**Module 1: Introduction to Machine Learning**

Machine learning models are algorithms trained on historical data to recognize patterns and make predictions without explicit programming. They're widely used in various industries:

Retail: Optimizing inventory, boosting sales with personalized recommendations, and adjusting pricing based on customer behavior.

Manufacturing: Enhancing predictive maintenance, quality control, and process optimization through sensor data analysis.

Traditional programming relies on predefined rules and offers more control but may lack flexibility. Machine learning, on the other hand, learns from data, offering adaptability but sometimes lacks interpretability. Performance depends on data quality and algorithm choice.

There are four types of machine learning algorithms:

**Supervised learning:** Trains models on labeled data for regression and classification tasks.

**Unsupervised learning:** Finds patterns in unlabeled data, useful for clustering and anomaly detection.

**Semi-supervised learning:** Combines labeled and unlabeled data for mapping functions, as seen in pseudo-labelling.

**Reinforcement learning:** Teaches agents to make decisions by interacting with an environment and receiving feedback.

These algorithms revolutionize decision-making processes across industries by learning from data to make accurate predictions and decisions.

**Outlines:**

1. Evaluate the suitability and estimate the performance of various machine learning models for specific tasks.

2. Formulate business problems and discern relevant data required for modeling frameworks.

3. Develop critical thinking skills to apply machine learning algorithms effectively in solving business problems.

**Module 2 : Introduction to R**

R programming is widely used for statistical computing and graphics, offering a broad range of techniques for data analysis, visualization, and modeling. It's commonly used by statisticians and data analysts.

To utilize R effectively, users typically install both R and RStudio, the latter providing an integrated development environment for R.

R packages extend R's functionality by bundling code, data, and documentation. They can be installed using the install.packages() function and loaded into the R session with library().

R features five basic atomic classes of objects: character, numeric, integer, complex, and logical. Additionally, it offers various data structures like atomic vectors, lists, matrices, data frames, factors, and tables.

* Atomic vector: it uses the c() function
* List: it is considered a special type of vector, and can contain elements of different classes.
* Factors: it is used to represent categorical data and can be unordered or ordered.
* Data frames: it used to store tabular data in R
* Matrices: they are two-dimensional vectors. The shape of a matrix is defended by two integers (number of rows, number of columns)

Data input and output operations often involve functions like read.csv() for reading CSV files and the library(readr) package.

Conditional statements in R, like "if-else" statements, utilize functions such as runif().

For loops and while loops are common control flow structures in R. A simple example of a for loop and a while loop is provided.

* Here are the samples for “for loop” and “while loop” functions:

# Simple for loop

for (i in 1:10) {

print(i)

}

# Simple while loop

i <- 1

while (i <= 10) {

print(i)

i <- i + 1}

**Outlines:** This module aims to equip learners with essential skills in utilizing R for implementing machine learning algorithms, data mining, visualization, and analytics. By the module's completion, participants will be proficient in installing Base R and its dependent libraries, enabling them to navigate the R-studio environment seamlessly. Additionally, learners will develop a keen understanding of key variable types in R and how to manipulate them effectively. Through practical exercises, participants will gain confidence in executing basic commands in R and importing/exporting data to and from the platform.

**Module 3: Fundamental Concepts in Supervised Learning**

Supervised machine learning involves finding a mapping function ℎ, which accurately predicts the output y based on input data x. A supervised learning model comprises three key components: problem representation, cost function (e.g., sum square of errors), and optimization method.

Overfitting occurs when a model excessively fits the training data, leading to poor generalization on new data, while underfitting arises when the model is too simplistic to capture data patterns.

To address overfitting, data is split into training, validation, and test sets. The training set trains the model, the validation set evaluates performance during hyperparameter tuning, and the test set provides an unbiased performance evaluation.

Leave-one-out cross-validation (LOOCV) trains the model on all data points except one, iteratively evaluating each omitted point's performance.

The caret package in R offers a comprehensive toolkit for training and assessing machine learning models, providing functions for data preprocessing, feature selection, model tuning, and performance evaluation. By loading the ISLR library and utilizing functions like createDataPartition(), caret enables efficient comparison and selection of models for diverse datasets.

**Outlines**: By the conclusion of this module, we will acquire the capability to discern and apply different methods for evaluating the performance of supervised learning models effectively. They will develop practical skills in implementing performance benchmarking techniques using the "Caret" package in R. Moreover, participants will comprehend the significance and role of supervised learning models in addressing various business challenges. emphasizing critical thinking in the application of machine learning algorithms, adeptness in formulating and solving business problems using relevant data, proficiency in evaluating model appropriateness and performance, effective utilization of software tools like R for implementing algorithms, and fostering clear communication and presentation of statistical findings.

**Module 4: k-NN Classification**

Classification and error metrics are fundamental concepts in evaluating the performance of classification models. In a binary classification scenario, observations can be categorized into true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). These metrics help quantify a model's accuracy, precision, recall, and specificity.

True Positive (TP): The observed value is positive, and we correctly predict it as positive.

True Negative (TN): The observed value is negative, and we correctly predict it as negative.

False Negative (FN): The observed value is positive, but we incorrectly predict it as negative.

False Positive (FP): The observed value is negative, but we incorrectly predict it as positive.

1.Accuracy = Ratio of (TP + TN) / N

2. Recall (sensitivity) = TP / (TP + FN)

3. Precision (positive predictive value) = TP / (TP + FP)

4. Specificity (true negative rate) = TN / (TN + FP)

A confusion matrix and crossTable() provides a tabular summary of model predictions compared to actual labels, offering insights into model performance in this we use “gmodels” package.

Forex: CrossTable(x=Test\_labels,y=Predicted\_Test\_labels, prop.chisq = FALSE)

ROC curves visualize the trade-off between true positive rate and false positive rate across different threshold settings in binary classification models.

K-Nearest Neighbors (KNN) is a supervised learning algorithm used for classification tasks, determining the class of a new data point based on the majority class of its K nearest neighbors in the training set. Common distance metrics include Euclidean, Manhattan, and Minkowski distances.

Normalization is a preprocessing step used to scale numeric features to a standard range, enhancing model performance and interpretability. Normalization in R is performed to scale numeric features to a standard range, ensuring all features contribute equally to the analysis. Various functions from different packages, such as scale() from base R, preProcess() from caret, and preProcess() from recipes, can be used for normalization. These functions help prepare data for analysis or modeling by standardizing numeric features. For normalization in r

Ex: Default\_norm <-predict(norm\_mdl, Default)

Dummy variables, or one-hot encoding, convert categorical data into numerical format, crucial for regression analysis and machine learning algorithms. Dummy variables are binary variables used to represent categorical data in a quantitative format. In R, they can be created using functions like model.matrix() or dummyVars() from the caret package. Dummy variables are important for regression analysis and machine learning algorithms that require numerical input data.

Parameters are internal variables optimized during model training, while hyperparameters are external settings controlling model behavior, typically tuned prior to training.

The value of K in KNN affects model bias, variance, and performance. Smaller K values lead to complex decision boundaries, potentially overfitting, while larger K values produce smoother boundaries but may oversimplify the model. Tuning K is essential for optimizing model performance, often done through techniques like grid search.

Implementing KNN in R involves data preparation, model training, evaluation, hyperparameter tuning, and prediction. Techniques such as cross-validation or grid search can be used for hyperparameter tuning. We use train() function to find the value of k.

Here's a basic overview of how to implement KNN in R:

Data Preparation: Ensure your dataset is properly formatted and preprocessed. Split the data into training and testing sets if necessary. For partitioning data into train and test we use1:

Index\_trn < - createDataPartition(Default\_norm$default, p=0.8, list = FALSE)

Model Training: Train the KNN model using the knn() function, specifying the training data, predictors, and the number of nearest neighbors (K).

Model Evaluation: Evaluate the performance of the KNN model using appropriate metrics such as accuracy, precision, recall, or ROC curves.

Hyperparameter Tuning: Optimize the performance of the KNN model by tuning the value of K using techniques like cross-validation or grid search.

Prediction: Once the model is trained and tuned, use it to make predictions on new data.

**Outlines**: By the module's end, we will master methods for evaluating supervised learning models, implementing performance benchmarks using the "Caret" package, and understanding the significance of these models in business. These outcomes support critical thinking, problem-solving, proficiency in model evaluation, effective software utilization, and clear communication of statistical results.

**Module 5: Naïve Bayes Classification**

Conditional probability is the probability of an event occurring given that another event has already occurred. It is denoted as P(A|B), read as "the probability of event A given event B."

Mathematically, conditional probability is calculated using the formula:

P(A∣B)= P(A∩B)/ P(B)

​Where:

P(A∣B) is the conditional probability of event A given event B.

P(A∩B) is the probability of both events A and B occurring (the intersection of A and B).

P(B) is the probability of event B occurring.

Independent events are events where the occurrence of one event does not affect the probability of the other event happening. In other words, the outcome of one event does not influence the outcome of the other.

Mathematically, two events A and B are independent if and only if:

P(A∩B)=P(A)×P(B) This means that the probability of both events A and B occurring is equal to the product of their individual probabilities.

The Naive Bayes classifier is a simple yet powerful probabilistic machine learning algorithm used for classification tasks. It is based on Bayes' theorem with an assumption of independence between the features. Despite its simplicity, Naive Bayes often performs well in practice, particularly for text classification and other tasks with high-dimensional feature spaces.

P(y|x1,….,xn) =p(x1,….,xn)p(y) / p(x1,….,xn)

The Naive Bayes model is a probabilistic classifier based on Bayes' theorem with the "naive" assumption of feature independence. It's used for classification tasks, where the goal is to predict the class label of a given instance based on its features.

Here's an overview of how the Naive Bayes model works:

Bayes' Theorem: The model is based on Bayes' theorem, which states:

P(y∣x)= P(x∣y)×P(y)/ P(x)

Where:

P(y∣x) is the probability of class y given the features x.

P(x∣y) is the probability of observing features x given class y.

P(y) is the prior probability of class y.

P(x) is the evidence, the probability of observing the features x.

In R, the Naive Bayes classifier is implemented through packages like e1071 or caret, which provide functions for model training, prediction, and evaluation.

#To build a naïve bayes classifiers we can use :

Nd\_model <- naiveBayes(default~balance+income,data = Train)

The AUC (Area Under the Curve) value and ROC (Receiver Operating Characteristic) curves are both metrics used to evaluate the performance of classification models, particularly binary classifiers, including Naive Bayes. It requires library(pROC).

AUC Value: The AUC represents the area under the ROC curve. It quantifies the model's ability to distinguish between positive and negative classes. A higher AUC value (closer to 1) indicates better discrimination performance, while a lower value suggests poorer performance. An AUC of 0.5 indicates a classifier that performs no better than random chance.

ROC Curves: ROC curves are graphical plots that illustrate the performance of a binary classification model across different threshold settings. They plot the true positive rate (sensitivity) against the false positive rate (1 - specificity) at various threshold values. A model with good discriminatory power will have an ROC curve that hugs the upper left corner of the plot, indicating high sensitivity and low false positive rate across various thresholds.

#to plot the roc curve we can use:

Plot.roc()

The Box-Cox transformation is a statistical method used to stabilize the variance and make data more normally distributed. It transforms data through a power transformation with a parameter λ, optimizing it to maximize normality. This technique is beneficial for addressing non-normality and heteroscedasticity in data, commonly applied in regression analysis and other statistical modeling tasks.

# to create a box-cox transformation model we can use below formula:

B\_C\_Trans <- preprocess(Default, method = “BoxCox”)

Print(B\_C\_Trans)

After print we will be able to get the lamda value

# to apply the model we can use:

Default\_Transformed =predict(B\_C\_Trans,Default)

Hypertuning these models is crucial for improving performance, typically involving techniques like grid search or random search to optimize hyperparameters.

**Outliers:** By the module's end, we will proficiently implement and optimize Naive Bayes classification models, utilizing them for predictive analysis. These skills support critical thinking in applying machine learning to business problems, formulating problems and identifying relevant data, evaluating model appropriateness and performance, effectively using software tools like R for implementation and communicating statistical findings clearly.

**Module 6: K-mean Clustering Algorithm**

Clustering is a type of unsupervised machine-learning technique used to group similar data points based on their characteristics or features. The goal of clustering is to partition a dataset into groups, or clusters, such that data points within the same cluster are more similar to each other than to those in other clusters. The main objective of clustering is to discover hidden patterns or structures in the data without prior knowledge of the class labels, the most common measure of distance is the Euclidean distance,

Clustering algorithms can be **hierarchical** or **non-hierarchical**. Hierarchical methods, such as agglomerative or divisive clustering, organize clusters into a natural hierarchy. Non-hierarchical methods, like k-means, require specifying the number of clusters beforehand and are preferred for large datasets due to lower computational intensity. However, determining the optimal number of clusters can be a challenge, which we'll address in this module.

**K-means** clustering is a widely-used unsupervised learning technique for dividing a dataset into a predefined number of clusters. Its goal is to group data points into K clusters, with each cluster represented by its centroid. The algorithm works by iteratively assigning data points to the nearest centroid and then updating the centroids based on the mean of the assigned points. It's commonly used in tasks like customer segmentation and image compression. However, K-means has drawbacks such as sensitivity to initial centroid selection and the assumption of spherical clusters. In R, you can implement it using the kmeans() function from the "**stats**" package.

**We learnt about the advantages and disadvantages of kmeans:**

Advantages:

Simple and efficient for large datasets.

Scales well to high-dimensional data.

Faster than hierarchical methods.

Suitable for datasets with spherical clusters.

Results are easy to interpret and visualize.

Disadvantages:

Sensitive to initial centroids.

Assumes clusters are spherical and similar in size.

Requires specifying K beforehand.

Not suitable for non-linear or irregular clusters.

Results may vary between runs, reducing robustness.

For clustering algorithms and visualization, we require a package called “factoextra” as it contains no. of useful functions:

1.fviz\_cluster(): to show the results of clustering.

2.fviz\_dist():finds the distance b/w the observations.

3.fviz\_nbclust(): visualize the clusters by using different methods.

Ex Formula: k6 <- kmeans(df, centers = 6, nstart = 30)

K6$centers # we can use this to find the centriods

K6$size # size of each clusters.

K6$cluster[135] # find the cluster of 135th observation in the dataset

fviz\_cluster(k6, data = df) #visualize the cluster

with the help of “flexclust” package we can apply K-means clustering using manhattan distance

Ex formula: k6 <- kcca(df, k=6, kccaFamily("kmedians"))

The number of clusters (K) is crucial in k-means clustering. It can be determined based on external considerations or data-driven methods. The goal is to minimize within-cluster variation (WSS), indicating smaller differences within clusters compared to between clusters.

Elbow Method: The elbow chart illustrates how WSS declines as more clusters are added. The "elbow point" where the rate of decline slows down signifies the optimal K. It balances model complexity (overfitting) with bias, with the elbow point representing the best compromise.

Ex formula: fviz\_nbclust(df, kmeans, method = "wss") # output would be in elbow shape by that we can find the number of clusters.

Silhouette Method: To validate the clustering assignment, the Silhouette Method measures how well each data point fits its assigned cluster compared to others. High silhouette widths indicate good clustering. In this case, a high value at K=6 suggests optimal clustering.

EX formula: fviz\_nbclust(df, kmeans, method = "silhouette")

Overall, k-means clustering requires careful consideration of K, balancing model complexity and bias. Both the elbow and silhouette methods offer data-driven approaches to determine the optimal number of clusters for a given dataset.

**Outlines**: By the module's conclusion, we will be proficient in applying the K-means clustering algorithm, understanding the role and tradeoffs of choosing the number of clusters (k), and interpreting the output of the model. They will also discuss real-world applications of distance clustering. These outcomes support critical thinking in applying machine learning to business problems, formulating problems and identifying relevant data, evaluating model appropriateness and performance, effectively using software tools like R for implementation, and communicating statistical findings clearly.

**Module 7: DBSCAN Clustering Algorithm**

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm designed to identify clusters in datasets with varying densities and shapes. Unlike K-means, DBSCAN does not require specifying the number of clusters beforehand and can find non-linearly separable clusters. It classifies points as core, border, or outlier based on their density within a specified neighborhood radius (ε) and minimum number of points (minPts). Core points form the core of clusters, while border points may also belong to clusters but lie on the edges, and outliers are points not part of any cluster.

DBSCAN iteratively expands clusters by adding neighboring points within the ε-neighborhood until all density-connected clusters are found. Despite its advantages, DBSCAN has disadvantages such as non-determinism for border points, dependency on distance measures, and challenges with datasets having large density differences. However, it offers benefits including no need to specify cluster number beforehand, ability to find arbitrarily shaped clusters, robustness to outliers, and requiring only two parameters (minPts and ε). Overall, while K-means is efficient and simple, DBSCAN provides more flexibility and robustness for handling complex datasets, albeit requiring careful parameter consideration.

Implementation of DBSCAN in r:

We require two packages called “dbscan and fpc”

EX formula: db <- dbscan::dbscan(moons, eps = 0.5, minPts = 5) # by using DBSCAN

using this we can find the cluster details.

plot(db, moons, main="DBSCAN", frame = FALSE) # plot the cluster details

db <- fpc::dbscan(df, eps = 0.15, MinPts = 5)# by using fpc package

db <- dbscan::kNNdistplot(df, k = 5) # by using kNNdisplot we can used for visualizing the distribution of k-nearest neighbor distances in a dataset.

**Outlines**: By the module's conclusion, we will be equipped to how we apply the DBSCAN clustering algorithm by fostering critical thinking using machine learning to business problems.

**Module 8: Hierarchical Clustering**

Hierarchical clustering is a method used to group similar objects into clusters based on their characteristics. This technique creates a hierarchical structure of clusters, where clusters at higher levels of the hierarchy contain fewer and more dissimilar objects, while clusters at lower levels are more specific and homogeneous. There are two main types of hierarchical clustering: agglomerative and divisive.

The result of hierarchical clustering is often represented visually using a dendrogram, which is a tree-like diagram showing the hierarchical relationships between clusters. Hierarchical clustering does not require specifying the number of clusters beforehand, making it useful for exploratory analysis. However, it can be computationally intensive, especially for large datasets, and sensitive to the choice of distance metric and linkage method used to measure the similarity between clusters. Despite these challenges, hierarchical clustering is widely used in various fields, including biology, social sciences, and data analysis.

Hierarchical clustering offers two main approaches: agglomerative (bottom-up) and divisive (top-down). In agglomerative clustering, clusters are merged based on their proximity, starting with individual data points and gradually combining them into larger clusters. Conversely, divisive clustering begins with all data points in a single cluster and recursively splits them into smaller clusters. Both approaches can yield similar results, but their time complexity may differ based on implementation.

The choice of distance measure is crucial, with options like Euclidean, Manhattan, and Mahalanobis distances, as well as similarity coefficients for categorical variables. Distances between clusters are calculated using methods like minimum, maximum, average, or centroid distance. Various linkage methods, such as single, complete, average, and centroid linkage, determine how clusters are merged based on inter-cluster distances. Additionally, Ward's method minimizes the loss of information when merging clusters, often resulting in convex clusters of roughly equal size. Overall, while both agglomerative and divisive clustering can be effective, agglomerative clustering is more commonly used in practice.

Implementation in R

Install hclust function in the “Stats” package and Agnes and Diana from “cluster” package

For Example:

# finding agnes and with different linkage methods

hc\_sin <- agnes(df, method = "single")

hc\_com <- agnes(df, method = "complete")

hc\_avg <- agnes(df, method = "average") # with the help of these we can able to find the best linkage method.

pltree(hc\_com, cex = 0.6, hang = -1, main = "Dendrogram of agnes") # with this we can plot dendogram of agnes

Using diana()function:

hc\_diana <- diana(df)

hc\_diana$dc # coefficient; amount of clustering structure found

pltree(hc\_diana, cex = 0.6, hang = -1, main = "Dendrogram of diana") with this we can plot dendogram of diana

**Outlines:** By module's end, we will comprehend the theoretical underpinnings of hierarchical clustering algorithms, elucidating their advantages such as flexibility in handling various cluster shapes and sizes, and inherent visualization of cluster hierarchy and a comparative analysis against k-means and DBSCAN methods, delineating the differences in approach and suitability for different data structures. Additionally, we will acquire proficiency in implementing hierarchical clustering models in R, interpreting dendrograms and cluster structures effectively. These theoretical insights and practical skills empower learners to think critically about algorithm selection, formulate problems, evaluate model performance, utilize software tools for implementation, and communicate findings clearly, aligning with the course learning outcomes.

**Module 9: Recommended Systems**

Association rules are utilized in data mining and machine learning to unveil meaningful relationships or patterns within extensive datasets. These rules, often presented as "if-then" statements, highlight associations between items in the data. A prevalent application of association rules lies in market basket analysis, where they unearth connections between products frequently bought together, thereby offering insights into consumer behavior and a study of “what goes with what.”

The Apriori algorithm stands as the foremost method for generating association rules. It operates through iterative steps, constructing candidate itemsets and subsequently trimming them based on a minimum support threshold to identify frequent itemsets. From these frequent itemsets, association rules are derived, guided by a minimum confidence threshold. Through this process, the algorithm efficiently uncovers valuable associations within transactional data.

Several measures are commonly used to evaluate association rules:

Support measures the frequency of a specific association:

Support = Count(X∩Y)/n

Confidence quantifies the strength of the association:

Confidence(X🡪Y) = Support(X∩Y)/SupportX

Lift compares observed support to expected independence:

Lift(X🡪Y) = Support(X∩Y)/Support(X)\*Support(Y)

Leverage measures deviation from independence:

Leverage(X🡪Y) = Support(X∩Y) - Support(X)\*Support(Y)

These formulas provide concise representations of each evaluation measure used in association rule mining.

Implementation in R:

To use Apriori algorithm we required two packages: “arules” and “arulesViz”

For Example: we can use this command :

Ar <- apriori(Groceries,parameter = list(supp = 0.001, conf = 0.80))

plot(Ar) # if we plot this rules we will get the scatter plot.

The scatterplot from the Groceries dataset displays support on the horizontal axis and confidence on the vertical axis. The shading represents lift, indicating the strength of association between antecedent and consequent itemsets. Interestingly, the plot reveals that the highest lift occurs at a mid-level of support and confidence, suggesting a balanced relationship between the frequency of occurrence and the strength of association in the dataset.

If we want to inspect anything we can use this command: for EX: inspect(Ar[1:10])

If you want to display the top 20 items, you can use this command:

for EX: itemFrequencyPlot(Groceries,topN=20,main='Relative Item Frequency

Plot',type="relative",ylab="Item Frequency (Relative)")

Inorder to visualize we can use this methods: for Ex:

plot(Ar[1:20], method = "paracoord")

plot(Ar [1:20], method = "graph")

Outlines: By module's end, we will grasp the distinction between different methods for estimating supervised learning model performance and will be able to implement various performance benchmarking techniques using the "Caret" package in R and also comprehend the role of Apriori algorithm by using arules” and “arulesViz packages and also its