Machine Learning Basics: HR Analytics

Understanding Attrition in HR

Elton Grivith D Souza 5 September 2021

Machine learning tasks can be gathered into the four following categories:

	Supervised learning	Unsupervised learning
Discrete	Classification or categorization	Clustering
Continuous	Regression	Dimensionality reduction

This article mainly focuses on classification. Specifically, this article describes the basis of this task and Illustrates the main concepts onto the IBM_HR_Attrition_Rate_Analytics dataset.

The structure of this article is as following:

- Problem Definition
- Data Analysis
- Pre-processing
- Building Machine Learning (ML) models
- Hyper - Parameter Tuning

Problem Definition

Every year a lot of companies hire a number of employees. The companies invest time and money in training those employees, not just this but there are training programs within the companies for their existing employees as well. The aim of these programs is to increase the effectiveness of their employees. But where HR Analytics fit in this? and is it just about improving the performance of employees?

HR Analytics

Human resource analytics (HR analytics) is an area in the field of analytics that refers to applying analytic processes to the human resource department of an organisation in the hope of improving employee performance and therefore getting a better return on investment. HR analytics does not just deal with gathering data on employee efficiency. Instead, it aims to provide insight into each process by gathering data and then using it to make relevant decisions about how to improve these processes.

HR Attrition

Attrition in human resources refers to the gradual loss of employees overtime. In general, relatively high attrition is problematic for companies. HR professionals often assume a leadership role in designing company compensation programs, work culture, and motivation systems that help the organisation retain top employees.

How does Attrition affect companies? and how does HR Analytics help in analysing attrition? We will discuss the first question here and for the second question, we will write the code and try to understand the process step by step.

Attrition Affecting Companies

A major problem in high employee attrition is its cost to an organisation. Job postings, hiring processes, paperwork, and new hire training are some of the common expenses of losing employees and replacing them. Additionally, regular employee turnover prohibits

your organisation from increasing its collective knowledge base and experience over time. This is especially concerning if your business is customer-facing, as customers often prefer to interact with familiar people. Errors and issues are more likely if you constantly have new workers.

Data Analysis

Step 1: Importing the libraries

We first import most of the necessary libraries.

```
# Importing Libraries and filtering warnings
import pandas as pd
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from scipy import stats
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report,roc_curve,roc_auc_s
core,plot_roc_curve
import scikitplot as skplt
from sklearn.model_selection import GridSearchCV

import warnings
warnings.filterwarnings('ignore')
```

Step 2: Reading the Dataset

```
# Reading Dataset
df=pd.read_csv('ibm-hr-analytics-employee-attrition-performance/WA_Fn-
UseC_-HR-Employee-Attrition.csv')
```

Our dataset is in the csv format so we use the read_csv function to read our dataset.

You can then use `df.head()` or `df.tail()` to view some of the data in the dataframe.

Step 3: Dataset Information

This step allows us to view basic information about our dataframe. We can then plan our approach to exploring, processing and modelling our data based on this understanding.

```
# (Rows, Columns in df)
> df.shape
# Memory size of df
> df.size
# Base information of df
> df.info()
# Statistical description of numerical columns in df
> df.describe()
```

The output for df.info() is provided below:

```
> <class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
#   Column                    Non-Null Count  Dtype
---  ------                    --------------  --
0   Age                       1470 non-null   int64
1   Attrition                 1470 non-null   object
2   BusinessTravel            1470 non-null   object
3   DailyRate                 1470 non-null   int64
4   Department                1470 non-null   object
5   DistanceFromHome          1470 non-null   int64
6   Education                 1470 non-null   int64
7   EducationField            1470 non-null   object
8   EmployeeCount             1470 non-null   int64
9   EmployeeNumber            1470 non-null   int64
10  EnvironmentSatisfaction   1470 non-null   int64
11  Gender                    1470 non-null   object
12  HourlyRate                1470 non-null   int64
13  JobInvolvement            1470 non-null   int64
14  JobLevel                  1470 non-null   int64
15  JobRole                   1470 non-null   object
16  JobSatisfaction           1470 non-null   int64
17  MaritalStatus             1470 non-null   object
18  MonthlyIncome             1470 non-null   int64
19  MonthlyRate               1470 non-null   int64
20  NumCompaniesWorked        1470 non-null   int64
21  Over18                    1470 non-null   object
22  OverTime                  1470 non-null   object
23  PercentSalaryHike         1470 non-null   int64
24  PerformanceRating         1470 non-null   int64
25  RelationshipSatisfaction  1470 non-null   int64
26  StandardHours             1470 non-null   int64
27  StockOptionLevel          1470 non-null   int64
28  TotalWorkingYears         1470 non-null   int64
29  TrainingTimesLastYear     1470 non-null   int64
30  WorkLifeBalance           1470 non-null   int64
31  YearsAtCompany            1470 non-null   int64
32  YearsInCurrentRole        1470 non-null   int64
33  YearsSinceLastPromotion   1470 non-null   int64
34  YearsWithCurrManager      1470 non-null   int64
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

Step 4: Exploring the data

We now understand the dataframe well enough to explore it. This step involves univariate and multivariate analysis that can then be used to pre-process the data to give us better models. A few suggestions have been provided below. Feel free to further explore and visualise the data.

```
# Unique values in `column_name`
> df.`column_name`.unique()

# value count in `column_name`
> df.`column_name`.value_counts
```

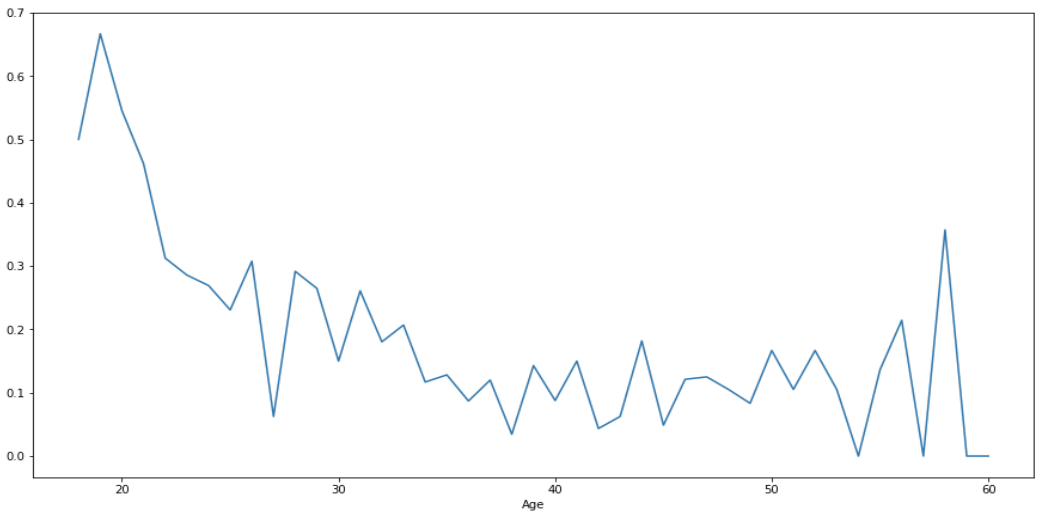
Machine learning models require all input and output variables to be numeric. This means that if your data contains categorical data, you must encode it to numbers before you can fit and evaluate a model. It is also easier to visualise numeric data. The snippet given below can be used to encode your labels. We will be encoding each label independently.

```
> lab_enc=LabelEncoder()

# Feature Encoding
> data=lab_enc.fit_transform(df['column_name'])
  df['column_name']=data
```

The snippet above converts the input column into `*int64*` *d*type using the Label Encoder instance.

Snippets given below can be used to obtain different plots. We can use this to observe different behaviour in our data. Examples of such observations have been provided below.

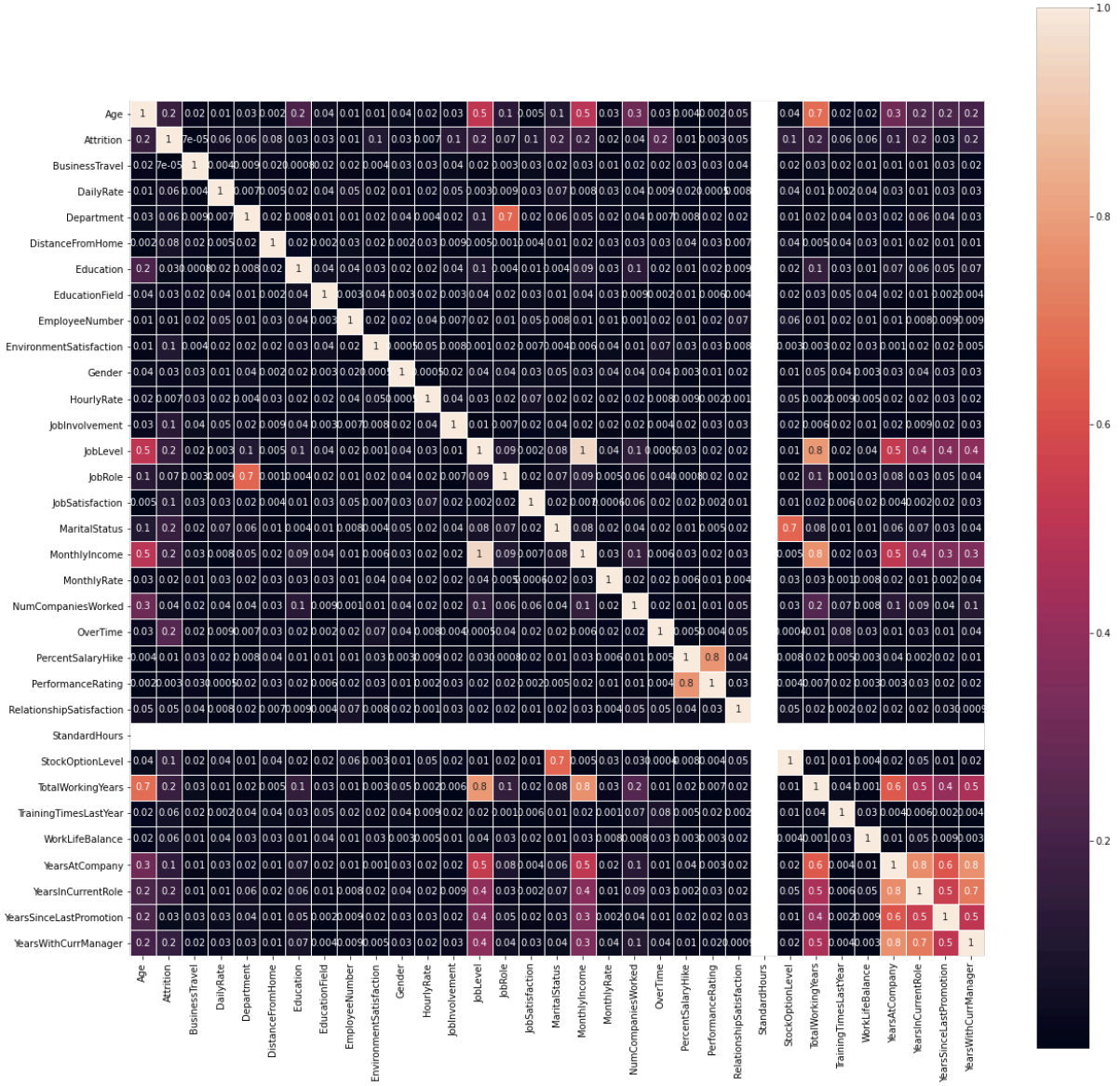
```
# Distribution of Age vs Attrition
> D=df.groupby('Age').mean()
  plt.figure(figsize=(15,8))
  D['Attrition'].plot()
```



The snippet above plots a graph depiction attrition among different age vector. From this, we can see that the attrition rate is higher for age between 18 and 25. This is an example of how observations can be made from different plots.

The snippet below can be used to plot a correlation matrix that defines how each column behaves with the other.

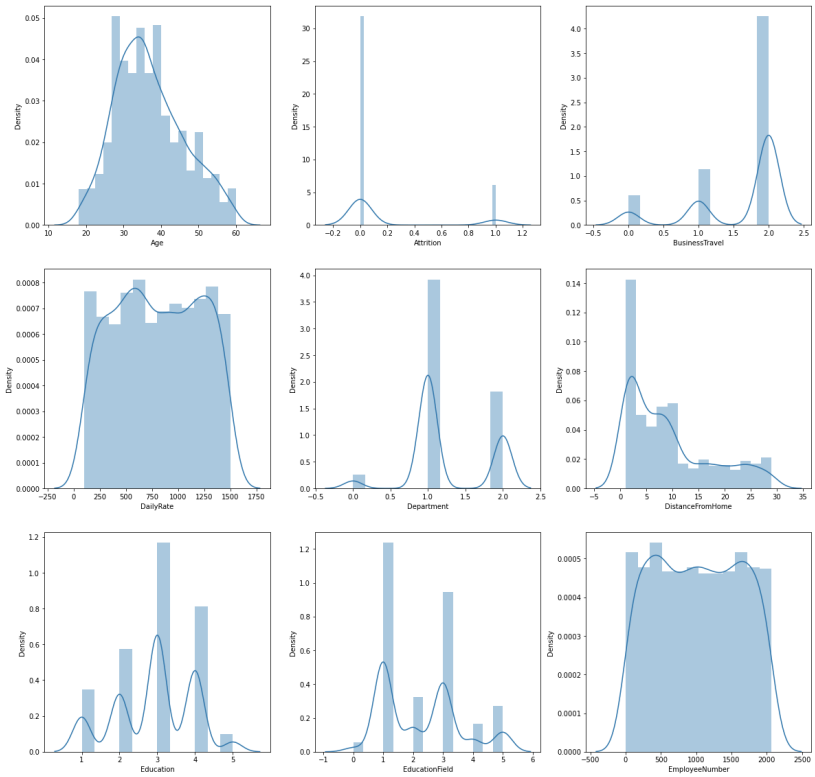
```
# Correlation plot
df_corr=df.corr().abs()
plt.figure(figsize=(20,20))
sns.heatmap(df_corr,annot=True,square=True,fmt='.1g',linewidth=1)
```



We can observe the most columns have very low correlation with our target label. And since multi-collinearity doesn't exist, we don't necessarily have to drop any columns.

We can also plot the distribution of our dataset. This can be used to check for skew, outliers, value scarcity etc.

```
#Distribution of the df
plt.figure(figsize=(20,20),facecolor='white')
plotnumber=1
for i in df:
    if plotnumber<=9:
        ax=plt.subplot(3,3,plotnumber)
        sns.distplot(df[i])
        plt.xlabel(i)
    plotnumber+=1
```



We can see that our data has some skew, feature value imbalance, data imbalance, outliers etc. We will need to process our data to treat these conditions.

Data Preprocessing

Data preprocessing can refer to manipulation or dropping of data before it is used in order to ensure or enhance performance, and is an important step in the data mining process. The phrase "garbage in, garbage out" is particularly applicable to data mining and machine learning projects. Thus we need to ensure that we fit our model with the best possible version of our data.

In our case, we first start this process by removing the outliers. For this, we have used Z-score method.

The Z-score method for outlier detection helps to understand if a data value is greater or lesser than mean and how far away it is from the mean. More specifically, Z score tells how many standard deviations away a data point is from the mean.

The snippet used in our code is provided below

```
#Removing outliers using zscore
z=np.abs(stats.zscore(df))
np.where(z>3)
```

This provides a numpy array where z is greater than 3. We can then remove that data using

```
index=(np.where(z>3)[0])
df=df.drop(df.index[index])
```

This will drop the rows where z is greater than 3. You can verify this step by using `df.shape()`. The difference in rows will tell you how many rows were dropped and the percentile can be calculated to monitor data loss.

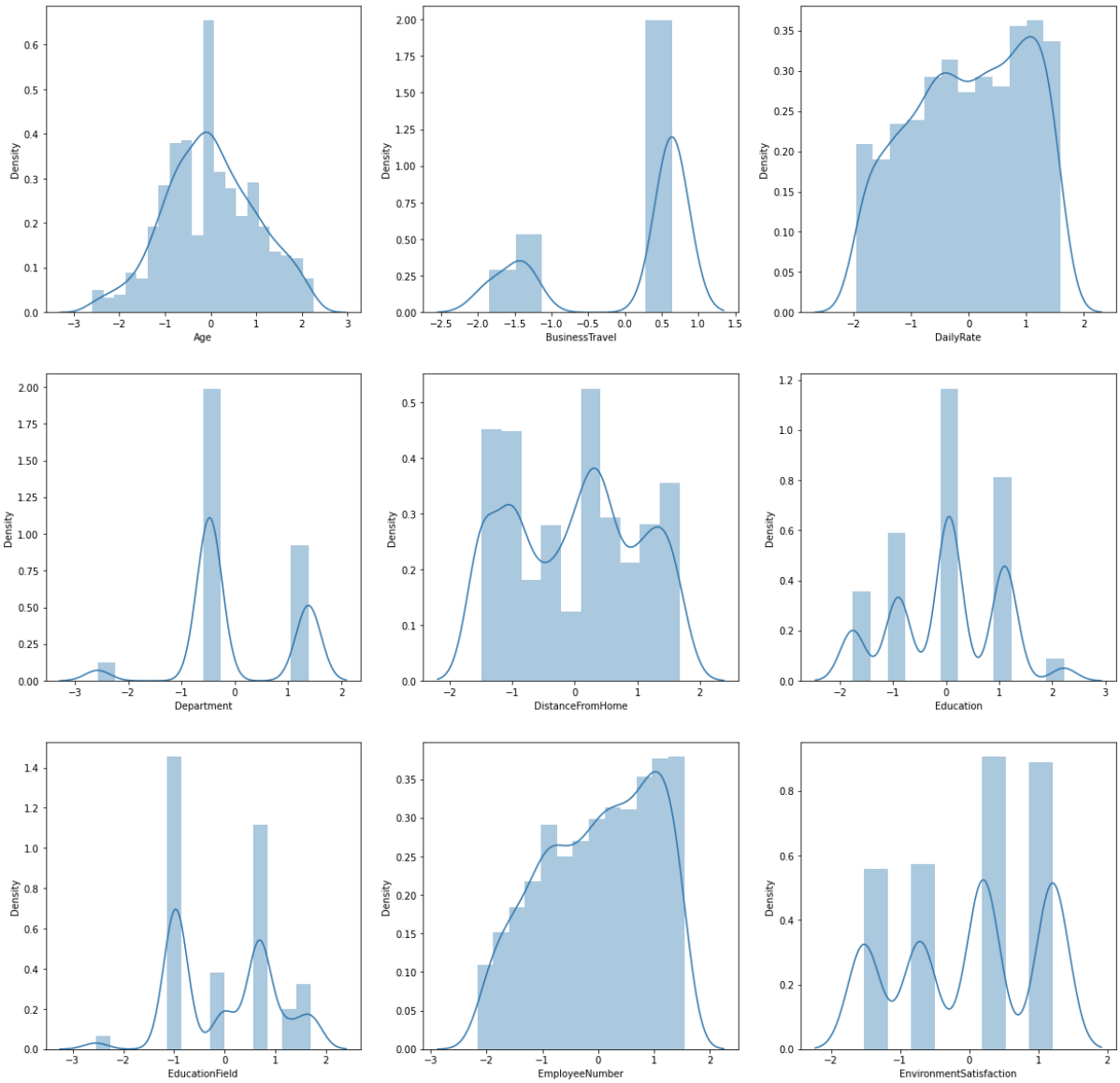
To fix the skew in our data, we first need to separate our target and feature columns. We can do this by

```
x=df.drop(columns=['Attrition'],axis=1)
y=df['Attrition']
```

We can then call the PowerTransformer method to fix the skew. You can execute `x.skew()` to monitor skew values of each feature. We have used the 'yeo-johnson' method to fix the skew in our data.

```
power = PowerTransformer(method='yeo-johnson', standardize=True)
data_new = power.fit_transform(x)
data_new=pd.DataFrame(data_new,columns=x.columns)
print(data_new.skew())
```

We can then plot the features to check their distributions. The snippet can be found in the data analysis section above.



From the above plot, we can observe the distribution of our features after removing the outliers and fixing skew. We can now use StandardScaler to transform our features. This is done because our estimators may behave badly if individual features do not look like standard normally distributed data.

```
x = data_new
scaler=StandardScaler()
x=scaler.fit_transform(x)
```

Building Machine Learning (ML) models

Now that our feature set looks satisfactory, we can start building our model.

In the snippet provided below, we have first split out data into training and testing data frames. The split size has been declared as 0.25, but, you can experiment and chose the split ratio that best suits your requirements. We have then created instances of our classifiers and built a list consisting of their identifiers.

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25)

#Classifiers :
knn=KNeighborsClassifier()
dt=DecisionTreeClassifier()
rf=RandomForestClassifier()
svm=SVC(probability=True)

m_list = [knn,dt,rf,svm]
```

We can now begin to fit our models. The following snippet fits our data with each of the instances in our `m_list`. It then prints the training and testing scores with CV and difference between testing accuracy and CV.

```
# Fitting Models
for m in m_list:
    m.fit(x_train,y_train)
#     Training------->>>>>
    print(m)
    train_preds = m.predict(x_train)
    print('Accuracy Score of
KNN :',accuracy_score(y_train,train_preds)*100)

#     Testing------->>>>>>
    print(m)
    test_preds = m.predict(x_test)
    te_acc = accuracy_score(y_test,test_preds)*100
    print('Accuracy Score of
KNN :',accuracy_score(y_test,test_preds)*100)


#     CV-------->>>>>
    print('cross validation scores below:--  \n',m)
    scr=cross_val_score(m,x,y,cv=5)
    print('Cross validation score of KNN: ',scr.mean()*100)
    print('Difference between CV_score and acc of KNN: ',(te_acc-
scr.mean()*100))
    print('\n')
    print('\n')
```

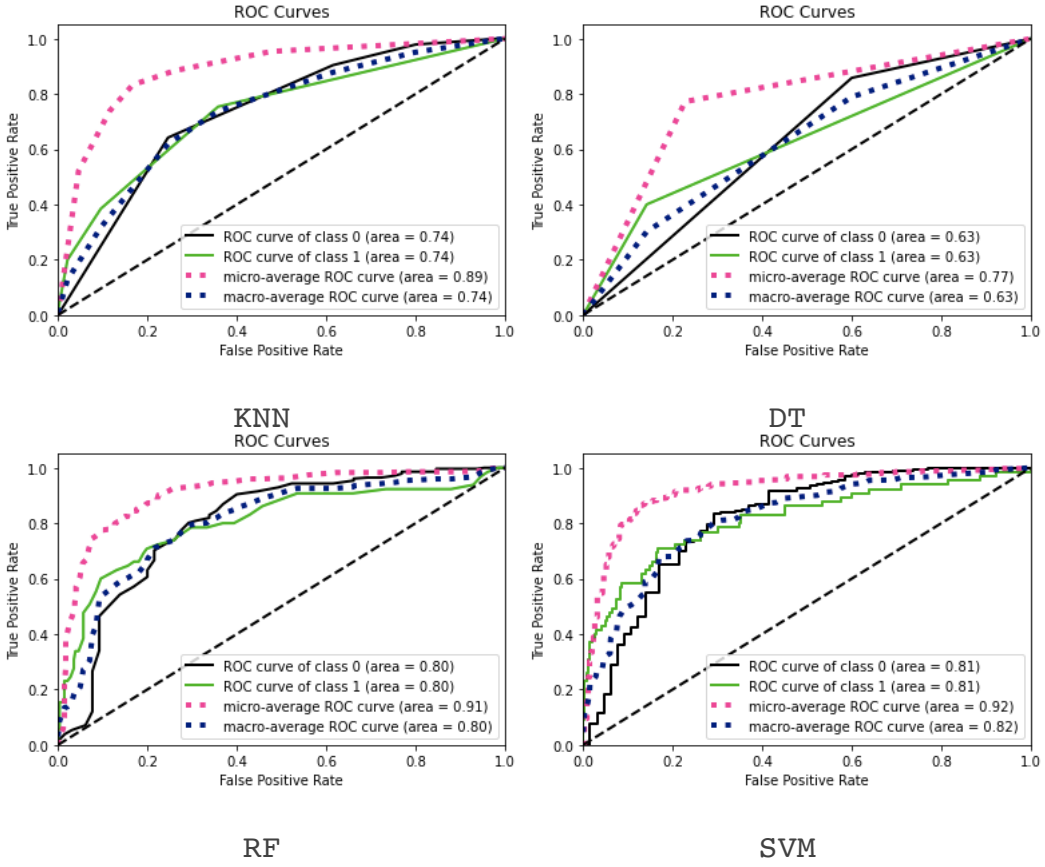
The output is provided below.

```
KNeighborsClassifier()
Accuracy Score of KNN : 87.01923076923077
KNeighborsClassifier()
Accuracy Score of KNN : 83.28530259365994
cross validation scores below:--
 KNeighborsClassifier()
Cross validation score of KNN:  84.71521699607824
Difference between CV_score and acc of KNN:  -1.429914402418305


DecisionTreeClassifier()
Accuracy Score of KNN : 100.0
DecisionTreeClassifier()
Accuracy Score of KNN : 77.23342939481267
cross validation scores below:--
 DecisionTreeClassifier()
Cross validation score of KNN:  77.57629275640859
Difference between CV_score and acc of KNN:  -0.3428633615959171


RandomForestClassifier()
Accuracy Score of KNN : 100.0
RandomForestClassifier()
Accuracy Score of KNN : 83.5734870317003
cross validation scores below:--
 RandomForestClassifier()
Cross validation score of KNN:  85.36373789055398
Difference between CV_score and acc of KNN:  -1.7902508588536818


SVC(probability=True)
Accuracy Score of KNN : 92.21153846153847
SVC(probability=True)
Accuracy Score of KNN : 85.5907780979827
cross validation scores below:--
 SVC(probability=True)
Cross validation score of KNN:  86.73454016570136
Difference between CV_score and acc of KNN:  -1.1437620677186544
```

From the output above we can see the accuracy, CV and the difference between CV_score and acc. We can use this data along with the ROC Curves to choose the optimal model. To find information about how we choose the optimal model, click here.

```
probas= `model_identifier`.predict_proba(x_test)
skplt.metrics.plot_roc(y_test,probas)
plt.show()
```

The roc curves obtained has been given below.



Hyper-parameter Tuning (HPT)

In this experiment we have we have used GridSearchCV for HPT. The example is demonstrated with Decision Tree classifier. The provided snippet is extensible to other classifiers when the appropriate parameter grid is used. Check the documentation of the classifier to build an appropriate parameter grid.

```
# Parameter Grid
grid_param={'criterion':['gini','entropy'],
            'max_depth':range(12,24,3),
            'min_samples_leaf':range(22,40,2),
            'min_samples_split':range(10,50,2)}
```

```
grid_search=GridSearchCV(dt,param_grid=grid_param,cv=5,n_jobs=-1,verbose
= 2)
grid_search.fit(x_train,y_train)
```

We can then fetch the best parameters from the conducted search by excuting

```
grid_search.best_params_
```

We can then build our model using this optimal parameter configuration.

```
# Model Fitted with best Parameters
dt=DecisionTreeClassifier(criterion='entropy', max_depth=12,
min_samples_leaf=24, min_samples_split=18)
dt.fit(x_train,y_train)
y_pred=dt.predict(x_test)
accuracy_score(y_test,y_pred)*100
```

This model is now fitted to provide the best possible result with the provided parameter space. In our experiment this Iteration of HPT improved the results from 77.23% to 83.57% (6.34% increase in accuracy metric).

We can now save and load our model accordingly.

```
import joblib

joblib.dump(dt,'DT_best.obj')
joblib.load('DT_best.obj')
```

To view this project in GitHub, click here.

References

1. https://numpy.org/doc/stable/
2. https://pandas.pydata.org/docs/
3. https://scikit-learn.org/stable/
4. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
5. https://www.geeksforgeeks.org/z-score-for-outlier-detection-python/
6. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.yeojohnson.html
7. https://towardsdatascience.com/introduction-to-regression-analysis-9151d8ac14b3
8. https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5
9. https://github.com/elton-dsza/DTrained_Projects/tree/main/HR%20Analytics%20Project-%20Understanding%20the%20Attrition%20in%20HR
10. https://towardsdatascience.com/machine-learning-basics-random-forest-regression-be3e1e3bb91a
11. https://github.com/dsrscientist/IBM_HR_Attrition_Rate_Analytics
12. https://towardsdatascience.com/gridsearchcv-for-beginners-db48a90114ee