CS 235 - Data Mining Techniques

Student Performance Predictor

Implementation Correctness Report

Team Members - Madhusudhan Tungamitta (mtung007), Balaji Arunachalam (barun001), Shreyas Ghuge (sghug001), Naman Mittal (nmitt005), Imran Shahid (ishah011)

Introduction

The task of predicting a student's performance was not as easy as we thought. Thanks to machine learning, we can finally reach 68.35% accuracy. We pre-processed data taken from the kaggle (Also available in UCI Machine Learning Repository). This data was then used to train machine learning models using scikit learn. We used accuracy obtained with these predefined libraries and implemented the same algorithms from scratch in python to achieve accuracy similar to them. Having done the same, We are proud to achieve 95% similar accuracies with in-depth knowledge of each algorithm. This report shows how we proceeded with each algorithm and improved results with our constant efforts and optimizations.

Algorithms Used

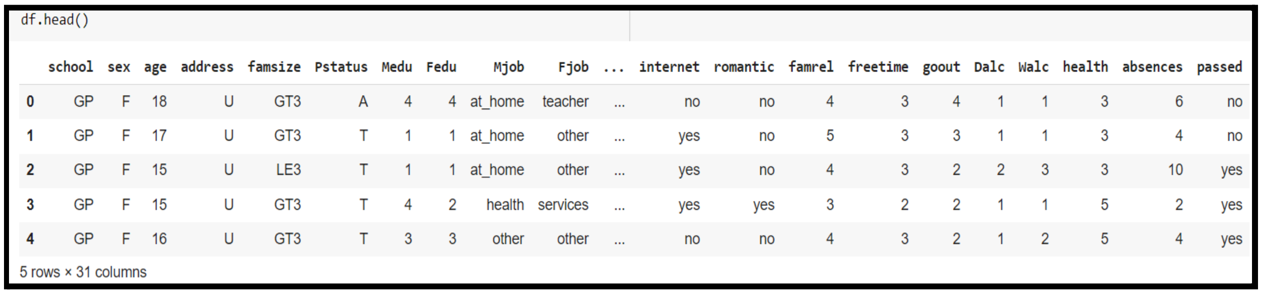
Our problem can be called a binary classification problem as the output will be either Pass or Fail. So we chose the following algorithms -

1) Decision Trees
2) Random Forest Classifiers
3) SVM - Support Vector Machine
4) Naive Bayes Classifiers
5) Logistic Regression

Steps followed in the project

1) Preprocessing the data

Initially, data contained 31 columns in which the last column, 'passed', is our target variable i.e we will use machine learning to use the other 30 columns and predict the value of 'passed'.



```
df.head()
```

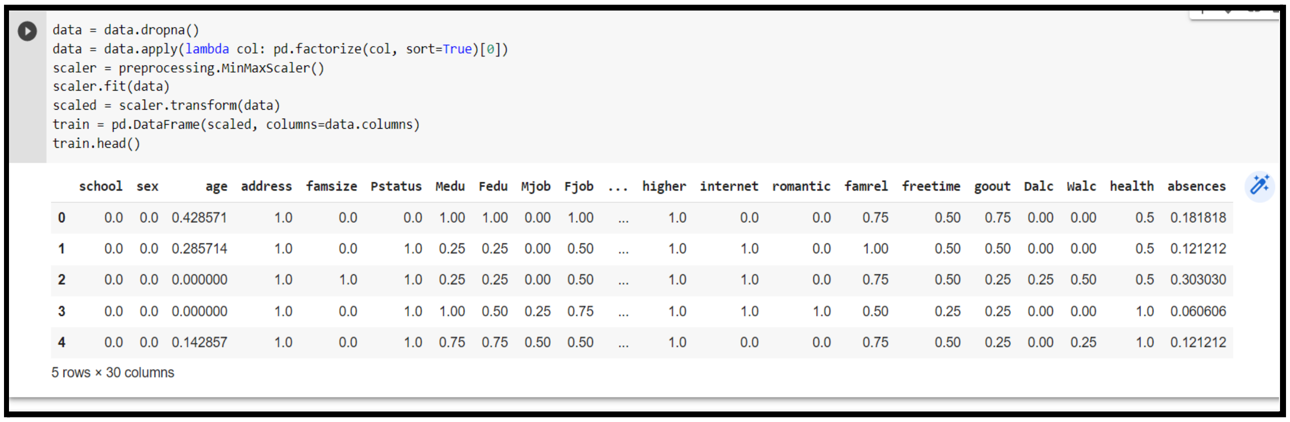
	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	internet	romantic	famrel	freetime	goout	Dalc	Walc	health	absences	passed
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	no	no	4	3	4	1	1	3	6	no
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	yes	no	5	3	3	1	1	3	4	no
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	yes	no	4	3	2	2	3	3	10	yes
3	GP	F	15	U	GT3	T	4	2	health	services	...	yes	yes	3	2	2	1	1	5	2	yes
4	GP	F	16	U	GT3	T	3	3	other	other	...	no	no	4	3	2	1	2	5	4	yes

5 rows × 31 columns

Fig 1: Sample data points

Further, we removed the last column and called it ‘outcomes’. The rest data is preprocessed as it will be our training data.

First, We removed the null values from the data. So if any of the values is null or empty then the whole row is removed. We also tried replacing null with 0 values but the best accuracy is achieved by having those rows removed from our data.



```
data = data.dropna()
data = data.apply(lambda col: pd.factorize(col, sort=True)[0])
scaler = preprocessing.MinMaxScaler()
scaler.fit(data)
scaled = scaler.transform(data)
train = pd.DataFrame(scaled, columns=data.columns)
train.head()
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	higher	internet	romantic	famrel	freetime	goout	Dalc	Walc	health	absences
0	0.0	0.0	0.428571	1.0	0.0	0.0	1.00	1.00	0.00	1.00	...	1.0	0.0	0.0	0.75	0.50	0.75	0.00	0.00	0.5	0.181818
1	0.0	0.0	0.285714	1.0	0.0	1.0	0.25	0.25	0.00	0.50	...	1.0	1.0	0.0	1.00	0.50	0.50	0.00	0.00	0.5	0.121212
2	0.0	0.0	0.000000	1.0	1.0	1.0	0.25	0.25	0.00	0.50	...	1.0	1.0	0.0	0.75	0.50	0.25	0.25	0.50	0.5	0.303030
3	0.0	0.0	0.000000	1.0	0.0	1.0	1.00	0.50	0.25	0.75	...	1.0	1.0	1.0	0.50	0.25	0.25	0.00	0.00	1.0	0.060606
4	0.0	0.0	0.142857	1.0	0.0	1.0	0.75	0.75	0.50	0.50	...	1.0	0.0	0.0	0.75	0.50	0.25	0.00	0.25	1.0	0.121212

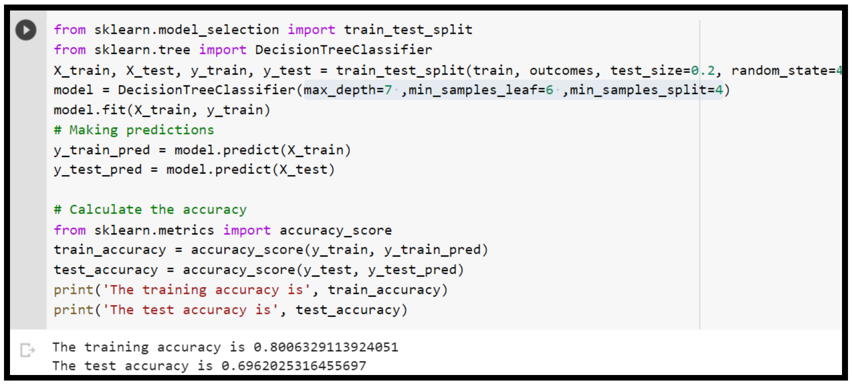
5 rows × 30 columns

Fig 3 - Training and Testing accuracies of DecisionTree Classifier

Secondly, We performed categorical encoding in the whole data where numerical representation is obtained from text data. For example, yes is given 1 and No is represented as 0.
Lastly, We scaled these numerical values as it helps to train faster and avoid local minima in numerous machine learning algorithms.

2) Classification using Decision Trees

First, we used the sklearn python library to train the decision tree classifier. After preprocessing, Our data was perfectly fine to directly use with sklearn. Multiple hyperparameters govern the accuracy of Decision Trees. We chose to play with ‘max_depth’, ‘min_samples_leaf’, and ‘min_samples_split’.
The test accuracy using sklearn came out to be 69.62%.



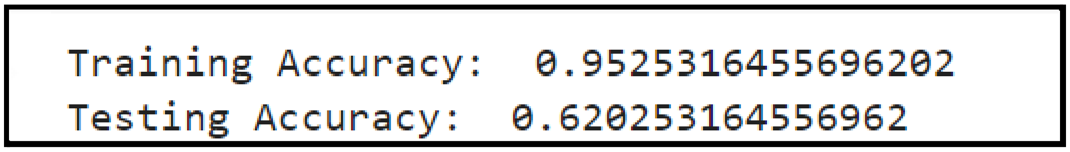
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
X_train, X_test, y_train, y_test = train_test_split(train, outcomes, test_size=0.2, random_state=4
model = DecisionTreeClassifier(max_depth=7 ,min_samples_leaf=6 ,min_samples_split=4)
model.fit(X_train, y_train)
# Making predictions
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# Calculate the accuracy
from sklearn.metrics import accuracy_score
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
print('The training accuracy is', train_accuracy)
print('The test accuracy is', test_accuracy)
```

```
The training accuracy is 0.8006329113924051
The test accuracy is 0.6962025316455697
```

Fig 3 - Training and Testing accuracies of DecisionTree Classifier using SKLearn

Now, we implemented Decision Trees Classifiers from scratch. Code for which is submitted in a separate file.

The accuracy obtained using our code is 62% which is almost equal to that of sklearn. Also, an interesting thing to note here is that testing accuracy for both was way too high than testing accuracy which hints at Overfitting making Decision Trees not the best algorithm for this data and application.

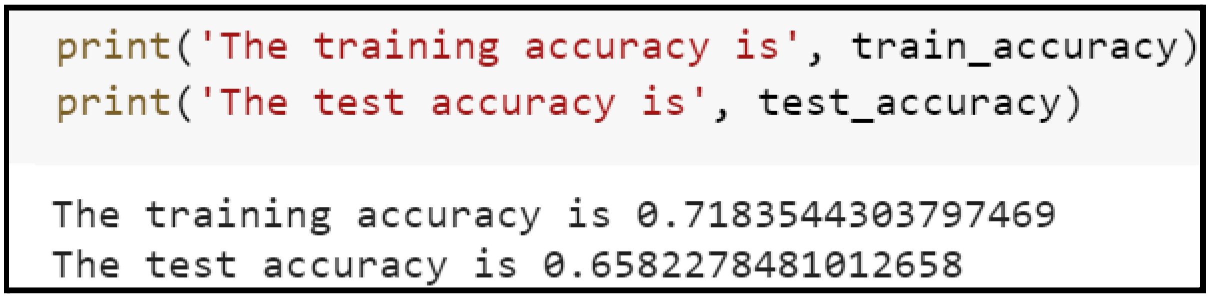


```
Training Accuracy:  0.9525316455696202
Testing Accuracy:  0.620253164556962
```

Fig 4 - Training and Testing accuracies of DecisionTree Classifier without SKLearn

3) Classification using Random Forest Classifiers

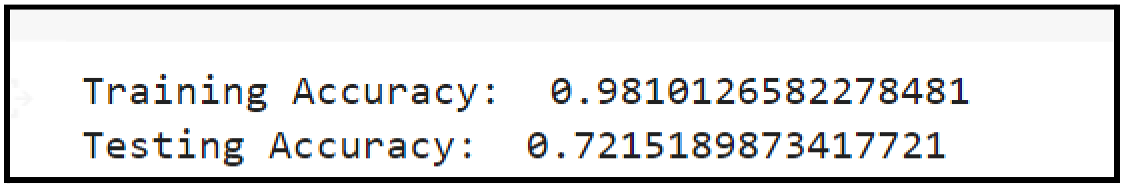
Unlike Decision Trees, Random Forest Classifiers had a little difference in testing and training accuracies. Testing accuracy using sklearn is 65.82% which is less than we got from Decision Trees but at least there is no sign of overfitting!



```
print('The training accuracy is', train_accuracy)
print('The test accuracy is', test_accuracy)

The training accuracy is 0.7183544303797469
The test accuracy is 0.6582278481012658
```

Fig 5 - Training and Testing accuracies of RandomForest Classifier using SKLearn

Unfortunately, after multiple attempts of hyperparameter tuning, we were able to achieve accuracy higher than sklearn but the overfitting appeared again. Our implementation suggested that Random forest though having good testing accuracy may not be the perfect algorithm for the task.

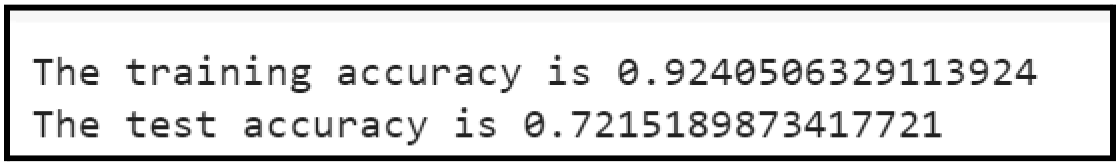


```
Training Accuracy:  0.9810126582278481
Testing Accuracy:  0.7215189873417721
```

Fig 6 - Training and Testing accuracies of RandomForest Classifier not using SKLearn

4) <u>**Classification using Support Vector Machines (SVM)**</u>

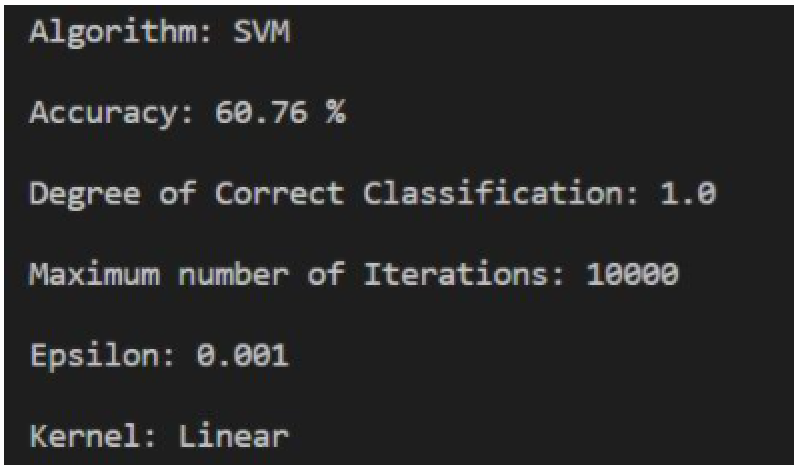
SVM classifiers gave us 65% accuracy with the 'linear' kernel. We implemented the same from scratch and were able to reach 60.76% accuracy. However, when we used the 'Poly' kernel in sklearn, we were able to reach 72.15% accuracy.



```
The training accuracy is 0.9240506329113924
The test accuracy is 0.7215189873417721
```

Fig 7 - Training and Testing accuracies of SVM while SKLearn features were used

With our implementation -

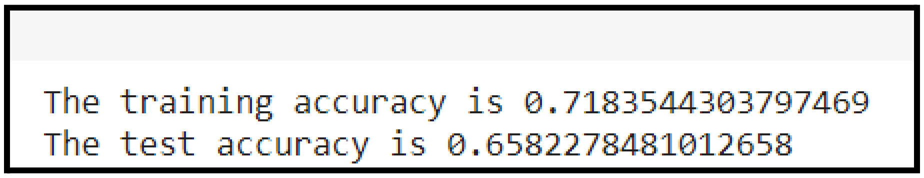


```
Algorithm: SVM

Accuracy: 60.76 %

Degree of Correct Classification: 1.0

Maximum number of Iterations: 10000

Epsilon: 0.001

Kernel: Linear
```

Fig 8 - Training and Testing accuracies of SVM

5) <u>**Naive Bayes Algorithm for Classification**</u>

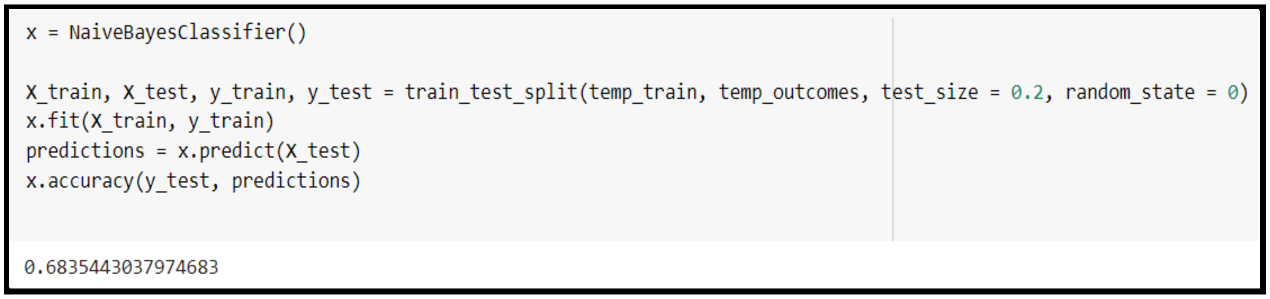
The Naive Bayes algorithm proved to be one of the best methods. It had a training accuracy of 71% and a Testing accuracy of 65.8% which shows no signs of overfitting and 65.8% is decent accuracy as compared to other algorithms.



```
The training accuracy is 0.7183544303797469
The test accuracy is 0.6582278481012658
```

Fig 9 - Training and Testing accuracies of NB at the beginning using SKLearn

When implemented from scratch, we achieved even better than sklearn i.e 68.35% accuracy.

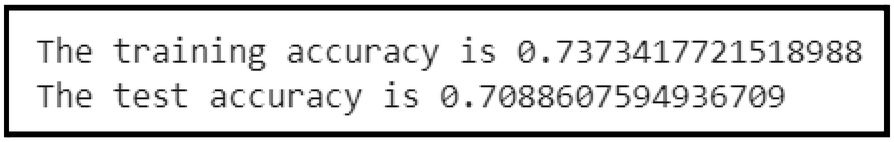


```
x = NaiveBayesClassifier()

X_train, X_test, y_train, y_test = train_test_split(temp_train, temp_outcomes, test_size = 0.2, random_state = 0)
x.fit(X_train, y_train)
predictions = x.predict(X_test)
x.accuracy(y_test, predictions)

0.6835443037974683
```

Fig 10 - Testing accuracies of NB

6) Logistic Regression Classifier

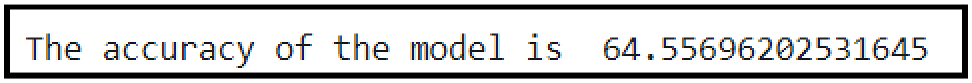
We could see the average accuracy scores for Logistic Regression as well. It performed almost similarly to other designs we followed. Comparing our output to Scikit learn's algorithm, we had a deficit of 6 %. During the usage of the out-of-the-box version of the algorithm, we were able to produce a training accuracy of 73% and a testing accuracy of close to 70 %. We managed to replicate the algorithm and availed a test accuracy of 64.5 %.
Using sklearn -



```
The training accuracy is 0.7373417721518988
The test accuracy is 0.7088607594936709
```

Fig 11 - SKLearn accuracies for Logistic Regression Algorithm

Using code from Scratch -



```
The accuracy of the model is  64.55696202531645
```

Fig 12 - Test accuracy for Logistic Regression when implemented from scratch

Evaluations & Graphs

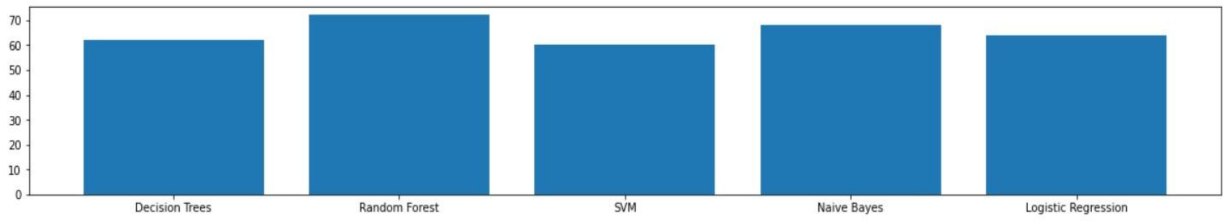


Fig 13 - Accuracy comparison of all ML models

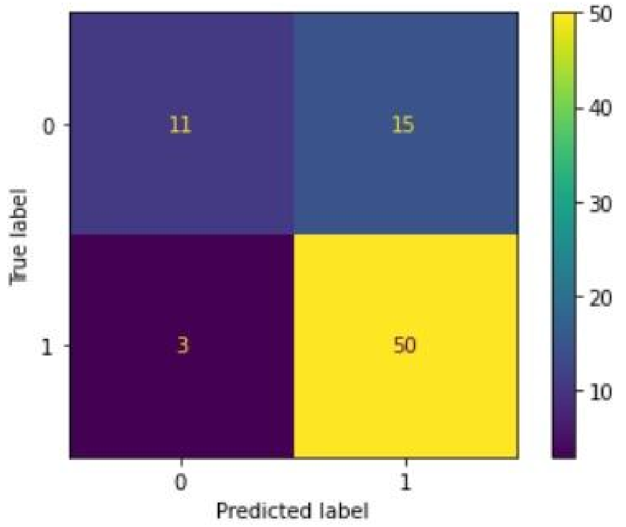


Fig 13 - Confusion matrix for Logistic Regression

Looking at the confusion matrix of the Logistic Regression model, it is evident that the sample TN = 11, TP = 50, FN = 4, FP = 15.

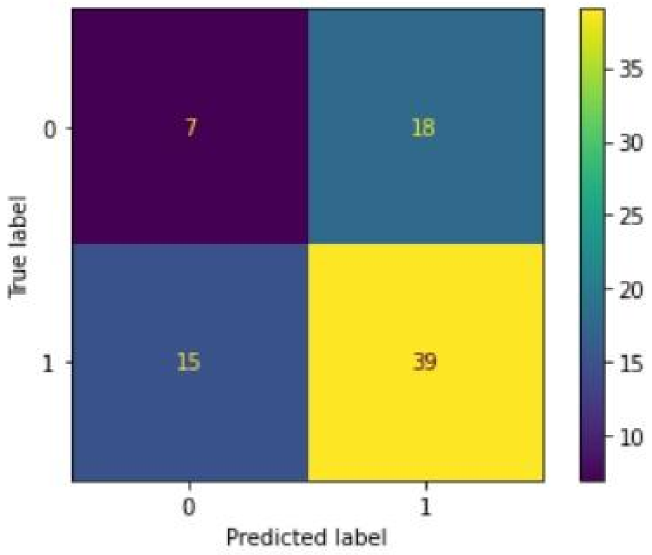


Fig 14 - Confusion matrix for Decision Tree

Looking at the confusion matrix of the Logistic Regression model, it is evident that the sample TN = 7, TP = 39, FN = 15, FP = 18.

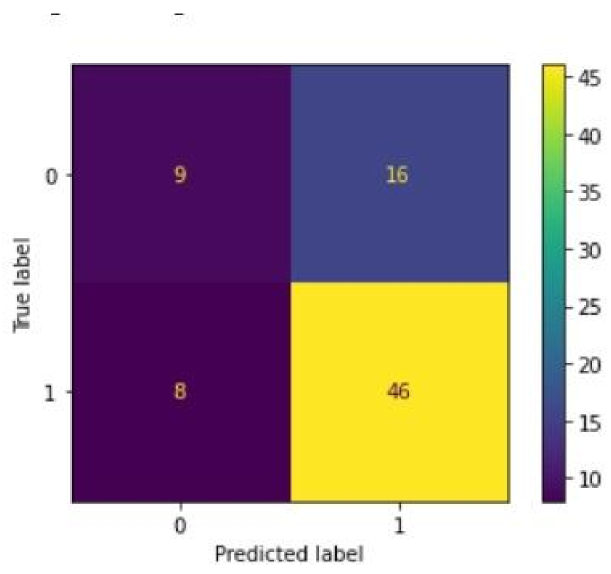


Fig 15 - Confusion matrix for Random Forest

Looking at the confusion matrix of the Logistic Regression model, it is evident that the sample TN = 9, TP = 46, FN = 8, FP = 16.

Conclusion

The performance of different machine learning models was compared to predict whether a student will pass or fail. For each algorithm, an in-built Python library, as well as Python code from scratch, was written and accuracy for both of the implementations is compared.