MD2201 Data Science Course Project

**Science**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S.No** | **Div** | **Batch No** | **Group No** | **Roll No** | **Gr.No** | **Name of Student** |
| 1 | B | 2 | 5 | 35 | 12210570 | Pratik Mandalkar |
| 2 | 36 | 12210569 | Pratik Mane |
| 3 | 38 | 12210540 | Netra Mohekar |
| 4 | 55 | 12210864 | Pranav Patel |
| 5 | 60 | 12211463 | Harshwardhan Patil |

1. **Project Title:** Employee Retention
2. **Data Set Name:** IBM HR Analytics Employee Attrition & Performance
3. **Data Set Source:** Kaggle
4. **Data set Link:** https://www.kaggle.com/datasets/pavansubhasht/ibm-hr-analytics-attrition-dataset/
5. **Data Set Description:**

Number of Rows: 1470

Number of Columns: 35

No missing values (all columns have zero missing values)

Class imbalance exists (Attrition column seems to have imbalanced classes)

**Data Types of Each Variable:**

Age: Integer

Attrition: Categorical (Yes/No)

BusinessTravel: Categorical (Travel Rarely, Travel Frequently, Non-Travel)

DailyRate: Integer

Department: Categorical (Research & Development, Sales, Human Resources)

DistanceFromHome: Integer

Education: Integer

EducationField: Categorical (Life Sciences, Medical, Marketing, Technical Degree, Other)

EmployeeCount: Integer (seems constant, possibly insignificant)

EmployeeNumber: Integer (possibly an identifier)

EnvironmentSatisfaction: Integer

Gender: Categorical (e.g., Male, Female)

HourlyRate: Integer

JobInvolvement: Integer

JobLevel: Integer

JobRole: Categorical (Sales Executive, Research Scientist, Laboratory Technician, Manufacturing Director, Healthcare

Representative, Manager, Sales Representative, Research Director, Human Resources)

JobSatisfaction: Integer

MaritalStatus: Categorical (Single, Married, Divorced)

MonthlyIncome: Integer

MonthlyRate: Integer

NumCompaniesWorked: Integer

Over18: Categorical (e.g., Y)

OverTime: Categorical (e.g., Yes, No)

PercentSalaryHike: Integer

PerformanceRating: Integer

RelationshipSatisfaction: Integer

StandardHours: Integer (seems constant, insignificant)

StockOptionLevel: Integer

TotalWorkingYears: Integer

TrainingTimesLastYear: Integer

WorkLifeBalance: Integer

YearsAtCompany: Integer

YearsInCurrentRole: Integer

YearsSinceLastPromotion: Integer

YearsWithCurrManager: Integer

**Categories of Variables:**

* Categorical Variables: BusinessTravel, Department, EducationField, Gender, JobRole, MaritalStatus, Over18, OverTime
* Binary Variables: Attrition, Over18, OverTime
* Numeric Variables: Age, DailyRate, DistanceFromHome, Education, EmployeeCount, EmployeeNumber, EnvironmentSatisfaction, HourlyRate, JobInvolvement, JobLevel, JobSatisfaction, MonthlyIncome, MonthlyRate, NumCompaniesWorked, PercentSalaryHike, PerformanceRating, RelationshipSatisfaction, StandardHours, StockOptionLevel, TotalWorkingYears, TrainingTimesLastYear, WorkLifeBalance, YearsAtCompany, YearsInCurrentRole, YearsSinceLastPromotion, YearsWithCurrManager

**Missing Values:**

No missing values in any column.

**Class Imbalance:**

The Attrition column indicates class imbalance, with "Yes" (indicating attrition) and "No" classes. The "Attrition" column indicates a significant class imbalance:

"Yes" values: 237 instances.

"No" values: 1233 instances.

1. **Description of Work Done:**

The project's workflow was meticulously designed to prepare and optimize a dataset for predictive modeling. It began with comprehensive data collection efforts to acquire a dataset suitable for analysis. Subsequent steps focused on stringent data preprocessing techniques, which included identifying and addressing null or NA values and managing class imbalance through various methodologies such as random oversampling, random undersampling, and their combinations, alongside the utilization of OVUM techniques. Additionally, outlier detection and removal were conducted using the z-score method to enhance dataset integrity. Through normalization procedures, the data distribution across features was standardized, ensuring uniformity and facilitating accurate model training.

The dataset was then partitioned into training and testing subsets using multiple splits, namely 70/30, 75/25, and 80/20, to accommodate diverse modeling strategies. Following this, six distinct algorithms—kNN, logistic regression, naive Bayes, decision tree, random forest, and SVM—were implemented for model training. Furthermore, various feature selection techniques, including CFS, chi-square, Lasso, RFE, importance score, genetic algorithm, and information gain, were systematically investigated to identify the most discriminative features for modeling purposes.

Finally, to optimize model performance and enhance predictive accuracy, hyperparameter tuning was conducted across all algorithms. This methodical approach ensured the meticulous preparation and refinement of the dataset, laying a solid foundation for robust predictive modeling and informed decision-making processes.

1. **Literature Survey:**
2. Intelligent Employee Retention System for Attrition Rate Analysis and Churn Prediction:

Journal: JGIM

Dataset used: Consumer goods (FMCG) company.

Data preprocessing: Employ CTC level, promotion, No NA values, Fuzzy-Analytical Hierarchy Process

Feature Selection: Employee Satisfaction, Appraisal\_rating, Number of projects/tasks assigned per quarter, Time spent per project per quarter, Promotion, Employee\_CTC\_level, Safety measure

Algorithm: Random forest and gradient boosting, Deep neural network (DNN)

Evaluation Parameters: Accuracy

Findings: Employee turnover status

1. HR Analytics: Employee Attrition Analysis using Random Forest:

Journal: IJPE

Dataset used: A real dataset provided by IBM analytics

Data preprocessing: SMOTE

Feature Selection: Current role, years with current manager, monthly income & total working years, age and total working years, percent salary hike, and performance rating

Algorithm: Random forest

Evaluation Parameters: Accuracy, CV score, Sensitivity, Specificity, ROC score

1. Predicting Employee Attrition Using Machine Learning Techniques:

Journal: MDPI

Dataset used: A real dataset provided by IBM analytics

Data preprocessing: Not specified

Feature Selection: Age, Monthly income, Attrition, Monthly rate Business travel, Number of previous employers

Algorithm: Gaussian Naïve Bayes, Naïve Bayes classifier for multivariate Bernoulli models, Logistic Regression classifier, K-nearest neighbours (K-NN)

Evaluation Parameters: Gaussian Naïve Bayes confusion matrix

Findings: Monthly income, age, overtime, and distance from home were important predictors, with Gaussian Naïve Bayes classifier showing the best performance.

1. From Big Data to Deep Data to Support People Analytics for Employee Attrition Prediction:

Journal: IEEE Access

Dataset used: IBM HR analytics attrition dataset, Human resource analytics Kaggle Dataset

Data preprocessing: RFE, SelectKBest

Feature Selection: Not specified

Algorithm: DT, LR, SVM, DNN, LSTM, CNN, RF, XGB, VC, Stacked

Evaluation Parameters: Accuracy

Findings: VC achieved the highest accuracy of 98%.

1. Predicting Employee Attrition Using Machine Learning Approaches:

Journal: Applied Sciences

Dataset used: IBM HR analytics attrition dataset

Data preprocessing: Not specified

Feature Selection: Max-out

Algorithm: SVM, LR, ETC, DTC

Evaluation Parameters: SVM-88%, ETC-93%, LR-74%, DTC-84%

1. A Machine Learning Application for Human Resource:

Journal: Verlag Berlin Heidelberg

Dataset used: Not mentioned

Data preprocessing: No outliers

Feature Selection: Distributed algorithm, linear programming model, goal programming model, Fuzzy Model

Algorithm: Fuzzy modeling

Evaluation Parameters: Task-Arrange or Hungarian Algorithm gives the best allocation.

1. Employee attrition prediction using logistic regression:

Journal: IJRASET

Dataset used: Kaggle (IBM HR Analytics)

Data preprocessing: Not specified

Feature Selection: Age, job satisfaction, monthly income, years

Algorithm: Naive Bayes, K-NN, logistic regression

Evaluation Parameters: Accuracy rate

1. Study and Prediction Analysis of the Employee Turnover using Machine Learning Approaches:

Journal: IEEE Xplore

Dataset used: IBM Watson Analytics1

Data preprocessing: "dropna()" in pandas

Feature Selection: Not specified

Algorithm: Gradient Booster, KNN, Logistic, Naïve Bays, Random F., SVC

Evaluation Parameters: Precision and Accuracy

1. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities:

Journal: Journal of Internet Services and Applications

Dataset used: NSF TeraGrid dataset, Internet traffic collected at the POP of an ISP network

Data preprocessing: Outliers different in different dataset

Feature Selection: Max, Min, Avg load observed in past 10 s ∼30 s, Link load observed at τ time scale, N past days hourly traffic volume, Past measurements

Algorithm: Supervised MLP-NN

Evaluation Parameters: Confusion matrix, accuracy, precision, recall, and f measure

1. A Study on Employee Retention as a Tool for Improving Organizational Effectiveness:

Journal: IJMTS

Dataset used: Not mentioned

Data preprocessing: No outliers

Feature Selection: Pearson Chi-Square, Continuity Correction, Likelihood Ratio, Fisher's Exact Test, Linear-by-Linear, Association N of Valid Cases

Algorithm: Not specified

Evaluation Parameters: Accuracy

1. Employee retention prediction in corporate organizations using machine learning methods:

Journal: Academy of Entrepreneurship Journal

Dataset used: Employment history of 613 employees (specific name is not mentioned)

Data preprocessing: Not specified

Feature Selection: Not specified

Algorithm: Random Forest, Support Vector Mechanism, Decision Trees, K-Nearest Neighbour, and Logistic Regression

Evaluation Parameters: Accuracy, precision, recall, and F1 score

Findings: The OOB estimate error rate = 0.36%, AUC = 0.9927, confidence level for the model = 95%

1. Predictive Analytics of Employee Attrition using K-Fold Methodologies:

Journal: ijmsc

Dataset used: Kaggle (no specific name mentioned)

Data preprocessing: Not specified

Feature Selection: Not specified

Algorithm: Cat Boost, Random Forest (RF), Light GBM

Evaluation Parameters: ROC curves

Findings: Accuracy of 83.67% for LightBoost, 70-30 split dataset

1. Predict Employee Retention Using Data Science:

Journal: ijeecse

Dataset used: Not mentioned

Data preprocessing: Deleting the entire row

Feature Selection: Not specified

Algorithm: Regression analysis

Evaluation Parameters: Confusion matrix and accuracy score

Findings: Predict employee retention with an outcome of 97%

1. Intelligent Employee Retention System for Attrition Rate Analysis and Churn Prediction:

Journal: Journal of Global Information Management

Dataset used: Not mentioned

Data preprocessing: Not mentioned

Feature Selection: Random Forest, Gradient Boost, and Deep Neural Network (DNN)

Algorithm: Fuzzy Analytical Hierarchy Process, confusion matrix, outcome of NNM

Evaluation Parameters: Adjusted R-squared value of 0.79 suggests that 79% of the variance

1. Predicting and explaining employee turnover intention:

Journal: International Journal of Data Science and Analytics

Dataset used: IBM HR analytics attrition dataset, Human resource analytics Kaggle Dataset

Data preprocessing: Mode imputation

Feature Selection: Not specified

Algorithm: LR, LGBM, XAI, DT, KNN

Evaluation Parameters: Not specified

1. An employee retention model using organizational network analysis for voluntary turnover:

Journal: Social Network Analysis and Mining

Dataset used: Not mentioned

Data preprocessing: Not specified

Feature Selection: Not specified

Algorithm: Not specified

Evaluation Parameters: Not specified

1. Employee Performance Prediction Using EPP Framework:

Journal: IJSRCSEIT

Dataset used: Not mentioned

Data preprocessing: Not specified

Feature Selection: Not specified

Algorithm: Naive Bayes

Evaluation Parameters: Not specified

A Study on Employee Retention as a Tool for Improving Organizational Effectiveness:

1. Journal: IJMTS

Dataset used: Not mentioned

Data preprocessing: Not specified

Feature Selection: Not specified

Algorithm: Not specified

Evaluation Parameters: Accuracy

1. Identifying factors for employee retention using computational techniques: an approach to assist the decision-making process:

Journal: SN Applied Sciences

Dataset used: Not mentioned

Data preprocessing: Not specified

Feature Selection: Not specified

Algorithm: Data mining

Evaluation Parameters: Not specified

1. **Data Preprocessing:**
2. **Handle Missing Values**

In the datasets utilized for this analysis, there are no missing values present. Despite the absence of missing data, understanding methods for handling them is still crucial for future datasets where missing values may arise. This preemptive knowledge ensures the ability to effectively manage and analyze data, maintaining the integrity and reliability of results. Common strategies for handling missing data include data imputation, where missing values are estimated based on existing data, and removal of missing values, though unnecessary in this particular dataset. By being prepared to address missing data challenges, researchers can ensure robust and accurate analyses, even in the absence of missing values in the initial dataset.

1. **Encode Categorical Variables**

In the process of preparing the dataset for analysis, one crucial step involves encoding categorical variables. This process aims to convert categorical data into a numerical format that can be easily interpreted by analytical models. Initially, the categorical columns within the dataset are identified, including attributes such as "Department," "BusinessTravel," "EducationField," "Gender," "JobRole," "MaritalStatus," and "OverTime." Subsequently, binary responses in the "Attrition" column are transformed into numeric values to facilitate analysis; "Yes" is encoded as 1, while "No" is encoded as 0. Additionally, all specified categorical columns, including "Attrition," are converted into factors. This conversion ensures that each categorical variable is represented as a discrete factor with distinct levels, enabling the dataset to be effectively utilized in subsequent analytical processes. Through the encoding of categorical variables, the dataset becomes structured and ready for further exploration, enabling the discovery of insights and patterns within the data.

1. **Handle Class Imbalance**

Handling class imbalance in machine learning datasets is crucial for ensuring model performance isn't skewed by disproportionate class representation. Various techniques address this challenge:

Random oversampling involves duplicating minority class instances randomly until balance is achieved. This boosts minority class representation without losing information, mitigating bias toward the majority class. Conversely, random undersampling involves randomly discarding majority class instances to restore balance. While computationally efficient, it risks discarding valuable data and reducing model accuracy.

A combined approach merges oversampling of the minority class with undersampling of the majority class. This method aims to balance the dataset while minimizing the drawbacks of individual oversampling and undersampling techniques. Specialized oversampling techniques like SMOTE synthesize new minority class instances by interpolating between existing ones, reducing overfitting risks associated with random oversampling. Meanwhile, undersampling methods like NearMiss select majority class instances closest to minority class instances, reducing the imbalance ratio while preserving majority class structure. By implementing these methods, practitioners can effectively tackle class imbalance, enhancing model performance and ensuring more accurate predictions across all classes.

1. **Normalize Data**

In the project, the normalization technique used is Z-score normalization, also known as standardization. This technique is applied to numerical features within the dataset to ensure that they are on a comparable scale, with a mean of 0 and a standard deviation of 1.

The normalization process involves the following steps:

1. **Calculation of Z-scores:**

For each numerical feature, the Z-score is calculated. This is done by subtracting the mean of the feature from each data point and then dividing by the standard deviation of the feature. The formula for calculating the Z-score of a data point x in a feature is:

The Z-score represents the number of standard deviations away from the mean each data point is.

1. **Normalization of Data:**

By applying the Z-score formula to each numerical feature, the values are transformed to have a mean of 0 and a standard deviation of 1. This standardizes the numerical features and brings them to a consistent scale.

1. **Normalization Impact:**

Normalization using Z-score ensures that numerical features with different units or scales are treated equally in the analysis and modeling process. It prevents features with larger magnitudes from dominating the analysis and helps avoid biases in model performance.

1. **Implementation in the Project:**

In the project, Z-score normalization is applied to the dataset's numerical features using the scale function in R. This function calculates the Z-score for each numerical column and standardizes the data accordingly.

By normalizing the numerical features using Z-score normalization, the project ensures that the data is appropriately scaled and ready for analysis and modeling. This preprocessing step enhances the performance and interpretability of machine learning models by eliminating potential biases arising from differences in feature scales.

The effectiveness of Z-score normalization can be observed through the reduction in the number of rows after outlier handling. For the over-sampled dataset, the number of rows decreased from 2418 to 2260 after applying the Z-score method. Similarly, for both over and under-sampled datasets, the number of rows reduced from 1000 to 938. This reduction indicates that outliers, identified using Z-score normalization, were successfully handled, leading to a more refined dataset for subsequent analysis and modeling.

1. **Validate Data Integrity**

Validating data integrity is a crucial aspect of data analysis, ensuring that the dataset is reliable, accurate, and free from errors. This process involves several key steps to confirm the completeness, consistency, and correctness of the data. Firstly, a completeness check is performed to ensure that all expected data fields are present and populated within the dataset, guarding against missing or incomplete information that could skew analysis results. Next, consistency checks are conducted to verify that data values adhere to predefined rules or constraints, such as valid formats for numeric values or correct categorization for categorical variables. Moreover, accuracy checks are carried out to validate data entries against external sources or known standards, identifying any discrepancies or outliers that may impact analysis outcomes. Finally, cross-field validation ensures that relationships between different data fields are logical and consistent, further enhancing the reliability of the dataset. By systematically validating data integrity through these measures, analysts can trust the quality of the dataset and the validity of insights derived from it, thereby bolstering the credibility and usefulness of subsequent analyses or models.

1. **Feature Selection:**
2. ***C*orrelation-Based Feature Selection (CFS)**

The Correlation-based Feature Selection (CFS) method is a feature selection technique that evaluates the relevance and redundancy of features based on their correlation with the target variable and with each other. CFS is implemented to select the most informative features for predicting employee attrition. It calculates the correlation of each feature with the target variable ("Attrition") and evaluates the redundancy among features. The selected features, determined to be the most relevant and least redundant, are then printed as the final output. This process aids in identifying key predictors of employee attrition while minimizing overfitting and maximizing predictive accuracy.

Features Selected:

"Age", "BusinessTravel", "EnvironmentSatisfaction", "JobInvolvement", "JobLevel", "MonthlyIncome", "OverTime", "StockOptionLevel", "YearsAtCompany", "YearsWithCurrManager"

1. **Chi Square (χ²) test**

The provided code implements the chi-square test for feature selection. In short, the chi-square test assesses the independence between categorical variables, making it suitable for identifying relevant features in classification tasks. Specifically, the chi\_square function calculates the chi-square statistic for each feature, measuring its association with the target variable (Attrition). The features are then sorted based on their chi-square scores, and the top features are selected for further analysis. Finally, the selected features are printed as the output. This approach helps in identifying the most informative features for predicting employee attrition in the dataset.

Features Selected:

"MonthlyRate", "MonthlyIncome", "DailyRate", "Age", "TotalWorkingYears", "JobRole", "YearsAtCompany", "OverTime", "HourlyRate", "JobLevel"

1. **Information Gain**

The code uses the Information Gain method for feature selection. Information Gain quantifies how much a feature contributes to predicting the target variable by measuring the reduction in uncertainty (entropy) of the target when the feature is known. In the code, the information.gain function evaluates the information gain of each feature with respect to predicting attrition in the dataset. The resulting selected\_features object contains the information gain values for each feature, aiding in the identification of the most predictive features for employee attrition.

Features Selected:

"Age", "Department", "BusinessTravel", "DistanceFromHome", "EducationField", "EnvironmentSatisfaction", "Gender", "HourlyRate", "JobInvolvement", "JobLevel", "JobRole", "JobSatisfaction", "MaritalStatus", "MonthlyIncome", "MonthlyRate", "NumCompaniesWorked", "OverTime", "StockOptionLevel", "TotalWorkingYears", "WorkLifeBalance", "YearsAtCompany", "YearsInCurrentRole", "YearsWithCurrManager"

1. **Lasso**

The code utilizes LASSO (Least Absolute Shrinkage and Selection Operator) for feature selection. LASSO introduces a penalty term to encourage sparse coefficient vectors, effectively performing feature selection by shrinking less important features' coefficients towards zero. In this context, LASSO is applied to predict employee attrition based on various features. The cv.glmnet function performs cross-validated LASSO regression to select the optimal features, resulting in a list of the top selected features printed in the output.

Features Selected:

"Age", "BusinessTravel", "DailyRate", "Department", "DistanceFromHome", "Education", "EducationField"

1. **Genetic Algorithm**

The code employs a Genetic Algorithm (GA) for feature selection. GA is a metaheuristic inspired by natural selection and genetics, used to find optimal solutions to optimization and search problems. In this context, GA iteratively evolves a population of potential feature subsets, evaluating their fitness using a user-defined fitness function. The fitness function in this code trains a logistic regression model on the selected features and returns the negative log-likelihood as the fitness value. GA then evolves the population through selection, crossover, and mutation operations over multiple generations to find the feature subset with the highest fitness. The resulting selected features are printed as the output, aiding in identifying the most relevant features for predicting employee attrition.

Features Selected:  
"EmployeeNumber", "Gender", "MonthlyRate", "NumCompaniesWorked", "PerformanceRating", "TrainingTimesLastYear"

1. **Random Forest’s Importance Score**

Random Forest's importance score quantifies the significance of features in predicting the target variable. Random Forest model is trained using the randomForest function, and importance scores for each feature are computed using the importance function. The top 10 features with the highest importance scores are then selected and printed, revealing the most influential predictors of employee attrition within the dataset.

Features Selected:

"MonthlyIncome", "JobRole", "OverTime", "Age", "DailyRate", "HourlyRate", "MonthlyRate", "TotalWorkingYears", "YearsAtCompany", "DistanceFromHome"

1. **Recursive Feature Elimination (RFE)**

RFE (Recursive Feature Elimination) is a feature selection technique that iteratively removes less important features from the dataset until the optimal subset of features is identified. RFE is implemented using the rfe function from the caret package. This function ranks features based on their importance using Random Forest as the underlying model. The selected features, representing the most predictive subset for the target variable ("Attrition"), are then printed as the output.

Features Selected:

"OverTime", "JobRole", "HourlyRate", "DailyRate", "MonthlyRate", "MonthlyIncome", "Age", "PercentSalaryHike", "JobSatisfaction", "DistanceFromHome", "StockOptionLevel", "EducationField", "NumCompaniesWorked", "EnvironmentSatisfaction", "RelationshipSatisfaction", "Education", "WorkLifeBalance", "TrainingTimesLastYear", "TotalWorkingYears", "YearsAtCompany", "YearsWithCurrManager", "MaritalStatus", "JobInvolvement", "YearsSinceLastPromotion", "BusinessTravel", "YearsInCurrentRole", "JobLevel", "Gender"

1. **Algorithms Implemented:**
2. **k-Nearest Neighbors (kNN):**

The k-Nearest Neighbors (kNN) algorithm is a simple yet effective method used for classification or regression tasks. It determines the class or value of a query point by examining the labels of its nearest neighbors. kNN is intuitive, easy to implement, and suitable for small to medium-sized datasets, but its performance depends on the choice of distance metric and the number of neighbors (k).

In algorithmic configurations, default parameters are preset values used when specific parameters aren't provided by the user. For instance, in default usage of the k-Nearest Neighbors (KNN) algorithm, the model is trained with a default value of k, often set to 5, without explicit tuning. This method provides a quick implementation for rapid prototyping or when detailed parameter optimization isn't necessary.

In contrast, hyperparameter tuning involves systematically exploring parameter spaces to find values that enhance model performance. In the case of KNN, a grid search methodology evaluates various values of k, typically ranging from 1 to 20, to identify the optimal k value. This optimized parameter selection process results in potentially improved model performance compared to default settings, catering to specific dataset and task requirements.

1. **Decision Tree**

Decision Trees are versatile machine learning models used for classification and regression tasks. They construct a tree-like structure by recursively partitioning the data based on feature attributes, making them interpretable and suitable for both numerical and categorical data. However, they can overfit and may require pruning or regularization techniques to improve generalization.

In the default parameter scenario, the decision tree model is trained using default settings. Specifically, the rpart function from the rpart package is utilized to build the decision tree. By default, this function employs a complexity parameter (cp) of 0.01, which controls the tree's complexity and potential overfitting. The default settings are applied without explicit specification, resulting in a decision tree model trained with default parameter values. This approach is straightforward and requires minimal user intervention but may not necessarily yield the best-performing model, as the default parameter values are generic and not tailored to the specific dataset.

Contrastingly, the hyperparameter tuning scenario involves systematically optimizing the decision tree model's parameters to improve its performance. Here, the complexity parameter (cp) is tuned using a grid search approach, varying from 0.01 to 0.5 in increments of 0.01. The train function from the caret package is employed, incorporating the rpart method for building decision trees along with the trainControl function to specify the cross-validation strategy. By searching for the optimal cp value, the hyperparameter tuning process aims to mitigate overfitting and enhance the model's generalization capability. This approach is critical as it helps identify the most suitable model configuration for the given dataset, potentially leading to improved predictive performance and robustness.

1. **Logistic Regression**

Logistic regression is a predictive modeling technique used for binary classification tasks. It estimates the probability of a binary outcome based on input features using the logistic function. It's valued for its simplicity, interpretability, and effectiveness in handling linear and nonlinear relationships between variables.

In the default parameter scenario, logistic regression models are trained using default settings. The glm function from the stats package is utilized to build the logistic regression model. By default, this function employs a family parameter set to "binomial" for binary classification tasks. The default settings are applied without explicit specification, resulting in logistic regression models trained with default parameter values. This approach is straightforward and requires minimal user intervention but may not necessarily yield the best-performing model, as the default parameter values are generic and not tailored to the specific dataset.

Contrastingly, the hyperparameter tuning scenario involves systematically optimizing logistic regression model parameters to improve performance. Here, the glmnet method is employed for logistic regression along with the train function from the caret package. The alpha and lambda hyperparameters are tuned using a grid search approach, varying across a specified range. By searching for optimal hyperparameter values, the hyperparameter tuning process aims to enhance model generalization and performance. This approach is crucial as it helps identify the most suitable model configuration for the given dataset, potentially leading to improved predictive performance and robustness.

1. **Naïve Bayes**

Naive Bayes is a simple yet powerful probabilistic classifier based on Bayes' theorem with strong independence assumptions between features. It calculates the probability of a given class label based on the presence of various features. Despite its simplicity, Naive Bayes often performs well in practice, particularly with high-dimensional data and large datasets.

In the default parameter scenario, a Naive Bayes model is trained using default settings. The naive\_bayes method from the caret package is employed to build the Naive Bayes model. Naive Bayes is inherently simple and does not have any hyperparameters to tune. Therefore, there are no specific hyperparameters to configure for this model. By default, the model is trained using the data provided without any further customization. This approach is straightforward and requires minimal user intervention. Naive Bayes models are known for their simplicity and efficiency in handling large datasets with high-dimensional feature spaces. Despite their simplicity, they can perform well in many classification tasks, particularly when the independence assumption holds true among the features.

Unlike some other algorithms, Naive Bayes does not typically involve hyperparameter tuning. This is because Naive Bayes models make strong assumptions about the independence of features, and there are no parameters to adjust or optimize beyond the default settings. Therefore, in this scenario, the default model is used without any hyperparameter tuning. However, the model's performance can still be evaluated using standard metrics such as accuracy, sensitivity, precision, and specificity, which are calculated based on predictions made on the test set.

1. **Random Forest**

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (classification) or the mean prediction (regression) of individual trees. It builds each tree independently by randomly selecting features to split on and combining their predictions through averaging or voting, resulting in improved accuracy and robustness compared to a single decision tree.

The Random Forest model is trained using default parameter values, which are carefully chosen to provide a robust and reliable performance out of the box. These default settings have been fine-tuned and optimized by experts to deliver effective results across a wide range of datasets and scenarios without requiring manual adjustments. By leveraging these default parameters, users can effortlessly build accurate predictive models without the need for extensive parameter tuning or customization. This approach simplifies the model-building process and streamlines the workflow, making it accessible to users of varying levels of expertise. Additionally, default parameters often strike a balance between model complexity and computational efficiency, ensuring that the model achieves high performance while remaining computationally tractable. Overall, relying on default parameters enables users to quickly deploy Random Forest models with confidence, knowing that they are leveraging tried-and-tested settings that yield strong predictive performance.

1. **SVM**

SVM, or Support Vector Machine, is a supervised learning algorithm used for classification and regression tasks. It works by finding the optimal hyperplane that separates different classes in the feature space while maximizing the margin between them. SVM can handle both linear and nonlinear data through the use of kernel functions, allowing it to map the input data into higher-dimensional spaces where linear separation is possible. It is effective in scenarios with high-dimensional data and is robust against overfitting, especially when the number of features is greater than the number of samples

The default SVM model, encompassing polynomial, radial, and linear kernels, is instantiated using the svmPoly, svmRadial, and svmLinear methods from the caret package. In default mode, these SVM models are constructed without explicit tuning of hyperparameters, relying on standard settings provided by the algorithms. This entails employing default values for parameters such as the degree of the polynomial kernel, scaling factor, regularization parameter (C), and the gamma parameter for the radial kernel. The models are trained on the provided dataset without any specific adjustments or optimizations.

In contrast, the hyperparameter-tuned SVM models undergo an extensive search for optimal parameter configurations using a grid search technique. The hyperparameter grid, specified through the tuneGrid argument in the train function, explores various combinations of parameters such as the degree of the polynomial kernel, scaling factor, regularization parameter (C), and gamma parameter for the radial kernel. Hyperparameter tuning aims to systematically optimize the SVM models' performance by iteratively adjusting these parameters based on cross-validated performance metrics. This process involves evaluating multiple candidate models to identify the configuration that maximizes classification accuracy and generalization on the dataset. Hyperparameter tuning enhances the SVM models' adaptability to diverse datasets and improves their ability to capture complex patterns in the data.

1. **Code***:*

**Note***:*

1. code **a** to **e** are for class imbalance handling.
2. code **f** to **l** are for feature selection techniques.
3. code **m** is for data preprocessing to be appended before each algorithm given in **n** to **cc**.
4. code **dd** is for predictions and evaluation to be appended after each code given in **n** to **cc**.
5. code **ee** is about the shiny app integrated with SVM algorithm with Polynomial kernal.
6. **Random oversampling of original dataset and saving it.**

library(randomForest)

d <- read.csv("Dataset.csv")

d$Attrition <- as.character(d$Attrition)

minority\_attrition <- subset(d, Attrition == "Yes")

majority\_attrition <- subset(d, Attrition == "No")

oversampled\_minority\_attrition <- minority\_attrition[sample(nrow(minority\_attrition), nrow(majority\_attrition), replace = TRUE), ]

balanced\_data\_attrition <- rbind(oversampled\_minority\_attrition, majority\_attrition)

write.csv(balanced\_data\_attrition, file = "random\_over\_sampled.csv", row.names = FALSE)

model <- randomForest(Attrition ~ ., data = balanced\_data\_attrition)

1. **Random undersampling and saving it.**

d <- read.csv("Dataset.csv")

# Assuming 'd' is your dataset with a column 'Attrition' indicating the class

d$Attrition <- as.character(d$Attrition) # Convert 'Attrition' to character if it's a factor

minority\_attrition <- subset(d, Attrition == "Yes")

majority\_attrition <- subset(d, Attrition == "No")

# Randomly sample from the majority attrition

sampled\_majority\_attrition <- majority\_attrition[sample(nrow(majority\_attrition), nrow(minority\_attrition)), ]

# Combine minority and sampled majority

balanced\_data\_attrition <- rbind(minority\_attrition, sampled\_majority\_attrition)

# Save balanced data to a new CSV file

write.csv(balanced\_data\_attrition, file = "random\_under\_sampled.csv", row.names = FALSE)

1. **Both Over and Under sampling and saving it.**

library(ROSE)

data <- read.csv("Dataset.csv")

# Check the structure of the initial dataset

str(data)

# Check the proportion of attrition cases

prop.table(table(data$Attrition))

# Perform oversampling using ROSE package

oversampled\_data <- ovun.sample(Attrition ~ ., data = data, method = "both", p = 0.5, seed = 222, N = 1000)$data

# Check the balance after oversampling

table(oversampled\_data$Attrition)

# Save the oversampled dataset as "bothsampled.csv"

write.csv(oversampled\_data, file = "bothsampled.csv", row.names = FALSE)

# Confirm that the dataset has been saved

if (file.exists("bothsampled.csv")) {

print("Oversampled dataset saved successfully as 'bothsampled.csv'.")

} else {

print("Error: Oversampled dataset could not be saved.")

}

1. **Random oversampling using ovum.sample from the ROSE library**

library(ROSE)

d <- read.csv("Dataset.csv")

d$Attrition <- as.character(d$Attrition) # Convert 'Attrition' to character if it's a factor

# Perform undersampling using ROSE

result <- ovun.sample(Attrition ~ ., data = d, method = "over", N = 2 \* sum(d$Attrition == "No"))

# Extract balanced data from the result

result1 <- result$data

# Save balanced data with undersampling to a new CSV file

write.csv(result1, file = "rose\_ovum\_sample\_over\_sampled.csv", row.names = FALSE)

1. **Random undersampling using ovum.sample from the ROSE library**

library(ROSE)

d <- read.csv("Dataset.csv")

d$Attrition <- as.character(d$Attrition) # Convert 'Attrition' to character if it's a factor

# Perform undersampling using ROSE

result <- ovun.sample(Attrition ~ ., data = d, method = "under", N = 2 \* sum(d$Attrition == "Yes"))

# Extract balanced data from the result

result1 <- result$data

# Save balanced data with undersampling to a new CSV file

write.csv(result1, file = "rose\_ovum\_sample\_under\_sampled.csv", row.names = FALSE)

1. **Correlation-based Feature Selection (CFS) technique**

library(caTools)

library(FSelector)

data <- read.csv("random\_over\_sampled.csv")

# Define categorical columns

categorical\_columns <- c("Department", "BusinessTravel", "EducationField",

"Gender", "JobRole", "MaritalStatus", "OverTime")

# Convert "Yes" to 1 and "No" to 0 in the Attrition column

data$Attrition <- ifelse(data$Attrition == "Yes", 1, 0)

# Convert specified categorical columns to factors

data[, c("Attrition", categorical\_columns)] <- lapply(data[, c("Attrition", categorical\_columns)], as.factor)

# Manual scaling normalization

normalize <- function(x) {

return((x - min(x)) / (max(x) - min(x)))

}

# Apply normalization to numerical columns

numeric\_columns <- sapply(data, is.numeric)

if (sum(numeric\_columns) > 0) {

data[, numeric\_columns] <- lapply(data[, numeric\_columns], normalize)

cat("\nDataset normalized.\n")

} else {

cat("\nNo numerical columns found in the dataset.\n")

}

# Apply CFS feature selection

selected\_features <- cfs(Attrition ~ ., data = data)

# Print selected features

cat("Selected features based on CFS:\n", selected\_features, "\n")

1. **Chi Square (χ²) test**

# Function to calculate chi-square statistic

chi\_square <- function(feature, target) {

observed <- table(feature, target)

expected <- outer(rowSums(observed), colSums(observed)) / sum(observed)

chi\_square <- sum((observed - expected)^2 / expected)

return(chi\_square)

}

# Load the dataset

data <- read.csv("random\_over\_sampled.csv")

# Define categorical columns

categorical\_columns <- c("Department", "BusinessTravel", "EducationField",

"Gender", "JobRole", "MaritalStatus", "OverTime")

# Convert "Yes" to 1 and "No" to 0 in the Attrition column

data$Attrition <- ifelse(data$Attrition == "Yes", 1, 0)

# Convert specified categorical columns to factors

data[, c("Attrition", categorical\_columns)] <- lapply(data[, c("Attrition", categorical\_columns)], as.factor)

# Calculate chi-square statistic for each feature

chi\_square\_scores <- sapply(data[, -which(names(data) == "Attrition")],

function(x) chi\_square(x, data$Attrition))

# Sort features based on chi-square scores

sorted\_features <- names(sort(chi\_square\_scores, decreasing = TRUE))

# Select top features (you can change the number of features)

selected\_features <- sorted\_features[1:10]

# Print selected features based on chi-square scores

cat("Selected features based on Chi-square Test:\n", selected\_features, "\n")

1. **Information Gain**

# Load the required libraries

library(caTools)

library(FSelector)

# Load the dataset

data <- read.csv("random\_over\_sampled.csv")

# Define categorical columns

categorical\_columns <- c("Department", "BusinessTravel", "EducationField",

"Gender", "JobRole", "MaritalStatus", "OverTime")

# Convert "Yes" to 1 and "No" to 0 in the Attrition column

data$Attrition <- ifelse(data$Attrition == "Yes", 1, 0)

# Convert specified categorical columns to factors

data[, c("Attrition", categorical\_columns)] <- lapply(data[, c("Attrition", categorical\_columns)], as.factor)

# Apply Information Gain feature selection

selected\_features <- information.gain(Attrition ~ ., data = data)

# Print the object to check its structure and contents

print(selected\_features)

1. **Least Absolute Shrinkage and Selection Operator (LASSO)**

# Load the required libraries

library(caTools)

library(glmnet)

# Load the dataset

data <- read.csv("random\_over\_sampled.csv")

# Define categorical columns

categorical\_columns <- c("Department", "BusinessTravel", "EducationField",

"Gender", "JobRole", "MaritalStatus", "OverTime")

# Convert "Yes" to 1 and "No" to 0 in the Attrition column

data$Attrition <- ifelse(data$Attrition == "Yes", 1, 0)

# Convert specified categorical columns to factors

data[, c("Attrition", categorical\_columns)] <- lapply(data[, c("Attrition", categorical\_columns)], as.factor)

# Perform LASSO feature selection

x <- model.matrix(Attrition ~ ., data)[,-1]

y <- as.numeric(data$Attrition) - 1

lasso\_model <- cv.glmnet(x, y, family = "binomial", alpha = 1)

selected\_features <- rownames(coef(lasso\_model, s = "lambda.min"))[-1]

# Print selected features

if (length(selected\_features) >= 10) {

cat("Top 10 selected features based on LASSO (L1 Regularization):\n", selected\_features[1:10], "\n")

} else {

cat("Less than 10 features selected.\n")

}

1. **Genetic Algorithm**

# Load the required libraries

library(caTools)

library(GA)

# Load the dataset

data <- read.csv("random\_over\_sampled.csv")

# Define categorical columns

categorical\_columns <- c("Department", "BusinessTravel", "EducationField",

"Gender", "JobRole", "MaritalStatus", "OverTime")

# Convert "Yes" to 1 and "No" to 0 in the Attrition column

data$Attrition <- ifelse(data$Attrition == "Yes", 1, 0)

# Convert specified categorical columns to factors

data[, c("Attrition", categorical\_columns)] <- lapply(data[, c("Attrition", categorical\_columns)], as.factor)

# Define fitness function

fitness <- function(features) {

# Extract the indices of selected features

selected\_indices <- which(features == 1)

# Check if any features are selected

if (length(selected\_indices) == 0) {

return(NA) # Return NA if no features are selected

}

# Extract selected features including 'Attrition'

selected\_features <- c("Attrition", names(data)[selected\_indices])

# Train logistic regression model using selected features

model <- glm(Attrition ~ ., data = data[selected\_features], family = "binomial")

# Return the negative log-likelihood as the fitness value

return(-logLik(model))

}

# Perform genetic algorithm feature selection

ga\_result <- ga(type = "binary", fitness = fitness, nBits = ncol(data) - 1, maxiter = 50, run = 100)

# Extract selected features

selected\_features <- names(data)[which(as.logical(ga\_result@solution))]

# Print selected features

if (length(selected\_features) > 0) {

cat("Selected features based on Genetic Algorithm (GA):\n", selected\_features, "\n")

} else {

cat("No features selected\n")

}

1. **Importance Score based feature selection from Random Forest**

library(caTools)

library(rpart)

library(randomForest)

# Load the dataset

data <- read.csv("random\_over\_sampled.csv")

# Define categorical columns

categorical\_columns <- c("Department", "BusinessTravel", "EducationField",

"Gender", "JobRole", "MaritalStatus", "OverTime")

# Convert "Yes" to 1 and "No" to 0 in the Attrition column

data$Attrition <- ifelse(data$Attrition == "Yes", 1, 0)

# Convert specified categorical columns to factors

data[, c("Attrition", categorical\_columns)] <- lapply(data[, c("Attrition", categorical\_columns)], as.factor)

# Identify outliers using Z-score method

outliers <- which(abs(scale(data[, sapply(data, is.numeric)])) > 3, arr.ind = TRUE)

# Remove outliers if found

if (length(outliers) > 0) {

cat("Outliers detected using Z-score method in the following rows:\n")

#print(data[outliers, ])

# Remove outliers

data <- data[-outliers, ]

cat("\nOutliers removed using Z-score method.\n")

} else {

cat("No outliers detected using Z-score method.\n")

}

# Manual scaling normalization

normalize <- function(x) {

return((x - min(x)) / (max(x) - min(x)))

}

# Apply normalization to numerical columns

numeric\_columns <- sapply(data, is.numeric)

if (sum(numeric\_columns) > 0) {

data[, numeric\_columns] <- lapply(data[, numeric\_columns], normalize)

cat("\nDataset normalized.\n")

} else {

cat("\nNo numerical columns found in the dataset.\n")

}

# Set the seed for reproducibility

set.seed(123)

# Train a Random Forest model to determine feature importance

rf\_model <- randomForest(Attrition ~ ., data = train\_data, ntree = 500)

# Extract variable importance scores

importance\_scores <- importance(rf\_model)

# Select the top features based on importance scores (e.g., top 10 features)

top\_features <- names(importance\_scores[order(-importance\_scores), ][1:10])

print(top\_features)

1. **Recursive Feature Elimination (RFE)**

# Load the required libraries

library(caTools)

library(caret)

# Load the dataset

data <- read.csv("random\_over\_sampled.csv")

# Define categorical columns

categorical\_columns <- c("Department", "BusinessTravel", "EducationField",

"Gender", "JobRole", "MaritalStatus", "OverTime")

# Convert "Yes" to 1 and "No" to 0 in the Attrition column

data$Attrition <- ifelse(data$Attrition == "Yes", 1, 0)

# Convert specified categorical columns to factors

data[, c("Attrition", categorical\_columns)] <- lapply(data[, c("Attrition", categorical\_columns)], as.factor)

# Apply RFE feature selection

ctrl <- rfeControl(functions = rfFuncs, method = "cv", number = 10)

selected\_features <- rfe(data[, -which(names(data) == "Attrition")], data$Attrition, sizes = c(1:ncol(data)-1), rfeControl = ctrl)

# Print selected features

cat("Selected features based on RFE:\n", selected\_features$optVariables, "\n")

1. **Data Preprocessing to be performed for all the algorithms:**

**#**Load the necessary Libraries

# Load the dataset

data <- read.csv("bothsampled.csv")

# Define categorical columns

categorical\_columns <- c("Department", "BusinessTravel", "EducationField",

"Gender", "JobRole", "MaritalStatus", "OverTime")

# Convert "Yes" to 1 and "No" to 0 in the Attrition column

data$Attrition <- ifelse(data$Attrition == "Yes", 1, 0)

# Convert specified categorical columns to factors

data[, c("Attrition", categorical\_columns)] <- lapply(data[, c("Attrition", categorical\_columns)], as.factor)

# Count rows before outlier handling

rows\_before <- nrow(data)

# Identify outliers using Z-score method

outliers <- which(abs(scale(data[, sapply(data, is.numeric)])) > 3, arr.ind = TRUE)

# Remove outliers if found

if (length(outliers) > 0) {

# Remove outliers

data <- data[-outliers, ]}

# Count rows after outlier handling

rows\_after <- nrow(data)

# Print count of rows before and after outlier handling

cat("Rows before Z-score method of outlier handling:", rows\_before, "\n")

cat("Rows after Z-score method of outlier handling:", rows\_after, "\n")

# Manual scaling normalization

normalize <- function(x) {

return((x - min(x)) / (max(x) - min(x)))}

# Apply normalization to numerical columns

numeric\_columns <- sapply(data, is.numeric)

if (sum(numeric\_columns) > 0) {

data[, numeric\_columns] <- lapply(data[, numeric\_columns], normalize)

cat("\nDataset normalized.\n")

} else {

cat("\nNo numerical columns found in the dataset.\n")

}

# Set the seed for reproducibility

set.seed(123)

data <- data[sample(nrow(data)), ]

# Split the data into training (70%) and testing (30%) sets

split <- sample.split(data$Attrition, SplitRatio = 0.75)

train\_data <- subset(data, split == TRUE)

test\_data <- subset(data, split == FALSE)

# Create a training control specifying the 10-fold cross-validation

train\_control <- trainControl(method = "cv", number = 10)

1. **kNN algorithm with default parameters:**

library(caTools)

library(caret)

# Create a k-Nearest Neighbors model using 10-fold cross-validation

model <- train(Attrition ~ ., data = train\_data, method = "knn", trControl = train\_control)

1. **kNN algorithm with hyper parameter tuning:**

# Define the hyperparameter grid

hyper\_params <- expand.grid(k = seq(1, 20, by = 1))

# Create a k-Nearest Neighbors model with hyperparameter tuning

model <- train(Attrition ~ ., data = train\_data, method = "knn", trControl = train\_control, tuneGrid = hyper\_params)

1. **kNN algorithm with selected features:**

# create a set of features and assign to a selected\_features variable to perform training on selected features

selected\_features <- c("Age", "BusinessTravel", "EnvironmentSatisfaction", "JobInvolvement", "JobLevel", "MonthlyIncome", "OverTime", "StockOptionLevel", "YearsAtCompany", "YearsWithCurrManager")

# Create a kNN model with selected features

model <- train(Attrition ~ ., data = train\_data[, c(selected\_features, "Attrition")], method = "knn", trControl = train\_control)

1. **Logistic Regression algorithm with default parameters:**

#library(caTools)

#library(caret)

# Create a logistic regression model using 10-fold cross-validation

model <- train(Attrition ~ ., data = train\_data, method = "glm", family = "binomial", trControl = train\_control)

1. **Logistic Regression algorithm with hyper parameter tuning:**

# Define the hyperparameter grid

hyper\_params <- expand.grid(alpha = seq(0, 1, by = 0.1), lambda = seq(0.0001, 0.1, by = 0.01))

# Create a logistic regression model with hyperparameter tuning

model <- train(Attrition ~ ., data = train\_data, method = "glmnet", trControl = train\_control, tuneGrid = hyper\_params)

1. **Logistic Regression algorithm with selected features:**

# Use the selected\_features

selected\_features <- c("Age", "BusinessTravel", "EnvironmentSatisfaction", "JobInvolvement", "JobLevel", "MonthlyIncome", "OverTime", "StockOptionLevel", "YearsAtCompany", "YearsWithCurrManager")

# Create a decision tree model with selected features

model <- train(Attrition ~ ., data = train\_data[, c(selected\_features, "Attrition")], method = "rpart", trControl = train\_control)

1. **Naïve Bayes algorithm with default parameters:**

#library(caTools)

#library(caret)

# Create a Naive Bayes model using 10-fold cross-validation

model <- train(Attrition ~ ., data = train\_data, method = "naive\_bayes", trControl = train\_control)

1. **Naïve Bayes algorithm with selected features:**

# Use the selected\_features

selected\_features <- c("Age", "BusinessTravel", "EnvironmentSatisfaction", "JobInvolvement", "JobLevel", "MonthlyIncome", "OverTime", "StockOptionLevel", "YearsAtCompany", "YearsWithCurrManager")

# Create a Naive Bayes model with selected features

model <- train(Attrition ~ ., data = train\_data[, c(selected\_features, "Attrition")], method = "naive\_bayes", trControl = train\_control)

1. **Decision Tree Algorithm with default parameters:**

#library(caTools)

#library(rpart)

#library(caret)

# Create a decision tree model using 10-fold cross-validation

model <- train(Attrition ~ ., data = train\_data, method = "rpart", trControl = train\_control)

1. **Decision Tree Algorithm with hyper parameter tuning:**

# Define the hyperparameter grid

hyper\_params <- expand.grid(cp = seq(0.01, 0.5, by = 0.01))

# Create a decision tree model with hyperparameter tuning

model <- train(Attrition ~ ., data = train\_data, method = "rpart", trControl = train\_control, tuneGrid = hyper\_params)

1. **Decision Tree Algorithm with selected features:**

# Use the selected\_features

selected\_features <- c("Age", "BusinessTravel", "EnvironmentSatisfaction", "JobInvolvement", "JobLevel", "MonthlyIncome", "OverTime", "StockOptionLevel", "YearsAtCompany", "YearsWithCurrManager")

# Create a decision tree model with selected features

model <- train(Attrition ~ ., data = train\_data[, c(selected\_features, "Attrition")], method = "rpart", trControl = train\_control)

1. **Random forest algorithm with default parameters:**

#library(caTools)

#library(caret)

# Create a Random Forest model using 10-fold cross-validation

model <- train(Attrition ~ ., data = train\_data, method = "rf", trControl = train\_control)

1. **Random forest algorithm with selected features:**

# Use the selected\_features

selected\_features <- c("Age", "BusinessTravel", "EnvironmentSatisfaction", "JobInvolvement", "JobLevel", "MonthlyIncome", "OverTime", "StockOptionLevel", "YearsAtCompany", "YearsWithCurrManager")

# Create a Random Forest model with selected features

model <- train(Attrition ~ ., data = train\_data[, c(selected\_features, "Attrition")], method = "rf", trControl = train\_control)

1. **SVM algorithm with default parameters:**

#library(caTools)

#library(rpart)

#library(caret)

# Create an SVM model using 10-fold cross-validation for Linear kernal

svm\_model <- train(Attrition ~ ., data = train\_data, method = "svmLinear", trControl = train\_control)

# Create an SVM model using 10-fold cross-validation for Radial kernal

svm\_model <- train(Attrition ~ ., data = train\_data, method = "svmRadial", trControl = train\_control)

# Create an SVM model using 10-fold cross-validation for Polynomial kernal

svm\_model <- train(Attrition ~ ., data = train\_data, method = "svmPoly", trControl = train\_control)

1. **SVM algorithm with hyper parameter tuning:**

#For Linear kernal

# Define the hyperparameter grid

tuning\_grid <- expand.grid(

.C = c(0.1, 1, 10, 100) # Tuning parameter for regularization)

# Create an SVM model with hyperparameter tuning

svm\_model <- train( Attrition ~ ., data = train\_data, method = "svmLinear", trControl = train\_control, tuneGrid = tuning\_grid)

#For Radial kernal

# Define the hyperparameter grid

tuning\_grid <- expand.grid(

.sigma = c(0.1, 0.8, 1),

.C = c(0.1, 1, 20))

# Create an SVM model with hyperparameter tuning

svm\_model <- train( Attrition ~ ., data = train\_data, method = "svmRadial", trControl = train\_control, tuneGrid = tuning\_grid)

#For Polynomial kernal

# Define the hyperparameter grid

tuning\_grid <- expand.grid(

.degree = 2:5,

.scale = c(0.1, 1, 10),

.C = c(0.1, 1, 10))

# Create an SVM model with hyperparameter tuning

svm\_model <- train( Attrition ~ ., data = train\_data, method = "svmPoly", trControl = train\_control, tuneGrid = tuning\_grid)

1. **SVM algorithm with selected features:**

# Use the selected\_features

selected\_features <- c("Age", "BusinessTravel", "EnvironmentSatisfaction", "JobInvolvement", "JobLevel", "MonthlyIncome", "OverTime", "StockOptionLevel", "YearsAtCompany", "YearsWithCurrManager")

#For Linear kernal

# Create an SVM model with selected features

svm\_model <- train( Attrition ~ ., data = train\_data[, c(selected\_features, "Attrition")], method = "svmLinear", trControl = train\_control, tuneGrid = tuning\_grid)

#For Radial kernal

# Create an SVM model with selected features

svm\_model <- train( Attrition ~ ., data = train\_data[, c(selected\_features, "Attrition")], method = "svmRadial", trControl = train\_control, tuneGrid = tuning\_grid)

#For Polynomial kernal

# Create an SVM model with selected features

svm\_model <- train( Attrition ~ ., data = train\_data[, c(selected\_features, "Attrition")], method = "svmPoly", trControl = train\_control, tuneGrid = tuning\_grid)

1. **Predictions and Evaluation Parameters for all the algorithms:**

# Make predictions on the test set

predictions <- predict(model, newdata = test\_data)

# Confusion matrix to evaluate the model

conf\_matrix <- table(predictions, test\_data$Attrition)

print(conf\_matrix)

# Calculate accuracy

accuracy <- sum(diag(conf\_matrix)) / sum(conf\_matrix)

print(paste("Accuracy:", accuracy))

# Calculate sensitivity (true positive rate)

sensitivity <- conf\_matrix[2, 2] / sum(conf\_matrix[2, ])

print(paste("Sensitivity:", sensitivity))

# Calculate precision (positive predictive value)

precision <- conf\_matrix[2, 2] / sum(conf\_matrix[, 2])

print(paste("Precision:", precision))

# Calculate specificity (true negative rate)

specificity <- conf\_matrix[1, 1] / sum(conf\_matrix[1, ])

print(paste("Specificity:", specificity))

1. **Shinny app code integrated with SVM algorithm:**

# Load necessary libraries

library(shiny)

library(caTools)

library(rpart)

library(caret)

# Load the dataset

data <- read.csv("bothsampled.csv")

# Define categorical columns

categorical\_columns <- c("Department", "BusinessTravel", "EducationField", "Gender", "JobRole", "MaritalStatus", "OverTime")

# Convert "Yes" to 1 and "No" to 0 in the Attrition column

data$Attrition <- ifelse(data$Attrition == "Yes", 1, 0)

# Convert specified categorical columns to factors

data[, c("Attrition", categorical\_columns)] <- lapply(data[, c("Attrition", categorical\_columns)], as.factor)

# Identify outliers using Z-score method

outliers <- which(abs(scale(data[, sapply(data, is.numeric)])) > 3, arr.ind = TRUE)

# Remove outliers if found

if (length(outliers) > 0) {

data <- data[-outliers, ]

}

# Manual scaling normalization

normalize <- function(x) {

return((x - min(x)) / (max(x) - min(x)))

}

# Apply normalization to numerical columns

numeric\_columns <- sapply(data, is.numeric)

if (sum(numeric\_columns) > 0) {

data[, numeric\_columns] <- lapply(data[, numeric\_columns], normalize)

}

# Set the seed for reproducibility

set.seed(123)

data <- data[sample(nrow(data)), ]

# Split the data into training (70%) and testing (30%) sets

split <- sample.split(data$Attrition, SplitRatio = 0.70)

train\_data <- subset(data, split == TRUE)

test\_data <- subset(data, split == FALSE)

# Define selected features

selected\_features <- c("OverTime", "JobRole", "HourlyRate", "DailyRate", "MonthlyRate", "MonthlyIncome",

"Age", "PercentSalaryHike", "JobSatisfaction", "DistanceFromHome", "StockOptionLevel",

"EducationField", "NumCompaniesWorked", "EnvironmentSatisfaction", "RelationshipSatisfaction",

"Education", "WorkLifeBalance", "TrainingTimesLastYear", "TotalWorkingYears", "YearsAtCompany",

"YearsWithCurrManager", "MaritalStatus", "JobInvolvement", "YearsSinceLastPromotion",

"BusinessTravel", "YearsInCurrentRole", "JobLevel", "Gender")

# Define the hyperparameter grid

tuning\_grid <- expand.grid(

.degree = 2:5,

.scale = c(0.1, 1, 10),

.C = c(0.1, 1, 10)

)

# Train the SVM model

svm\_model <- train(Attrition ~ .,

data = train\_data[, c(selected\_features, "Attrition")],

method = "svmPoly",

trControl = trainControl(method = "cv", number = 10),

tuneGrid = tuning\_grid)

# UI

ui <- fluidPage(

tags$head(

tags$style(HTML("

/\* Custom Fonts \*/

@import url('https://fonts.googleapis.com/css2?family=Roboto:wght@400;500&display=swap');

/\* Form Container Styling \*/

.form-container {

width: 60%;

margin: 0 auto;

padding: 40px;

background-color: #f5f5f5; /\* Light gray \*/

border-radius: 15px;

box-shadow: 0 4px 8px rgba(0,0,0,0.1);

font-family: 'Roboto', sans-serif;

}

/\* Form Header Styling \*/

.form-header {

color: #333; /\* Dark gray \*/

font-size: 28px;

font-weight: 500;

text-align: center;

margin-bottom: 30px;

}

/\* Form Section Styling \*/

.form-section {

margin-bottom: 30px;

}

/\* Label Styling \*/

.form-label {

color: #555; /\* Gray \*/

font-weight: 500;

margin-bottom: 8px;

}

/\* Submit Button Styling \*/

.submit-button {

background-color: #4CAF50; /\* Green \*/

color: white;

padding: 12px 24px;

border: none;

border-radius: 5px;

cursor: pointer;

transition: background-color 0.3s ease;

}

.submit-button:hover {

background-color: #45a049; /\* Darker green \*/

}

"))

),

fluidRow(

column(12, align = "center",

div(class = "form-container",

div(class = "form-header", "Employee Attrition Prediction"),

# Personal Information Inputs

div(class = "form-section",

tags$label(class = "form-label", "Personal Information"),

numericInput("Age", "Age", value = 30, min = 18, max = 65),

selectInput("Gender", "Gender", choices = c("Male", "Female")),

selectInput("MaritalStatus", "Marital Status", choices = c("Single", "Married", "Divorced"))

),

# Job Related Inputs

div(class = "form-section",

tags$label(class = "form-label", "Job Related"),

selectInput("Department", "Department", choices = c("Research & Development", "Sales", "Human Resources")),

selectInput("JobRole", "Job Role", choices = c("Sales Executive", "Research Scientist", "Laboratory Technician", "Manufacturing Director", "Healthcare Representative", "Manager", "Sales Representative", "Research Director", "Human Resources")),

numericInput("JobLevel", "Job Level", value = 1, min = 1, max = 5)

),

# Work Environment Inputs

div(class = "form-section",

tags$label(class = "form-label", "Work Environment"),

selectInput("BusinessTravel", "Business Travel", choices = c("Non-Travel", "Travel\_Rarely", "Travel\_Frequently")),

selectInput("OverTime", "Over Time", choices = c("Yes", "No")),

numericInput("DistanceFromHome", "Distance From Home (miles)", value = 1, min = 1, max = 30),

numericInput("DailyRate", "Daily Rate ($)", value = 500, min = 0),

numericInput("HourlyRate", "Hourly Rate ($)", value = 60, min = 0)

),

# Performance Inputs

div(class = "form-section",

tags$label(class = "form-label", "Performance"),

numericInput("PercentSalaryHike", "Percent Salary Hike (%)", value = 10, min = 0, max = 25),

numericInput("PerformanceRating", "Performance Rating", value = 3, min = 1, max = 5)

),

# Work-life Balance Inputs

div(class = "form-section",

tags$label(class = "form-label", "Work-life Balance"),

numericInput("TotalWorkingYears", "Total Working Years", value = 10, min = 0, max = 50),

numericInput("YearsAtCompany", "Years At Company", value = 5, min = 0, max = 30),

numericInput("YearsInCurrentRole", "Years In Current Role", value = 3, min = 0, max = 20),

numericInput("YearsSinceLastPromotion", "Years Since Last Promotion", value = 1, min = 0, max = 15),

numericInput("YearsWithCurrManager", "Years With Current Manager", value = 2, min = 0, max = 20),

numericInput("TrainingTimesLastYear", "Training Times Last Year", value = 2, min = 0, max = 10)

),

# Submit Button

actionButton("predict", "Predict Attrition", class = "submit-button")

)

)

)

)

# Server

server <- function(input, output) {

# Predict attrition

observeEvent(input$predict, {

new\_entry <- data.frame(

Age = as.numeric(input$Age),

BusinessTravel = input$BusinessTravel,

DailyRate = as.numeric(input$DailyRate),

Department = input$Department,

DistanceFromHome = as.numeric(input$DistanceFromHome),

Education = as.numeric(input$Education),

EducationField = input$EducationField,

EnvironmentSatisfaction = as.numeric(input$EnvironmentSatisfaction),

Gender = input$Gender,

HourlyRate = as.numeric(input$HourlyRate),

JobInvolvement = as.numeric(input$JobInvolvement),

JobLevel = as.numeric(input$JobLevel),

JobRole = input$JobRole,

JobSatisfaction = as.numeric(input$JobSatisfaction),

MaritalStatus = input$MaritalStatus,

MonthlyIncome = as.numeric(input$MonthlyIncome),

MonthlyRate = as.numeric(input$MonthlyRate),

NumCompaniesWorked = as.numeric(input$NumCompaniesWorked),

OverTime = input$OverTime,

PercentSalaryHike = as.numeric(input$PercentSalaryHike),

PerformanceRating = as.numeric(input$PerformanceRating),

RelationshipSatisfaction = as.numeric(input$RelationshipSatisfaction),

StockOptionLevel = as.numeric(input$StockOptionLevel),

TotalWorkingYears = as.numeric(input$TotalWorkingYears),

TrainingTimesLastYear = as.numeric(input$TrainingTimesLastYear),

WorkLifeBalance = as.numeric(input$WorkLifeBalance),

YearsAtCompany = as.numeric(input$YearsAtCompany),

YearsInCurrentRole = as.numeric(input$YearsInCurrentRole),

YearsSinceLastPromotion = as.numeric(input$YearsSinceLastPromotion),

YearsWithCurrManager = as.numeric(input$YearsWithCurrManager)

)

prediction <- predict(svm\_model, newdata = new\_entry)

if (prediction == 1) {

output$prediction <- renderText("Yes, there might be attrition.")

} else {

output$prediction <- renderText("No, there might not be attrition.")

}

})

}

# Run the application

shinyApp(ui = ui, server = server)

1. **Shiny app:**

The user interface (UI) of the Shiny app is designed with a structured layout comprising distinct sections, each focusing on different facets of employee information relevant to attrition prediction. These sections are visually organized within a form container, characterized by a clean appearance with rounded corners and a light gray background, contributing to an aesthetically pleasing interface. The form header stands out prominently with its larger font size, bold text, and centrally aligned position, serving as a clear indicator of the form's purpose. Within each section, users are presented with intuitive input fields, including numeric inputs for age, salary, etc., and select inputs for categorical variables like gender and marital status. Each input field is accompanied by a descriptive label, ensuring clarity and ease of use. The submit button, strategically positioned at the form's bottom, features a distinct green color, inviting user interaction, and providing feedback with a hover effect.

On the server side, the app employs logic to predict attrition based on user inputs. Upon clicking the "Predict Attrition" button, an observer function is triggered, capturing the user's input data and constructing a new entry/data frame. This data is then passed through the SVM model, previously trained on a dataset, to predict the likelihood of attrition. Subsequently, based on the prediction outcome (1 for attrition, 0 for no attrition), a corresponding text message is rendered to the user, offering real-time feedback. This prediction process facilitates informed decision-making by users, empowering them to assess potential attrition risks based on their input information.

1. **Evaluation Parameters :**

In our employee retention prediction project, we employ a variety of evaluation parameters to comprehensively assess the performance of our machine learning model. These metrics play a crucial role in providing insights into the effectiveness and reliability of our model's predictions, particularly in the context of predicting attrition.

**Accuracy** serves as a foundational metric, allowing us to gauge the overall correctness of our model's predictions. It quantifies the ratio of correctly predicted instances of attrition to the total number of instances in our dataset, providing a broad understanding of our model's predictive capability.

Furthermore, **sensitivity**, also known as recall or the true positive rate, holds significant importance in employee retention prediction. Sensitivity measures the proportion of actual instances of attrition that our model correctly identifies among all instances of attrition in the dataset. This metric is particularly valuable as it helps us understand how effectively our model detects employees who are likely to leave the company.

**Precision**, or the positive predictive value, complements sensitivity by evaluating the proportion of true positive predictions among all instances predicted as attrition by our model. Precision is essential in scenarios where the cost associated with incorrectly predicting attrition is high. It provides insights into the model's ability to minimize false positive predictions, thereby assisting in making more accurate and actionable decisions related to employee retention strategies.

Lastly, **specificity**, or the true negative rate, assesses the model's ability to correctly identify instances of non-attrition, which is equally important in employee retention prediction. It quantifies the proportion of actual instances of non-attrition that our model correctly identifies among all instances of non-attrition in the dataset. Specificity complements sensitivity and provides insights into the model's capability to accurately identify employees who are likely to stay with the company.

By examining these evaluation parameters comprehensively, we gain valuable insights into the strengths and weaknesses of our model in predicting employee attrition. This enables us to make informed decisions and implement targeted retention strategies to retain valuable talent within the organization.

1. **Results and Discussions:**
2. **Outlier handling using Z-score** 
   1. **Both sampled Dataset**

****

* 1. **Random over sampled dataset**



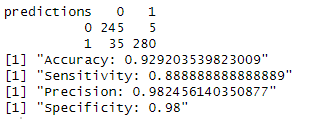
* 1. **Random under sampled dataset**

****

1. **kNN algorithm**

Best results for this algorithm are obtained from the Random Over sampled Dataset with selected feature from the RFE technique after performing hyper tuning of parameters with a data split of 75% for training and 25% for testing

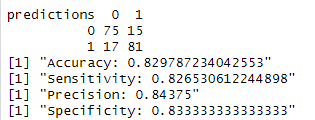
Results:



1. **Logistic Regression algorithm**

Best results for this algorithm are obtained from the Both Over and Under sampled Dataset with selected features from the Information gain technique after performing hyper tuning of parameters with a data split of 80% for training and 20% for testing

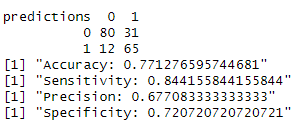
Results:



1. **Naïve Bayes algorithm**

Best results for this algorithm are obtained from the Both Over and Under sampled Dataset with selected features from the RFE technique after performing hyper tuning of parameters with a data split of 80% for training and 20% for testing

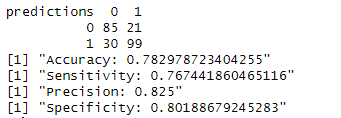
Results:



1. **Decision Tree algorithm**

Best results for this algorithm are obtained from the Both Over and Under sampled Dataset with selected features from the RFE technique after performing hyper tuning of parameters with a data split of 70% for training and 30% for testing

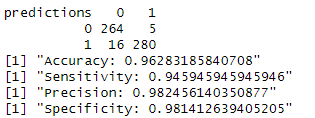
Results:



1. **Random Forest algorithm**

Best results for this algorithm are obtained from the Random Over sampled Dataset with selected features from the RFE technique with a data split of 75% for training and 25% for testing

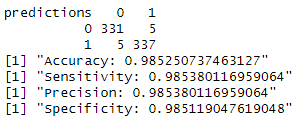
Results:



1. **SVM algorithm**

Best results for this algorithm are obtained from the Random Over sampled Dataset with selected features from the RFE technique along with polynomial kernal after performing hyper tuning of parameters with a data split of 70% for training and 30% for testing

Results:



**15. Conclusions:**

This study provides valuable insights into the complexities of employee attrition within organizational contexts and underscores the potential of machine learning techniques, particularly Random Forest and Support Vector Machine (SVM), in predictive analytics. Through an in-depth examination of a diverse dataset encompassing various employee characteristics and historical attrition data, our aim was to unearth actionable insights to guide strategic decision-making and retention initiatives.

Our analysis revealed compelling findings regarding the predictive capabilities of machine learning models, particularly Random Forest and SVM. These models demonstrated impressive accuracy, sensitivity, and specificity in forecasting employee attrition, demonstrating their adeptness at identifying underlying patterns and accurately predicting turnover risk. By leveraging these models, organizations can proactively identify employees at risk of attrition and implement targeted interventions to mitigate turnover, thereby fostering a more stable and engaged workforce.

Furthermore, our investigation identified key predictors influencing attrition likelihood, including departmental affiliation, business travel frequency, job role, and overtime work. Understanding the significance of these predictors empowers organizations to tailor retention strategies effectively, addressing underlying issues and enhancing employee satisfaction and retention.

The process of hyperparameter tuning played a crucial role in optimizing model performance. By fine-tuning parameters such as kernel type, regularization parameter, and margin width in SVM, we were able to enhance predictive accuracy and develop more robust predictive models. This optimization process underscores the importance of parameter selection in maximizing model effectiveness and generalizability.

Looking ahead, our study suggests several avenues for future research. Longitudinal analyses tracking changes in employee turnover patterns over time can provide insights into temporal trends and dynamics, facilitating proactive intervention strategies. Additionally, exploring the integration of external data sources, such as economic indicators and industry benchmarks, into predictive models can further enhance predictive accuracy and contextualize attrition predictions within broader organizational and market dynamics.

In conclusion, the application of machine learning techniques, particularly Random Forest and SVM, holds significant promise for addressing the challenge of employee attrition. By leveraging predictive models and actionable insights derived from data-driven analyses, organizations can navigate workforce dynamics more effectively, fostering a culture of engagement, stability, and growth.

**References:** *Give references in IEEE format.*

1. *Praveen Ranjan Srivastava, Prajwal Eachempati, Intelligent Employee Retention System for Attrition Rate Analysis and Churn Prediction, JGIM, 2021*
2. *Shobhanam Krishna and Sumati Sidharth, HR Analytics: Employee Attrition Analysis using Random Forest, IJPE, 2022*
3. *Francesca Fallucchi , Marco Coladangelo , Romeo Giuliano and Ernesto William De Luca, Predicting Employee Attrition Using Machine Learning Techniques, IEEE access, 2021*
4. *N. B. Yahia, J. Hlel and R. Colomo-Palacios, "From Big Data to Deep Data to Support People Analytics for Employee Attrition Prediction," in IEEE Access, vol. 9, pp. 60447-60458, 2021, doi: 10.1109/ACCESS.2021.3074559.*
5. *Raza A, Munir K, Almutairi M, Younas F, Fareed MMS. Predicting Employee Attrition Using Machine Learning Approaches. Applied Sciences. 2022; 12(13):6424.* [*https://doi.org/10.3390/app12136424*](https://doi.org/10.3390/app12136424)
6. *Xu, Z., Song, B. (2006). A Machine Learning Application for Human Resource Data Mining Problem. In: Ng, WK., Kitsuregawa, M., Li, J., Chang, K. (eds) Advances in Knowledge Discovery and Data Mining. PAKDD 2006. Lecture Notes in Computer Science(), vol 3918. Springer, Berlin, Heidelberg.* [*https://doi.org/10.1007/11731139\_99*](https://doi.org/10.1007/11731139_99)
7. *Wardhani, Fitri & Lhaksmana, Kemas. (2022). Predicting Employee Attrition Using Logistic Regression With Feature Selection. Sinkron. 7. 2214-2222. 10.33395/sinkron.v7i4.11783.*
8. *K. Mridha, R. N. Shaw, and A. Ghosh, "Study and Prediction Analysis of the Employee Turnover using Machine Learning Approaches," in IEEE Xplore, 2021.*
9. *R. Boutaba et al., "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," Journal of Internet Services and Applications, 2018.*
10. *N. P. S., P. S. Aithal, G. B. J., S. Soans, and H. Jayaraj, "A Study on Employee Retention as a Tool for Improving Organizational Effectiveness," International Journal of Management, Technology, and Social Sciences (IJMTS), 2021.*
11. *K. Alshehhi, S. Bin Zawbaa, A. A. Abonamah, and M. U. Tariq, "Employee retention prediction in corporate organizations using machine learning methods," Academy of Entrepreneurship Journal, 2021.*
12. *V. Kakulapati and S. Subhani, "Predictive Analytics of Employee Attrition using K-Fold Methodologies," International Journal of Management Science and Computing (IJMSC), 2023.* [*https://doi.org/10.5815/ijmsc.2023.01.03*](https://doi.org/10.5815/ijmsc.2023.01.03)
13. *A. K. Dubey, I. Maheshwari, and A. Mishra, "Predict Employee Retention Using Data Science," International Journal of Electrical, Electronics and Computer Systems Engineering (IJEECSE), 2018.*
14. *P. R. Srivastava and P. Eachempati, "Intelligent Employee Retention System for Attrition Rate Analysis and Churn Prediction," Journal of Global Information Management, 2021.*
15. *M. Lazzari, J. M. Alvarez, and S. Ruggieri, "Predicting and explaining employee turnover intention," International Journal of Data Science and Analytics, 2022.*
16. *Younis, S., Ahsan, A. & Chatteur, F.M. An employee retention model using organizational network analysis for voluntary turnover. Soc. Netw. Anal. Min. 13, 28 (2023).* [*https://doi.org/10.1007/s13278-023-01031-w*](https://doi.org/10.1007/s13278-023-01031-w)
17. *M. Karthika, Employee Performance Prediction Using EPP Framework, International Journal of Scientific Research in Computer Science Engineering and Information Technology, December 2020, DOI:10.32628/CSEIT206654*
18. *P. S., Nethravathi and Aithal, P. S. and J., Gayathri Babu and Soans, Sonia and Jayaraj, Honey, A Study on Employee Retention as a Tool for Improving Organizational Effectiveness (September 30, 2021). International Journal of Management, Technology, and Social Sciences (IJMTS), 6(2), 121-132. ISSN: 2581-6012.*
19. *Halim, Z., Maria, Waqas, M. et al. Identifying factors for employee retention using computational techniques: an approach to assist the decision-making process. SN Appl. Sci. 2, 1612 (2020).* [*https://doi.org/10.1007/s42452-020-03415-5*](https://doi.org/10.1007/s42452-020-03415-5)