Abbreviation

KDD **K**nowledge **D**iscovery in **D**atabase

… …

List of Figures

Figure 17: Dropping currently unneeded attributes 7

Figure 18: Simplifying “Parental level of education” attribute 7

Figure 19: Dummifying values and dropping their first row 8

Figure 20: Mapping “Ethnicity” attribute values with numbers 8

Figure 21: Splitting up the dataset 8

Figure 22: Constructing Decision Tree with training set 9

Figure 23: Student Tier Decision Tree 10

Figure 24: Testing the Decision Tree with testing set 10

Figure 25: Checking the Decision Tree’s Accuracy 10

List of Tables

Table 1: A comparison between X and Y 10

Table of Contents

I…. 3

II… 4

III…. 5

*(Removing this italic red paragraph below after you read it)*

*Student please use font Times New Roman, size 13, Justify Text Alignment Mode for the report content.*

*If you insert the code into the report, please keep its original format with font, size and colors.*

*Please insert only the code useful for the report in explanation or clarification, and limit lengthening the report by pasting unnecessary code. You can insert the path of the code files, and I will search it in your attached project code to check them.*

**You will improve the below sections time by time until the final submission.**

1. **INTRODUCTION**

*[In this section, students please insert the detailed description of your project with its objectives and data mining purposes. You can give the samples to make it clearer.]*

1. **DATA UNDERSTANDING**
2. **Dataset**

*[*

* *Stored in the flat file .csv or in the relational database MySQL, …*
* *Dimensional description: number of attributes, number of objects.*
* *If there is any hierarchical structure in any attribute value, that is, the value is a structure, not a single value, even you need another dataset to store its values.*
* *If there is extra storage for media files such as image, video, audio, etc.*
* *Dataset size (KB, MB,…)*
* *Dataset type*
* *…. (students insert more details in here)…*

*]*

1. **Attribute**

*[Make a table to list the attributes and their attribute type, description, example, operations, … (see the lecture 2)*

*Which attributes have flaws like empty, missing, …. Make a discussion on these attributes if they are important or have important meaning to the data mining because it affect to the result?*

*……… students fill more with your own ideas at here…..*

*]*

1. **Objects**

*[*

* *How many objects do you have in the dataset?*
* *…*

*]*

1. **DATA CLEANING**

*[*

* *Analyze and do the statistics of dataset on missing values, duplicate, redundant, irrelevant data in your dataset.*
* *How you remove them, modified them (you can add code here)*
* *Why you keep them*
* *…*

*]*

1. **DATA PREPROCESSING**

*[*

* *Reduce the objects, the attributes.*
* *Discuss on the size, attributes and the objects of the subset and*
* *How do you do it (you can add code here)*
* *…*

*]*

***--- Students please think a bit and add more contents into the below sections. The red lines are just some hints for you to understand what that sections are about. So, you need to add more your own ideas.***

***--- You can change the position of the hints between the sections if you think their new positions are more rational.***

***--- Please add more your own ideas on the report contents that help you gain higher score.***

***--- IN THIS REQUEST, you can start with 1 algorithm first, the other two can be done in the next REQUEST (request 4)***

1. **MODEL BUILDING**

In this section, we will be using some algorithms for detecting pattern, relationship between attributes in our dataset.

Before we start building our model, we will need to simplify our preprocessed dataset a bit more. First, we will drop some attribute that we will not need for now.

***Please be noted that these attributes will be used in another section so we are not really getting rid of it.***

attribute\_To\_Drop = ["math score", "reading score", "writing score", "Extra math study time", "Extra reading study time", "Extra writing study time", "math\_guess\_min", "math\_guess\_max", "reading\_guess\_min", "reading\_guess\_max", "writing\_guess\_min", "writing\_guess\_max"]

data = dataSet.drop(attribute\_To\_Drop, 1)

***Figure 17: Dropping currently unneeded attributes***

Second, we will simplify the attribute “Parental level of education” into 2 values:

* Higher education which is consisted of:
* Associate’s degree
* College
* Master’s degree
* Bachelor’s degree
* High school

data["parental level of education"] = data["parental level of education"].replace(["bachelor's degree", "college", "master's degree", "associate's degree"], "higher education")

***Figure 18: Simplifying “Parental level of education” attribute***

We are doing this so that the table will only have binary value of 0 and 1.

Third, we will dummify the following attributes to 0 and 1:

* Math tier
* Writing tier
* Reading tier
* Parental level of education
* Lunch
* Test preparation course
* Total tier

newSet = pd.get\_dummies(data, columns=["Math tier", "Reading tier", "Writing tier", "parental level of education", "lunch", "test preparation course"],

prefix=["MathTier", "ReadTier", "WriteTier", "ParentEdu", "Lunch", "TestPrep"], drop\_first=True)

***Figure 19: Dummifying values and dropping their first row***

Finally, we will use numbers to represent the “Ethnic” attribute values because the algorithm we are using is unable to read string. We will map the values like below:

* Group A = 0
* Group B = 1
* Group C = 2
* Group D = 3
* Group E = 4

    temp = {"group A": 0, "group B": 1, "group C": 2, "group D": 3, "group E": 4}

    newSet["Ethnicity"] = newSet["Ethnicity"].map(temp)

***Figure 20: Mapping “Ethnicity” attribute values with numbers***

After that, we will split our dataset into 2 parts, training set and test set. The training set will take 80% of the dataset, while the test set take 20%.

train, test = train\_test\_split(newSet, test\_size=0.2, shuffle=False)

***Figure 21: Splitting up the dataset***

Now that our data has been simplified even more, we can now start the model building process.

Our first algorithm will be the Decision Tree Algorithm that we will use to classify the students who passed or failed their exam.

Our decision tree will be based on these attributes:

* Math tier.
* Reading tier.
* Writing tier.

We will be using the “Total tier” attribute as our class for classifying the data. First, we will build the tree with the training set and the sklearn library.

Y = trainSet["Total tier"]

    X = trainSet.drop("Total tier", axis=1)

    clf = tree.DecisionTreeClassifier()

    clf = clf.fit(X, Y)

    dot\_data = tree.export\_graphviz(clf, feature\_names=X.columns, class\_names=["Fail", "Pass"], filled=True, out\_file=None)

    graph = pydotplus.graph\_from\_dot\_data(dot\_data)

    graph.write\_png("StudentTreeTrain.png")

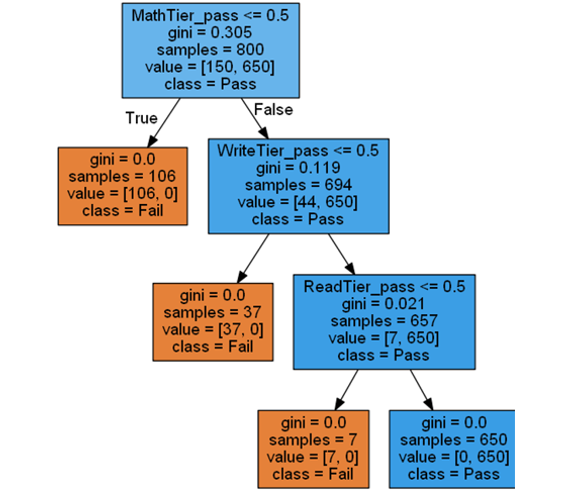
    img = mpimg.imread('StudentTreeTrain.png')

    imgplot = plt.imshow(img)

    plt.show()

***Figure 22: Constructing Decision Tree with training set***

The code in ***figure 22*** will give us the decision tree like in the figure below.

***Figure 23:***** ***Student Tier Decision Tree***

Next, we will test the tree with the testing set to see how well it performed.

 testY = testSet["Total tier"]

 testX = testSet.drop("Total tier", axis=1)

 predY = clf.predict(testX)

 testX = testX.reset\_index()

 prediction = pd.concat([testX['ID'], pd.Series(predY, name="Predicted Tier")], axis=1)

 print("Prediction:\n", prediction)

***Figure 24: Testing the Decision Tree with testing set***

The resulted prediction will then be passed to an accuracy test to see its accuracy.

print("Accuracy of test data: %.2f" % (accuracy\_score(testY, predY)))

***Figure 25: Checking the Decision Tree’s Accuracy***

Running the accuracy test code in ***figure 25*** will give us the value of 1.0 so the Decision Tree is accurate.

**The second algorithms we use coverage data of all students**

We will used enum to set class name easier

Students are divided into 2 groups: Pass and Fail

* Tails explanation:
* L : Standard lunch
* P : Parents have higher education
* PL : Standard lunch, Parents have higher education
* Pass:
* Ace : No Test preparation, Free/reduced lunch, Highschool education Parents
* Good : Completed Test preparation, Free/reduced lunch, Highschool education Parents

class PassGroup(enum.Enum):  
 Ace = 1  
 Good = 2  
 Ace\_L = 3  
 Good\_L = 4  
 Ace\_P = 5  
 Good\_P = 6  
 Ace\_PL = 7  
 Good\_PL = 8

***Figure 26: Pass Group classification***

* Fail:
* Awful : No Test preparation, Free/reduced lunch, Highschool education Parents
* Bad : Completed Test preparation, Free/reduced lunch, Highschool education Parents

class FailGroup(enum.Enum):  
 Awful = 1  
 Bad = 2  
 Awful\_L = 3  
 Bad\_L = 4  
 Awful\_P = 5  
 Bad\_P = 6  
 Awful\_PL = 7  
 Bad\_PL = 8

***Figure 27: Fail group classification***

By taking training 80%, testing 20%, we select 800 in 1000 objects for training data

* **Training**

PassStudent = (temp["Total tier"] == "pass")  
FailStudent = (temp["Total tier"] == "fail")

***Figure 28: Pass and Fail Student training simplication***

if PassStudent and temp["ParentEdu\_higher education"] == 0 and temp["Lunch\_standard"] == 0 and \  
 temp["TestPrep\_none"] == 0:  
 studentClass.append(PassGroup.Good)  
elif PassStudent and temp["ParentEdu\_higher education"] == 0 and temp[  
 "Lunch\_standard"] == 0 and temp["TestPrep\_none"] == 1:  
 studentClass.append(PassGroup.Ace)  
elif PassStudent and temp["ParentEdu\_higher education"] == 0 and temp[  
 "Lunch\_standard"] == 1 and temp["TestPrep\_none"] == 0:  
 studentClass.append(PassGroup.Good\_L)  
elif PassStudent and temp["ParentEdu\_higher education"] == 0 and temp[  
 "Lunch\_standard"] == 1 and temp["TestPrep\_none"] == 1:  
 studentClass.append(PassGroup.Ace\_L)  
elif PassStudent and temp["ParentEdu\_higher education"] == 1 and temp[  
 "Lunch\_standard"] == 0 and temp["TestPrep\_none"] == 0:  
 studentClass.append(PassGroup.Good\_P)  
elif PassStudent and temp["ParentEdu\_higher education"] == 1 and temp[  
 "Lunch\_standard"] == 0 and temp["TestPrep\_none"] == 1:  
 studentClass.append(PassGroup.Ace\_P)  
elif PassStudent and temp["ParentEdu\_higher education"] == 1 and temp[  
 "Lunch\_standard"] == 1 and temp["TestPrep\_none"] == 0:  
 studentClass.append(PassGroup.Good\_PL)  
elif PassStudent and temp["ParentEdu\_higher education"] == 1 and temp[  
 "Lunch\_standard"] == 1 and temp["TestPrep\_none"] == 1:  
 studentClass.append(PassGroup.Ace\_PL)  
  
  
# Classifying students that failed the exam  
elif FailStudent and temp["ParentEdu\_higher education"] == 0 and temp[  
 "Lunch\_standard"] == 0 and temp["TestPrep\_none"] == 0:  
 studentClass.append(FailGroup.Bad)  
elif FailStudent and temp["ParentEdu\_higher education"] == 0 and temp[  
 "Lunch\_standard"] == 0 and temp["TestPrep\_none"] == 1:  
 studentClass.append(FailGroup.Awful)  
elif FailStudent and temp["ParentEdu\_higher education"] == 0 and temp[  
 "Lunch\_standard"] == 1 and temp["TestPrep\_none"] == 0:  
 studentClass.append(FailGroup.Bad\_L)  
elif FailStudent and temp["ParentEdu\_higher education"] == 0 and temp[  
 "Lunch\_standard"] == 1 and temp["TestPrep\_none"] == 1:  
 studentClass.append(FailGroup.Awful\_L)  
elif FailStudent and temp["ParentEdu\_higher education"] == 1 and temp[  
 "Lunch\_standard"] == 0 and temp["TestPrep\_none"] == 0:  
 studentClass.append(FailGroup.Bad\_P)  
elif FailStudent and temp["ParentEdu\_higher education"] == 1 and temp[  
 "Lunch\_standard"] == 0 and temp["TestPrep\_none"] == 1:  
 studentClass.append(FailGroup.Awful\_P)  
elif FailStudent and temp["ParentEdu\_higher education"] == 1 and temp[  
 "Lunch\_standard"] == 1 and temp["TestPrep\_none"] == 0:  
 studentClass.append(FailGroup.Bad\_PL)  
elif FailStudent and temp["ParentEdu\_higher education"] == 1 and temp[  
 "Lunch\_standard"] == 1 and temp["TestPrep\_none"] == 1:  
 studentClass.append(FailGroup.Awful\_PL)  
else:  
 studentClass.append("Error")

***Figure 29: Pass and Fail students training classification***

* **Testing**

PassStudent = (temp["Total tier"] == "pass")  
FailStudent = (temp["Total tier"] == "fail")

***Figure 30: Pass and Fail Student testing simplication***

if PassStudent and temptest["ParentEdu\_higher education"] == 0 and temptest["Lunch\_standard"] == 0 and \  
 temptest["TestPrep\_none"] == 0:  
 studentClasstest.append(PassGroup.Good)  
elif PassStudent and temptest["ParentEdu\_higher education"] == 0 and temptest[  
 "Lunch\_standard"] == 0 and temptest["TestPrep\_none"] == 1:  
 studentClasstest.append(PassGroup.Ace)  
elif PassStudent and temptest["ParentEdu\_higher education"] == 0 and temptest[  
 "Lunch\_standard"] == 1 and temptest["TestPrep\_none"] == 0:  
 studentClasstest.append(PassGroup.Good\_L)  
elif PassStudent and temptest["ParentEdu\_higher education"] == 0 and temptest[  
 "Lunch\_standard"] == 1 and temptest["TestPrep\_none"] == 1:  
 studentClasstest.append(PassGroup.Ace\_L)  
elif PassStudent and temptest["ParentEdu\_higher education"] == 1 and temptest[  
 "Lunch\_standard"] == 0 and temptest["TestPrep\_none"] == 0:  
 studentClasstest.append(PassGroup.Good\_P)  
elif PassStudent and temptest["ParentEdu\_higher education"] == 1 and temptest[  
 "Lunch\_standard"] == 0 and temptest["TestPrep\_none"] == 1:  
 studentClasstest.append(PassGroup.Ace\_P)  
elif PassStudent and temptest["ParentEdu\_higher education"] == 1 and temptest[  
 "Lunch\_standard"] == 1 and temptest["TestPrep\_none"] == 0:  
 studentClasstest.append(PassGroup.Good\_PL)  
elif PassStudent and temptest["ParentEdu\_higher education"] == 1 and temptest[  
 "Lunch\_standard"] == 1 and temptest["TestPrep\_none"] == 1:  
 studentClasstest.append(PassGroup.Ace\_PL)  
  
  
# Classifying students that failed the exam  
elif FailStudent and temptest["ParentEdu\_higher education"] == 0 and temptest[  
 "Lunch\_standard"] == 0 and temptest["TestPrep\_none"] == 0:  
 studentClasstest.append(FailGroup.Bad)  
elif FailStudent and temptest["ParentEdu\_higher education"] == 0 and temptest[  
 "Lunch\_standard"] == 0 and temptest["TestPrep\_none"] == 1:  
 studentClasstest.append(FailGroup.Awful)  
elif FailStudent and temptest["ParentEdu\_higher education"] == 0 and temptest[  
 "Lunch\_standard"] == 1 and temptest["TestPrep\_none"] == 0:  
 studentClasstest.append(FailGroup.Bad\_L)  
elif FailStudent and temptest["ParentEdu\_higher education"] == 0 and temptest[  
 "Lunch\_standard"] == 1 and temptest["TestPrep\_none"] == 1:  
 studentClasstest.append(FailGroup.Awful\_L)  
elif FailStudent and temptest["ParentEdu\_higher education"] == 1 and temptest[  
 "Lunch\_standard"] == 0 and temptest["TestPrep\_none"] == 0:  
 studentClasstest.append(FailGroup.Bad\_P)  
elif FailStudent and temptest["ParentEdu\_higher education"] == 1 and temptest[  
 "Lunch\_standard"] == 0 and temptest["TestPrep\_none"] == 1:  
 studentClasstest.append(FailGroup.Awful\_P)  
elif FailStudent and temptest["ParentEdu\_higher education"] == 1 and temptest[  
 "Lunch\_standard"] == 1 and temptest["TestPrep\_none"] == 0:  
 studentClasstest.append(FailGroup.Bad\_PL)  
elif FailStudent and temptest["ParentEdu\_higher education"] == 1 and temptest[  
 "Lunch\_standard"] == 1 and temptest["TestPrep\_none"] == 1:  
 studentClasstest.append(FailGroup.Awful\_PL)  
else:  
 studentClasstest.append("Error")

***Figure 31: Pass and Fail students testing classification***

Algorithms 3:

We will used decision tree to predict the time for extra study and guessing the result in the future of 3 subjects: Math, Reading and Writing.

In order to test the accuracy of decision tree, we seperate the data of 1000 objects into 80% in training and 20% in testing.

The percentage of time to study extra:

By calculation : the percentage of time to study extra is calculated by

Predict subject = 100 – (subject score -49)\*(100/22) %

Math:

trainmathtimeY = trainSet["Extra math study time"]  
trainmathtimeX = trainSet.drop("Extra math study time", axis=1)  
clfmathtime = tree.DecisionTreeClassifier()  
clfmathtime = clfmathtime.fit(trainmathtimeX, trainmathtimeY)  
testmathtimeY = testSet["Extra math study time"]  
testmathtimeX = testSet.drop("Extra math study time", axis=1)  
predmathtimeY = clfmathtime.predict(testmathtimeX)

Reading:

trainreadingtimeY = trainSet["Extra reading study time"]  
trainreadingtimeX = trainSet.drop("Extra reading study time", axis=1)  
clfreadingtime = tree.DecisionTreeClassifier()  
clfreadingtime = clfreadingtime.fit(trainreadingtimeX, trainreadingtimeY)  
testreadingtimeY = testSet["Extra reading study time"]  
testreadingtimeX = testSet.drop("Extra reading study time", axis=1)  
predreadingtimeY = clfreadingtime.predict(testreadingtimeX)

Writing:

trainwritingtimeY = trainSet["Extra writing study time"]  
trainwritingtimeX = trainSet.drop("Extra writing study time", axis=1)  
clfwritingtime = tree.DecisionTreeClassifier()  
clfwritingtime = clfwritingtime.fit(trainwritingtimeX, trainwritingtimeY)  
testwritingtimeY = testSet["Extra writing study time"]  
testwritingtimeX = testSet.drop("Extra writing study time", axis=1)  
predwritingtimeY = clfwritingtime.predict(testwritingtimeX)

Guessing the result based on the current result:

We guess the result in range from min to max

Min : Min score = subject’s current score – 10

Max: Max score = subject’s current score + 20 + 0.05\* predict subject

Math:

Min:

trainmathminY = trainSet["math\_guess\_min"]  
trainmathminX = trainSet.drop("math\_guess\_min", axis=1)  
clfmathmin = tree.DecisionTreeClassifier()  
clfmathmin = clfmathmin.fit(trainmathminX, trainmathminY)  
testmathminY = testSet["math\_guess\_min"]  
testmathminX = testSet.drop("math\_guess\_min", axis=1)  
predmathminY = clfmathmin.predict(testmathminX)  
testmathminX = testmathminX.reset\_index()

Max:

trainmathmaxY = trainSet["math\_guess\_max"]  
trainmathmaxX = trainSet.drop("math\_guess\_max", axis=1)  
clfmathmax = tree.DecisionTreeClassifier()  
clfmathmax = clfmathmax.fit(trainmathmaxX, trainmathmaxY)  
testmathmaxY = testSet["math\_guess\_max"]  
testmathmaxX = testSet.drop("math\_guess\_max", axis=1)  
predmathmaxY = clfmathmax.predict(testmathmaxX)

Reading:

Min:

trainreadingminY = trainSet["reading\_guess\_min"]  
trainreadingminX = trainSet.drop("reading\_guess\_min", axis=1)  
clfreadingmin = tree.DecisionTreeClassifier()  
clfreadingmin = clfreadingmin.fit(trainreadingminX, trainreadingminY)  
testreadingminY = testSet["reading\_guess\_min"]  
testreadingminX = testSet.drop("reading\_guess\_min", axis=1)  
predreadingminY = clfreadingmin.predict(testreadingminX)

Max:

trainreadingmaxY = trainSet["reading\_guess\_max"]  
trainreadingmaxX = trainSet.drop("reading\_guess\_max", axis=1)  
clfreadingmax = tree.DecisionTreeClassifier()  
clfreadingmax = clfreadingmax.fit(trainreadingmaxX, trainreadingmaxY)  
testreadingmaxY = testSet["reading\_guess\_max"]  
testreadingmaxX = testSet.drop("reading\_guess\_max", axis=1)  
predreadingmaxY = clfreadingmax.predict(testreadingmaxX)

Writing

Min:

trainwritingminY = trainSet["writing\_guess\_min"]  
trainwritingminX = trainSet.drop("writing\_guess\_min", axis=1)  
clfwritingmin = tree.DecisionTreeClassifier()  
clfwritingmin = clfwritingmin.fit(trainwritingminX, trainwritingminY)  
testwritingminY = testSet["writing\_guess\_min"]  
testwritingminX = testSet.drop("writing\_guess\_min", axis=1)  
predwritingminY = clfwritingmin.predict(testwritingminX)

Max:

trainwritingmaxY = trainSet["writing\_guess\_max"]  
trainwritingmaxX = trainSet.drop("writing\_guess\_max", axis=1)  
clfwritingmax = tree.DecisionTreeClassifier()  
clfwritingmax = clfwritingmax.fit(trainwritingmaxX, trainwritingmaxY)  
testwritingmaxY = testSet["writing\_guess\_max"]  
testwritingmaxX = testSet.drop("writing\_guess\_max", axis=1)  
predwritingmaxY = clfwritingmax.predict(testwritingmaxX)

Physical resources:

* 1 PC
* CPU:
* Speed: 0.7 GHz
* Processes: 4
* Threads: 40
* 0.1-0.2 GB RAM

Our purpose is training the machine to test how it learning the data and then test the data to check the accuracy, this will help us create the machine to test the data instead of using basic algorithms code, our project in this moment run properly the algorithms 1 that training and testing the student total tier by training the result of math, reading and writing.

Our 2 next algorithms is classified the student based on external factors and guessing the result using decision (Our 2 next algorithms might be changed in the next report).

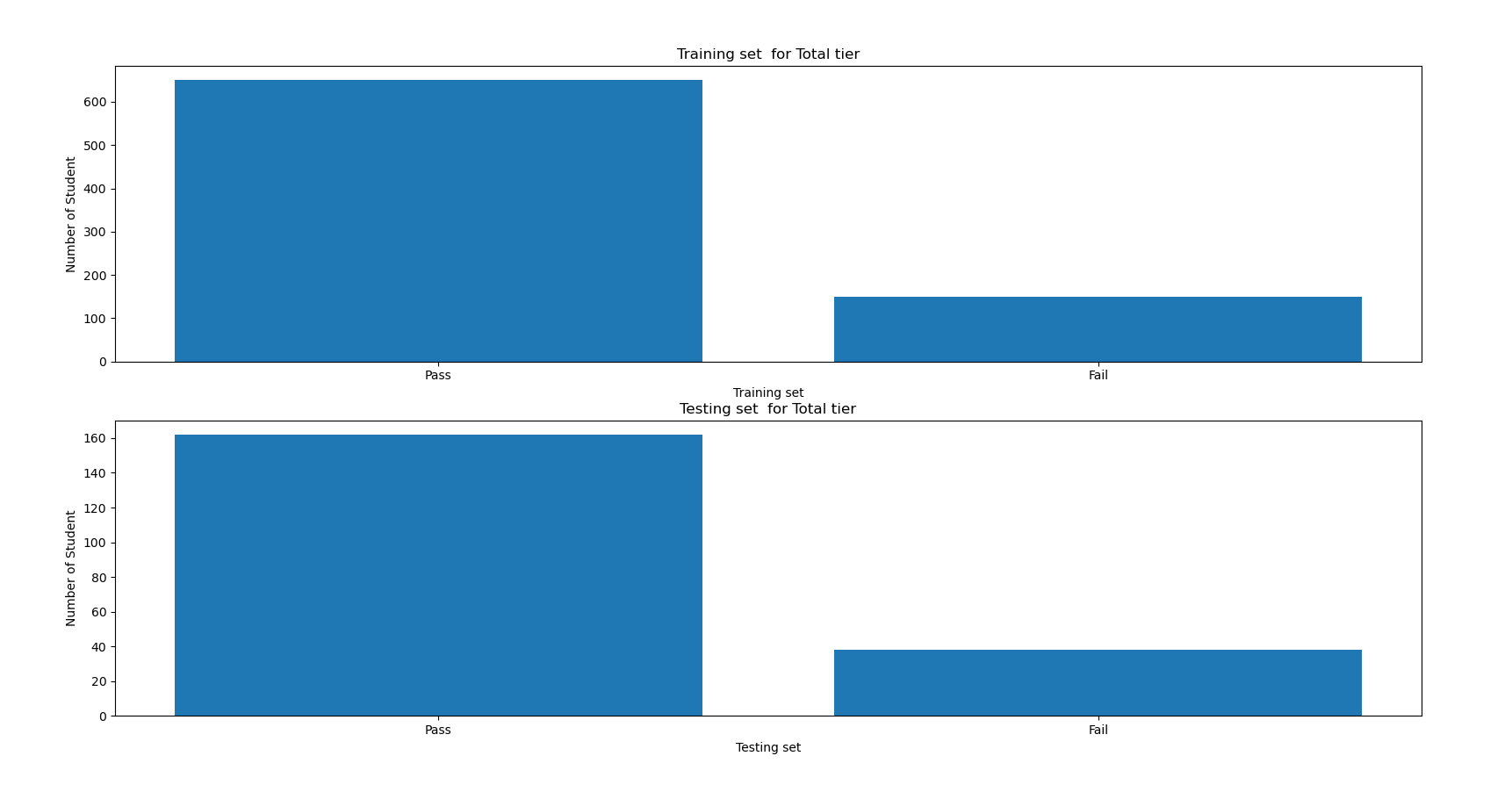
Algorithms 1:

Train: Pass: 650, Fail:150 (From student 1 to 800)

Test: Pass:162, Fail: 38 (From student 801 to 1000)

Prediction: Pass: 162, Fail:38 (From student 801 to 1000)

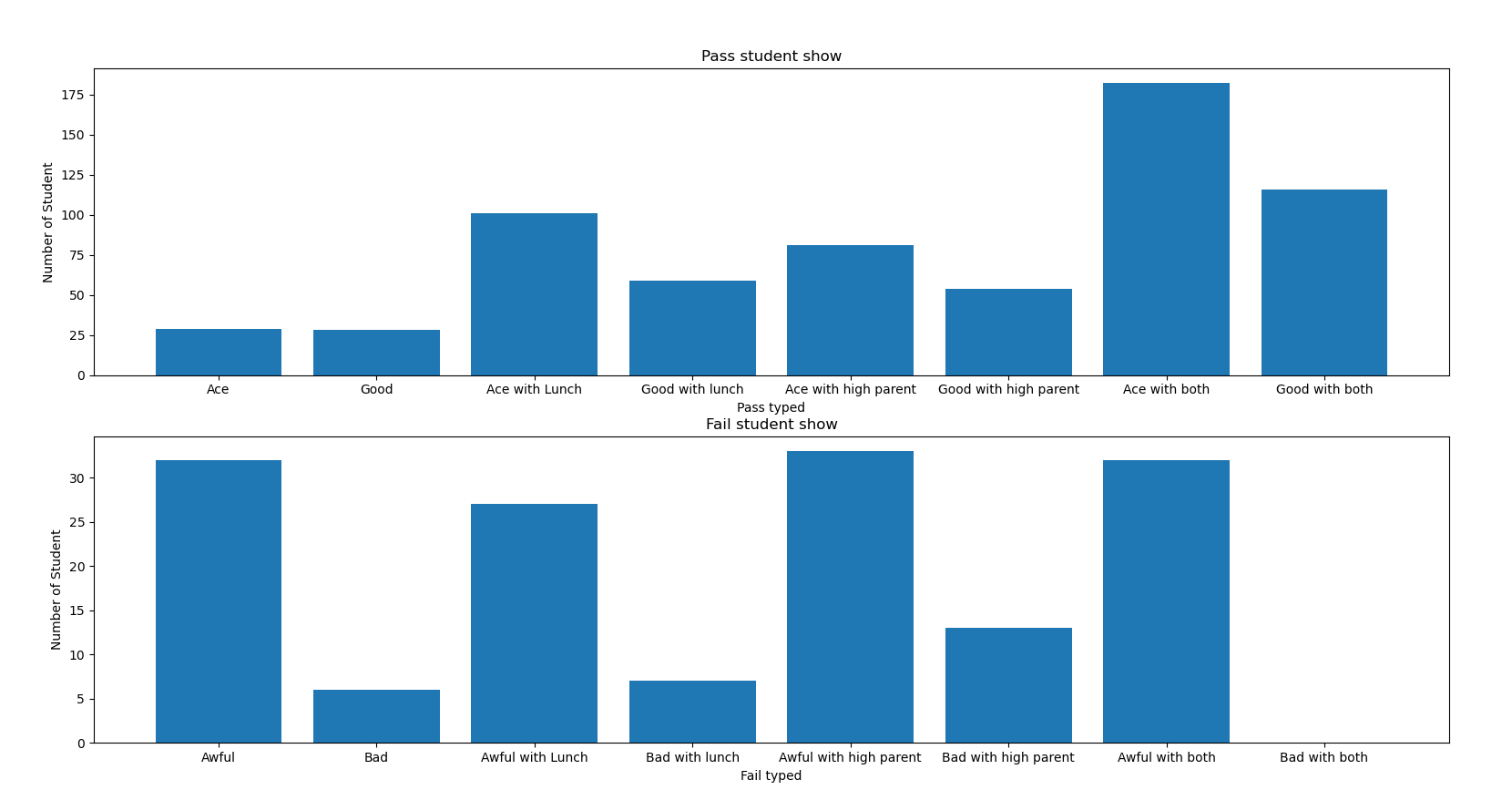
Accuracy : 100%



***Figure 32 : Number of pass and fail student in Training and Testing set***

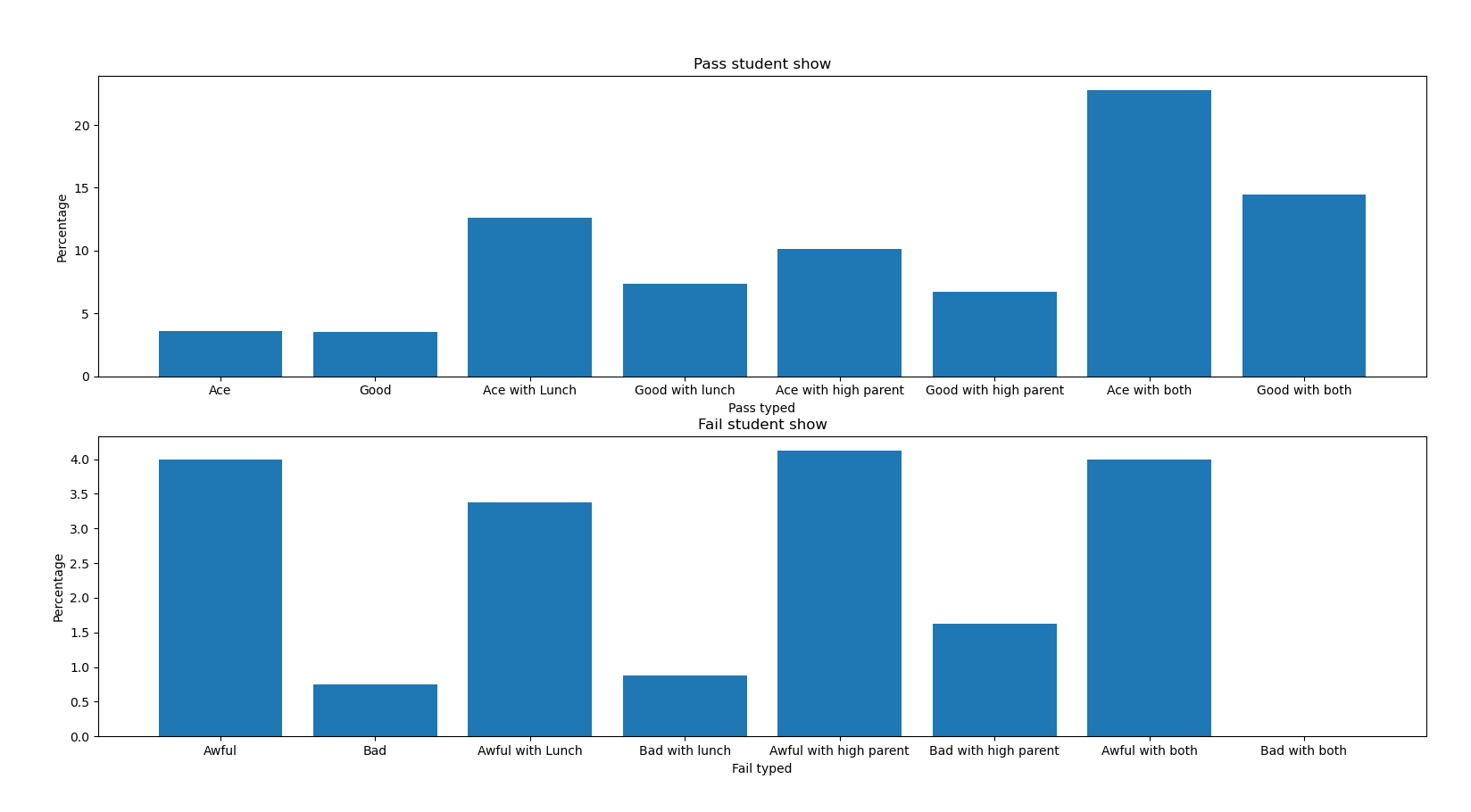
Algorithms 2:

* Training data:
  + Train index = 800
  + Number of Ace: 29
  + Number of Good: 28
  + Number of Ace with lunch: 101
  + Number of Good with lunch: 59
  + Number of Ace with high parent: 81
  + Number of Good with high parent: 54
  + Number of Ace with lunch and high parent: 182
  + Number of Good with lunch and high parent: 116
  + Number of Awful: 32
  + Number of Bad: 6
  + Number of Awful with lunch: 27
  + Number of Bad with lunch: 7
  + Number of awful with high parent: 33
  + Number of bad with high parent: 13
  + Number of awful with lunch and high parent: 32
  + Number of bad with lunch and high parent: 0



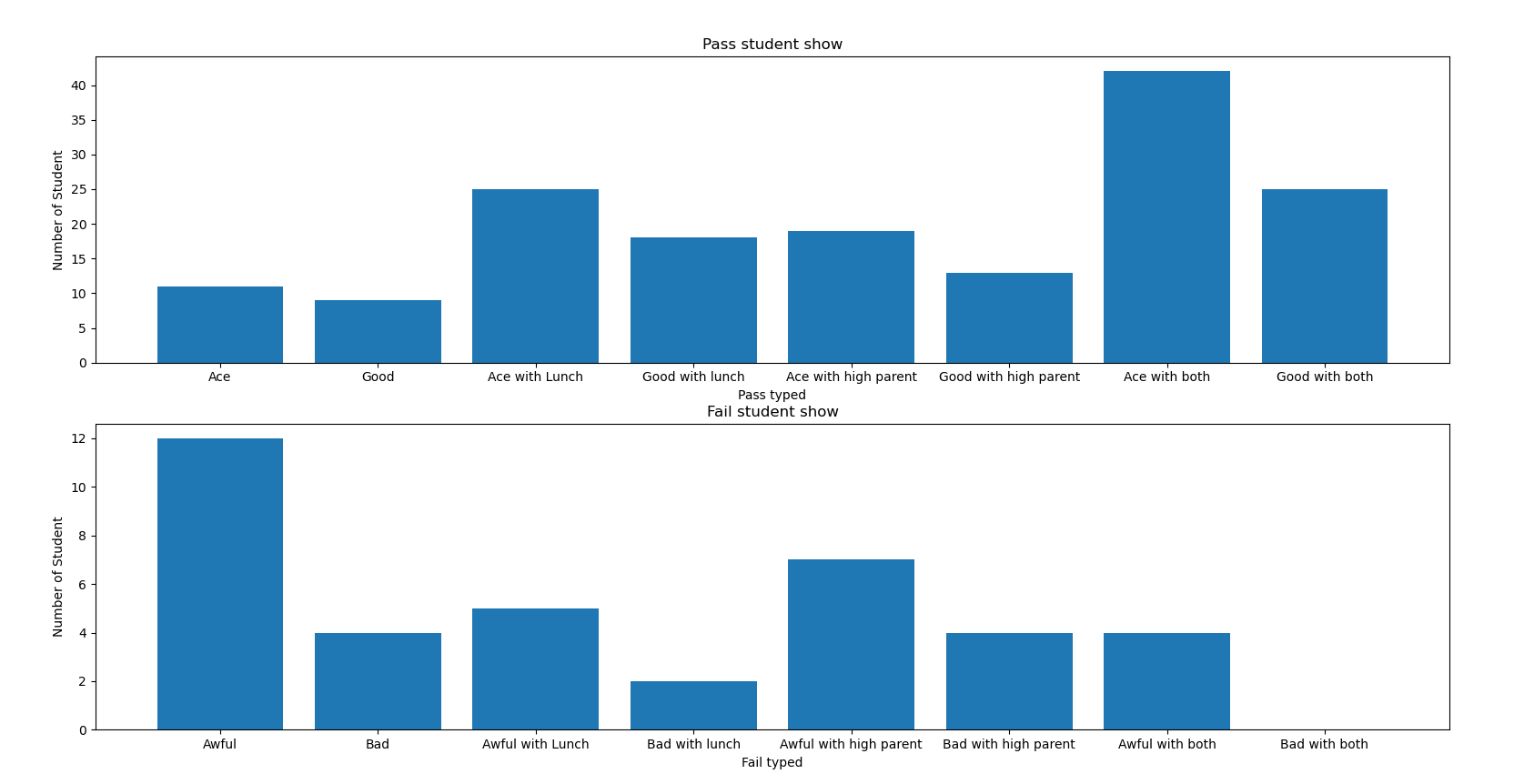
***Figure 33: Number of Student in classification in training data***

* Coverage of ace Student: 3.625 %
* Coverage of good Student: 3.5 %
* Coverage of ace with lunch Student: 12.625 %
* Coverage of good with lunch Student: 7.375 %
* Coverage of ace with high parent Student: 10.125 %
* Coverage of good with high parent Student: 6.75 %
* Coverage of ace with lunch and high parent Student: 22.75 %
* Coverage of good with lunch and high parent Student: 14.5 %
* Coverage of awful Student: 4.0 %
* Coverage of bad Student: 0.75 %
* Coverage of awful with lunch Student: 3.375 %
* Coverage of bad with lunch Student: 0.875 %
* Coverage of awful with high parent Student: 4.125 %
* Coverage of bad with high parent Student: 1.625 %
* Coverage of awful with lunch and high parent Student: 4.0 %
* Coverage of bad with lunch and high parent Student: 0.0 %

******

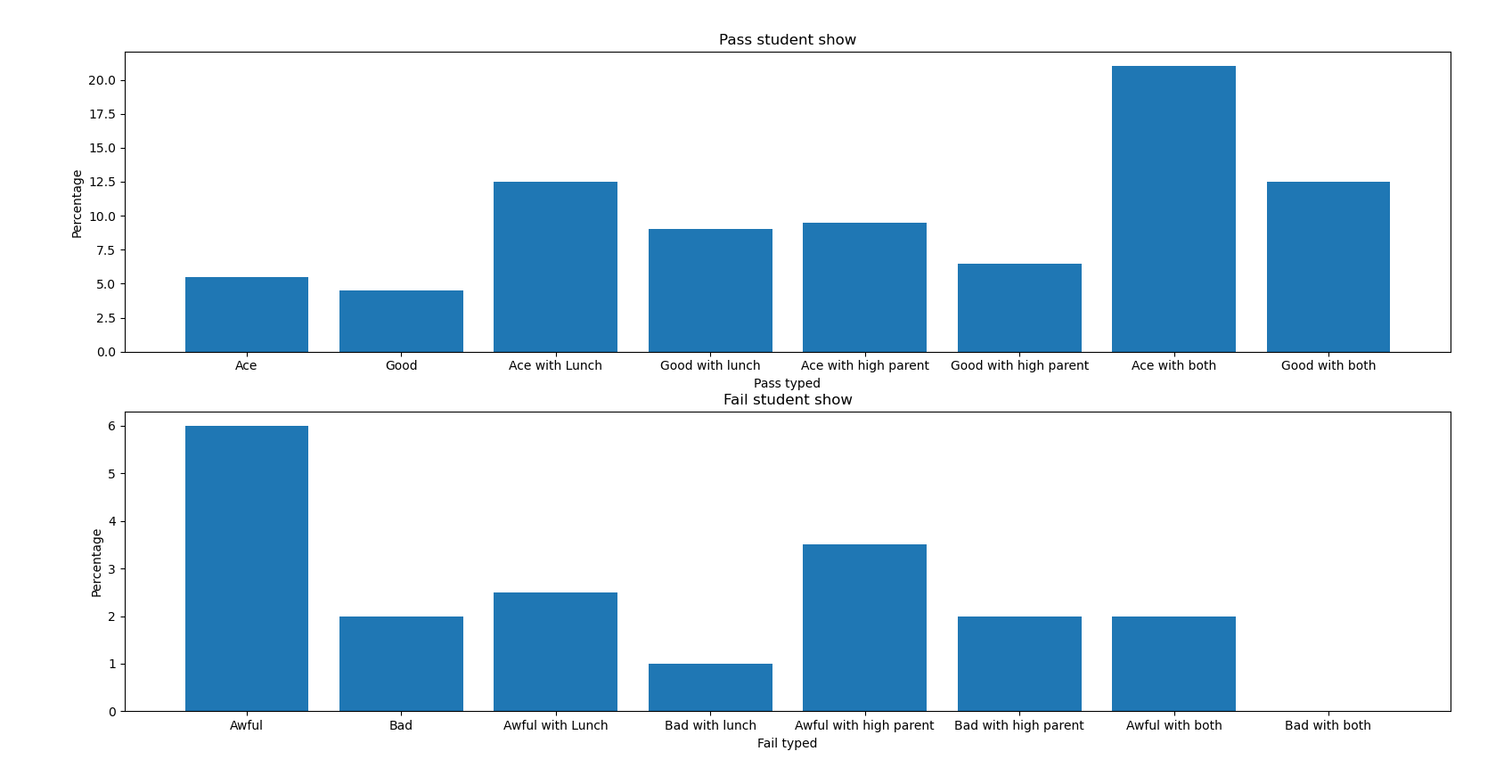
***Figure 34: Coverage percentage of student classification in training data***

* Testing data:
* Test index = 200
* Number of Ace: 11
* Number of Good: 9
* Number of Ace with lunch: 25
* Number of Good with lunch: 18
* Number of Ace with high parent: 19
* Number of Good with high parent: 13
* Number of Ace with lunch and high parent: 42
* Number of Good with lunch and high parent: 25
* Number of Awful: 12
* Number of Bad: 4
* Number of Awful with lunch: 5
* Number of Bad with lunch: 2
* Number of awful with high parent: 7
* Number of bad with high parent: 4
* Number of awful with lunch and high parent: 4
* Number of bad with lunch and high parent: 0



***Figure 35: Number of Student in classification in testing data***

* Coverage of ace Student: 5.5
* Coverage of good Student: 4.5
* Coverage of ace with lunch Student: 12.5
* Coverage of good with lunch Student: 9.0
* Coverage of ace with high parent Student: 9.5
* Coverage of good with high parent Student: 6.5
* Coverage of ace with lunch and high parent Student: 21.0
* Coverage of good with lunch and high parent Student: 12.5
* Coverage of awful Student: 6.0
* Coverage of bad Student: 2.0
* Coverage of awful with lunch Student: 2.5
* Coverage of bad with lunch Student: 1.0
* Coverage of awful with high parent Student: 3.5
* Coverage of bad with high parent Student: 2.0
* Coverage of awful with lunch and high parent Student: 2.0
* Coverage of bad with lunch and high parent Student: 0.0

**

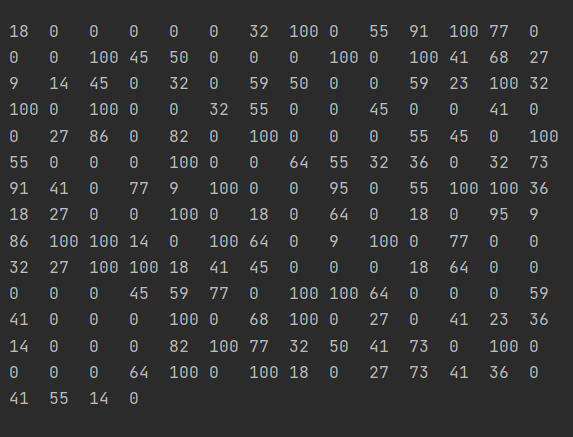
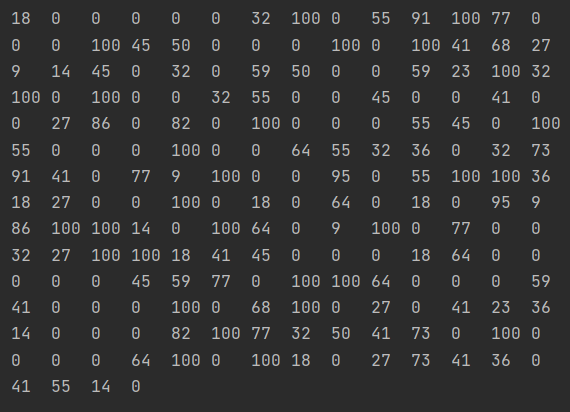
***Figure 36: Coverage percentage of student classification in testing data***

**Algorithm 3:**

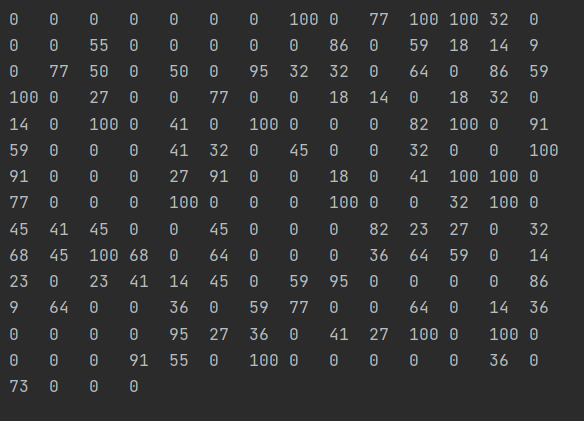
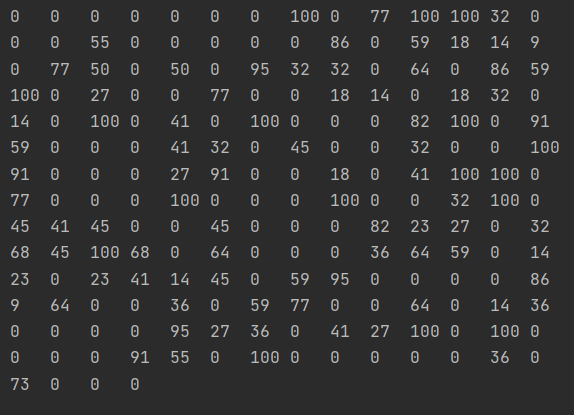
We collect 200 object data from 801 to 1000 to test the accuracy, all the result of test and prediction will convert into the matrix 15\*14

The percentage of time to study extra:

Math: Algorithm accuracy: 100% ( No error)

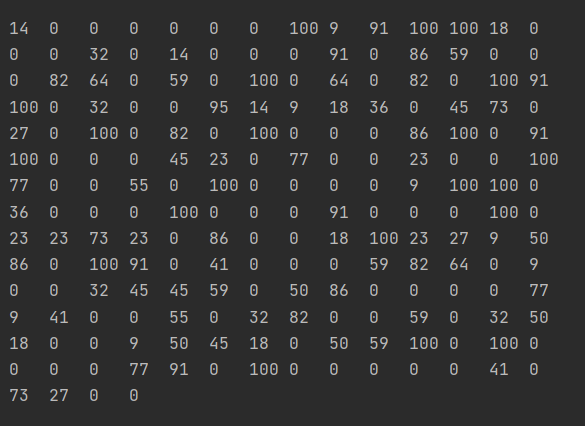
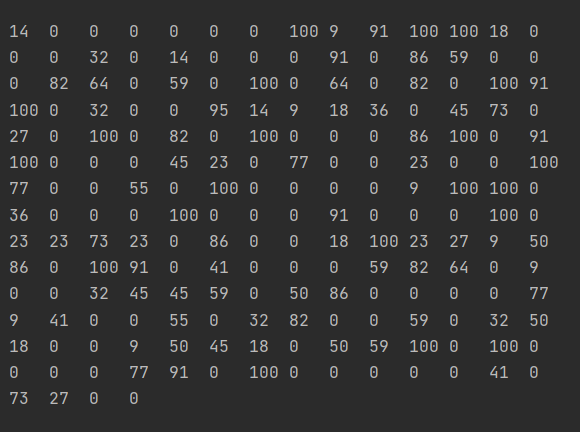
 Test Predict

Reading: Algorithm accuracy: 100% ( No error)

Test Predict

Writing: Algorithm accuracy: 100% ( No error)

Test Predict



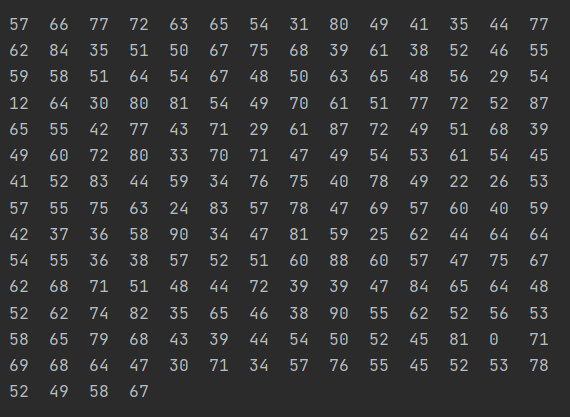
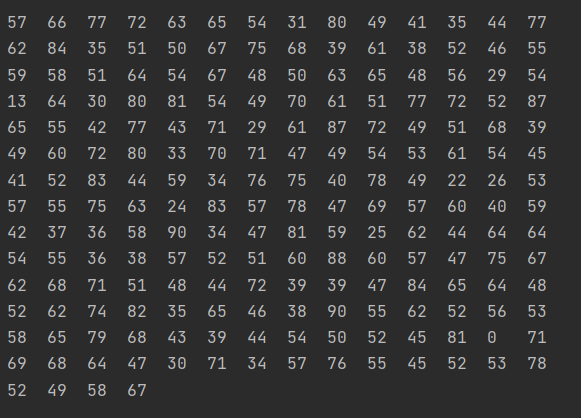
Guessing the result based on the current result:

We guess the result in range from min to max

Math:

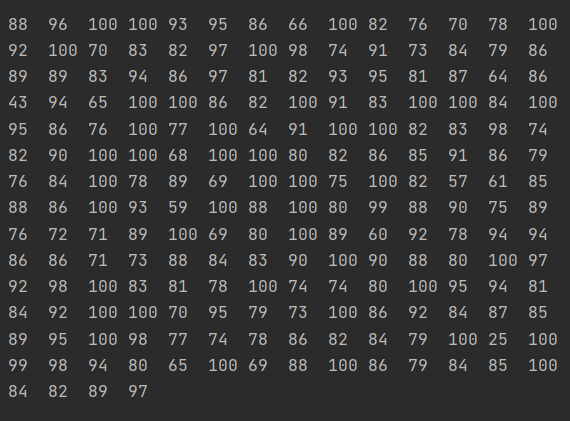
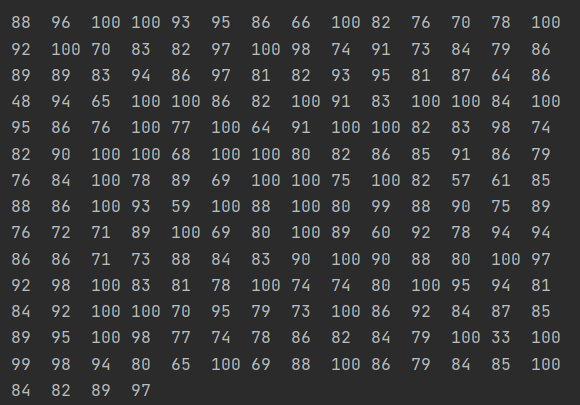
Min: Accuracy: 99.5% ( 1 error )

Test Predict



Max: 99% (2 errors)

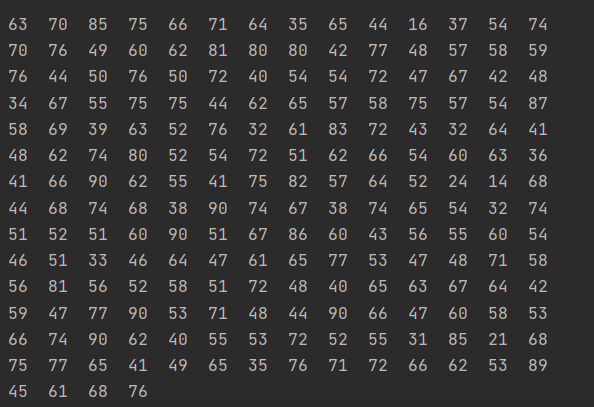
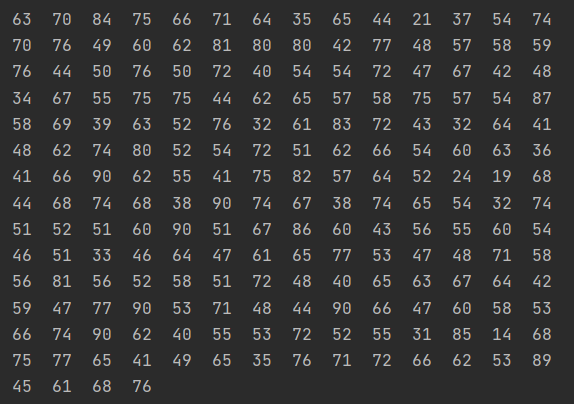
Test Predict



Reading:

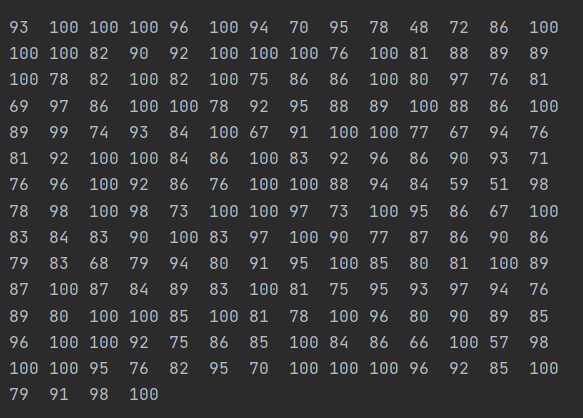
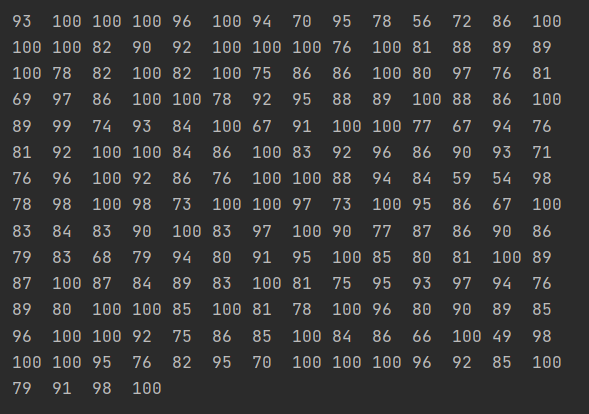
Min: Accuracy : 98% (4 errors )

Test Predict



Max: Accuracy: 98.5% (3 errors )

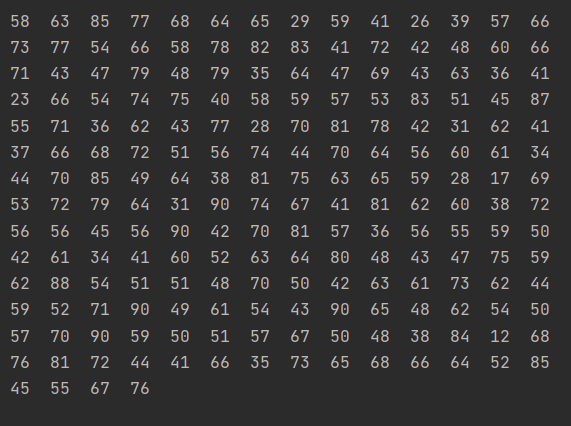
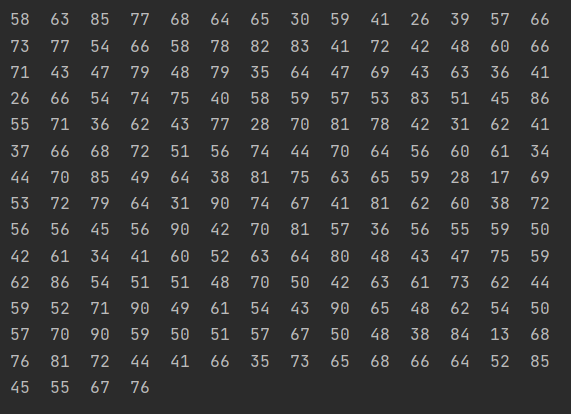
Test Predict



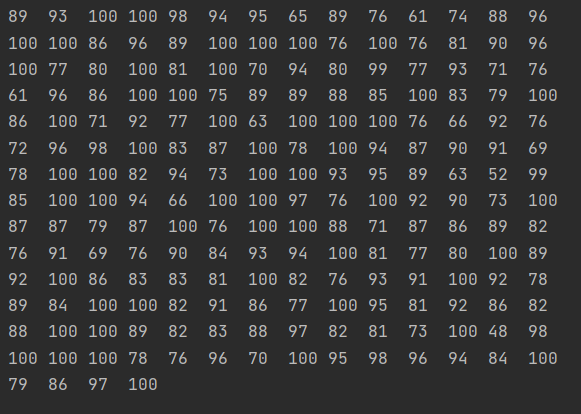
Writing:

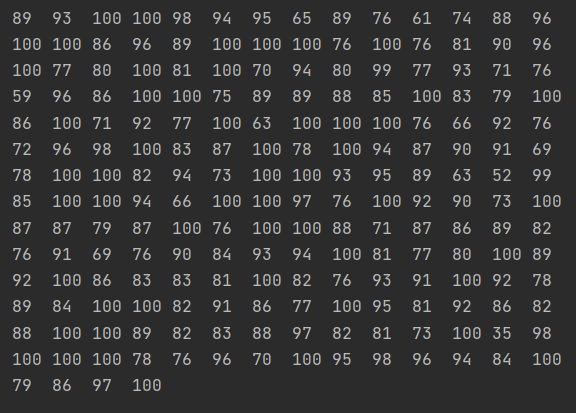
` Min: 97.5% (5 errors )

Test Predict



Max: 99% ( 2 errors )

Test Predict



* + *Accuracy visual data of 3 algorithms. Analyse why it is good or bad (what factor such as dataset, params, ….)*

*Other visual data analyse the data mining purpose (e.g., heart disease is effected by attributes X1, X2, …. More specificially, when X1, X2, … increases, the heart disease of these ages … are affected seriousely, etc…*

* + *Compare three of them using charts or tables.*

1. **CONCLUSION AND FUTURE WORKS**

- Pros:

+Easy to construct

+Quick run and response

+It can be used with a wide range of objects.

+Given a general assessment of the student classification, finding and giving time to low-scoring students, and guessing the point using simple math and simple algorithms,This is a common misunderstanding.

+The data can be displayed easily by bar chart and pie chart

- Cons:

+The data depends mostly on mathematic.

+Not diversity (the data focuses solely on classifying, guessing points, and assisting the weak student, and has yet to release some strategies to assist the student in improving as well as fostering the good and excellent students).

-Future works:

+Adding more requests to increase the diversity of data solutions

+Using the code to determine other data like VGU or some other schools, colleges, universities, etc.

+Given some specific direction for students based on students, create algorithms to foster the good and excellent students.

**DUTY ROSTER**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Task** | **In Charge** | **Start** | **End** | **State** | **Note** |
| 1 | Design Class A, B, C | Nguyen Van A  Le Thi B | 2-Dec-18 | 1-Jan-19 | Done |  |
| 2 | Code Function 1 | Nguyen Van A | 01-Jan-19 | ~~17-Jan-19~~  20-Jan-19 | Delayed | Overlapped time with the other course project |
| 3 | Report Section II | Le Thi B | 02-Jan-19 |  | In progress |  |
| … | … | … | … | … | … | … |
| n | … | … | … | … | … | … |

**REFERENCE**

1. Tutorial Page, Oracle https://...
2. …

*[Students, please put here whatever sources you referred or used in the project]*