EE769 : Report

Assignment 2: Two class prediction (Kaggle Competition)

Submitted by: Vivek P Revi (19310076)

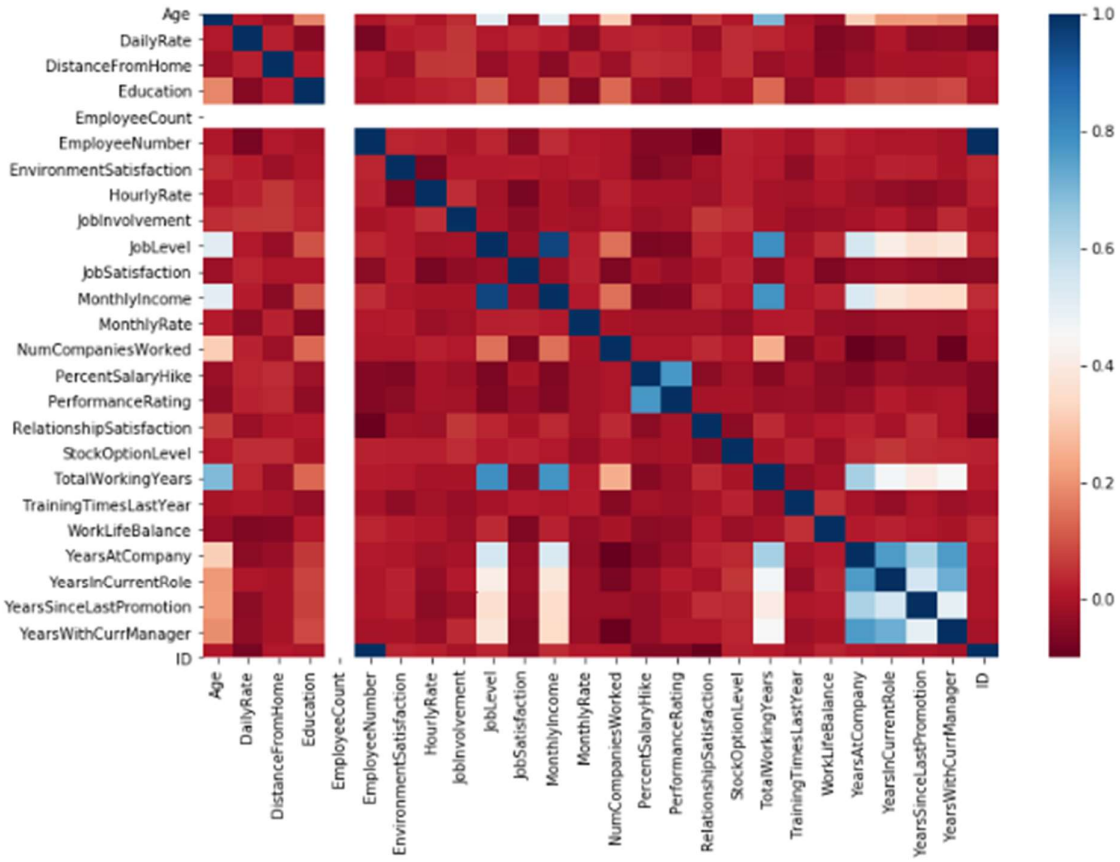
Studying the Data:

The training data set contains 34 columns and 1028 rows. Of the 34 columns, one is the output "Attrition" and we have 33 features left to create our model upon. There are no "null" values in any of the columns, so we do not need any Imputation to be done. The data type of the values in each column is to be looked at first.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1028 entries, 0 to 1027
Data columns (total 34 columns):
Age                       1028 non-null int64
Attrition                 1028 non-null int64
BusinessTravel            1028 non-null object
DailyRate                 1028 non-null int64
Department                1028 non-null object
DistanceFromHome          1028 non-null int64
Education                 1028 non-null int64
EducationField            1028 non-null object
EmployeeCount             1028 non-null int64
EmployeeNumber            1028 non-null int64
EnvironmentSatisfaction   1028 non-null int64
Gender                    1028 non-null object
HourlyRate                1028 non-null int64
JobInvolvement            1028 non-null int64
JobLevel                  1028 non-null int64
JobRole                   1028 non-null object
JobSatisfaction           1028 non-null int64
MaritalStatus             1028 non-null object
MonthlyIncome             1028 non-null int64
MonthlyRate               1028 non-null int64
NumCompaniesWorked        1028 non-null int64
OverTime                  1028 non-null object
PercentSalaryHike         1028 non-null int64
PerformanceRating         1028 non-null int64
RelationshipSatisfaction  1028 non-null int64
StockOptionLevel          1028 non-null int64
TotalWorkingYears         1028 non-null int64
TrainingTimesLastYear     1028 non-null int64
WorkLifeBalance           1028 non-null int64
YearsAtCompany            1028 non-null int64
YearsInCurrentRole        1028 non-null int64
YearsSinceLastPromotion   1028 non-null int64
YearsWithCurrManager      1028 non-null int64
ID                        1028 non-null int64
dtypes: int64(27), object(7)
memory usage: 273.2+ KB
```

From the above, we can see that 7 features are of "object" data type and remaining 26 (excluding attrition) is of "int64". In order to train on these features we will need to change the dtype of "object" into "categorical" values.

Now to analyse the relationship between all the features and to find out whether any correlation exists, we plot the **heatmap** using the seaborn library.



From the above heat map, we can conclude that the features like “EmployeeCount”, “EmployeeNumber” and “ID” does not have any impact on whether the employee decides to leave or stay in the company. So we drop those features from our dataframe.

```
# dropping less important features
ID = features["ID"]
df_fs_1 = features.drop(["EmployeeCount","EmployeeNumber","ID"], axis=1)
```

Now the features with “object” dtype need to be changed to **“category”.**

```
# encoding
df_fs_1['BusinessTravel'] = df_fs_1['BusinessTravel'].astype('category')
df_fs_1['Department'] = df_fs_1['Department'].astype('category')
df_fs_1['EducationField'] = df_fs_1['EducationField'].astype('category')
df_fs_1['Gender'] = df_fs_1['Gender'].astype('category')
df_fs_1['JobRole'] = df_fs_1['JobRole'].astype('category')
df_fs_1['MaritalStatus'] = df_fs_1['MaritalStatus'].astype('category')
df_fs_1['OverTime'] = df_fs_1['OverTime'].astype('category')
df_fs_1['JobRole'] = df_fs_1['JobRole'].astype('category')
```

Data Preprocessing:

The categorical features now need to be encoded into numerical values for training on a machine learning model. Here I have decided to use **"One Hot Encoding"**, because it considers every features equally important and avoids false assumptions as in the case of "Label Encoding".

```
# One hot encoding for some important features
df_fs_2 = df_fs_1.copy()
df_fs_3 = pd.get_dummies(df_fs_2, columns=['BusinessTravel'], prefix = ['b_trav']) #one hot for business travel
df_fs_3 = pd.get_dummies(df_fs_3, columns=['Department'], prefix = ['dept']) #one hot for department
df_fs_3 = pd.get_dummies(df_fs_3, columns=['EducationField'], prefix = ['edu']) #one hot for education
df_fs_3 = pd.get_dummies(df_fs_3, columns=['JobRole'], prefix = ['job']) #one hot for job
df_fs_3 = pd.get_dummies(df_fs_3, columns=['MaritalStatus'], prefix = ['marr']) #one hot for marital status
df_fs_3 = pd.get_dummies(df_fs_3, columns=['Gender'], prefix = ['gend']) #one hot for gender
df_fs_3 = pd.get_dummies(df_fs_3, columns=['OverTime'], prefix = ['ot']) #one hot for over time
```

Now since the features are all independent, they are having a different scale of values. To reduce the bias for a feature and thus to improve model prediction, we need to **"standardize"** the features. This brings every feature to a mean '0' and standard deviation '1'.

```
from sklearn import preprocessing

train_norm = X_train[X_train.columns[:]]
test_norm = X_test[X_test.columns[:]]

std_scale = preprocessing.StandardScaler().fit(train_norm)
X_train_norm = std_scale.transform(train_norm)
X_test_norm = std_scale.transform(test_norm)
```

Train and Validation Split:

To figure out which model performs best on the above processed data, I had split the dataset into train and validation(test set). Here 70% of the data was used for training and 30% was set aside for validation.

```
# Split data to test and train
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df_fs_3, attrition, train_size = 0.7, random_state = 2)
```

Now the above data was used for creating a model using various machine learning techiques.

Model using ANN:

Artificial Neural Network with 2 hidden layers and “relu” activation was found to give the best result on the given setup.

```
# Create Model
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(solver='lbfgs', alpha=1e-5,
                           hidden_layer_sizes=(2,), activation='relu', random_state=1)
classifier.fit(X_train_norm, y_train)

# Predicting the test set result
y_pred = classifier.predict(X_test_norm)

# Making the Confusion Matrix
cm = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {asc(y_pred, y_test)}")
print(("\nConfusion Matrix:"))
print(cm)
```

```
Accuracy: 0.8576051779935275

Confusion Matrix:
[[245  21]
 [ 23  20]]
```

Model using LDA:

Linear Discriminant Analysis was carried out on the above normalized data.

```
# Create model
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
classifier = LinearDiscriminantAnalysis()
classifier.fit(X_train_norm, y_train)

# Predicting the test set result
y_pred = classifier.predict(X_test_norm)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {asc(y_pred, y_test)}")
print(("\nConfusion Matrix:"))
print(cm)
```

```
Accuracy: 0.8737864077669902

Confusion Matrix:
[[253  13]
 [ 26  17]]
```

Model using SVM:

Support Vector Machine algorithm was also tried. Since output is two class, I have used “sigmoid” activation for the output.

```
# Create model
from sklearn.svm import SVC
classifier = SVC(kernel = 'sigmoid') #kernal can be sigmoid, polynomial and gaussian
# classifier = SVC(kernel = 'rbf') # for gaussian
# classifier = SVC(kernel = 'sigmoid') # for sigmoid [ONLY FOR 2 OUTPUTS]
# classifier = SVC(kernel = 'poly', degree = 8) # for polynomial, specify degree [SHOWS ERROR]
classifier.fit(X_train_norm, y_train)

# Predicting the test set result
y_pred = classifier.predict(X_test_norm)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {asc(y_pred, y_test)}")
print(("\nConfusion Matrix:"))
print(cm)
```

```
Accuracy: 0.8640776699029126

Confusion Matrix:
[[258   8]
 [ 34   9]]
```

Findings and Inferences:

Apart from LDA, SVM and ANN, various other models were also tried such as,

- Decision Tree
- QDA
- K-Nearest Neighbours
- Random Forrest

But these methods were either giving low accuracy or inconsistent result, even after hyperparameter tuning. So I decided to stick with LDA, SVM and ANN.

Of the three methods, LDA was giving better accuracy even with the non-standardized data. SVM and ANN give better accuracy only when the data has been standardized. This indicates that the data is **Linearly Separable** because we know that LDA works best for linearly separable data. It was observed that the LDA gives the same accuracy on standardized and non-standardized data.

Apart from “one-hot-encoding”, “label-encoding” was also tried. But most of the models were giving better results when “one-hot-encoding” was used. This may be because label encoding initializes categories into random values and while training model may make false assumptions that one value is greater than other in the same feature. This will result in poor accuracy. The downside of using “one-hot-encoding” is that it increases the number of features in the dataset.

Resampling the Data:

It was found that the above training dataset was highly **imbalanced**. Because the output "Attrition" column, consists of 590 "0s" and 129 "1s". This, in theory, will result in a bias for the model. To eliminate this and hopefully to improve accuracy, **Upsampling** and **Downsampling** methods were implemented, using "sklearn – resample"

Upsampling increases the number of minority class ("1s" in this case) by introducing dummy rows with the available data. On the other hand, Downsampling decreases the majority class("0s" in this case) by dropping random rows.

But the obtained model accuracy did not improve on trying these techniques, so it was not used in the final model.

Results using Best Models for Test Data Prediction:

All the above three models were done on test data set provided to see the accuracy. The accuracy obtained from these is listed in the table below.

Model	Accuracy on Kaggle
LDA	0.88383
SVM	0.89898
ANN	0.86363

Out of the three, we can see SVM gave the best accuracy. Thus it was decided to be used as the final model.

Learnings:

- To use scikit-learn library and its various API elements.
- To analyse and study, given a dataset.
- To preprocess the given dataset so that several machine learning models can be performed on the data.
- Importance of Normalizing/Standardizing the dataset to obtain good results.
- Using correlation matrix and heat map to find the relation between various features.
- Importance of train, validation and test data splits to compare various machine learning models.
- Various parameters involved in each machine learning modes.
- To tune the hyperparameters in some of the models.
- Sampling methods to balance highly imbalanced dataset.
- Read and analyse the confusion matrix obtained from each model.
- Various encoding methods "onehot", "label", "custom-label" etc.. and its advantages and disadvantages in data preprocessing.
- Getting insights into the dataset after analysing the results obtained from various models.