Department of Information Technology
V R Siddhartha Engineering College

Machine Learning Project

Assignment -2

Presented by
Rizwanullah Mohammad (208W1A1299)
Harshita Sambana (208W1A12C1)
Abhi Venkat Sai Samsani (208W1A12C2)

Submitted to
Dr.T.Anuradha PhD ,Professor

Task-1:
CLASSIFICATION - HR ATTRITION DATA BASED ON IBM ATTRITION

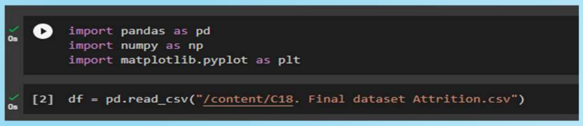
The dataset contains the same features as the IBM dataset with a few added features to help us with the project, which includes the survival analysis and prediction of an employee, whether he/she would attrite or not.

Preprocessing on the Dataset

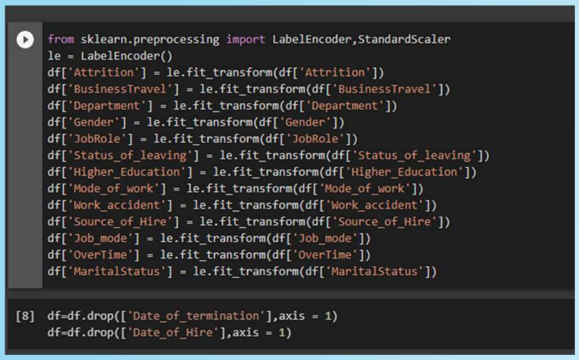
Steps for preprocessing the dataset:

- Import the necessary libraries such as pandas, numpy, sklearn, etc.
- Load the dataset using pandas read_csv() function.
- Check for missing values in the dataset using isnull() function and handle them if any.
- Convert the categorical variables to numerical using label encoding.
- Split the dataset into training and testing sets using train_test_split() function from sklearn.
- Scale the numerical variables to have a mean of 0 and standard deviation of 1 using StandardScaler() function from sklearn.

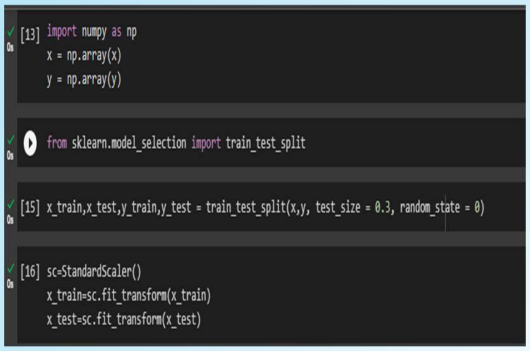
Code for Preprocessing the Attrition prediction



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

[2] df = pd.read_csv("/content/C18. Final dataset Attrition.csv")
```



```
from sklearn.preprocessing import LabelEncoder,StandardScaler
le = LabelEncoder()
df['Attrition'] = le.fit_transform(df['Attrition'])
df['BusinessTravel'] = le.fit_transform(df['BusinessTravel'])
df['Department'] = le.fit_transform(df['Department'])
df['Gender'] = le.fit_transform(df['Gender'])
df['JobRole'] = le.fit_transform(df['JobRole'])
df['Status_of_leaving'] = le.fit_transform(df['Status_of_leaving'])
df['Higher_Education'] = le.fit_transform(df['Higher_Education'])
df['Mode_of_work'] = le.fit_transform(df['Mode_of_work'])
df['Work_accident'] = le.fit_transform(df['Work_accident'])
df['Source_of_Hire'] = le.fit_transform(df['Source_of_Hire'])
df['Job_mode'] = le.fit_transform(df['Job_mode'])
df['OverTime'] = le.fit_transform(df['OverTime'])
df['MaritalStatus'] = le.fit_transform(df['MaritalStatus'])

[8] df=df.drop(['Date_of_termination'],axis = 1)
    df=df.drop(['Date_of_Hire'],axis = 1)
```



```
[13] import numpy as np
     x = np.array(x)
     y = np.array(y)

     from sklearn.model_selection import train_test_split

[15] x_train,x_test,y_train,y_test = train_test_split(x,y, test_size = 0.3, random_state = 0)

[16] sc=StandardScaler()
     x_train=sc.fit_transform(x_train)
     x_test=sc.fit_transform(x_test)
```

Classification model with Random Forest Classifier and Decision Tree

- **Decision tree :**It is a tree-like model used in machine learning for classification and regression analysis. It works by recursively splitting the data into subsets based on significant attributes until a final decision or prediction is made. It is known for its interpretability and ability to handle both categorical and numerical data.
- **Random Forest Classifier** is an ensemble learning algorithm that constructs multiple decision trees during the training phase and combines them to make a prediction. It is a popular machine learning algorithm used for classification tasks, and its ability to handle large datasets with high dimensionality makes it a popular choice in many domains. The algorithm's performance is usually high, as it reduces the overfitting problem present in decision trees, provides high accuracy, and can handle missing data. The algorithm is easy to use and can work with both categorical and numerical data.

DECISION TREE ALGORITHM

Step 1: Select the best attribute:

- Calculate the information gain for each attribute using the following formula:
  Information Gain = Entropy(S) - ∑ [(|Sv| / |S|) * Entropy(Sv)]
- Calculate the entropy of the set S using the following formula: Entropy(S) = - ∑ [(|Si| / |S|) * log2(|Si| / |S|)]
- Select the attribute with the highest information gain.

Step 2: Split the dataset:

- Split the dataset based on the values of the selected attribute.
- Each branch of the tree represents a different value of the selected attribute.

Step 3: Recursively repeat the process:

- Repeat the above two steps for each subset of the data created by the split.
- Continue until a stopping criterion is met, such as a certain depth of the tree or a minimum number of instances in each leaf node.

Step 4: Assign class labels:

- Determine the class label for each leaf node by the majority class of the instances in that node.

RANDOM FOREST ALGORITHM

Step 1: Select random samples:

- Randomly select a subset of the data (with replacement) from the original dataset.
- This subset is used as the training set for each tree in the forest.

Step 2: Build decision trees:

- Build a decision tree for each subset of the data created in Step 1, using the decision tree algorithm.
- The number of trees in the forest is determined by the user.

Step 3: Make predictions:

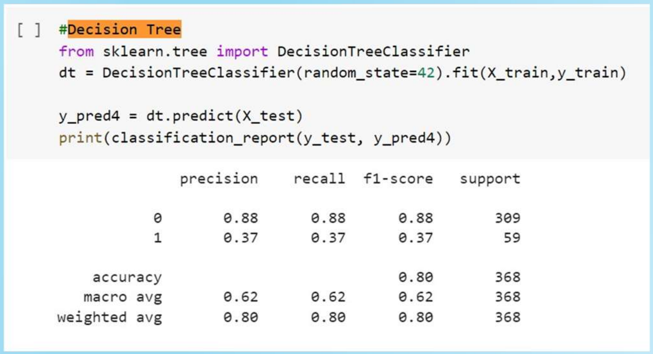
- For a new input, make a prediction by aggregating the predictions of all the decision trees in the forest.
- The most common prediction is taken as the final prediction.

Step 4: Calculate the overall accuracy:

- Calculate the overall accuracy of the random forest by comparing the predicted output to the actual output for each input.

Classification model with Random Forest Classifier and Decision Tree

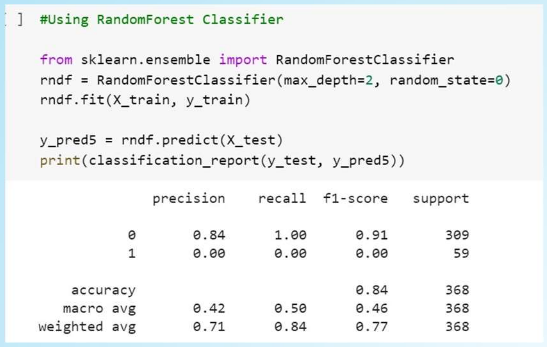
Code for Decision Tree :



```
[ ] #Decision Tree
    from sklearn.tree import DecisionTreeClassifier
    dt = DecisionTreeClassifier(random_state=42).fit(X_train,y_train)

    y_pred4 = dt.predict(X_test)
    print(classification_report(y_test, y_pred4))
```

```
              precision    recall  f1-score   support

           0       0.88      0.88      0.88       309
           1       0.37      0.37      0.37        59

    accuracy                           0.80       368
   macro avg       0.62      0.62      0.62       368
weighted avg       0.80      0.80      0.80       368
```

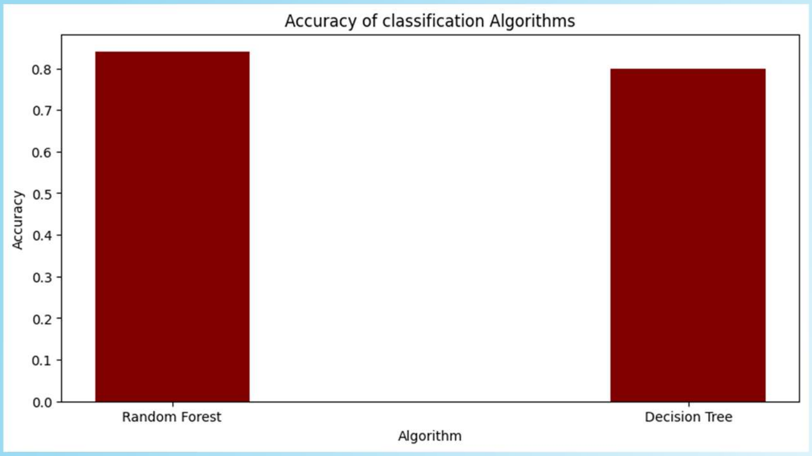
Code for Random Forest Classifier:



```
[ ] #Using RandomForest Classifier

    from sklearn.ensemble import RandomForestClassifier
    rndf = RandomForestClassifier(max_depth=2, random_state=0)
    rndf.fit(X_train, y_train)

    y_pred5 = rndf.predict(X_test)
    print(classification_report(y_test, y_pred5))
```

```
              precision    recall  f1-score   support

           0       0.84      1.00      0.91       309
           1       0.00      0.00      0.00        59

    accuracy                           0.84       368
   macro avg       0.42      0.50      0.46       368
weighted avg       0.71      0.84      0.77       368
```

Evaluating the model with measures.



Comparison Analysis of Result

Decision trees and random forests are popular machine learning algorithms that are widely used for classification and regression tasks

- In a study using decision tree of 1470 rows of final attrition dataset it achieved an accuracy of 80%.
- In a study using Random Forest of 1470 rows of final attrition dataset it achieved an accuracy of 84%.
- While comparing all the algorithms Random Forest shows good accuracy.

Conclusion

- In conclusion, the problem statement of building a classification model for predicting employee attrition based on HR data is important for organizations that aim to retain their valuable employees. By analyzing the data and building a model, organizations can gain insights into the factors that contribute to attrition and take measures to prevent it. The addition of survival analysis features in the dataset can provide a deeper understanding of how long an employee is likely to stay with the organization. Overall, this project can help organizations make data-driven decisions to improve employee retention and create a more stable workforce.

Task-2:
REGRESSION - INSURANCE PREMIUM PREDICTION

All of us wish to achieve financial freedom at some point in our life, and when it comes to doing that, we tend to believe that savings are enough to be financially stable. But, if you look at life from a practical perspective, you would understand that savings alone are not enough to achieve financial freedom; insuring your assets with general insurance policies is equally important. Hence predict the insurance premium amount to be paid of the persons based on their data. Perform the following tasks on the dataset given:

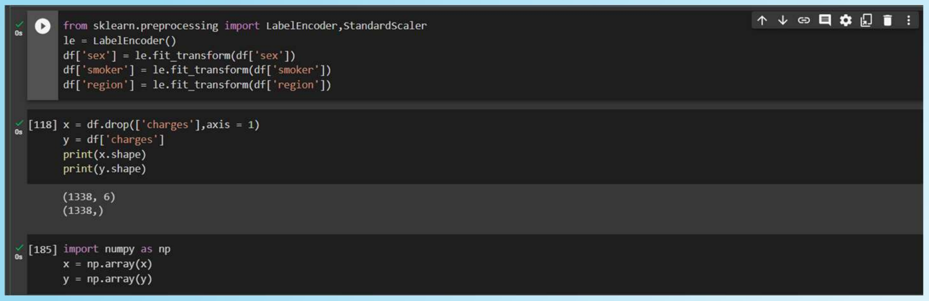
1. Apply the required preprocessing on the dataset to apply classification / regression.
2. 2. Construct the classification/regression model with any 2 algorithms by writing the code in your preferable language.
3. 3. Evaluate the model with suitable measures.
4. 4. Perform the comparison analysis of your results.
5. 5. Discuss what your observations and conclusions.

Preprocessing on the Dataset

Steps for preprocessing the dataset:

- Import the necessary libraries such as pandas, numpy, sklearn, etc.
- Load the dataset using pandas read_csv() function.
- Check for missing values in the dataset using isnull() function and handle them if any.
- Convert the categorical variables to numerical using label encoding.
- Split the dataset into training and testing sets using train_test_split() function from sklearn.
- Scale the numerical variables to have a mean of 0 and standard deviation of 1 using StandardScaler() function from sklearn.

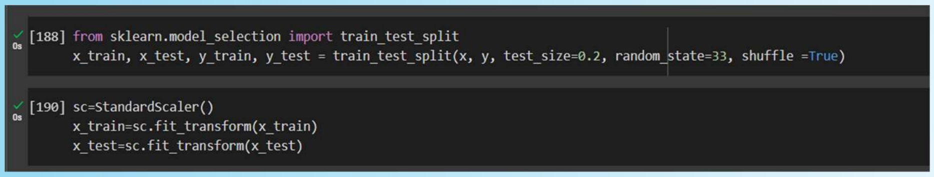
Code for Preprocessing the Insurance prediction



```
from sklearn.preprocessing import LabelEncoder,StandardScaler
le = LabelEncoder()
df['sex'] = le.fit_transform(df['sex'])
df['smoker'] = le.fit_transform(df['smoker'])
df['region'] = le.fit_transform(df['region'])
```

```
[118] x = df.drop(['charges'],axis = 1)
      y = df['charges']
      print(x.shape)
      print(y.shape)

      (1338, 6)
      (1338,)
```

```
[185] import numpy as np
      x = np.array(x)
      y = np.array(y)
```



```
[188] from sklearn.model_selection import train_test_split
      x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=33, shuffle =True)
```

```
[190] sc=StandardScaler()
      x_train=sc.fit_transform(x_train)
      x_test=sc.fit_transform(x_test)
```

Regression model with Ridge Regressor and Random Forest Regressor

- **RIDGE REGRESSION** is a regularization technique used in linear regression to prevent overfitting by introducing a penalty term to the cost function. The penalty term, which is a function of the square of the magnitude of the coefficients, forces the model to prioritize simpler models with smaller coefficient values over more complex models with larger coefficient values. This helps to reduce the impact of noise in the data and improve the generalizability of the model. Ridge regression is particularly useful when dealing with high-dimensional datasets with multicollinearity among the independent variables, where the standard linear regression algorithm may fail due to overfitting. The strength of the regularization can be controlled using a hyperparameter, alpha, which balances the trade-off between bias and variance.
- **RANDOM FOREST REGRESSOR** is a decision tree-based ensemble learning algorithm that builds multiple decision trees and aggregates their predictions to produce a more accurate and robust prediction. The algorithm works by randomly selecting a subset of features and a subset of samples from the training data to build each decision tree. This randomness helps to reduce overfitting and increase the diversity of the decision trees, improving the overall accuracy of the model. The final prediction of the model is the average of the predictions of all the decision trees. Random Forest Regressor is a powerful and widely used algorithm for regression tasks due to its ability to handle complex and high-dimensional data, detect feature importance, and avoid overfitting. However, the interpretability of the model may be compromised due to the black-box nature of decision trees.

RIDGE REGRESSION ALGORITHM

Step 1: Preprocessing the data:

- Scale the features to have zero mean and unit variance.
- Split the data into training and testing sets.

Step 2: Initialize the model:

- Initialize the regression coefficients to zero or small random values.

Step 3: Define the loss function:

- Define the loss function as the sum of squared errors between the predicted and actual output, plus the regularization term.
- The regularization term is a penalty for large coefficients and is defined as the product of the regularization parameter and the L2 norm of the coefficients.
- The loss function is given by: Loss = $||y - Xw||^2 + \text{alpha} * ||w||^2$

Step 4: Minimize the loss function:

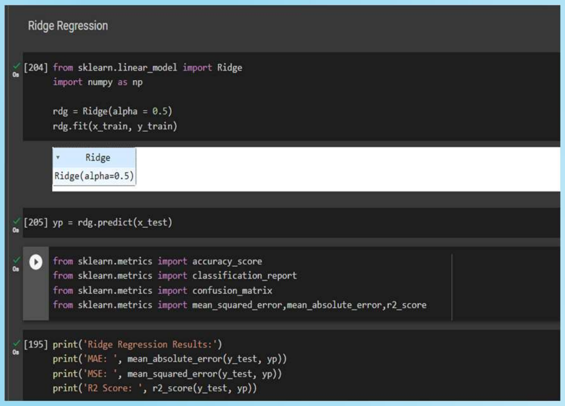
- Minimize the loss function using the closed-form solution or gradient descent.
- The closed-form solution is given by: w = (X^T X + alpha * I)^-1 X^T y
- where X is the training data, y is the target output, and I is the identity matrix.
- If using gradient descent, update the coefficients using the following formula: w = w - learning_rate * (X^T (Xw - y) + alpha * w)

Step 5: Evaluate the model:

- Evaluate the model on the testing set using a performance metric such as mean squared error (MSE) or R-squared.

Regression model with Ridge Regressor and Random Forest Regressor

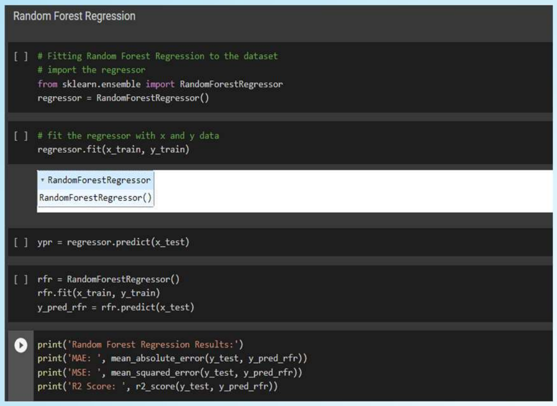
Code for Ridge Regressor :



Ridge Regression

```
[204] from sklearn.linear_model import Ridge
      import numpy as np

      rdg = Ridge(alpha = 0.5)
      rdg.fit(x_train, y_train)

      Ridge
      Ridge(alpha=0.5)

[205] yp = rdg.predict(x_test)

      from sklearn.metrics import accuracy_score
      from sklearn.metrics import classification_report
      from sklearn.metrics import confusion_matrix
      from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score

[195] print('Ridge Regression Results:')
      print('MAE: ', mean_absolute_error(y_test, yp))
      print('MSE: ', mean_squared_error(y_test, yp))
      print('R2 Score: ', r2_score(y_test, yp))
```

Code for Random Forest Regressor:



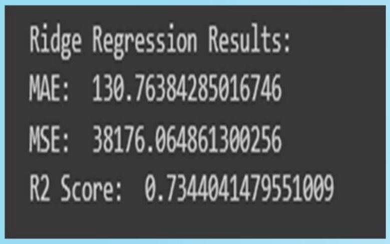
Random Forest Regression

```
[ ] # Fitting Random Forest Regression to the dataset
    # import the regressor
    from sklearn.ensemble import RandomForestRegressor
    regressor = RandomForestRegressor()

[ ] # fit the regressor with x and y data
    regressor.fit(x_train, y_train)

    RandomForestRegressor
    RandomForestRegressor()

[ ] ypr = regressor.predict(x_test)

[ ] rfr = RandomForestRegressor()
    rfr.fit(x_train, y_train)
    y_pred_rfr = rfr.predict(x_test)

    print('Random Forest Regression Results:')
    print('MAE: ', mean_absolute_error(y_test, y_pred_rfr))
    print('MSE: ', mean_squared_error(y_test, y_pred_rfr))
    print('R2 Score: ', r2_score(y_test, y_pred_rfr))
```

Evaluating the model with measures.

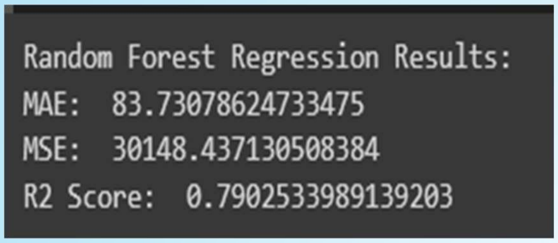
- After training the regression models on the dataset, we can evaluate their performance using various evaluation metrics such as Mean Squared Error, R2 Score, Mean Absolute Error

Output for Ridge Regression model :



```
Ridge Regression Results:
MAE:  130.76384285016746
MSE:  38176.064861300256
R2 Score:  0.7344041479551009
```

Output for Random Forest Regression model :



```
Random Forest Regression Results:
MAE:   83.73078624733475
MSE:   30148.437130508384
R2 Score:  0.7902533989139203
```

Comparison Analysis of Result

- After applying both Ridge Regression and Random Forest Regression on the insurance dataset, we found that Random Forest Regression performed better than Ridge Regression. This was observed in the R2 scores, with the Random Forest Regression model achieving a higher R2 score than the Ridge Regression model.
- The R2 score for the Random Forest Regression model was 0.79, while the R2 score for the Ridge Regression model was 0.73. This indicates that the Random Forest Regression model explains more of the variance in the target variable, charges, than the Ridge Regression model.
- Although both models performed well, the Random Forest Regression model was able to capture more complex relationships between the independent and dependent variables due to its ability to handle non-linear relationships and interactions. Overall, the Random Forest Regression algorithm seems to be a better choice for this particular dataset, as it produced a more accurate and robust prediction of insurance premiums.

Observation and Conclusion

- From our analysis of the insurance dataset, we observed that both Ridge Regression and Random Forest Regression algorithms can be used to predict insurance premiums. However, the Random Forest Regression model produced better results than the Ridge Regression model, achieving a higher R2 score. This could be attributed to the Random Forest Regression's ability to handle complex relationships and interactions between the independent and dependent variables.
- We also observed that certain features of the dataset, such as age and smoking status, had a strong correlation with the insurance premiums, while other features like region had a weaker correlation. This indicates that certain factors have a bigger impact on the insurance premiums than others, and these factors should be given more weight when predicting insurance premiums.
- In conclusion, the choice of regression algorithm to predict insurance premiums depends on the specific characteristics of the dataset and the complexity of the relationships between the independent and dependent variables. In this particular case, Random Forest Regression was found to be more effective in predicting insurance premiums due to its ability to handle non-linear relationships and interactions. Overall, insurance companies can benefit from using regression models to predict insurance premiums as it can help them make more accurate pricing decisions and improve their risk management strategies.