INTRODUCTION

The aim of this report is to analyze a dataset in order to discover hidden patterns from the raw data.

In particular, we explore datasets , provided in the form of csv file or JSON file, focusing on certain attributes in order to find out new features about the data that are not attributes of the given dataset.

In fact, in this case we analyze the files commits.csv to find out new patterns regarding the commits.csv such as evenness, earliness, lateness and even contribution.

Moreover, after have been discovered those features , we can use a tree algorithm, a supervised learning model used in predicting a dependent variable with a series of training variables.

For instance, we should choose some of the new patterns found in the data analysis to predict the grades of the commits, given in the file grades.JSON.

For example, we could do some experiments researching a tree regarding the score representing how well the tree performs by changing the numbers of training variables of the dataset csv.

To sum up, in this report, there will descripted several tasks regarding the data commits.csv and grades.JSON.

TASK 1 :CALCULATING EVENNESS OF COMMITS OVER TIME

The aim of this task is to calculate the evenness of commits over time .

To accomplish this task ,we create a method called calculateEvenness that has as parameters a filename of the dataset file, representing the dataset to be analyzed, and the number of the project of the commits.

```
def calculateEvenness(filename,project_id):
```

In this method, the first step involves loading into a data frame, called df from the CSV file commits.CSV.

```
df=pd.read_csv(filename,delimiter=";")#Load the dataframe reading the file commits.csv
```

To load df , we have to import the pandas library ,useful for data manipulation and analysis.

```
import pandas as pd#use shorthand pd for  the pandas library
```

Then we select the column time of the data frame df ,applying a lambda to its value ,that converts in the dataframe the values of time from hours to weeks.

```
df['time'] = df['time'].apply(lambda x: int((x/24)/7))#converts hours into weeks in the dataframe
```

After, we introduce the variable values in which there are selected the columns named time and project_id of the dataframe and we create a variable x that selects the column named time where the project_id is the same project_id given as a parameter of the method invoked.

```
values=df[['project_id','time']]
x= values[values.project_id==project_id ]
```

Therefore ,the total_commits is found by calculating the number of rows of commits in the dataframe x .

```
total_commits=x.shape[0]#total commits
print("Number of total commits  is:",total_commits)
```

The next step is calculating the total weeks in which all commits are done.

This could be performed by creating a list (declared as z), collecting all the weeks of commits ,and then compute the difference between the maximum value of this list and the minimum of the list.

```
z=df['time'].tolist()
#the variable z is the list of values in the column time in the given project_id
total_weeks=max(z)-min(z)
#the number of weeks it is defined as  the difference between the max of the week and the min of the week
```

Then we create a list such that the variance of the values is maximum.

Basically, the list declared as max_var_values is a list , given by an array of zeros which length equals to total_weeks-1.Finally,the values of total_commits is appended to max_var_values making the variance of the list the greatest.

```
max_var_values=list(np.zeros(total_weeks-1, dtype=int))
max_var_values.append(total_commits)
print(max_var_values)
```

We introduce the variable max_variance to indicate the variance of the list max_var_values.

```
max_variance=np.var(max_var_values)
print("The maximum variance is:",max_variance)
```

In order to calculate the variance is necessary importing the numpy library, a general-purpose array-processing package that provides a high-performance multidimensional array object, and tools for working with these arrays.

```
import numpy as np#use shorthand np for  the numpy library
```

Instead to calculate the actual variance of the commits belonging to a certain project_id ,is needed to represent y, a list of the weeks in which commits are done.

```
y=x['time'].tolist()
```

After we compute an histogram given input data and bins, intervals to split the data ,returns the values of the histogram and the edges of the bins used by the histogram function .

Thus ,we find the commits_per_week and edges ,using the function numpy.histogram.

In this case ,y is the input data while the bins are founded ,using np.arange((0, 1+total_weeks)).

In fact,bins is an array of evenly spaced values starting from 0,included and ending to total_weeks+1 ,excluded.

Thus, to find the actual_variance, it is used the np.var of the commits_per_week.

```
commits_per_week, edges = np.histogram(y, bins=np.arange(0, 1+total_weeks))
actual_variance=np.var(commits_per_week)
```

Finally, after has been founded the value of the max_variance and the actual_variance of commits ,given a project id ,we simply use the formula :

evennessScore = 1-(actual_variance/max_variance)

This score represents a number belonging to the interval [0,1] that measure evenness of commits of a certain project_id over time , expressed in terms of weeks .


```
commits_per_week, edges = np.histogram(y, bins=np.arange(0, 1+total_weeks))
actual_variance=np.var(commits_per_week)
```


TASK 2 :CALCULATING LATENESS OF COMMITS

The aim of this task is to calculate the lateness of commits given the filename of the dataset of commits and a certain project_id.

```
def calculateLateness(filename,project_id):
```

Has already done in the first task, it is loaded in a dataframe after a dataset has been read, time is converted from hours to weeks, and finally finding the total_commits and the number_of_weeks.

```
df=pd.read_csv(filename,delimiter=";")#Load the dataframe reading the file commits.csv
df['time'] = df['time'].apply(lambda x: int((x/24)/7))#converts hours into weeks in the dataframe
values=df[['project_id','time']]
x= values[values.project_id==project_id ]
total_commits=x.shape[0]#total commits
#print("Number of total commits  is:",total_commits)
# approach of finding the number of weeks
z=x['time'].tolist()#the variable z is the list of values in the column time inthe given project_id
number_of_weeks=max(z)-min(z)
```

Then ,using np.linspace it is possible to create a list of evenly spaced numbers over a specified interval ϵ[0,1] in which the number of samples to generate ,corresponds to the number_of_weeks.

```
weights=np.linspace(start=0, stop=1, num=number_of_weeks)
```

We can refer to this list as weights ,and using a for loop enable to find lateness as the sum of values given from the relation z[i]/total_commits*weights[i] where i is the index of the element to be taken into consideration.

```
for i in range(0,len(weights)):
    value=z[i]/total_commits*weights[i]
    lateness+=value

return lateness
```

To sum up, the lateness is a number that indicates how late are commits with the same project_id in an interval from 0 to 1 .

TASK 3 :CALCULATING EARLYNESS OF COMMITS

Regarding the way to find earliness of commits ,it is possible to make assumption about it.

Firstly ,as lateness, could be represent a number belonging to the interval [0,1] with the extreme values included.

Moreover, there is a relation between earliness and lateness.

In fact ,the sum between earliness an lateness correspond to 1 .

e.g earliness +lateness=1

Therefore by shifting lateness on the 2nd member of the previous relation ,it is possible to calculate earliness and the difference between 1 and lateness.

```
def calculateEarlyness(filename,project_id):
    return 1-calculateLateness(filename,project_id)
```

TASK 4:CALCULATE EVEN CONTRIBUTION DONE BY A TEAM

Basically, the aim of this task if to find a score that states even contribution by members of a team working for commits with the same project.id

In order to find even contribution by the members of a team it will be helpful to create a procedure that enable to compute the contribution for a member of a team .

Such method has 3 parameters :

a)the filename of the dataset

b)the project_id represented as an integer

c)the member identified by a number

Firstly , the data source it is been read and loaded into a dataframe .

```
def calculateContributionForAMember(filename,project_id,member):
    df=pd.read_csv(filename,delimiter=";")#Load the dataframe reading the file commits.csv
```

Basically, the even contribution by a member is the sum between the total additions and the total deletions done my a member.

The way to find all the addition or all deletions done by a members is using the .loc that it allows to filter the values of the dataframe under multiple Boolean conditions along the rows and make column selections by label

Therefore, to calculate total_additions and total_deletions it is enough to apply sum(),a function that given iterable items returns the sum of all items in an iterable sequence.

```
def calculateContributionForAMember(filename,project_id,member):
    df=pd.read_csv(filename,delimiter=";")#Load the dataframe reading the file commits.csv
   # select the values of dataframe  to use Boolean indexing.
    total_additions=df.loc[(df['project_id']==project_id)&(df['committer']==member)  , 'additions'].sum()
    total_deletions=df.loc[(df['project_id']==project_id)&(df['committer']==member), 'deletions'].sum()
    print("User",member)
    print("Number of additions",total_additions)
    print("Number of deletions",total_deletions)
    return total_additions+total_deletions
```

Then ,after it has been created the method calculateContributionForAMember(filename,project_id,member),we can create a procedure that given the filename of the dataset and a project id find the even contribution for a team.

```
def calculateEvenContribution(filename,project_id): # amount of work per member
```

Firstly , after loading the dataset into a dataframe df, a new dataframe, named ,values select the column labelled project_id, committer, additions and deletions. At the end, the variable x collects the values in which the project_id is the one given as a parameter

```
df=pd.read_csv(filename,delimiter=";")#Load the dataframe reading the file commits.csv
values=df[['project_id','committer','additions','deletions']]
x=values[values.project_id==project_id ]
```

First of all, we find the members that are working for the project.

Then a list of committers is created using the dataframe x.

After, it is initialize an empty list declared as works. In this list ,for each member it is executed calculateContributionForAMember(filename,project_id,members[i]).

Therefore ,the total_contribution is achieved by applying the function sum to the list works.

```
members=list(set(x['committer']))#list of members in a team
works=[]#list wich  contains the values of the amount of work done by each member of the team
for i in range(0,len(members)):
    works.append(calculateContributionForAMember(filename,project_id,members[i]))
total_contribution=sum(works)
```

The following step is to find both the maximum variance and the actual variance regarding contribution in order to calculate the even contribution for a team.

```
max_var_values=list(np.zeros(len(members)-1, dtype=int))
print(max_var_values)
max_var_values.append(total_contribution)
print(max_var_values)
max_variance=np.var(max_var_values)
print("The maximum variance is:",max_variance)
actual_variance=np.var(works)
print("The actual variance is:",actual_variance)
```

Basically, the max_var_values is the result of appending the total_contribution to list created by an array of zeros which shape equals to the length of members, decreased by 1.

Thus, the variable max_variance it is introduced to indicate the variance of the list max_var_values, while the actual_variance is the variance of works.

Finally, the value 1-(actual_variance/max_variance) is returned and this value represents the even contribution for a team working for commits of a given project_id

```
return  1-(actual_variance/max_variance)
```

TASK 5:PREDICT THE GRADE OF COMMITS BASED ON EVENNESS,EARLYNESS,LATENESS

The aim of this task is to predict the grade of commits based on attributes such as evenness, lateness and earliness.

First of all, we have to create a new dataset not inserting project_id of commits that are not present in the file grades.JSON, that is the file containing the grade for each project_id.

As a result ,the project_id 17 , will not be inserted in a new dataset in which each project _id is followed by the evenness, the earlyness, the lateness and the grade .

```
def createNewDataset(filename):
    df=pd.read_csv(filename,delimiter=";")#Load the dataframe reading the file commits.csv
    data= pd.read_json("C:\\Users\\Andrea\\Downloads\\grades.json", orient='columns')
    a=list(set(df['project_id']))
    print(a)
    row=[]
    print(len(a))
    new_data = data.sort_values('project_id')
    grades=new_data['grade'].tolist()
    del grades[16]#delete the grades of the project 17 as this does not exist in the file commits.csv
    print(len(grades))
    #print(new_data)
    header = ['project_id', 'evenness', 'lateness','earlyness','grade']
    lines = [[]] * len(a)
    #print(lines)
    for i in range(0,len(a)):
        row=[a[i],calculateEvenness(filename,a[i]),calculateLateness(filename,a[i]),calculateEarlyness(filename,a[i]),grades[i]]
        lines.append(row)
    print(lines)
    with open("C:\\Users\\Andrea\\Documents\commits_newdataset.csv", "w", newline='') as f:
        writer = csv.writer(f, delimiter=';')
        writer.writerow(header) # write the header
        # write the actual content line by line
        for l in lines:
            writer.writerow(l)
    f.close()
```

After created the new dataset it is possible to write a method that create a tree predicting grades.

Firstly, is loaded the data frame df ,reading the file commits_newdataset.csv.

```
def createTreePredictingGrades():# Predicting grade using a regression tree
    df=pd.read_csv("C:\\Users\\Andrea\\Documents\commits_newdataset.csv",delimiter=";")#read the dataset
```

The next step involves splitting the dataset vertically , so that the predicting column labeled with the grade is in y and the data such as evenness, earlyness, lateness are in X.

```
# Split the dataset vertically, so that we have the column we are predicting in y and the data in X
y = df['grade']
print(y.head())
X = df.drop('grade', axis=1)
X = X.drop('project_id', axis=1)
```

The next move is to split the dataset horizontally, so that we can use parts of it to train, or "grow", the tree, and the remaining part to test how well the tree performs. A normal percentage is to use 80% for training and 20% for testing to access to different functionality from the sklearn-library.

Thus, we import several functions from sklearn library,useful for machine learning and in this case to work with regression tree, a model that observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output

```
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
from IPython.display import Image
```

The train_test_split splits X and y into 4 variables :X_train, X_test, y_train, y_test.

```
# We split the dataset horizontally. We use 20% for testing and 80% for training
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10)
```

Then it is created a decision tree regressor, setting the maximum depth of the tree equals to 6 .

```
# Grow tree, using max_depth and/or min_impurity_decrease (mse, Mean Squared Error)
regr= DecisionTreeRegressor(max_depth=6)
regr.fit(X_train, y_train)
```

After,we fit the tree with the variable used for training.

Given the X_test the predicted values are returned in y_pred.

```
# Test tree
y_pred = regr.predict(X_test)
print(y_pred)
```

Moreover, it is displayed the testing score using the score coefficient of determination R^2 of the prediction.

```
print("Testing score: " + format(regr.score(X_test,y_test)))
```

1st attempt

```
0    3.2
1    4.3
2    4.1
3    4.3
4    3.7
Name: grade, dtype: float64
   evenness  lateness  earlyness
0  0.405615  0.152582   0.847418
1  0.688707  0.182219   0.817781
2  0.581888  0.169271   0.830729
3  0.612780  0.066335   0.933665
4  0.618691  0.130261   0.869739
[4.2  3.6  3.95 4.5 ]
Testing score: 0.8487654320987654
```

2nd attempt

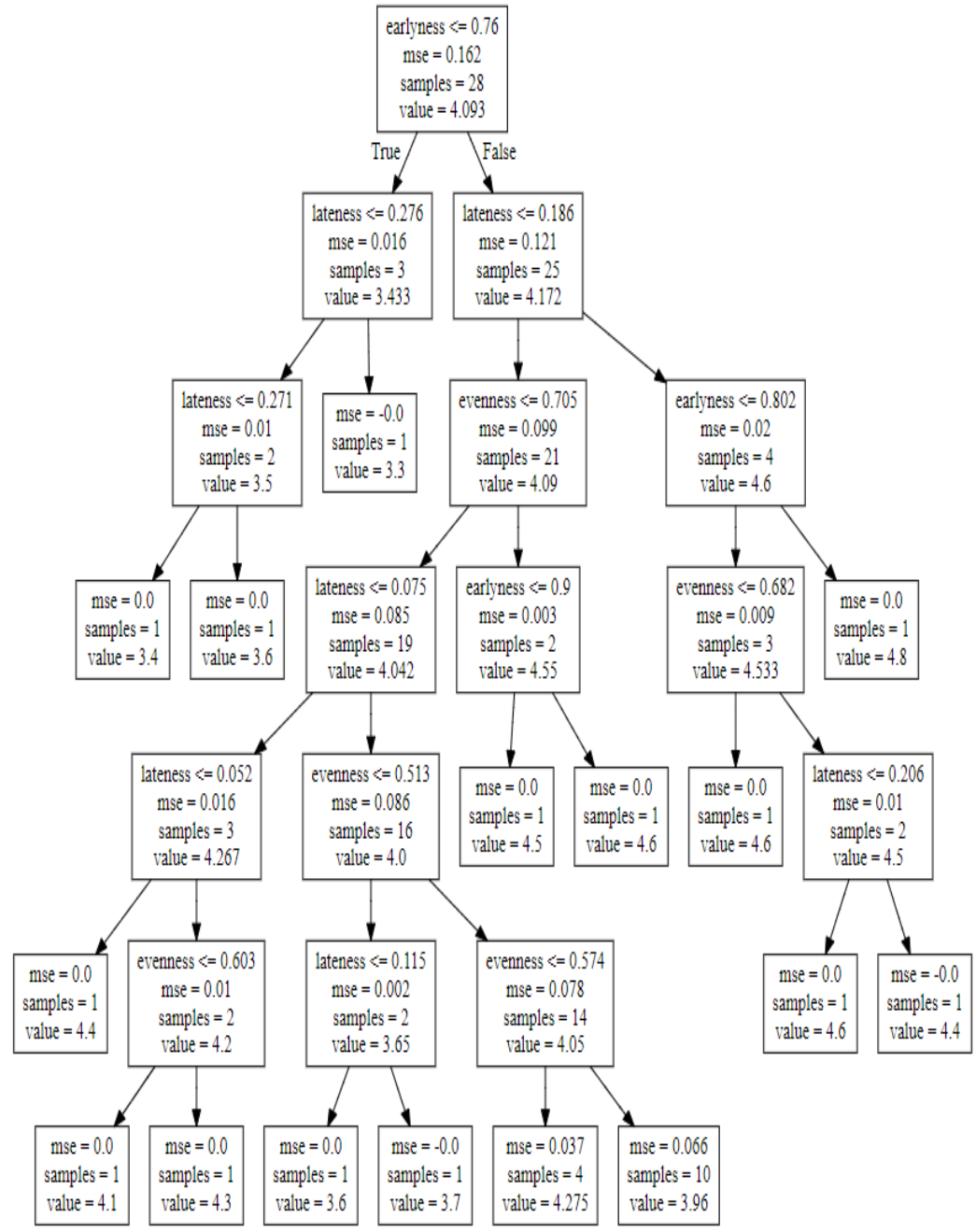
```
0    3.2
1    4.3
2    4.1
3    4.3
4    3.7
Name: grade, dtype: float64
   evenness  lateness  earlyness
0  0.405615  0.152582   0.847418
1  0.688707  0.182219   0.817781
2  0.581888  0.169271   0.830729
3  0.612780  0.066335   0.933665
4  0.618691  0.130261   0.869739
[4.16666667 3.94       4.16666667 4.16666667]
Testing score: -0.2393964334705081
```

Finally ,the code below display a tree .

```
# Display tree
#dotfile = open("dtree2.dot", 'w+')
dotfile = open("./dtree2.dot", 'w')
tree.export_graphviz(regr, out_file ="dtree2.dot" , feature_names = X.columns)
dotfile.close()
```

Then the content of the file (dtree2.dot), which should be in the project folder, can be copied and pasted into the textbox on the following webiste: http://webgraphviz.com/.

The result is below.



TASK 6:PREDICT THE GRADE OF COMMITS BASED ON EVENNESS,EARLYNESS,LATENESS AND EVEN CONTRIBUTION

Similary to what has done in the 5th task described in this report,a new dataset is created base on the files commits.csv and grades.JSON.

The only difference between the dataset created in the previous task is that in the new datasource there is added a column regarding the even contribution of commits.

```
def createNewDataset2(filename):
    df=pd.read_csv(filename,delimiter=";")#Load the dataframe reading the file commits.csv
    data= pd.read_json("C:\\Users\\Andrea\\Downloads\\grades.json", orient='columns')
    a=list(set(df['project_id']))
    print(a)
    row=[]
    print(len(a))
    new_data = data.sort_values('project_id')
    grades=new_data['grade'].tolist()
    del grades[16]#delete the grades of the project 17 as this does not exist in the file commits.csv
    print(len(grades))
    #print(new_data)
    header = ['project_id', 'evenness', 'lateness','earlyness','even_contribution','grade']
    lines = [[]] * len(a)
    #print(lines)
    for i in range(0,len(a)):
        row=[a[i],calculateEvenness(filename,a[i]),calculateLateness(filename,a[i]),calculateEarlyness(filename,a[i]),calculateEarlyness(filename,a[i]),calculateEvenContribution(filename,a[i]),grades[i]]
        lines.append(row)
    print(lines)
    with open("C:\\Users\\Andrea\\Documents\commits_newdataset2.csv", "w", newline='') as f:
         writer = csv.writer(f, delimiter=';')
         writer.writerow(header) # write the header
         # write the actual content line by line
         for l in lines:
             writer.writerow(l)
    f.close() '''
```

Below is the implementation of a tree predicting the grade according to the following attributes:

eveness,earlyness,lateness,even contribution

```
def createTreePredictingGrades2():# Predicting grade using a regression tree
    df=pd.read_csv("C:\\Users\\Andrea\\Documents\commits_newdataset2.csv",delimiter=";")#read the dataset
    print(df.head())
    print(df.tail())
    #print(df)
    # Split the dataset vertically, so that we have the column we are predicting in y and the data in X
    y = df['grade']
    print(y.head())
    X = df.drop('grade', axis=1)
    X = X.drop('project_id', axis=1)
    print(X.head())
    # We split the dataset horizontally. We use 20% for testing and 80% for training
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)
    # Grow tree, using max_depth and/or min_impurity_decrease (mse, Mean Squared Error)
    regr = DecisionTreeRegressor(max_depth=6)
    regr.fit(X_train, y_train)
    # Test tree
    y_pred = regr.predict(X_test)
    print(y_pred)
    print("Testing score: " + format(regr.score(X_test,y_test)))
    # Display tree
    # Display tree
    #dotfile = open("dtree2.dot", 'w+')
    dotfile = open("./dtree3.dot", 'w')
    tree.export_graphviz(regr, out_file ="dtree3.dot" , feature_names = X.columns)
    dotfile.close()
```

The code above is similar to the one used to implement createTreePredictingGrades.

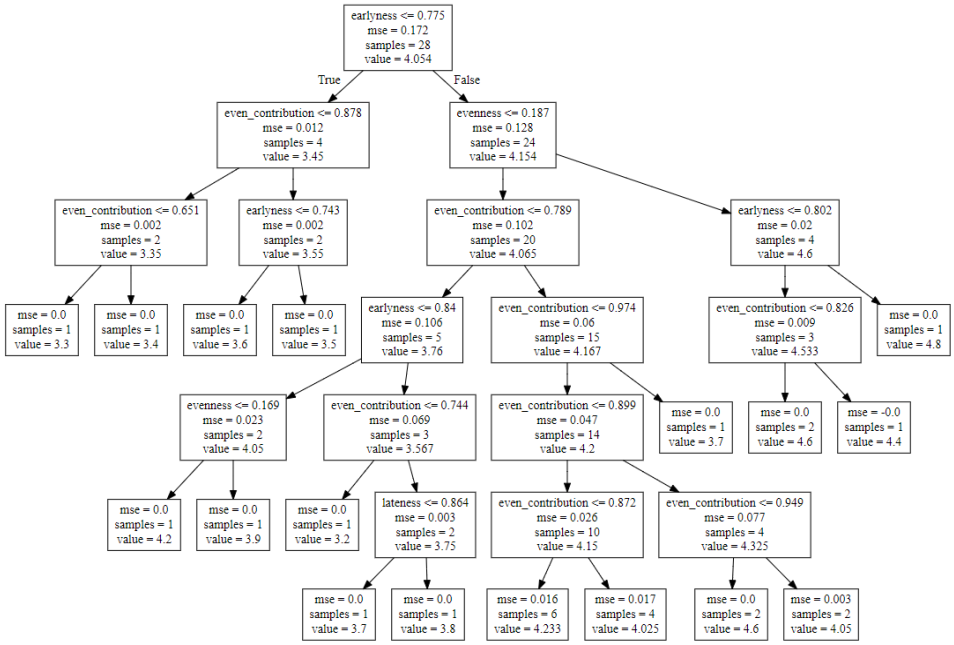
Below are some screenshots showing the values of the testing score of the tree.

```
1    3.2
2    4.3
3    4.1
4    4.3
5    3.7
Name: grade, dtype: float64
   evenness  lateness  earlyness  even_contribution
1  0.152582  0.847418   0.847418           0.733538
2  0.182219  0.817781   0.817781           0.839928
3  0.169271  0.830729   0.830729           0.897677
4  0.066335  0.933665   0.933665           0.832736
5  0.130261  0.869739   0.869739           0.990589
[4.12 4.12 3.6  4.12]
Testing score: 0.728065843621398
```

```
1    3.2
2    4.3
3    4.1
4    4.3
5    3.7
Name: grade, dtype: float64
   evenness  lateness  earlyness  even_contribution
1  0.152582  0.847418   0.847418           0.733538
2  0.182219  0.817781   0.817781           0.839928
3  0.169271  0.830729   0.830729           0.897677
4  0.066335  0.933665   0.933665           0.832736
5  0.130261  0.869739   0.869739           0.990589
[4.15 4.6  3.4  4.6 ]
Testing score: -1.3935185185185168
```

The only difference is that the data x used for prediction is made by 4 attributes rather then the 3 attributes taking into consideration in the previous task.

In the next page there is the image of the new tree ,as the result of executing the method createTreePredictingGrades2().



CONCLUSION

To sum up the tasks described in this report, is that we have used regression trees to do some experiments about how it is work the performance regarding prediction of new features such as the grades of commits.

Note that the testing score of a tree predicting grade, described in task 5 and task6 will differ each time the program is run. That is because there is an element of random involved when the tree is created.

1st execution of task 5 and task 6

```
0     3.2
1     4.3
2     4.1
3     4.3
4     3.7
Name: grade, dtype: float64
    evenness   lateness   earlyness
0   0.405615   0.152582    0.847418
1   0.688707   0.182219    0.817781
2   0.581888   0.169271    0.830729
3   0.612780   0.066335    0.933665
4   0.618691   0.130261    0.869739
[4. 4. 4. 4.]
Testing score: -0.45251396648044695
1     3.2
2     4.3
3     4.1
4     4.3
5     3.7
Name: grade, dtype: float64
    evenness   lateness   earlyness   even_contribution
1   0.152582   0.847418    0.847418            0.733538
2   0.182219   0.817781    0.817781            0.839928
3   0.169271   0.830729    0.830729            0.897677
4   0.066335   0.933665    0.933665            0.832736
5   0.130261   0.869739    0.869739            0.990589
[4.1  4.4  4.26 4.4 ]
Testing score: -0.10288770053476037
```

2nd execution of task 5 and task 6

```
0    3.2
1    4.3
2    4.1
3    4.3
4    3.7
Name: grade, dtype: float64
   evenness  lateness  earlyness
0  0.405615  0.152582   0.847418
1  0.688707  0.182219   0.817781
2  0.581888  0.169271   0.830729
3  0.612780  0.066335   0.933665
4  0.618691  0.130261   0.869739
[3.975      3.975      4.06       4.36666667]
Testing score: -1.6865694444444435
1    3.2
2    4.3
3    4.1
4    4.3
5    3.7
Name: grade, dtype: float64
   evenness  lateness  earlyness  even_contribution
1  0.152582  0.847418   0.847418           0.733538
2  0.182219  0.817781   0.817781           0.839928
3  0.169271  0.830729   0.830729           0.897677
4  0.066335  0.933665   0.933665           0.832736
5  0.130261  0.869739   0.869739           0.990589
[4.15 3.6  4.15 3.9 ]
Testing score: 0.3419354838709662
```

3rd execution of task 5 and task 6

```
0    3.2
1    4.3
2    4.1
3    4.3
4    3.7
Name: grade, dtype: float64
   evenness  lateness  earlyness
0  0.405615  0.152582   0.847418
1  0.688707  0.182219   0.817781
2  0.581888  0.169271   0.830729
3  0.612780  0.066335   0.933665
4  0.618691  0.130261   0.869739
[4.21111111 4.21111111 4.6        3.98333333]
Testing score: -0.33416949486006975
1    3.2
2    4.3
3    4.1
4    4.3
5    3.7
Name: grade, dtype: float64
   evenness  lateness  earlyness  even_contribution
1  0.152582  0.847418   0.847418           0.733538
2  0.182219  0.817781   0.817781           0.839928
3  0.169271  0.830729   0.830729           0.897677
4  0.066335  0.933665   0.933665           0.832736
5  0.130261  0.869739   0.869739           0.990589
[4.14166667 3.4        3.5        4.14166667]
Testing score: 0.6721897335932424
```

Therefore ,it seems that by running the methods described in task 5 and task 6 ,usually the testing score of the tree is better in the task in which more attributes are used to make the prediction of the grade.

Moreover, in the experiments done to create tree it is found that the testing score sometimes is negative and usually when positive it seems to be less than 1.

This happens due to the testing score is performed using the coefficient of determination R^2.

By the definition, this number R^2 =1-U/V where U and V are usually positive numbers .

Basically ,U is the squared sum residual of your prediction, and V is the total square sum(sample sum of square).

The coefficient of determination results negative if U is greater than V.

As a result, R^2 compares the fit of the chosen model with that of a horizontal straight line (the null hypothesis). If the chosen model fits worse than a horizontal line, then R^2 is negative.

In conclusion ,it seems that usually having more attributes for prediction improves the performance of the tree .