Student Performance Prediction

June 23, 2020

1 Student Performace in Secondary Education

This is the main project for Week-5 of this course. The dataset is from the UCI Machine Learning Repository (originating from the University of Minho, GuimarÃ£es, Portugal). You can download the dataset from the link above.

2 Objectives:

1. Data set explanation.
2. Data Preparation for Classification and Regression.
3. Defining Classification and Regression models.
4. Models Evaluation.
5. Feature Iportance.
6. Error Calculation.
7. Model Boosting.

2.1 Additional Requirements:

1. pip install xgboost

3 Data Set Information:

This data approaches student achievement in secondary education of two Portuguese schools. The data attributes include student grades, demographic, social and school related features) and it was collected by using school reports and questionnaires. Two datasets are provided regarding the performance in two distinct subjects: Mathematics (mat) and Portuguese language (por). In [Cortez and Silva, 2008] , the two datasets were modeled under binary/five-level classification and regression tasks. "In this notebook, I use one subject containg dataset i.e; Math". Important note: the target attribute G3 has a strong correlation with attributes G2 and G1. This occurs because G3 is the final year grade (issued at the 3rd period), while G1 and G2 correspond to the 1st and 2nd period grades. It is more difficult to predict G3 without G2 and G1, but such prediction is much more useful (see paper source for more details)

[1]:
```python
# Importing Libraries

import numpy as np
import pandas as pd
from collections import defaultdict
import sklearn
import csv
import time
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
import random
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.naive_bayes import MultinomialNB
from xgboost import XGBClassifier
from xgboost import plot_importance
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

The purpose of this dataset is to predict G3 (final grade) using G1, G2(period grades) and other attributes.

Below is the dataset description from the UCI Machine Learning Repository.

4 Attributes for student-mat.csv (Math course) dataset:

1 school - student's school (binary: "GP" - Gabriel Pereira or "MS" - Mousinho da Silveira)

2 sex - student's sex (binary: "F" - female or "M" - male)

3 age - student's age (numeric: from 15 to 22)

4 address - student's home address type (binary: "U" - urban or "R" - rural)

5 famsize - family size (binary: "LE3" - less or equal to 3 or "GT3" - greater than 3)

6 Pstatus - parent’s cohabitation status (binary: ”T” - living together or ”A” - apart)

7 Medu - mother’s education (numeric: 0 - none, 1 - primary education (4th grade), 2 – 5th to 9th grade, 3 – secondary education or 4 – higher education)

8 Fedu - father’s education (numeric: 0 - none, 1 - primary education (4th grade), 2 – 5th to 9th grade, 3 – secondary education or 4 – higher education)

9 Mjob - mother’s job (nominal: ”teacher”, ”health” care related, civil ”services” (e.g. administrative or police), ”at_home” or ”other”)

10 Fjob - father’s job (nominal: ”teacher”, ”health” care related, civil ”services” (e.g. administrative or police), ”at_home” or ”other”)

11 reason - reason to choose this school (nominal: close to ”home”, school ”reputation”, ”course” preference or ”other”)

12 guardian - student’s guardian (nominal: ”mother”, ”father” or ”other”)

13 traveltime - home to school travel time (numeric: 1 - <15 min., 2 - 15 to 30 min., 3 - 30 min. to 1 hour, or 4 - >1 hour)

14 studytime - weekly study time (numeric: 1 - <2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 - >10 hours)

15 failures - number of past class failures (numeric: n if 1<=n<3, else 4)

16 schoolsup - extra educational support (binary: yes or no)

17 famsup - family educational support (binary: yes or no)

18 paid - extra paid classes within the course subject (Math or Portuguese) (binary: yes or no)

19 activities - extra-curricular activities (binary: yes or no)

20 nursery - attended nursery school (binary: yes or no)

21 higher - wants to take higher education (binary: yes or no)

22 internet - Internet access at home (binary: yes or no)

23 romantic - with a romantic relationship (binary: yes or no)

24 famrel - quality of family relationships (numeric: from 1 - very bad to 5 - excellent)

25 freetime - free time after school (numeric: from 1 - very low to 5 - very high)

26 goout - going out with friends (numeric: from 1 - very low to 5 - very high)

27 Dalc - workday alcohol consumption (numeric: from 1 - very low to 5 - very high)

28 Walc - weekend alcohol consumption (numeric: from 1 - very low to 5 - very high)

29 health - current health status (numeric: from 1 - very bad to 5 - very good)

30 absences - number of school absences (numeric: from 0 to 93)

5 These grades are related with the course subject, Math:

31 G1 - first period grade (numeric: from 0 to 20)

31 G2 - second period grade (numeric: from 0 to 20)

32 G3 - final grade (numeric: from 0 to 20, output target)

Run the cells below to examine the dataset.

6 Data Preparation

[2]:
```
# Load student dataset

df = pd.read_csv('student/student-mat.csv', sep = ";")
```

[3]:
```
df.head()
```

[3]:
```
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  ...  \
0     GP   F   18       U     GT3       A     4     4  at_home   teacher  ...
1     GP   F   17       U     GT3       T     1     1  at_home     other  ...
2     GP   F   15       U     LE3       T     1     1  at_home     other  ...
3     GP   F   15       U     GT3       T     4     2   health  services  ...
4     GP   F   16       U     GT3       T     3     3    other     other  ...

  famrel freetime  goout  Dalc  Walc health absences  G1  G2  G3
0      4        3      4     1     1      3        6   5   6   6
1      5        3      3     1     1      3        4   5   5   6
2      4        3      2     2     3      3       10   7   8  10
3      3        2      2     1     1      5        2  15  14  15
4      4        3      2     1     2      5        4   6  10  10

[5 rows x 33 columns]
```

[4]:
```
df_columns = list(df.columns)
df_columns
```

[4]:
```
['school',
 'sex',
 'age',
 'address',
 'famsize',
 'Pstatus',
 'Medu',
 'Fedu',
 'Mjob',
 'Fjob',
```

```
 'reason',
 'guardian',
 'traveltime',
 'studytime',
 'failures',
 'schoolsup',
 'famsup',
 'paid',
 'activities',
 'nursery',
 'higher',
 'internet',
 'romantic',
 'famrel',
 'freetime',
 'goout',
 'Dalc',
 'Walc',
 'health',
 'absences',
 'G1',
 'G2',
 'G3']
```

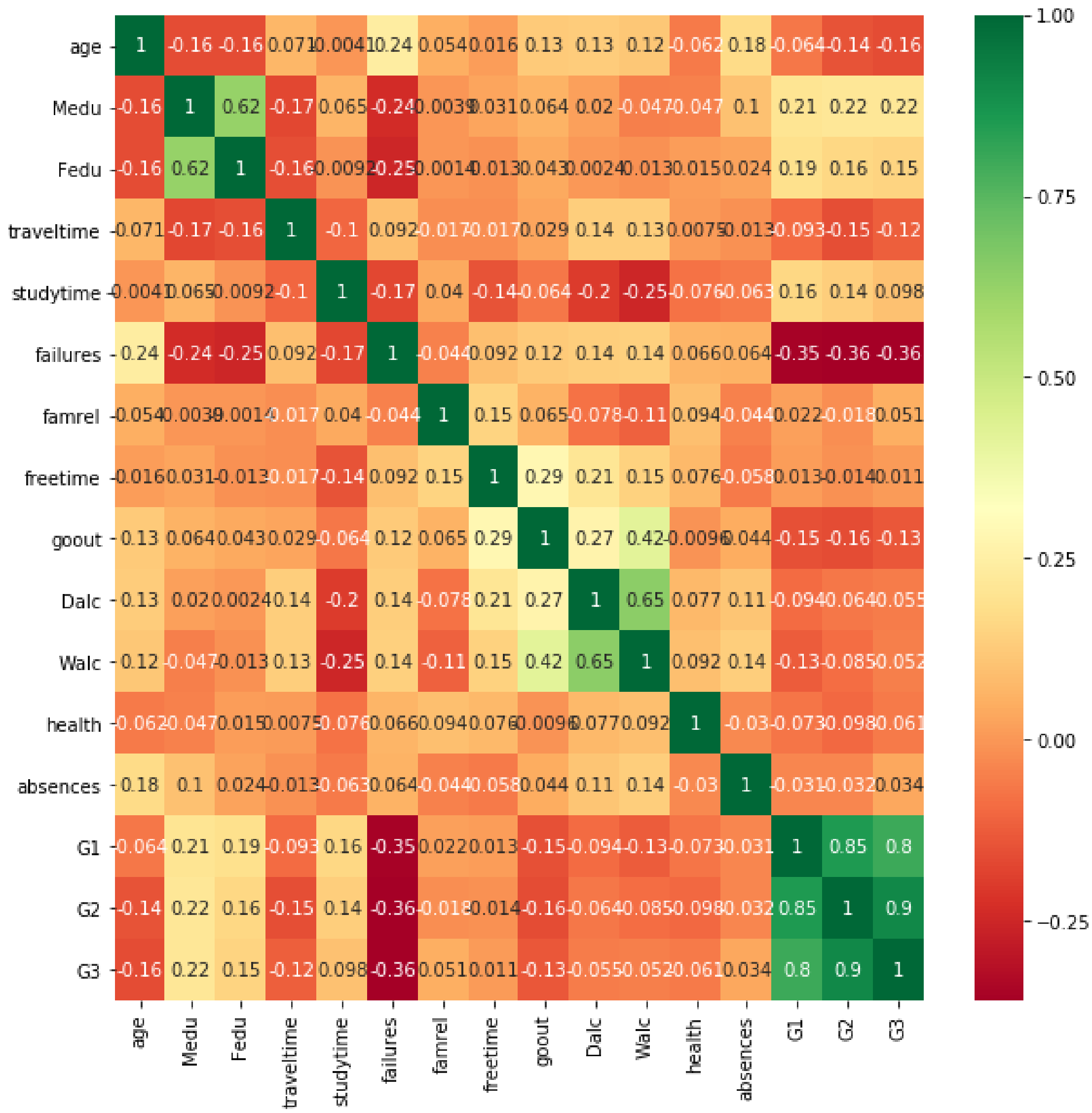
[5]:
```
df.describe()
```

[5]:
```
              age        Medu        Fedu  traveltime   studytime    failures  \
count  395.000000  395.000000  395.000000  395.000000  395.000000  395.000000
mean    16.696203    2.749367    2.521519    1.448101    2.035443    0.334177
std      1.276043    1.094735    1.088201    0.697505    0.839240    0.743651
min     15.000000    0.000000    0.000000    1.000000    1.000000    0.000000
25%     16.000000    2.000000    2.000000    1.000000    1.000000    0.000000
50%     17.000000    3.000000    2.000000    1.000000    2.000000    0.000000
75%     18.000000    4.000000    3.000000    2.000000    2.000000    0.000000
max     22.000000    4.000000    4.000000    4.000000    4.000000    3.000000

           famrel    freetime       goout        Dalc        Walc      health  \
count  395.000000  395.000000  395.000000  395.000000  395.000000  395.000000
mean     3.944304    3.235443    3.108861    1.481013    2.291139    3.554430
std      0.896659    0.998862    1.113278    0.890741    1.287897    1.390303
min      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000
25%      4.000000    3.000000    2.000000    1.000000    1.000000    3.000000
50%      4.000000    3.000000    3.000000    1.000000    2.000000    4.000000
75%      5.000000    4.000000    4.000000    2.000000    3.000000    5.000000
max      5.000000    5.000000    5.000000    5.000000    5.000000    5.000000

         absences          G1          G2          G3
```

```
count  395.000000  395.000000  395.000000  395.000000
mean     5.708861   10.908861   10.713924   10.415190
std      8.003096    3.319195    3.761505    4.581443
min      0.000000    3.000000    0.000000    0.000000
25%      0.000000    8.000000    9.000000    8.000000
50%      4.000000   11.000000   11.000000   11.000000
75%      8.000000   13.000000   13.000000   14.000000
max     75.000000   19.000000   19.000000   20.000000
```

[6]:

```
#get correlations of numerical features in dataset
corrmat = df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(10,10))
#plot heat map
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



[7]:
```
df = df.dropna()
df
```

[7]:
```
    school sex  age address famsize Pstatus  Medu  Fedu      Mjob      Fjob  \
0       GP   F   18       U     GT3       A     4     4   at_home   teacher
1       GP   F   17       U     GT3       T     1     1   at_home     other
2       GP   F   15       U     LE3       T     1     1   at_home     other
3       GP   F   15       U     GT3       T     4     2    health  services
4       GP   F   16       U     GT3       T     3     3     other     other
5       GP   M   16       U     LE3       T     4     3  services     other
6       GP   M   16       U     LE3       T     2     2     other     other
7       GP   F   17       U     GT3       A     4     4     other   teacher
8       GP   M   15       U     LE3       A     3     2  services     other
9       GP   M   15       U     GT3       T     3     4     other     other
10      GP   F   15       U     GT3       T     4     4   teacher    health
11      GP   F   15       U     GT3       T     2     1  services     other
12      GP   M   15       U     LE3       T     4     4    health  services
13      GP   M   15       U     GT3       T     4     3   teacher     other
14      GP   M   15       U     GT3       A     2     2     other     other
15      GP   F   16       U     GT3       T     4     4    health     other
16      GP   F   16       U     GT3       T     4     4  services  services
17      GP   F   16       U     GT3       T     3     3     other     other
18      GP   M   17       U     GT3       T     3     2  services  services
19      GP   M   16       U     LE3       T     4     3    health     other
20      GP   M   15       U     GT3       T     4     3   teacher     other
21      GP   M   15       U     GT3       T     4     4    health    health
22      GP   M   16       U     LE3       T     4     2   teacher     other
23      GP   M   16       U     LE3       T     2     2     other     other
24      GP   F   15       R     GT3       T     2     4  services    health
25      GP   F   16       U     GT3       T     2     2  services  services
26      GP   M   15       U     GT3       T     2     2     other     other
27      GP   M   15       U     GT3       T     4     2    health  services
28      GP   M   16       U     LE3       A     3     4  services     other
29      GP   M   16       U     GT3       T     4     4   teacher   teacher
..     …  ..  …     …     …     …   …   …       …         …
365     MS   M   18       R     GT3       T     1     3   at_home     other
366     MS   M   18       U     LE3       T     4     4   teacher  services
367     MS   F   17       R     GT3       T     1     1     other  services
368     MS   F   18       U     GT3       T     2     3   at_home  services
369     MS   F   18       R     GT3       T     4     4     other   teacher
370     MS   F   19       U     LE3       T     3     2  services  services
371     MS   M   18       R     LE3       T     1     2   at_home  services
372     MS   F   17       U     GT3       T     2     2     other   at_home
373     MS   F   17       R     GT3       T     1     2     other     other
374     MS   F   18       R     LE3       T     4     4     other     other
```

375	MS	F	18	R	GT3	T	1	1	other	other
376	MS	F	20	U	GT3	T	4	2	health	other
377	MS	F	18	R	LE3	T	4	4	teacher	services
378	MS	F	18	U	GT3	T	3	3	other	other
379	MS	F	17	R	GT3	T	3	1	at_home	other
380	MS	M	18	U	GT3	T	4	4	teacher	teacher
381	MS	M	18	R	GT3	T	2	1	other	other
382	MS	M	17	U	GT3	T	2	3	other	services
383	MS	M	19	R	GT3	T	1	1	other	services
384	MS	M	18	R	GT3	T	4	2	other	other
385	MS	F	18	R	GT3	T	2	2	at_home	other
386	MS	F	18	R	GT3	T	4	4	teacher	at_home
387	MS	F	19	R	GT3	T	2	3	services	other
388	MS	F	18	U	LE3	T	3	1	teacher	services
389	MS	F	18	U	GT3	T	1	1	other	other
390	MS	M	20	U	LE3	A	2	2	services	services
391	MS	M	17	U	LE3	T	3	1	services	services
392	MS	M	21	R	GT3	T	1	1	other	other
393	MS	M	18	R	LE3	T	3	2	services	other
394	MS	M	19	U	LE3	T	1	1	other	at_home

	…	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	…	4	3	4	1	1	3	6	5	6	6
1	…	5	3	3	1	1	3	4	5	5	6
2	…	4	3	2	2	3	3	10	7	8	10
3	…	3	2	2	1	1	5	2	15	14	15
4	…	4	3	2	1	2	5	4	6	10	10
5	…	5	4	2	1	2	5	10	15	15	15
6	…	4	4	4	1	1	3	0	12	12	11
7	…	4	1	4	1	1	1	6	6	5	6
8	…	4	2	2	1	1	1	0	16	18	19
9	…	5	5	1	1	1	5	0	14	15	15
10	…	3	3	3	1	2	2	0	10	8	9
11	…	5	2	2	1	1	4	4	10	12	12
12	…	4	3	3	1	3	5	2	14	14	14
13	…	5	4	3	1	2	3	2	10	10	11
14	…	4	5	2	1	1	3	0	14	16	16
15	…	4	4	4	1	2	2	4	14	14	14
16	…	3	2	3	1	2	2	6	13	14	14
17	…	5	3	2	1	1	4	4	8	10	10
18	…	5	5	5	2	4	5	16	6	5	5
19	…	3	1	3	1	3	5	4	8	10	10
20	…	4	4	1	1	1	1	0	13	14	15
21	…	5	4	2	1	1	5	0	12	15	15
22	…	4	5	1	1	3	5	2	15	15	16
23	…	5	4	4	2	4	5	0	13	13	12
24	…	4	3	2	1	1	5	2	10	9	8

```
25   …    1     2    2   1   3    5     14   6   9   8
26   …    4     2    2   1   2    5      2  12  12  11
27   …    2     2    4   2   4    1      4  15  16  15
28   …    5     3    3   1   1    5      4  11  11  11
29   …    4     4    5   5   5    5     16  10  12  11
..   …   …     …    …   …   …    …     …  ..  ..  ..
365  …    3     3    4   2   4    3      4  10  10  10
366  …    4     2    2   2   2    5      0  13  13  13
367  …    5     2    1   1   2    1      0   7   6   0
368  …    5     2    3   1   2    4      0  11  10  10
369  …    3     2    2   4   2    5     10  14  12  11
370  …    3     2    2   1   1    3      4   7   7   9
371  …    4     3    3   2   3    3      3  14  12  12
372  …    3     4    3   1   1    3      8  13  11  11
373  …    3     5    5   1   3    1     14   6   5   5
374  …    5     4    4   1   1    1      0  19  18  19
375  …    4     3    2   1   2    4      2   8   8  10
376  …    5     4    3   1   1    3      4  15  14  15
377  …    5     4    3   3   4    2      4   8   9  10
378  …    4     1    3   1   2    1      0  15  15  15
379  …    4     5    4   2   3    1     17  10  10  10
380  …    3     2    4   1   4    2      4  15  14  14
381  …    4     4    3   1   3    5      5   7   6   7
382  …    4     4    3   1   1    3      2  11  11  10
383  …    4     3    2   1   3    5      0   6   5   0
384  …    5     4    3   4   3    3     14   6   5   5
385  …    5     3    3   1   3    4      2  10   9  10
386  …    4     4    3   2   2    5      7   6   5   6
387  …    5     4    2   1   2    5      0   7   5   0
388  …    4     3    4   1   1    1      0   7   9   8
389  …    1     1    1   1   1    5      0   6   5   0
390  …    5     5    4   4   5    4     11   9   9   9
391  …    2     4    5   3   4    2      3  14  16  16
392  …    5     5    3   3   3    3      3  10   8   7
393  …    4     4    1   3   4    5      0  11  12  10
394  …    3     2    3   3   3    5      5   8   9   9

[395 rows x 33 columns]
```

[8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 395 entries, 0 to 394
Data columns (total 33 columns):
school        395 non-null object
sex           395 non-null object
age           395 non-null int64
```

```
address       395 non-null object
famsize       395 non-null object
Pstatus       395 non-null object
Medu          395 non-null int64
Fedu          395 non-null int64
Mjob          395 non-null object
Fjob          395 non-null object
reason        395 non-null object
guardian      395 non-null object
traveltime    395 non-null int64
studytime     395 non-null int64
failures      395 non-null int64
schoolsup     395 non-null object
famsup        395 non-null object
paid          395 non-null object
activities    395 non-null object
nursery       395 non-null object
higher        395 non-null object
internet      395 non-null object
romantic      395 non-null object
famrel        395 non-null int64
freetime      395 non-null int64
goout         395 non-null int64
Dalc          395 non-null int64
Walc          395 non-null int64
health        395 non-null int64
absences      395 non-null int64
G1            395 non-null int64
G2            395 non-null int64
G3            395 non-null int64
dtypes: int64(16), object(17)
memory usage: 104.9+ KB
```

[9]:
```
df.head()
```

[9]:
```
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  …  \
0     GP   F   18       U     GT3       A     4     4  at_home   teacher  …
1     GP   F   17       U     GT3       T     1     1  at_home     other  …
2     GP   F   15       U     LE3       T     1     1  at_home     other  …
3     GP   F   15       U     GT3       T     4     2   health  services  …
4     GP   F   16       U     GT3       T     3     3    other     other  …

  famrel freetime  goout  Dalc  Walc health absences  G1  G2  G3
0      4        3      4     1     1      3        6   5   6   6
1      5        3      3     1     1      3        4   5   5   6
2      4        3      2     2     3      3       10   7   8  10
3      3        2      2     1     1      5        2  15  14  15
```

```
4      4        3      2     1     2      5        4   6  10  10

[5 rows x 33 columns]
```

7 Data Preparation for Classification

7.1 Feature Selection

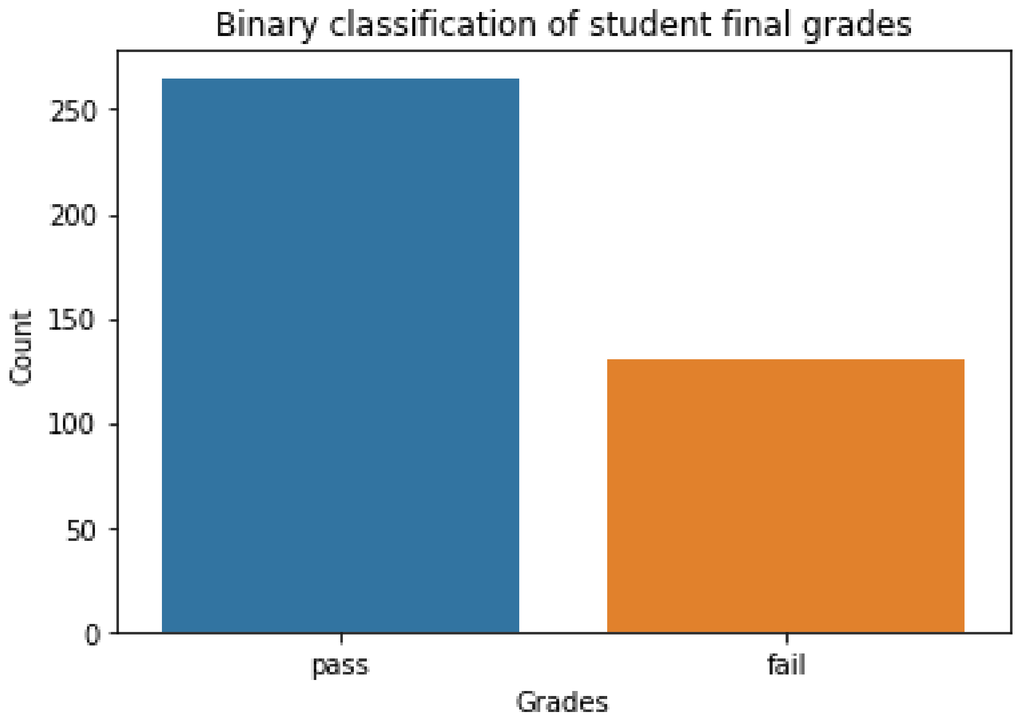
Feature Selection for binary classification – pass if G3 10, else fail.

[10]:
```
df['bin_G3'] = 'na'
df.loc[(df.G3 >= 10), 'bin_G3'] = 'pass'
df.loc[(df.G3 < 10), 'bin_G3'] = 'fail'
df.head(5)
```

[10]:
```
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  …  \
0     GP   F   18       U     GT3       A     4     4  at_home   teacher  …
1     GP   F   17       U     GT3       T     1     1  at_home     other  …
2     GP   F   15       U     LE3       T     1     1  at_home     other  …
3     GP   F   15       U     GT3       T     4     2   health  services  …
4     GP   F   16       U     GT3       T     3     3    other     other  …

  freetime goout  Dalc  Walc  health absences  G1  G2  G3 bin_G3
0        3     4     1     1       3        6   5   6   6   fail
1        3     3     1     1       3        4   5   5   6   fail
2        3     2     2     3       3       10   7   8  10   pass
3        2     2     1     1       5        2  15  14  15   pass
4        3     2     1     2       5        4   6  10  10   pass

[5 rows x 34 columns]
```

[11]:
```
f, ax = plt.subplots()
figure = sns.countplot(x = 'bin_G3', data=df, order=['pass','fail'])
ax = ax.set(ylabel="Count", xlabel = "Grades")
figure.grid(False)
plt.title('Binary classification of student final grades')
```

[11]:
```
Text(0.5, 1.0, 'Binary classification of student final grades')
```



As we can see student pass rate is higher but no.of failing students is also substantially high reaching nearly 150.

[12]:
```
# Encoding new feature as labels for this classification .
le = preprocessing.LabelEncoder()
df.bin_G3 = le.fit_transform(df.bin_G3)
```

[13]:
```
X = df.drop(labels = ['bin_G3','G3'],axis=1)
y = df.bin_G3
```

[14]:
```
# Train test splitting in 70-30 ratio
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3)
```

[15]:
```
# get dummy varibles
X_train = pd.get_dummies(X_train)
X_test = pd.get_dummies(X_test)
```

[16]:
```
X_train.shape
```

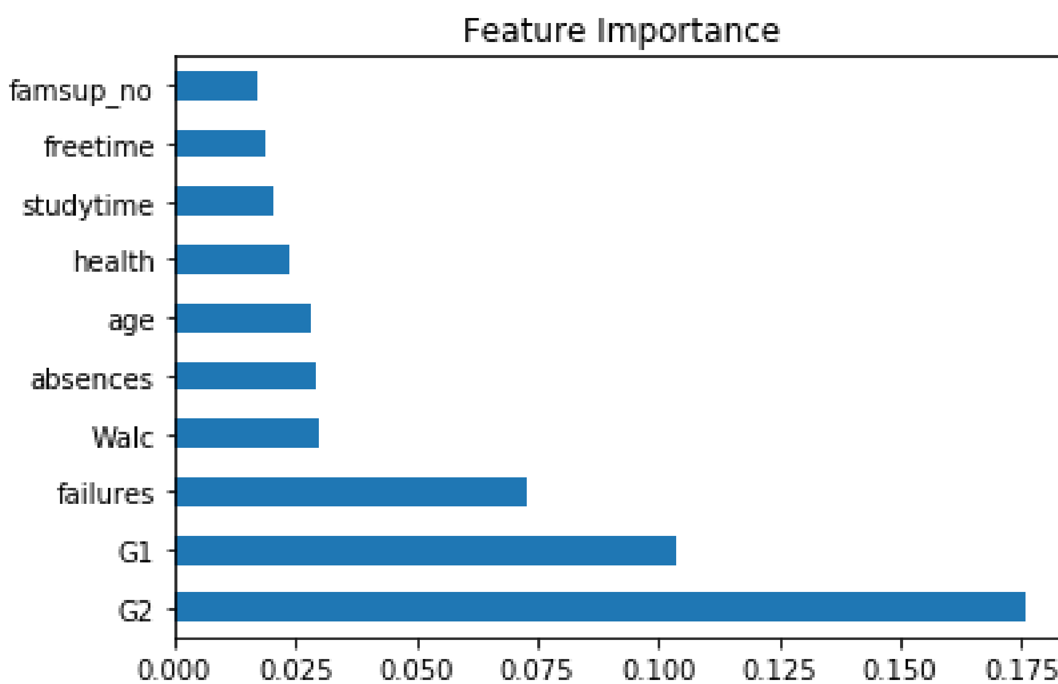
[16]:
```
(276, 58)
```

7.2 Feature Importance

In this project, I will try Feature importance using two models: Extra tree and XGboost classifier. Extra tree is an inbuilt class that comes with Tree Based Classifiers. Both have been used here to extract top features.

[17]:
```
def extclass(x,y):
    model = ExtraTreesClassifier()
    model.fit(x,y)

    feat_importances = pd.Series(model.feature_importances_, index=x.columns)
    feat_importances.nlargest(10).plot(kind='barh')
    plt.title('Feature Importance')
    plt.show()

    return model

etcmodel = extclass(X_train,y_train)
```

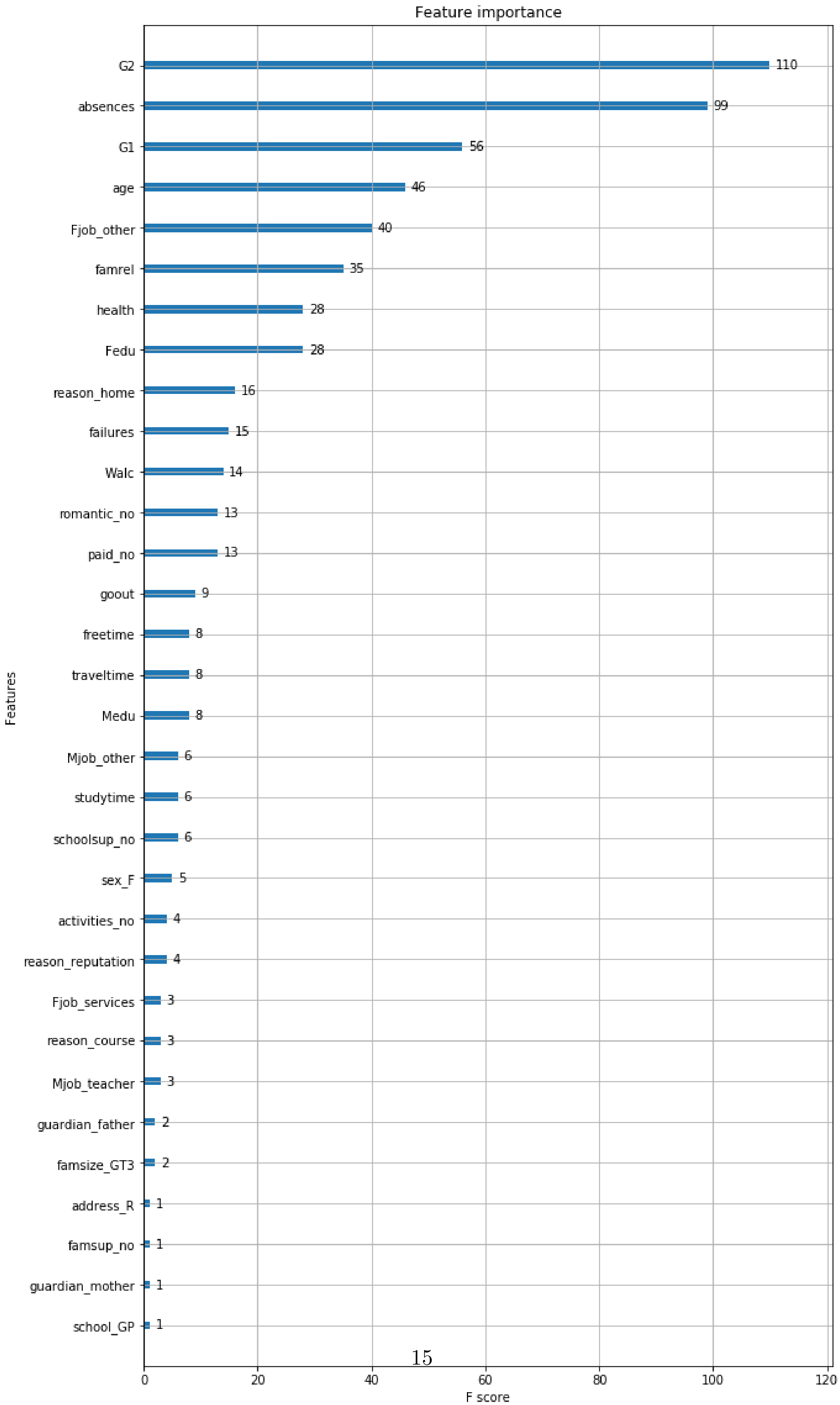
```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in version
0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```



From the graph above we can see that G1 and G2 grades are the most significant featueres and other than academics(grades, absences, study time) domestic features such as parents education and student roaming features affect his final grade 'G3'.

[18]:
```
def xgbclass(x_train,y_train,x_val,y_val):
    model = XGBClassifier()
    model = XGBClassifier(learning_rate=0.1,n_estimators=100)
    mf = model.fit(x_train,y_train)
#     y_pred=model.predict(x_val)
#     y_predict = mf.predict_proba(x_val)
    print("XGBoost Train data Score" , ":" , mf.score(x_train, y_train)
          , "," ,"Validation data Score" ,":" , mf.score(x_val, y_val))


    # plot feature importance
    fig, ax = plt.subplots(figsize=(10, 20))
    plot_importance(mf, ax=ax)

    plt.show()
    return model

xgmodel = xgbclass(X_train,y_train,X_test,y_test)
```

```
XGBoost Train data Score : 1.0 , Validation data Score : 0.8991596638655462
```



Feature importance
G2
110
absences
99
G1
56
age
46
Fjob_other
40
famrel
35
health
28
Fedu
28
reason_home
16
failures
15
Walc
14
romantic_no
13
paid_no
13
goout
9
freetime
8
traveltime
8
Medu
8
Mjob_other
6
studytime
6
schoolsup_no
6
sex_F
5
activities_no
4
reason_reputation
4
Fjob_services
3
reason_course
3
Mjob_teacher
3
guardian_father
2
famsize_GT3
2
address_R
1
famsup_no
1
guardian_mother
1
school_GP
1
Features
0
20
40
60
80
100
120
F score

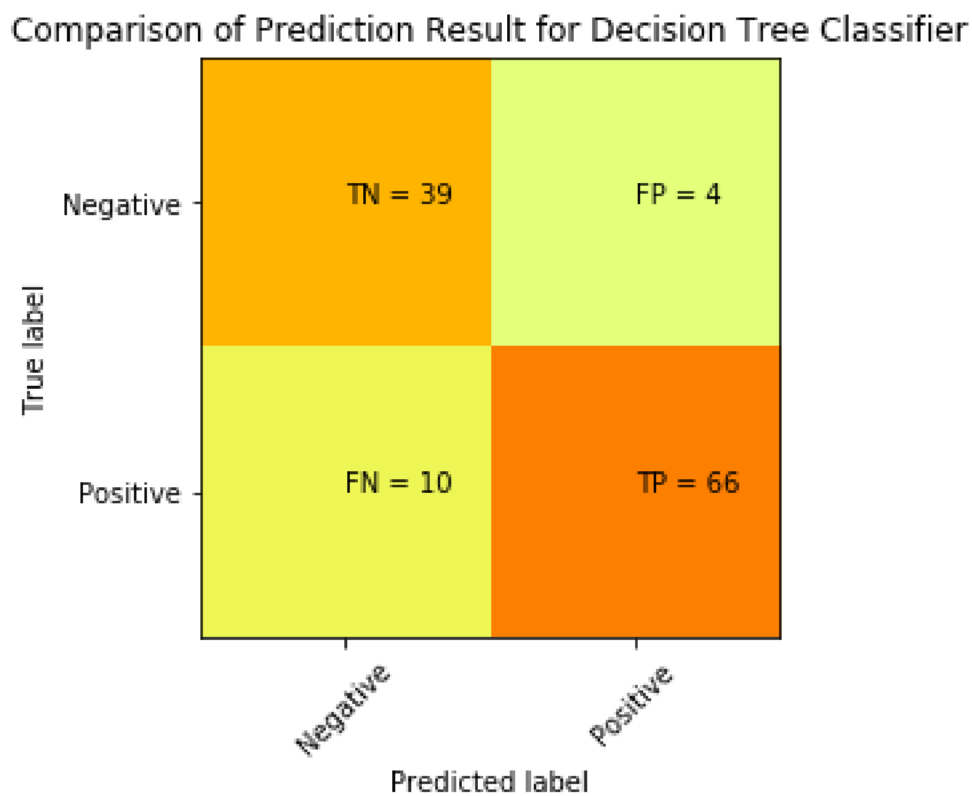
From this graph we can see the impact features on the validation/testing set. Here G2,G1 are the most affecting features, then some of the domestic features such as Father education,health,relation affect his final grade 'G3'.

7.3 Model Evaluation (Classification)

[19]:
```python
def modelscores(model,Model):
    classes = ['pass','fail']
    y_pred = model.predict(X_test)

#     Confusion Matrix
    cm = confusion_matrix(y_test,y_pred)
    accuracy = round(100*np.trace(cm)/np.sum(cm),1)


    #Plot/Display the results
    cm_plot(cm,Model)
    print('Accuracy of the Model' ,Model, str(accuracy)+'%')

    print('\n',classification_report(y_test,y_pred,target_names=classes))

    print("\nModel Training Score" , ":" , model.score(X_train, y_train) , "," ,
      "Cross Validation Score" ,":" , model.score(X_test, y_test))
#     print(y_pred)
    df['finalG3'] = pd.DataFrame(y_pred)
    df_out = pd.merge(df,df[['finalG3']],how = 'left',left_index = True,␣
 ↪right_index = True)
```

[20]:
```python
#Function to plot Confusion Matrix
def cm_plot(cm,Model):
    plt.clf()
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
    classNames = ['Negative','Positive']
    plt.title('Comparison of Prediction Result for '+ Model)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    tick_marks = np.arange(len(classNames))
    plt.xticks(tick_marks, classNames, rotation=45)
    plt.yticks(tick_marks, classNames)
    s = [['TN','FP'], ['FN', 'TP']]
    for i in range(2):
        for j in range(2):
            plt.text(j,i, str(s[i][j])+" = "+str(cm[i][j]))
    plt.show()
```

[21]:
```python
# Classification using Decision tree.
Model = "Decision Tree Classifier"

tree = DecisionTreeClassifier(min_samples_leaf=17)
dtcmodel= tree.fit(X_train, y_train)

y_pred = dtcmodel.predict(X_test)
# for i in range(len(X_test)):
#     print("X=%s, Predicted=%s" % (X_test[i], y_pred[i]))

modelscores(dtcmodel,Model)
df.head()
```




```
Accuracy of the Model Decision Tree Classifier 88.2%

              precision    recall  f1-score   support

        pass       0.80      0.91      0.85        43
        fail       0.94      0.87      0.90        76

    accuracy                           0.88       119
   macro avg       0.87      0.89      0.88       119
```

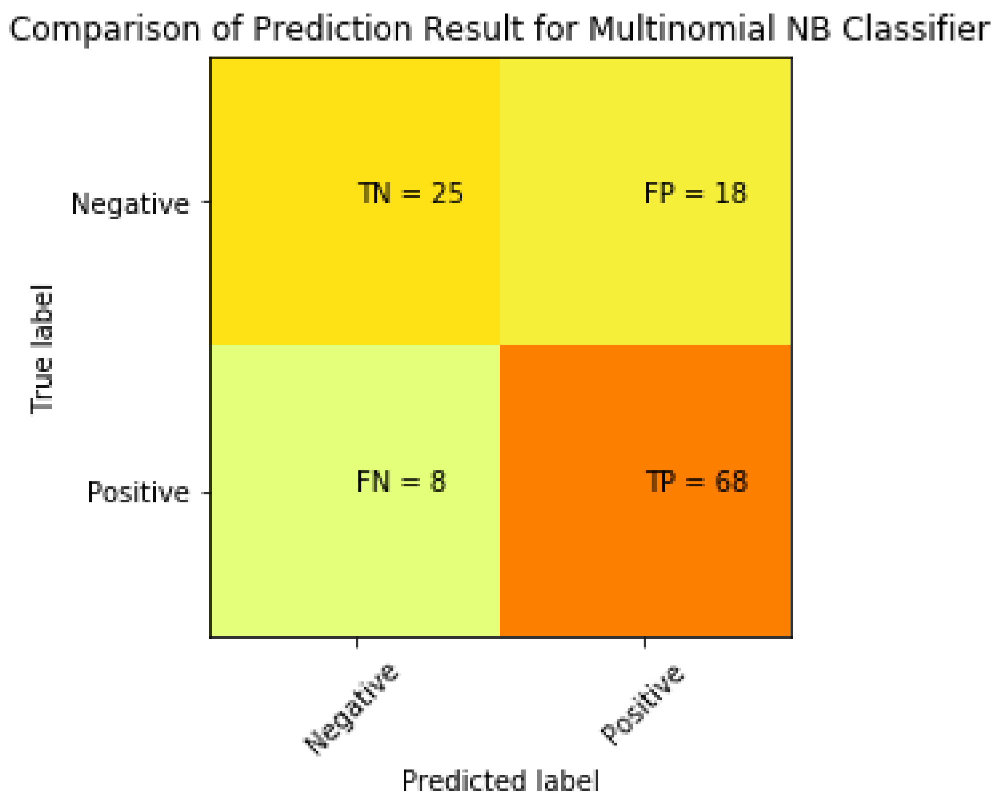
```
weighted avg       0.89      0.88      0.88       119


Model Training Score : 0.9347826086956522 , Cross Validation Score :
0.8823529411764706
```

[21]:

```
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  …  \
0     GP   F   18       U     GT3       A     4     4  at_home   teacher  …
1     GP   F   17       U     GT3       T     1     1  at_home     other  …
2     GP   F   15       U     LE3       T     1     1  at_home     other  …
3     GP   F   15       U     GT3       T     4     2   health  services  …
4     GP   F   16       U     GT3       T     3     3    other     other  …

  goout Dalc  Walc  health  absences  G1  G2  G3 bin_G3 finalG3
0     4    1     1       3         6   5   6   6      0     0.0
1     3    1     1       3         4   5   5   6      0     0.0
2     2    2     3       3        10   7   8  10      1     1.0
3     2    1     1       5         2  15  14  15      1     1.0
4     2    1     2       5         4   6  10  10      1     0.0

[5 rows x 35 columns]
```

[22]:

```
# Classification using Naive Bayes for multinomial models
Model = "Multinomial NB Classifier"

clf = MultinomialNB()
nbmodel= clf.fit(X_train, y_train)

modelscores(nbmodel,Model)
df.head()
```



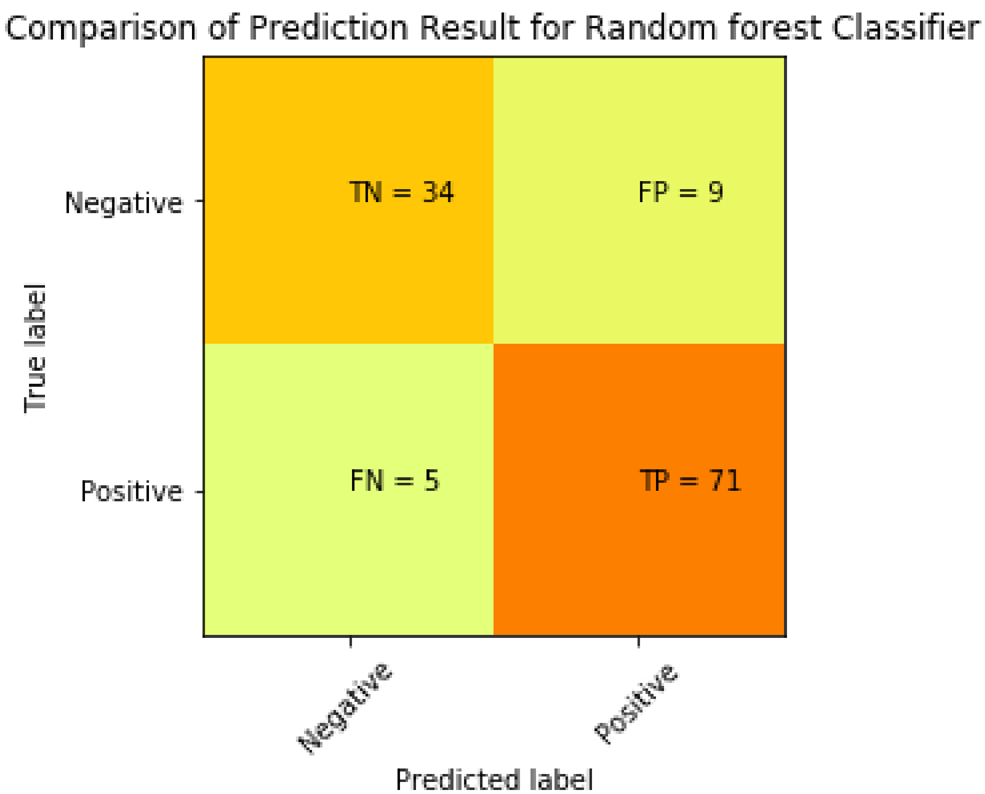

```
Accuracy of the Model Multinomial NB Classifier 78.2%

              precision    recall  f1-score   support

        pass       0.76      0.58      0.66        43
        fail       0.79      0.89      0.84        76

    accuracy                           0.78       119
   macro avg       0.77      0.74      0.75       119
weighted avg       0.78      0.78      0.77       119


Model Training Score : 0.8695652173913043 , Cross Validation Score :
0.7815126050420168
```

[22]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	…	\
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	…	
1	GP	F	17	U	GT3	T	1	1	at_home	other	…	
2	GP	F	15	U	LE3	T	1	1	at_home	other	…	
3	GP	F	15	U	GT3	T	4	2	health	services	…	
4	GP	F	16	U	GT3	T	3	3	other	other	…	

```
  goout Dalc  Walc  health  absences  G1  G2  G3 bin_G3 finalG3
0     4    1     1       3         6   5   6   6      0     0.0
1     3    1     1       3         4   5   5   6      0     0.0
2     2    2     3       3        10   7   8  10      1     1.0
3     2    1     1       5         2  15  14  15      1     1.0
4     2    1     2       5         4   6  10  10      1     0.0

[5 rows x 35 columns]
```

[23]:
```python
# Classification using Random forest classifier
Model = "Random forest Classifier"

clf = RandomForestClassifier(n_estimators=34, min_samples_leaf=2)
rfmodel= clf.fit(X_train, y_train)

modelscores(rfmodel,Model)
df.head()
```




```
Accuracy of the Model Random forest Classifier 88.2%

              precision    recall  f1-score   support
```

```
        pass       0.87      0.79      0.83        43
        fail       0.89      0.93      0.91        76

    accuracy                           0.88       119
   macro avg       0.88      0.86      0.87       119
weighted avg       0.88      0.88      0.88       119


Model Training Score : 0.9891304347826086 , Cross Validation Score :
0.8823529411764706
```

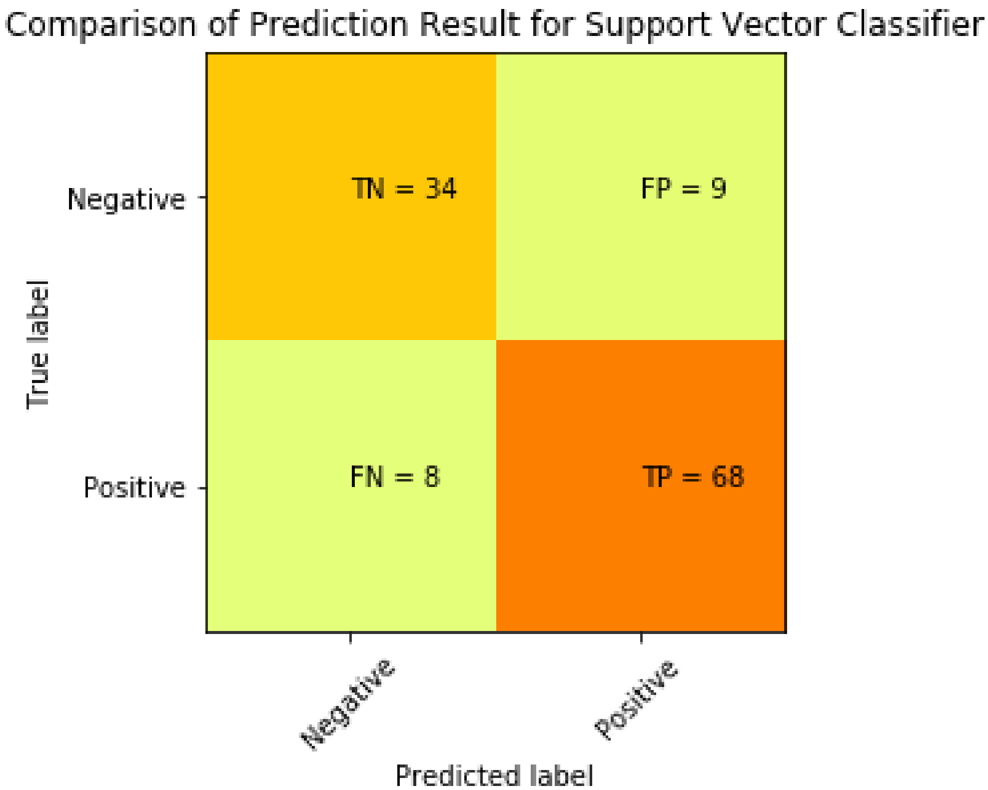
[23]:

```
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  ...  \
0     GP   F   18       U     GT3       A     4     4  at_home   teacher  ...
1     GP   F   17       U     GT3       T     1     1  at_home     other  ...
2     GP   F   15       U     LE3       T     1     1  at_home     other  ...
3     GP   F   15       U     GT3       T     4     2   health  services  ...
4     GP   F   16       U     GT3       T     3     3    other     other  ...

  goout Dalc  Walc  health  absences  G1  G2  G3 bin_G3 finalG3
0     4    1     1       3         6   5   6   6      0     0.0
1     3    1     1       3         4   5   5   6      0     0.0
2     2    2     3       3        10   7   8  10      1     1.0
3     2    1     1       5         2  15  14  15      1     1.0
4     2    1     2       5         4   6  10  10      1     1.0

[5 rows x 35 columns]
```

[24]:

```python
# Classification using SVC
Model = "Support Vector Classifier"

svc = SVC()
svcmodel= svc.fit(X_train, y_train)

modelscores(svcmodel,Model)
df.head()
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/svm/base.py:193: FutureWarning:
The default value of gamma will change from 'auto' to 'scale' in version 0.22 to
account better for unscaled features. Set gamma explicitly to 'auto' or 'scale'
to avoid this warning.
  "avoid this warning.", FutureWarning)
```




```
Accuracy of the Model Support Vector Classifier 85.7%

              precision    recall  f1-score   support

        pass       0.81      0.79      0.80        43
        fail       0.88      0.89      0.89        76

    accuracy                           0.86       119
   macro avg       0.85      0.84      0.84       119
weighted avg       0.86      0.86      0.86       119


Model Training Score : 0.9746376811594203 , Cross Validation Score :
0.8571428571428571
```

[24]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	…	\
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	…	
1	GP	F	17	U	GT3	T	1	1	at_home	other	…	
2	GP	F	15	U	LE3	T	1	1	at_home	other	…	
3	GP	F	15	U	GT3	T	4	2	health	services	…	
4	GP	F	16	U	GT3	T	3	3	other	other	…	

```
  goout Dalc  Walc  health  absences  G1  G2  G3 bin_G3 finalG3
0      4    1     1       3         6   5   6   6      0     0.0
1      3    1     1       3         4   5   5   6      0     1.0
2      2    2     3       3        10   7   8  10      1     1.0
3      2    1     1       5         2  15  14  15      1     1.0
4      2    1     2       5         4   6  10  10      1     0.0

[5 rows x 35 columns]
```

8 Data Preparation for Regression

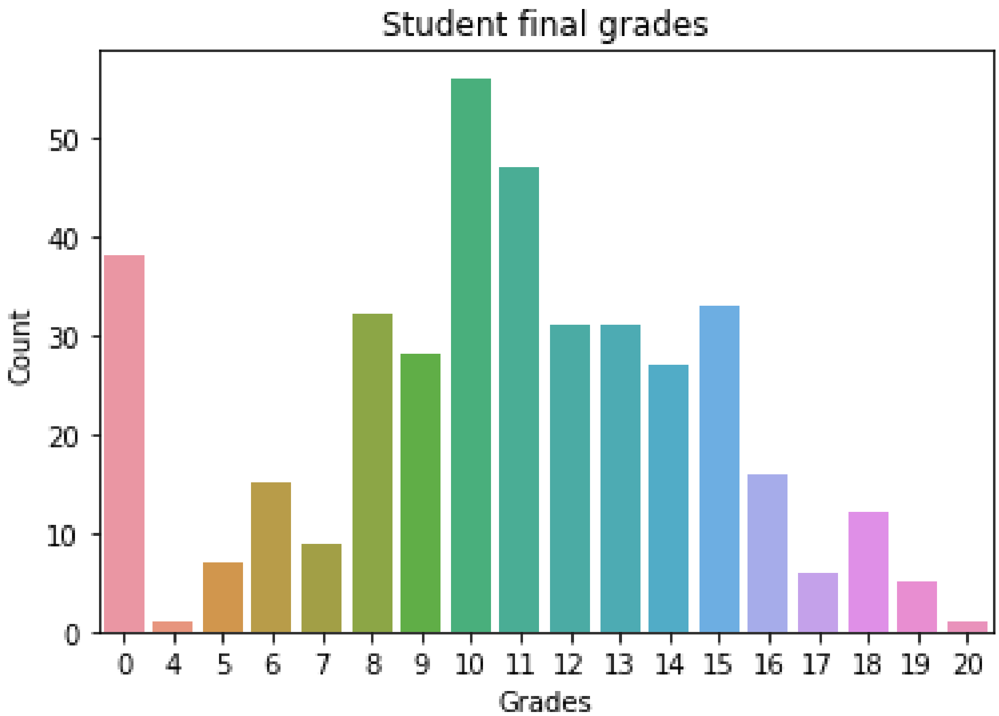
Since Data has to be continuous in Regression, I used 'G3' feature as labels to predict its results.

[25]:
```
df = df.drop(['bin_G3','finalG3'], axis = 1)
# df = df.drop(['bin_G3'], axis = 1)
df.head()
```

[25]:
```
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  …  \
0     GP   F   18       U     GT3       A     4     4  at_home   teacher  …
1     GP   F   17       U     GT3       T     1     1  at_home     other  …
2     GP   F   15       U     LE3       T     1     1  at_home     other  …
3     GP   F   15       U     GT3       T     4     2   health  services  …
4     GP   F   16       U     GT3       T     3     3    other     other  …

  famrel freetime  goout  Dalc  Walc health absences  G1  G2  G3
0      4        3      4     1     1      3        6   5   6   6
1      5        3      3     1     1      3        4   5   5   6
2      4        3      2     2     3      3       10   7   8  10
3      3        2      2     1     1      5        2  15  14  15
4      4        3      2     1     2      5        4   6  10  10

[5 rows x 33 columns]
```

[26]:
```
f, ax = plt.subplots()
figure = sns.countplot(x = 'G3', data=df)
ax = ax.set(ylabel="Count", xlabel = "Grades")
figure.grid(False)
plt.title('Student final grades')
```

[26]: Text(0.5, 1.0, 'Student final grades')



8.1 Encoding Categorical variables

[27]:
```
objlist = df.select_dtypes(include='object').columns
objlist
```

[27]:
```
Index(['school', 'sex', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob',
       'reason', 'guardian', 'schoolsup', 'famsup', 'paid', 'activities',
       'nursery', 'higher', 'internet', 'romantic'],
      dtype='object')
```

[28]:
```
le = preprocessing.LabelEncoder()
df['school']=pd.DataFrame(le.fit_transform(df.school))
df['sex']=pd.DataFrame(le.fit_transform(df.sex))
df['address']=pd.DataFrame(le.fit_transform(df.address))
df['famsize']=pd.DataFrame(le.fit_transform(df.famsize))
df['Pstatus']=pd.DataFrame(le.fit_transform(df.Pstatus))
df['Mjob']=pd.DataFrame(le.fit_transform(df.Mjob))
df['Fjob']=pd.DataFrame(le.fit_transform(df.Fjob))
df['reason']=pd.DataFrame(le.fit_transform(df.reason))
df['guardian']=pd.DataFrame(le.fit_transform(df.guardian))
df['schoolsup']=pd.DataFrame(le.fit_transform(df.schoolsup))
df['famsup']=pd.DataFrame(le.fit_transform(df.famsup))
```

```
df['paid']=pd.DataFrame(le.fit_transform(df.paid))
df['activities']=pd.DataFrame(le.fit_transform(df.activities))
df['nursery']=pd.DataFrame(le.fit_transform(df.nursery))
df['higher']=pd.DataFrame(le.fit_transform(df.higher))
df['internet']=pd.DataFrame(le.fit_transform(df.internet))
df['romantic']=pd.DataFrame(le.fit_transform(df.romantic))
df.head()
```

[28]:

```
   school  sex  age  address  famsize  Pstatus  Medu  Fedu  Mjob  Fjob  …  \
0       0    0   18        1        0        0     4     4     0     4  …
1       0    0   17        1        0        1     1     1     0     2  …
2       0    0   15        1        1        1     1     1     0     2  …
3       0    0   15        1        0        1     4     2     1     3  …
4       0    0   16        1        0        1     3     3     2     2  …

   famrel  freetime  goout  Dalc  Walc  health  absences  G1  G2  G3
0       4         3      4     1     1       3         6   5   6   6
1       5         3      3     1     1       3         4   5   5   6
2       4         3      2     2     3       3        10   7   8  10
3       3         2      2     1     1       5         2  15  14  15
4       4         3      2     1     2       5         4   6  10  10

[5 rows x 33 columns]
```

[29]:

```
# Standardization
scaler=StandardScaler()
scaler.fit(df.drop('G3',axis=1))
scaled_features=scaler.transform(df.drop('G3',axis=1))
dataset_scaled=pd.DataFrame(scaled_features,columns=df.columns[:-1])
dataset_scaled.head()
```

[29]:

```
     school       sex       age   address   famsize   Pstatus      Medu  \
0 -0.36305 -0.948176  1.023046  0.535392 -0.636941 -2.938392  1.143856
1 -0.36305 -0.948176  0.238380  0.535392 -0.636941  0.340322 -1.600009
2 -0.36305 -0.948176 -1.330954  0.535392  1.570004  0.340322 -1.600009
3 -0.36305 -0.948176 -1.330954  0.535392 -0.636941  0.340322  1.143856
4 -0.36305 -0.948176 -0.546287  0.535392 -0.636941  0.340322  0.229234

       Fedu      Mjob      Fjob  …  romantic    famrel  freetime     goout  \
0  1.360371 -1.769793  1.993149  … -0.708450  0.062194 -0.236010  0.801479
1 -1.399970 -1.769793 -0.325831  … -0.708450  1.178860 -0.236010 -0.097908
2 -1.399970 -1.769793 -0.325831  … -0.708450  0.062194 -0.236010 -0.997295
3 -0.479857 -0.954077  0.833659  …  1.411533 -1.054472 -1.238419 -0.997295
4  0.440257 -0.138362 -0.325831  … -0.708450  0.062194 -0.236010 -0.997295

       Dalc      Walc    health  absences        G1        G2
0 -0.540699 -1.003789 -0.399289  0.036424 -1.782467 -1.254791
```

```
1 -0.540699 -1.003789 -0.399289 -0.213796 -1.782467 -1.520979
2  0.583385  0.551100 -0.399289  0.536865 -1.179147 -0.722415
3 -0.540699 -1.003789  1.041070 -0.464016  1.234133  0.874715
4 -0.540699 -0.226345  1.041070 -0.213796 -1.480807 -0.190038

[5 rows x 32 columns]
```

[30]:
```
# Splitting dataset into train and test sets

X = df.drop(labels = ['G3'],axis=1)     #independent columns
y=df['G3']      #target column i.e final grade

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.
↪3,random_state=42)

# X_train = pd.get_dummies(X_train)
# X_test = pd.get_dummies(X_test)
print('Training Features Shape:', X_train.shape)
print('Training Labels Shape:', y_train.shape)
print('Testing Features Shape:', X_test.shape)
print('Testing Labels Shape:', y_test.shape)
# np.isnan(y_train.values.any())
# X_train.values[:,1]
```

```
Training Features Shape: (276, 32)
Training Labels Shape: (276,)
Testing Features Shape: (119, 32)
Testing Labels Shape: (119,)
```

8.2 Error Calculation and Graph Plotting

[31]:
```
def rmscores(model):
    scores = cross_val_score(model, X_train, y_train,␣
↪scoring="neg_mean_squared_error", cv=20)
    rmse_scores = np.sqrt(-scores)
    rmean = rmse_scores.mean()

    print("Root mean square error: ",rmean)

#     #combined rmse value
# rss=((y_test-y_pred)**2).sum()
# mse=np.mean((y_test-y_pred)**2)
# print("Final rmse value is =",np.sqrt(np.mean((y_test-y_pred)**2)))
# def map_acc(model):
#     # Performance metrics
#     errors = abs(y_pred - y_test)
```

```
#     print('Metrics for Random Forest Trained on Expanded Data')
#     print('Average absolute error:', round(np.mean(errors), 2), 'degrees.')
#     # Calculate mean absolute percentage error (MAPE)
#     mape = np.mean(100 * (errors / y_test))

    # Compare to baseline
#     improvement_baseline = 100 * abs(mape - baseline_mape) / baseline_mape
#     print('Improvement over baseline:', round(improvement_baseline, 2), '%.')
#     print(round(mape,2))
#     # Calculate and display accuracy
#     accuracy = 100 - rmean
#     print('Accuracy:', accuracy, '%.')
```

8.3 Feature Importance:

1. Selecting the features with high importance.
2. Selecting top 3 features.
3. Selecting these features among previous features via indices(X_train,X_test).

[66]:
```
def feat_imp(model):
    # Get numerical feature importances
    importances = list(model.feature_importances_)
    # List of tuples with variable and importance
    feature_importances = [(feature, round(importance, 2)) for feature,␣
 ↪importance in zip(df_columns, importances)]
    # Sort the feature importances by most important first
    feature_importances = sorted(feature_importances, key = lambda x: x[1],␣
 ↪reverse = True)
    # Print out the feature and importances
#     [print('Variable: {:20} Importance: {}'.format(*pair)) for pair in␣
 ↪feature_importances]
#     print(feature_importances[0:3])
#     fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2)

    # list of x locations for plotting
    x_values = list(range(len(importances)))
#     # Make a bar chart
# #     ax1.plot([1, 2])
#     plt.bar(x_values, importances, orientation = 'vertical', color = 'r',␣
 ↪edgecolor = 'k', linewidth = 1)
#     # Tick labels for x axis
#     plt.xticks(x_values, df_columns, rotation='vertical')
#     # Axis labels and title
#     plt.ylabel('Importance'); plt.xlabel('Variable'); plt.title('Variable␣
 ↪Importances')
```

```
    # List of features sorted from most to least important
    sorted_importances = [importance[1] for importance in feature_importances]
    sorted_features = [importance[0] for importance in feature_importances]
    # Cumulative importances
    cumulative_importances = np.cumsum(sorted_importances)
    # Make a line graph
#     ax2.plot([1, 2])
#     plt.plot(x_values, cumulative_importances, 'g-')
#     # Draw line at 95% of importance retained
#     plt.hlines(y = 0.90, xmin=0, xmax=len(sorted_importances), color = 'r',␣
↪linestyles = 'dashed')
#     # Format x ticks and labels
#     plt.xticks(x_values, sorted_features, rotation = 'vertical')
#     # Axis labels and title
#     plt.xlabel('Variable'); plt.ylabel('Cumulative Importance'); plt.
↪title('Cumulative Importances')

#     print('Number of features for 90% importance:', np.
↪where(cumulative_importances > 0.90)[0][0] + 1)

    # Extract the names of the most important features
    important_feature_names = [feature[0] for feature in feature_importances[0:
↪3]]
    # Find the columns of the most important features
    important_indices = [df_columns.index(feature) for feature in␣
↪important_feature_names]
#     print(important_indices)
    # Create training and testing sets with only the important features
    important_train_features = X_train.iloc[:, important_indices]
    important_test_features = X_test.iloc[:, important_indices]


#     print(important_train_features.shape)
    # Sanity check on operations
#     print('Important train features shape:', important_train_features.shape)
#     print('Important test features shape:', important_test_features.shape)

    return important_train_features,important_test_features
```

8.4 Model training and Evaluation

[87]:
```
# Random Forest Regressor
print("Model : Random forest Regressor")

regr = RandomForestRegressor()
```

```
rgrmodel = regr.fit(X_train,y_train)

#predict the test result
y_pred=rgrmodel.predict(X_test)
rmscores(rgrmodel)



df['finalG3'] = pd.DataFrame(y_pred)
df_out = pd.merge(df,df[['finalG3']],how = 'left',left_index = True,␣
↪right_index = True)
df.head()
```

```
Model : Random forest Regressor

/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in version
0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)

Root mean square error:  1.4891846307314445
```

[87]:

```
   school  sex  age  address  famsize  Pstatus  Medu  Fedu  Mjob  Fjob  …  \
0       0    0   18        1        0        0     4     4     0     4  …
1       0    0   17        1        0        1     1     1     0     2  …
2       0    0   15        1        1        1     1     1     0     2  …
3       0    0   15        1        0        1     4     2     1     3  …
4       0    0   16        1        0        1     3     3     2     2  …

   freetime  goout  Dalc  Walc  health  absences  G1  G2  G3  finalG3
0         3      4     1     1       3         6   5   6   6      7.9
1         3      3     1     1       3         4   5   5   6     11.9
2         3      2     2     3       3        10   7   8  10      7.5
3         2      2     1     1       5         2  15  14  15      9.4
4         3      2     1     2       5         4   6  10  10      8.5

[5 rows x 34 columns]
```

[88]:

```
important_train_features,important_test_features = feat_imp(rgrmodel)
```

[93]:

```
# Train the expanded model on only the important features
rgrmodel.fit(important_train_features, y_train)
# Make predictions on test data
y_pred = rgrmodel.predict(important_test_features)

scores = cross_val_score(rgrmodel, important_train_features, y_train,␣
↪scoring="neg_mean_squared_error", cv=20)
rmse_scores = np.sqrt(-scores)
```

```
rmean = rmse_scores.mean()
print("RMSE for Feature Engineered RF : ",rmean)
```

```
RMSE for Feature Engineered RF :  1.4611114669764524
```

[ ]:

```
```

[109]:

```
# Linear Regressor
print("Model : Linear Regressor")

lnr = LinearRegression()
lnrmodel = lnr.fit(X_train, y_train)

#predict the test result
y_pred=lnrmodel.predict(X_test)
# print(y_test.shape)
plt.title('Linear Regressor')
sns.regplot(y_test,y_pred)
plt.xlabel('Observed')
plt.ylabel('Predicted')
rmscores(lnrmodel)



df['finalG3'] = pd.DataFrame(y_pred)
df_out = pd.merge(df,df[['finalG3']],how = 'left',left_index = True,␣
↪right_index = True)
df.head()
```

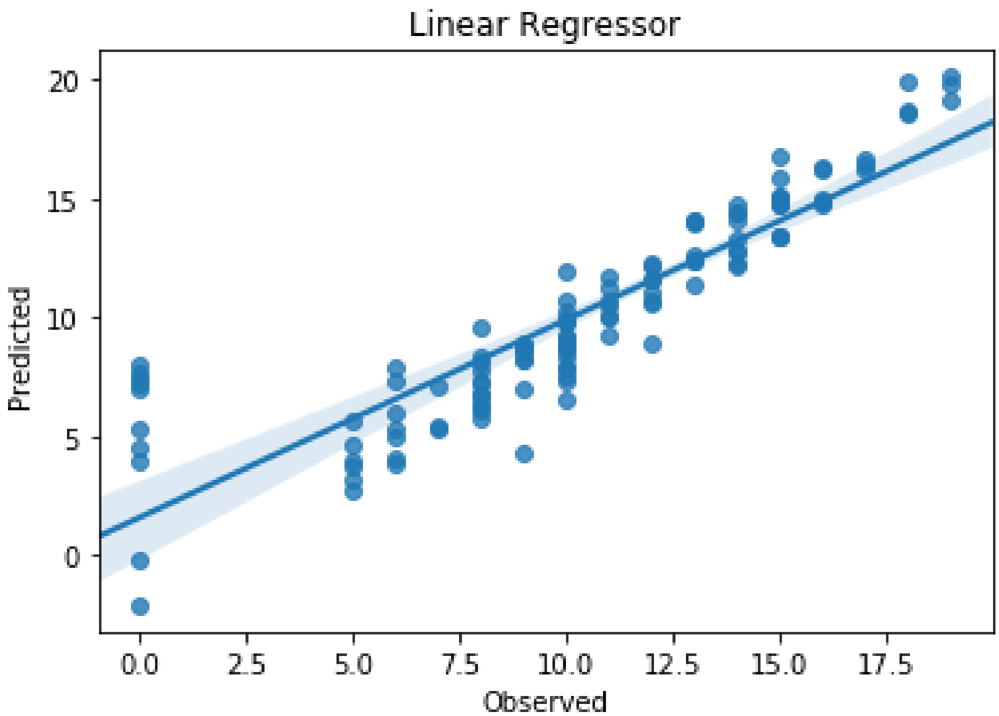
```
Model : Linear Regressor
Root mean square error:  1.7844463379416968
```

[109]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	…	\
0	0	0	18	1	0	0	4	4	0	4	…	
1	0	0	17	1	0	1	1	1	0	2	…	
2	0	0	15	1	1	1	1	1	0	2	…	
3	0	0	15	1	0	1	4	2	1	3	…	
4	0	0	16	1	0	1	3	3	2	2	…	

	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3	finalG3
0	3	4	1	1	3	6	5	6	6	6.599157
1	3	3	1	1	3	4	5	5	6	11.601092
2	3	2	2	3	3	10	7	8	10	2.714617
3	2	2	1	1	5	2	15	14	15	8.306609
4	3	2	1	2	5	4	6	10	10	8.576246

[5 rows x 34 columns]



[94]:
```
# Decision tree Regressor
print("Model : Decision tree Regressor")

dtr = DecisionTreeRegressor(random_state = 0)
dtrmodel = dtr.fit(X_train,y_train)

#predict the test result
y_pred=dtrmodel.predict(X_test)
# print(y_test.shape)
# plt.title('Decision Tree Regressor')
# graphs(dtrmodel)
rmscores(dtrmodel)

df['finalG3'] = pd.DataFrame(y_pred)
df_out = pd.merge(df,df[['finalG3']],how = 'left',left_index = True,␣
 ↪right_index = True)
df.head()
```

```
Model : Decision tree Regressor
Root mean square error:  1.9487467976467108
```

[94]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	…	\
0	0	0	18	1	0	0	4	4	0	4	…	
1	0	0	17	1	0	1	1	1	0	2	…	

```
2        0    0   15          1          1          1     1     1     0     2  ...
3        0    0   15          1          0          1     4     2     1     3  ...
4        0    0   16          1          0          1     3     3     2     2  ...

   freetime  goout  Dalc  Walc  health  absences  G1  G2  G3  finalG3
0         3      4     1     1       3         6   5   6   6      8.0
1         3      3     1     1       3         4   5   5   6     11.0
2         3      2     2     3       3        10   7   8  10      6.0
3         2      2     1     1       5         2  15  14  15     10.0
4         3      2     1     2       5         4   6  10  10      7.0

[5 rows x 34 columns]
```

[95]:

```
important_train_features,important_test_features = feat_imp(dtrmodel)
```

[100]:

```
# Train the expanded model on only the important features
dtrmodel.fit(important_train_features, y_train)
# Make predictions on test data
y_pred = dtrmodel.predict(important_test_features)

scores = cross_val_score(dtrmodel, important_train_features, y_train,␣
 ↪scoring="neg_mean_squared_error", cv=20)
rmse_scores = np.sqrt(-scores)
rmean = rmse_scores.mean()
print("RMSE: ",rmean)

# plt.title('FE Random forest')
# plt.scatter(y_test,y_pred)
# plt.plot([0,(np.max(y_test))],[0,(np.max(y_test))], 'r', alpha=0.5)
# plt.xlabel('Observed')
# plt.ylabel('Predicted')
```

```
RMSE:  1.6577707009804121
```

8.5 Model Tune up:

To tune up the hyperparameters we are going to do a grid_search. Using cross validaton grid_search will get the mean loss score for every combination of hyperparameters passed into the grid. The grid will then save the model which achieved the best score. This way we will be able to tune up our model without splitting the training set any more. Here I'm doing only for Random forest as it has least error rate.

[106]:

```
from sklearn.model_selection import GridSearchCV
param_grid = [{'n_estimators': [50,100,200,1000], 'max_features': [5,10,15,18],
               'bootstrap': [True,False], 'min_samples_leaf':[1,2],
               'min_samples_split':[2,3]}]
```

```
forest_reg = RandomForestRegressor()
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',verbose=1)

grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 128 candidates, totalling 640 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 640 out of 640 | elapsed:  2.8min finished
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_search.py:813:
DeprecationWarning: The default of the `iid` parameter will change from True to
False in version 0.22 and will be removed in 0.24. This will change numeric
results when test-set sizes are unequal.
  DeprecationWarning)
```

[106]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=RandomForestRegressor(bootstrap=True, criterion='mse',
                                             max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators='warn', n_jobs=None,
                                             oob_score=False, random_state=None,
                                             verbose=0, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'bootstrap': [True, False],
                          'max_features': [5, 10, 15, 18],
                          'min_samples_leaf': [1, 2],
                          'min_samples_split': [2, 3],
                          'n_estimators': [50, 100, 200, 1000]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=1)
```

[107]:
```
print(" ----------------- GRID_SEARCH INFO -----------------\n ")
print(grid_search.best_params_)
print(grid_search.best_estimator_)
print("Best CV score:", np.sqrt(-grid_search.best_score_))

cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
 ----------------- GRID_SEARCH INFO -----------------
```

```
{'bootstrap': False, 'max_features': 18, 'min_samples_leaf': 1,
'min_samples_split': 3, 'n_estimators': 200}
RandomForestRegressor(bootstrap=False, criterion='mse', max_depth=None,
                      max_features=18, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=3,
                      min_weight_fraction_leaf=0.0, n_estimators=200,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
Best CV score: 1.5599817605655208
2.1883035584141695 {'bootstrap': True, 'max_features': 5, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 50}
2.136847959705732 {'bootstrap': True, 'max_features': 5, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 100}
2.1273121053287554 {'bootstrap': True, 'max_features': 5, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 200}
2.13919152749452 {'bootstrap': True, 'max_features': 5, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 1000}
2.1444012114644364 {'bootstrap': True, 'max_features': 5, 'min_samples_leaf': 1,
'min_samples_split': 3, 'n_estimators': 50}
2.1570575137346135 {'bootstrap': True, 'max_features': 5, 'min_samples_leaf': 1,
'min_samples_split': 3, 'n_estimators': 100}
2.134691418429996 {'bootstrap': True, 'max_features': 5, 'min_samples_leaf': 1,
'min_samples_split': 3, 'n_estimators': 200}
2.145050134451814 {'bootstrap': True, 'max_features': 5, 'min_samples_leaf': 1,
'min_samples_split': 3, 'n_estimators': 1000}
2.2162486029452544 {'bootstrap': True, 'max_features': 5, 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 50}
2.2148632337905716 {'bootstrap': True, 'max_features': 5, 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 100}
2.187483570743321 {'bootstrap': True, 'max_features': 5, 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 200}
2.19679954481181 {'bootstrap': True, 'max_features': 5, 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 1000}
2.268376535953709 {'bootstrap': True, 'max_features': 5, 'min_samples_leaf': 2,
'min_samples_split': 3, 'n_estimators': 50}
2.2637667561019956 {'bootstrap': True, 'max_features': 5, 'min_samples_leaf': 2,
'min_samples_split': 3, 'n_estimators': 100}
2.20060614459739 {'bootstrap': True, 'max_features': 5, 'min_samples_leaf': 2,
'min_samples_split': 3, 'n_estimators': 200}
2.210662267215839 {'bootstrap': True, 'max_features': 5, 'min_samples_leaf': 2,
'min_samples_split': 3, 'n_estimators': 1000}
1.773730205126955 {'bootstrap': True, 'max_features': 10, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 50}
1.7813493298010386 {'bootstrap': True, 'max_features': 10, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 100}
1.7480905473098933 {'bootstrap': True, 'max_features': 10, 'min_samples_leaf':
```

```
1, 'min_samples_split': 2, 'n_estimators': 200}
1.7475296715359516 {'bootstrap': True, 'max_features': 10, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 1000}
1.8402214118201272 {'bootstrap': True, 'max_features': 10, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 50}
1.7518297081075374 {'bootstrap': True, 'max_features': 10, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 100}
1.7383116152998541 {'bootstrap': True, 'max_features': 10, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 200}
1.747394328456834 {'bootstrap': True, 'max_features': 10, 'min_samples_leaf': 1,
'min_samples_split': 3, 'n_estimators': 1000}
1.7618767182098527 {'bootstrap': True, 'max_features': 10, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 50}
1.8007089440176802 {'bootstrap': True, 'max_features': 10, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 100}
1.8250564215396667 {'bootstrap': True, 'max_features': 10, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 200}
1.787388147756086 {'bootstrap': True, 'max_features': 10, 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 1000}
1.8831138931710745 {'bootstrap': True, 'max_features': 10, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 50}
1.781190543393492 {'bootstrap': True, 'max_features': 10, 'min_samples_leaf': 2,
'min_samples_split': 3, 'n_estimators': 100}
1.778408988093299 {'bootstrap': True, 'max_features': 10, 'min_samples_leaf': 2,
'min_samples_split': 3, 'n_estimators': 200}
1.7973827710911656 {'bootstrap': True, 'max_features': 10, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 1000}
1.621439242583978 {'bootstrap': True, 'max_features': 15, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 50}
1.644235869394321 {'bootstrap': True, 'max_features': 15, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 100}
1.6416923046626022 {'bootstrap': True, 'max_features': 15, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 200}
1.623662734040861 {'bootstrap': True, 'max_features': 15, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 1000}
1.682255350049174 {'bootstrap': True, 'max_features': 15, 'min_samples_leaf': 1,
'min_samples_split': 3, 'n_estimators': 50}
1.6338803623469498 {'bootstrap': True, 'max_features': 15, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 100}
1.6022735269588817 {'bootstrap': True, 'max_features': 15, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 200}
1.6358171608012784 {'bootstrap': True, 'max_features': 15, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 1000}
1.6635935948647784 {'bootstrap': True, 'max_features': 15, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 50}
1.672536193359059 {'bootstrap': True, 'max_features': 15, 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 100}
1.6529763536216802 {'bootstrap': True, 'max_features': 15, 'min_samples_leaf':
```

```
2, 'min_samples_split': 2, 'n_estimators': 200}
1.6467602803969563 {'bootstrap': True, 'max_features': 15, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 1000}
1.6316942400265058 {'bootstrap': True, 'max_features': 15, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 50}
1.6529152229216564 {'bootstrap': True, 'max_features': 15, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 100}
1.644250239433227 {'bootstrap': True, 'max_features': 15, 'min_samples_leaf': 2,
'min_samples_split': 3, 'n_estimators': 200}
1.6503208587995193 {'bootstrap': True, 'max_features': 15, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 1000}
1.6721118296975512 {'bootstrap': True, 'max_features': 18, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 50}
1.6765420113859446 {'bootstrap': True, 'max_features': 18, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 100}
1.5965604447133181 {'bootstrap': True, 'max_features': 18, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 200}
1.598531421812452 {'bootstrap': True, 'max_features': 18, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 1000}
1.6089170119824079 {'bootstrap': True, 'max_features': 18, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 50}
1.6238091358820412 {'bootstrap': True, 'max_features': 18, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 100}
1.6054421925719267 {'bootstrap': True, 'max_features': 18, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 200}
1.6115581716897482 {'bootstrap': True, 'max_features': 18, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 1000}
1.6469309502358878 {'bootstrap': True, 'max_features': 18, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 50}
1.6143710361846213 {'bootstrap': True, 'max_features': 18, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 100}
1.615839276367172 {'bootstrap': True, 'max_features': 18, 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 200}
1.607978815552676 {'bootstrap': True, 'max_features': 18, 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 1000}
1.653082216223378 {'bootstrap': True, 'max_features': 18, 'min_samples_leaf': 2,
'min_samples_split': 3, 'n_estimators': 50}
1.6440266729521655 {'bootstrap': True, 'max_features': 18, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 100}
1.6092468444331556 {'bootstrap': True, 'max_features': 18, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 200}
1.6138286815051865 {'bootstrap': True, 'max_features': 18, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 1000}
2.026179026955753 {'bootstrap': False, 'max_features': 5, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 50}
2.034221533259544 {'bootstrap': False, 'max_features': 5, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 100}
2.026992533751434 {'bootstrap': False, 'max_features': 5, 'min_samples_leaf': 1,
```

```
'min_samples_split': 2, 'n_estimators': 200}
2.0327750780929854 {'bootstrap': False, 'max_features': 5, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 1000}
2.0691127413052883 {'bootstrap': False, 'max_features': 5, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 50}
2.0532487954317737 {'bootstrap': False, 'max_features': 5, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 100}
2.05616246895324 {'bootstrap': False, 'max_features': 5, 'min_samples_leaf': 1,
'min_samples_split': 3, 'n_estimators': 200}
2.0534625352206333 {'bootstrap': False, 'max_features': 5, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 1000}
2.137545168420393 {'bootstrap': False, 'max_features': 5, 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 50}
2.0845363010451097 {'bootstrap': False, 'max_features': 5, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 100}
2.122088215560941 {'bootstrap': False, 'max_features': 5, 'min_samples_leaf': 2,
'min_samples_split': 2, 'n_estimators': 200}
2.1036823684974952 {'bootstrap': False, 'max_features': 5, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 1000}
2.122733925136222 {'bootstrap': False, 'max_features': 5, 'min_samples_leaf': 2,
'min_samples_split': 3, 'n_estimators': 50}
2.062386590494838 {'bootstrap': False, 'max_features': 5, 'min_samples_leaf': 2,
'min_samples_split': 3, 'n_estimators': 100}
2.135256069918948 {'bootstrap': False, 'max_features': 5, 'min_samples_leaf': 2,
'min_samples_split': 3, 'n_estimators': 200}
2.1080976647984784 {'bootstrap': False, 'max_features': 5, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 1000}
1.7103482991331986 {'bootstrap': False, 'max_features': 10, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 50}
1.7112017982116832 {'bootstrap': False, 'max_features': 10, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 100}
1.6909767458698135 {'bootstrap': False, 'max_features': 10, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 200}
1.692063235447666 {'bootstrap': False, 'max_features': 10, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 1000}
1.746048852181089 {'bootstrap': False, 'max_features': 10, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 50}
1.7200992135187512 {'bootstrap': False, 'max_features': 10, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 100}
1.7332420065475187 {'bootstrap': False, 'max_features': 10, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 200}
1.691749569789825 {'bootstrap': False, 'max_features': 10, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 1000}
1.8079592004832177 {'bootstrap': False, 'max_features': 10, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 50}
1.7594824205386028 {'bootstrap': False, 'max_features': 10, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 100}
1.7386954950354363 {'bootstrap': False, 'max_features': 10, 'min_samples_leaf':
```

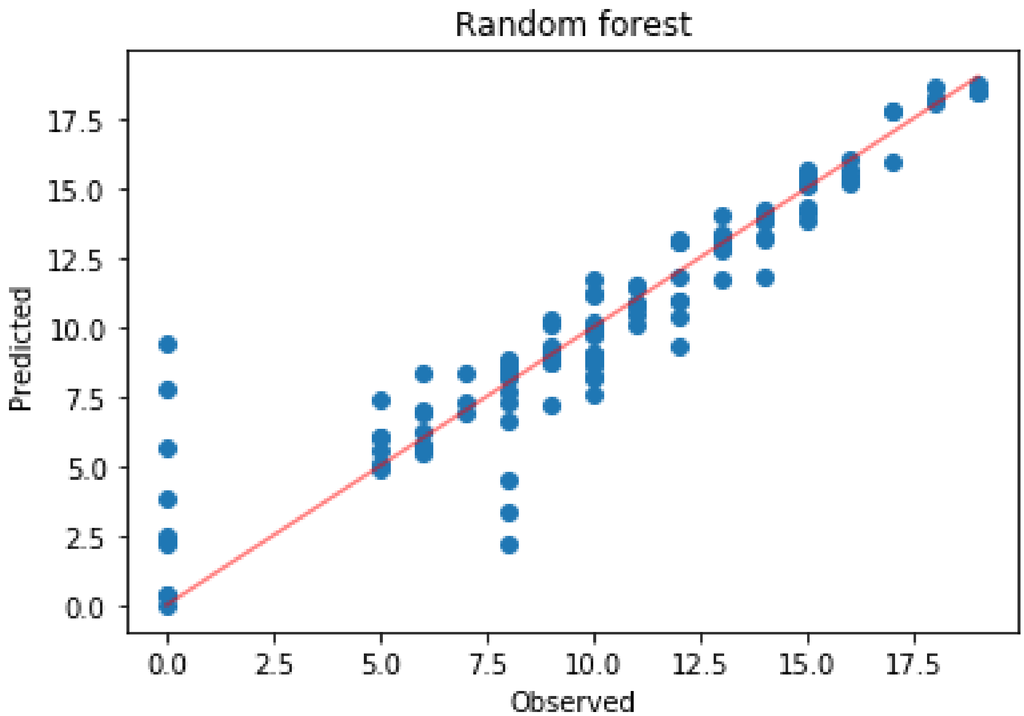
```
2, 'min_samples_split': 2, 'n_estimators': 200}
1.7343060522718754 {'bootstrap': False, 'max_features': 10, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 1000}
1.7552609181833254 {'bootstrap': False, 'max_features': 10, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 50}
1.7674428539983291 {'bootstrap': False, 'max_features': 10, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 100}
1.7434991818586283 {'bootstrap': False, 'max_features': 10, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 200}
1.7402964505341452 {'bootstrap': False, 'max_features': 10, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 1000}
1.6507104311587275 {'bootstrap': False, 'max_features': 15, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 50}
1.6108608959456918 {'bootstrap': False, 'max_features': 15, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 100}
1.5804737239286788 {'bootstrap': False, 'max_features': 15, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 200}
1.6070232642183901 {'bootstrap': False, 'max_features': 15, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 1000}
1.664489302363088 {'bootstrap': False, 'max_features': 15, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 50}
1.6199629569481082 {'bootstrap': False, 'max_features': 15, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 100}
1.6026844271996923 {'bootstrap': False, 'max_features': 15, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 200}
1.6076078907603006 {'bootstrap': False, 'max_features': 15, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 1000}
1.6494773393885105 {'bootstrap': False, 'max_features': 15, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 50}
1.6242207866383152 {'bootstrap': False, 'max_features': 15, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 100}
1.6287774335956458 {'bootstrap': False, 'max_features': 15, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 200}
1.6239137785904432 {'bootstrap': False, 'max_features': 15, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 1000}
1.6421437700253814 {'bootstrap': False, 'max_features': 15, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 50}
1.6246708170234365 {'bootstrap': False, 'max_features': 15, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 100}
1.6479768224437485 {'bootstrap': False, 'max_features': 15, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 200}
1.630835556607486 {'bootstrap': False, 'max_features': 15, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 1000}
1.6113011215622692 {'bootstrap': False, 'max_features': 18, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 50}
1.6019774917451153 {'bootstrap': False, 'max_features': 18, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 100}
1.6036690551375608 {'bootstrap': False, 'max_features': 18, 'min_samples_leaf':
```

```
1, 'min_samples_split': 2, 'n_estimators': 200}
1.5901256809319886 {'bootstrap': False, 'max_features': 18, 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 1000}
1.5996248427931967 {'bootstrap': False, 'max_features': 18, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 50}
1.580045432524198 {'bootstrap': False, 'max_features': 18, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 100}
1.5599817605655208 {'bootstrap': False, 'max_features': 18, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 200}
1.5817294656024692 {'bootstrap': False, 'max_features': 18, 'min_samples_leaf':
1, 'min_samples_split': 3, 'n_estimators': 1000}
1.6159447439374086 {'bootstrap': False, 'max_features': 18, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 50}
1.604295205497055 {'bootstrap': False, 'max_features': 18, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 100}
1.6185156497122686 {'bootstrap': False, 'max_features': 18, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 200}
1.5951005680637567 {'bootstrap': False, 'max_features': 18, 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 1000}
1.6015962749767256 {'bootstrap': False, 'max_features': 18, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 50}
1.6378184746416855 {'bootstrap': False, 'max_features': 18, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 100}
1.6121193946893309 {'bootstrap': False, 'max_features': 18, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 200}
1.6007438220846675 {'bootstrap': False, 'max_features': 18, 'min_samples_leaf':
2, 'min_samples_split': 3, 'n_estimators': 1000}
```

[108]:
```
final_model = grid_search.best_estimator_
# predictions
final_predictions = final_model.predict(X_test)
rmscores(final_model)

# Computing R^2 performance metric for the regression model
from sklearn import metrics
print('R2_Score: ', metrics.r2_score(y_test,final_predictions))

plt.title('Random forest')
plt.scatter(y_test,final_predictions)
plt.plot([0,(np.max(y_test))],[0,(np.max(y_test))], 'r', alpha=0.5)
plt.xlabel('Observed')
plt.ylabel('Predicted')
```

```
Root mean square error:  1.3809274488803933
R2_Score:  0.858689982447926
```

[108]:
```
Text(0, 0.5, 'Predicted')
```



9 Comments:

1. From the Feature importance we can see 'G2' affects the most in his final grade 'G3'.
2. Out of Classification models Decision tree Classifier has best Validation score. So it would be preferred.
3. Out of Regression models Random Forest regressor has least RMSE error, so it is preferred.
4. After Performing prediction for important feature for tree based models(Random Forest & Decision Tree). Again Random forest has least error rate.

```
[ ]:
```