CO544 - Machine Learning
Project Report

E/15/119
E/15/202
E/15/208

Introduction to the problem

In this project we were given a heterogeneous dataset, which has a large amount of data. And the expected output is to classify the data as ‘Success’ or ‘Failure’. Since there are only two class labels, ‘Success’ or ‘Failure’, this problem falls into the classification problem type. So we are supposed to apply machine learning and data mining to solve this classification problem.

This dataset includes sixteen number attributes. But the attributes' real names are altered. They are only named as A1, A2,.. and A16. So we do not have a real understanding of real use of these data. But we have given the real data types of each and every attribute. ‘A16’ is the class attribute. Apart from the size of the data, it includes missing values.

Features of the dataset consisted of different data types. They are object, float, int, bool types.

In this project we used in built python machine algorithms.First we train some machine learning algorithms with the given dataset and later, we use those trained models to classify the given test dataset as ‘Success’ or ‘Failure’. For different models trained, their predicted results differ with each model according to their level of accuracy.

Explorations conducted on features

1. We first load the Python libraries such as numpy, pandas, scikit-learn and matplotlib which will help to analyze the data and build up a suitable model. Also loaded the training dataset. The last column is our target variable, ‘A16’, and the rest are the features. In the target attribute there were only 2 classes: 'Success’ and ‘Failure’.

2. One thing that we noticed was most of the algorithms work better with numerical inputs. And the imputation method we use later on to compute missing values , does not work with object data types. But we had several nominal attributes also. So we converted them into numerical values using the label encoding method. But label encoding method can only be used for categorical data types other than numerical data types. They did not work with object data types. So first we converted nominal object data types to categorical data types and numerical object data types to float type. Then we used label encoding method to convert categorical values into numerical values. Better encoding of categorical data can mean better model performance.

```
# change object type to float of some columns which the data type
#has been identified as object,but they are of type float
dataset["A2"] = dataset["A2"].astype(float)
dataset["A14"] = dataset["A14"].astype(float)

# change object type to category
dataset["A1"] = dataset["A1"].astype('category')
dataset["A3"] = dataset["A3"].astype('category')
dataset["A4"] = dataset["A4"].astype('category')
dataset["A6"] = dataset["A6"].astype('category')
dataset["A9"] = dataset["A9"].astype('category')
dataset["A15"] = dataset["A15"].astype('category')
dataset["A16"] = dataset["A16"].astype('category')
```

```
# Label Encoding
dataset["A1"] = dataset["A1"].cat.codes
dataset["A3"] = dataset["A3"].cat.codes
dataset["A4"] = dataset["A4"].cat.codes
dataset["A6"] = dataset["A6"].cat.codes
dataset["A9"] = dataset["A9"].cat.codes
dataset["A15"] = dataset["A15"].cat.codes
dataset["A16"] = dataset["A16"].cat.codes
```

And also we tried one hot encoding in which the categorical variable is removed and a new binary variable is added for each unique integer value. But it didn't work as label encoding.

```
#one-hot encoding to object type columns
onehote_data = newer_data.copy()
onehote_data = pd.get_dummies(onehote_data, columns=['A3'], prefix=['A3'])
onehote_data = pd.get_dummies(onehote_data, columns=['A4'], prefix=['A4'])
onehote_data = pd.get_dummies(onehote_data, columns=['A6'], prefix=['A6'])
onehote_data = pd.get_dummies(onehote_data, columns=['A9'], prefix=['A9'])
onehote_data = pd.get_dummies(onehote_data, columns=['A15'], prefix=['A15'])
```

```
onehote_data.head()
```

	A1	A2	A5	A7	A8	A10	A11	A12	A13	A14	...	A9_ff	A9_h	A9_j	A9_n	A9_o	A9_v	A9_z	A15_g	A15_p	A15_s
0	b	30.83	0.00	0.0	True	1.25	True	1.0	False	202.0	...	0	0	0	0	0	1	0	1	0	0
1	a	58.67	4.46	560.0	True	3.04	True	6.0	False	43.0	...	0	1	0	0	0	0	0	1	0	0
2	a	24.50	0.50	824.0	False	1.50	True	0.0	False	280.0	...	0	1	0	0	0	0	0	1	0	0
3	b	27.83	1.54	3.0	True	3.75	True	5.0	True	100.0	...	0	0	0	0	0	1	0	1	0	0
4	b	25.00	11.25	1208.0	True	2.50	True	17.0	False	200.0	...	0	0	0	0	0	1	0	1	0	0

5 rows × 43 columns

3. This dataset included missing values as '?'. As a result of that some attributes that ideally should be in numeric data type (int or float), were represented as a string data type. So we replaced those missing values with a non string value (such as numpy.nan or 0) and then we converted those attributes into a numeric data type.

4. There were no missing values included in A16 (class attribute). So it was very easy in training the classification algorithms with the given dataset.

5. Checking for missing values in a Python Data can be accomplished using the isnull().sum() function, on the dataset.
   Output shows the number of missing values in each column of the dataset.

6. In the given training dataset, there were some missing values. So they have to be treated before we train the models.

   - We used an imputation method called Regression Imputation. What we did in this method was, we estimated the missing values by Regression using other variables as the parameters.

   - Initially we imputed all the features with missing values using a trivial method like 'Simple Random Imputation' (imputed the missing data with random observed values of the variable) which was later followed by Regression Imputation of each of the variables iteratively.

     For that we used a function called 'random_imputation' which replaces the missing values with some random observed values of the feature. This method was repeated for all the features containing missing values, after which they serve as parameters in the regression model to estimate other variable values. It was done as follows.

```
missing_columns = ["A1", "A2", "A3", "A4", "A6", "A9", "A14"]

def random_imputation(dataset, feature):
    number_missing = dataset[feature].isnull().sum()
    observed_values = dataset.loc[dataset[feature].notnull(), feature]
    dataset.loc[dataset[feature].isnull(), feature + '_imp'] = np.random.choice(observed_values, number_missing,
                                                                                replace=True)

    return dataset

for feature in missing_columns:
    dataset[feature + '_imp'] = dataset[feature]
    dataset = random_imputation(dataset, feature)
```

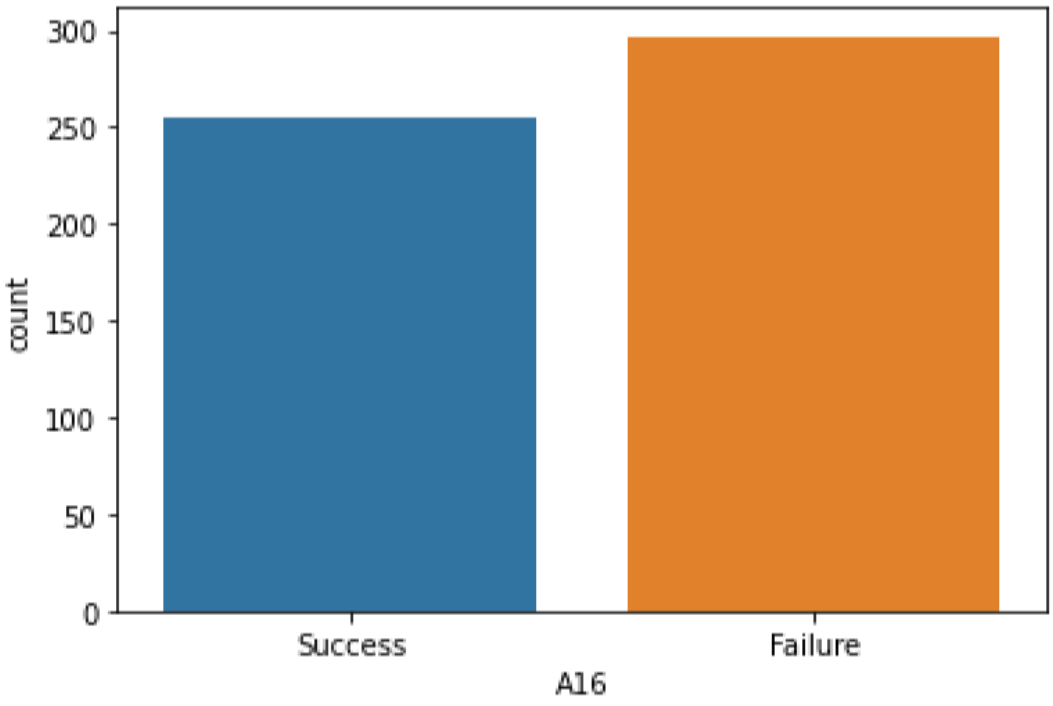
     Then we used Deterministic Regression Imputation to replace the missing data with the values predicted in our regression model and repeated this process for each variable. This was done as follows.

```
deter_data = pd.DataFrame(columns=["Det" + name for name in missing_columns])

for feature in missing_columns:
    deter_data["Det" + feature] = dataset[feature + "_imp"]
    parameters = list(set(dataset.columns) - set(missing_columns) - {feature + '_imp'})

    X = dataset[parameters]
    y = dataset[feature + '_imp']

    # Create a Linear Regression model to estimate the missing data
    model = linear_model.LinearRegression()
    model.fit(X, y)

    # observe that I preserve the index of the missing data from the original dataframe
    deter_data.loc[dataset[feature].isnull(), "Det" + feature] = model.predict(dataset[parameters])[
        dataset[feature].isnull()]
```

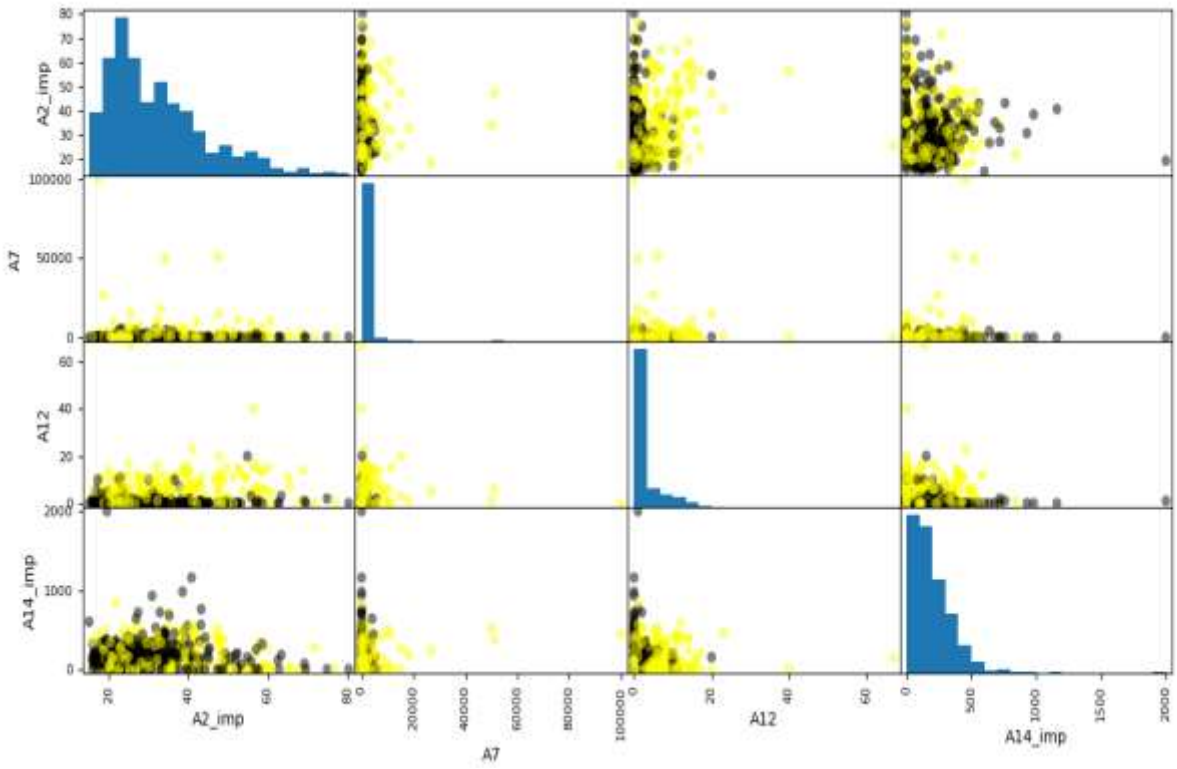
```
for feature in missing_columns:
    dataset[feature + '_imp'] = deter_data['Det' + feature]

for feature in missing_columns:
    dataset[feature] = dataset[feature + '_imp']
```

7. After computing the missing values, we checked whether there are any missing values remaining to handle using the same functions isnull().sum() on the dataset. (there were no missing values)

8. In the A16 attribute of the given dataset, there were 297 Failures and 255 Success. So it is not a biased classification. Therefore it was not needed to balance the class attribute.



9. We used a scatter matrix to plot some of the numeric variables we have in the dataset against each other to gain an idea about the correlation between them.
    - The diagonal shows the distribution of some numeric variables of our training dataset.
    - In the other cells of the plot matrix, we have the scatterplots (i.e. correlation plot) of each variable combination of our dataframe



10. We created training and test sets by splitting the training dataset into a proportion of 75% of data into training and rest for the test set, and applied scaling as a prior step for training the models.

```
# split the data set as training set and test set randomly
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state=0)

# apply scaling
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Final approach(s) to predictions

Motivation and Reasoning

- We trained different types of python machine learning algorithms with the given dataset. We used 7 different types of such algorithms (such as Logistic Regression, Decision Tree Classifier (used different maximum depths), K-Neighbors Classifier, Linear Discriminant Analysis, GaussianNB, Support Vector machine and Random Forest algorithm).

- To choose the best model to predict the given test dataset, we divided the given dataset into 2 sets as training data and test data. After we separate out the data as test and train, basically we build classifiers using the train set and evaluate it using the test set. Since we already know the correct class attribute result for this test data (as we extracted the test data from the training dataset itself to train the models), we were able to calculate the test accuracies for each and every model. Then the model which gave the highest accuracy was selected as the best model to get the final predictions.

- When dividing the given dataset as training data and test data, about 75% of data was taken as the training data and the remaining data was selected to test the accuracy of models. Based on the proportion we split the dataset, if the dataset is splitted such that there is more proportion for training data, that will give more accuracy. If the dataset is splitted such that there is more proportion for test data, that will give more reliability.

Final approach

LR
Accuracy of Logistic regression classifier on training set: 0.85
Accuracy of Logistic regression classifier on test set: 0.86

DT
Accuracy of Decision Tree classifier on training set: 1.00
Accuracy of Decision Tree classifier on test set: 0.89

DTdep3
Accuracy of Decision Tree classifier on training set with maximum depth of 3 : 0.84
Accuracy of Decision Tree classifier on test set with maximum depth of 3 : 0.84

DTdep4
Accuracy of Decision Tree classifier on training set with maximum depth of 4 : 0.88
Accuracy of Decision Tree classifier on test set with maximum depth of 4 : 0.88

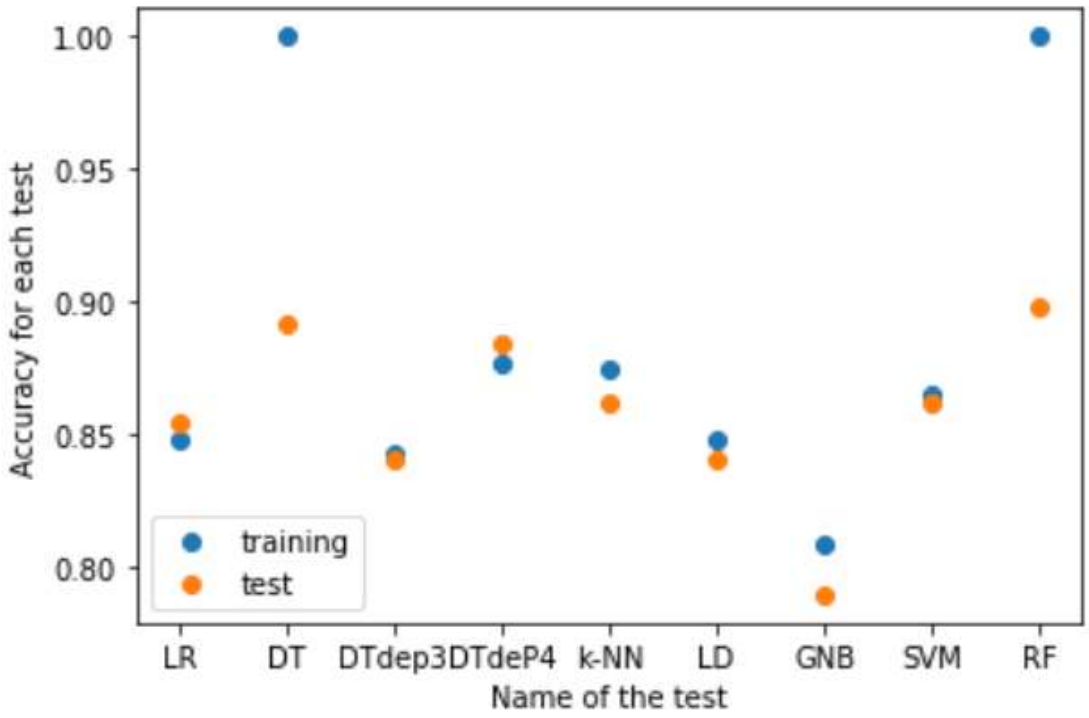
k-NN
Accuracy of K-NN classifier on training set: 0.87
Accuracy of K-NN classifier on test set: 0.86

LD
Accuracy of LDA classifier on training set: 0.85
Accuracy of LDA classifier on test set: 0.84

GNB
Accuracy of GNB classifier on training set: 0.81
Accuracy of GNB classifier on test set: 0.79

SVM
Accuracy of SVM classifier on training set: 0.86
Accuracy of SVM classifier on test set: 0.86

RF
Accuracy of Random forest classifier on training set: 1.00
Accuracy of Random forest classifier on test set: 0.90



So following the above steps, we got the highest test accuracy for the Random Forest classifier. Note :But the accuracies change according to the way how the dataset was divided into test and train.

- But the given test dataset (test dataset in kaggle) did not give a higher accuracy for Random Forest classifier as we expected. So we tried other algorithms one by one to get the final predictions.
- Then from those results, we choose the best model which gives the highest accuracy to predict the final results.
- That model was the **Support Vector Machine algorithm**.
- Source code can be found in **CO544-Project-Final-Result.py**

Alternatives considered

- There are several methods to compute missing values. Some commonly used methods are;
  - dropping instances
  - dropping attributes
  - imputing the attribute mean for all missing values
  - imputing the attribute median for all missing values
  - imputing the attribute mode for all missing values
  - using regression to impute attribute missing values
  - Using different algorithms like xgboost

  So we tried several methods out of these to get the best accuracy levels for the models.

- When handling missing values we also tried dropping the instances method. But a problem arose when the validation set also contained instances with missing values. And managed to handle it, by replacing missing values with attribute mode. And in that we used one hot encoding to replace all categorical data with binary values and it gave 94% accuracy for the last 50% of the validation dataset. But when considering the whole validation set the accuracy was lower than the final model.

- When handling missing values we can use algorithms which need to handle the missing values by ourselves or we can use algorithms which handle the missing values by itself. We used one such classifier called xgboost classifier (you can find the source code for this in **CO544-Project-xgboost.py**).
  - In there we only replaced the missing values as *numpy.nan* and then trained the model with the given dataset (no need of handling missing values by ourselves).
  - It gave a higher accuracy for the first 50% of the validation dataset (higher accuracy than the Support Vector Machine algorithm - 89.855%).
  - But accuracy dropped when considering the whole validation dataset.
  - So that we did not choose it for final predictions.

- We used another method to handle the missing values. It is by imputing the attribute mean and mode for all missing values. (you can find the source code for this in **CO544-Project-MeanMode.py**).
    - first we replaced the missing values as numpy.nan
    - Then, mean was taken for the attributes with numerical data types.(for instance: in training data - 'A2' , 'A14')
    - And mode was taken for the attributes with nominal data types. (for instance: in training data - 'A1' , 'A3', 'A4' , 'A6' , 'A9')
    - Then those values were substituted for the appropriate missing value places.
    - We got the highest accuracy for the Random Forest classifier algorithm.

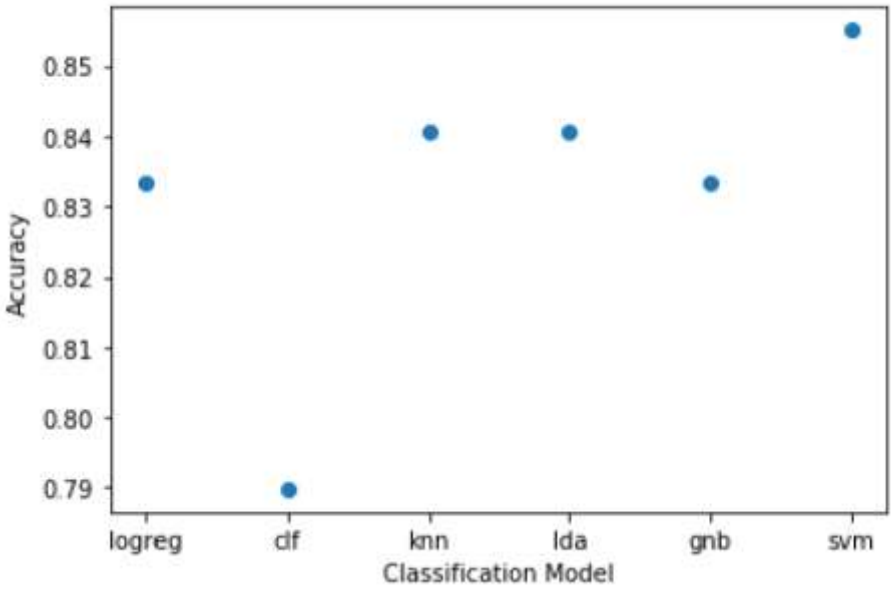
    - But in the final approach, Random Forest classifier algorithm gave 92.753% accuracy while K-NN algorithm gave the highest accuracy that we got as the final result(95.652%) for this implementation.

    - When we replace the missing data with some common value like mean and mode we might under/over estimate it. In other words, we add some bias to our estimation. So that we did not choose it for final predictions.

- Also tried to handle the missing values by dropping the instances which contain the missing values. This method is said to be the fastest and easiest method to handle the missing values. This was achieved by calling dropna() function. But we learnt that this method reduces the quality of our model as it reduces sample size because it works by deleting all other observations where any of the variables is missing. Since in the training data set there were 7 attributes out of 15 feature attributes with missing values. Removing all of them limited the space of the model. So we avoid using that in the final modal.

- As another method we used to encode the missing values as unique values. (source code can be found in **CO544-Project.py**)
    - For that, first we replaced all the numeric missing values with "0" and changed the attribute type to float.
    - We did not change other missing values in attributes which are of string data type
    - Then using *LabelEncoder* all the values are encoded with a numeric value so that the missing values are also given a unique value in their respective attribute field.
    - After doing this dataset is trained with 6 models (such as Logistic Regression, Decision Tree Classifier, K-Neighbors Classifier, Linear Discriminant Analysis, GaussianNB and Support Vector machine).
    - Then we selected the model which gave the highest accuracy for test data which was a part of the given dataset.
    - We got the highest accuracy for the **Support Vector Machine** algorithm.

- So as in the final approach this again gave the highest accuracy that we got as the final result(95.652%).



- Here logreg = Logistic Regression, clf = Decision Tree Classifier, knn = K-Neighbors Classifier, lda = Linear Discriminant Analysis, gnb = GaussianNB, svm = Support Vector machine

- So to avoid these kinds of problems with missing values we thought to use an imputation method called Regression Imputation. That was what we used for our final predictions.
- There are several methods of handling categorical data. We tried label encoding and one hot encoding and we thought label encoding is the most suitable one for this dataset. So we used it in the final modal.
- When training the random forest model, we tried both RandomForestRegressor and RandomForestClassifier classes. But RandomForestRegressor gave a very low accuracy whereas RandomForestClassifier gave a high accuracy for the test dataset. We realized that it was because we used the regressor for a classification problem. So it is a bad practice to use regression for a classification task.

```
#use model Random forest
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators=100, random_state=0)
regressor.fit(X_train, y_train)
```

```
Accuracy of Random forest regressor on training set: 0.95
Accuracy of Random forest regressor on test set: 0.60
```

```
# use model Random forest
from sklearn import model_selection
rfr = RandomForestClassifier(n_estimators=100, random_state=0)
rfr.fit(X_train, y_train)
```

```
Accuracy of Random forest classifier on training set: 1.00
Accuracy of Random forest classifier on test set: 0.91
```

Reasons to choose the final approach

- 88.405% accuracy for the first 50% of the validation data set and 95.652% accuracy for the other 50% of the validation data set.

Conclusion

From this project, we attempted to study comparative performances of different supervised machine learning algorithms in predicting the results of a dataset with new instances along with the python libraries; numpy, pandas, Scikit-learn etc. Here we used several machine learning classification models on the same dataset and analyzed their accuracy to choose the best model. We learned and obtained several insights about different classification models and the keys to develop one with a good performance.

Although classifiers like Decision Tree and Random Forest give higher training accuracy the test accuracies are very low. They are overfitting to the dataset. In the Decision Tree we avoided overfitting by setting the maximum depth limit.

The algorithms which have the highest accuracy may not always be the best. It depends on the number of missing values, how we handle missing values, number of data that is used to train the model etc.