PREDICTIONS.ELM REPORT

BY: KUMAIL NAQVI 400068946

For the Final Bonus Assignment:

CS1JC3 – Professor Christopher Anand

TABLE OF CONTENTS

Contents

REPORT

Introduction

University finals are extremely important in securing a high grade and GPA as they usually have heavy weightings, however most students are not aware of how prepared they are for the final examination. Students usually end up with low grades in their final even though their progress was seemingly good throughout the semester. A functional way to address this problem is to somehow predict how a user would do on their final with their current progress, that way a user can know what level they're at and how much they need to improve.

The Predictions.elm program is created to address this problem, by analyzing thousands of raw data entries containing user actions for the online CS 1JC3 Elm assignments. The program calculates a prediction of how each user present in the dataset file did on their final examination for CS 1JC3. This is achieved by analyzing user data and formulating what the user could have got on their final by passing the analyzed data into an experimentally derived formula.

REPORT

Functions Used & Implementation

The program and all the functions were made by me from scratch without importing any external modules.

userQuestionCorrect:

- List, Int ,Int-> Bool
- The most basic function, this function checks if a given user passed a given question

userUniqueTrueOccurances:

- List, Int ,Int-> Int
- Used to determine how many questions a given user has passed in total.

usersList:

- List -> List
- Creates a list of unique users present in the Anonymous.elm file

unique:

- List -> Set -> List
- Removes duplicate values form a list and orders the list.

listOfCompletedQuestions:

- List, Int ,Int-> List
- Compiles a list of questions a given user has completed

numUsersPassedQuestion:

- Int -> Float
- Finds the number of users in total who passed a given question

REPORT

numPercentPassedQuestion:

- Int -> Float
- Divides numUsersPassedQuestion by 117, the number of users, to find the percent of users who passed a given question

totalEaseOfQuestion:

- List, Int ,Int-> Bool

- Calculates the sum of numPercentPassedQuestion for each question a given user has passed

avgEaseOfQuestion;

- List, Int ,Int-> Bool
- Divides the output of totalEaseOfQuestion for a given user by the number of questions that user passed.

firstOccuranceTime:

- List, Int ,Int-> Bool
- Finds the first time a given user compiled a given question

firstCorrectOccuranceTime:

- List, Int ,Int-> Bool
- Finds the first time a given user correctly compiled a question

REPORT

timeTakenOnQuestion:

- Int-> Int
- Finds the difference between firstOccuranceTime and FirstCorrectOccuranceTime to find the time taken by a given user to pass a given question.

totalTimeTaken:

- Int-> Int
- Calculates the sum of timeTakenOnQuestion for all questions a given user has passed.

averageTimePerQuestion:

- Int-> Int
- Divides the value of totalTimeTaken for a given user by the number of questions they've passed.

globalTimeSum:

- List -> Int
- Sums averageTimePerQuestion for every user.

globalAverageTimePerQuestion:

- List -> Int
- Divides the value of globalTimeSum by 117

userTimeScore:

- Int -> Float
- Divides the averageTimePerQuestion for a given user by globalAverageTimePerQuestion

userScore:

- Int -> Float
- Applies the derived formula to arrive at a score for a given user.

trueUserScore:

- Int -> Float
- Calculates the final prediction for a user.

Methodology

The prediction for a user's score is given by the formula I have derived:

$$\mathbf{2.75e}^{\text{userUniqueTrueOccurances}/40} \mathbf{- e}^{\text{avgEaseOfQuestion}} \mathbf{-} \text{ userTimeScore}$$

The constant 2.75 was found experimentally. I ranked the three parameters by their value in predicting a user's final score, the most valuable parameter was the number of questions a user has completed so the first term in the equation is given the highest value, the number of questions completed is divided by 40 because we were required to complete 40 questions. The second parameter is subtracted by the first because the smaller value of "avgEaseOfQuestion", the better your score, so the less is subtracted from the first term. The last term simply is userTimeScore which is subtracted from the rest, the smaller the value someone got for "userTimeScore" the better they did. However, I did not keep "userTimeScore" as a logarithmic term because it's not as valuable as the first two because of the way users answered questions.

REPORT

The final prediction is given by the "trueUserScore" function which directly and indirectly encompasses all the other functions. It contains 3 checks, firstly if the user has done nearly no questions it outputs 0, as the formula above will give negative values. Then the function checks if the userTimeScore is less than 0.01, this is a very important check as most people have timeScores within ranges of 0.5 and 2, but there was a very anomalous set of users who had extraordinarily fast question completion times, they were able to answer all their questions without a single mistake, this is not possible, especially for such a large number of people, I ruled that these users must have copy/pasted code from other users who had already passed the question through trial and error. So, for users whose timeScores are less than 0.01 it outputs a prediction of 1/7 on the final exam since they have not learned how to write in Elm, they were simply copying code from others. Finally, this function caps the max value at 7, so if someone's "userScore" value exceeds 7, this function will output 7, otherwise it rounds the number down to an integer and returns the value.

Conclusion

As this report shows I've successfully created a program, Predictions.elm, which can predict what users may have got on their final Computer Science 1JC3 examination by analyzing raw data entries. The program does this by combining all the functions I've implemented and applies them to a formula which I derived through trial and error to give a prediction. The final outcome is a visual list of all unique users with their calculated prediction next to their user number.

Appendix

```
---kumail naqvi 40068946 2016
import Html
import GraphicSVG exposing (..)
import Anonymous exposing(..)
import Set exposing(toList, fromList)
type Message = GameTick Float GetKeyState
main = gameApp GameTick {
                    model = model
                  , view = view
                  , update = update
                  }
model = {
          t = 0
      }
update message model =
  case message of
    GameTick tick (getKeyState,changeP1,changeP2) -> {
                            t = tick
                        }
view model = (collage 1366 720 [frame])
-----------------------------------------------------------------------------------------------------------------------
getTime anonList questionNum userNum = case anonList of
              (x1, x2, x3, x4)::xs -> [(x1, x2, x3, x4)]
              otherwise -> otherwise
-----------------------------------------------------------------------------------------------------------------------
firstOccuranceTime anonList questionNum userNum = case anonList of
---Checks the first time a user compiled a given question
---Gives Int
```

```
---working
        (x1, x2, x3, x4)::xs -> if userNum == x1
                        then
                          if questionNum == x2
                           then x3
                           else firstOccuranceTime xs questionNum userNum
                        else firstOccuranceTime xs questionNum userNum
        otherwise -> 0
-------------------------------------------------------------------------------------------------------------
firstCorrectOccuranceTime anonList questionNum userNum = case anonList of
---Checks if a user ever got a certain question correct and Gives time of first pass
---Gives Int
---working
        (x1, x2, x3, x4)::xs -> if userNum == x1
                        then
                          if questionNum == x2
                           then
                             if x4 == True
                              then x3
                              else firstCorrectOccuranceTime xs questionNum userNum
                           else firstCorrectOccuranceTime xs questionNum userNum
                        else firstCorrectOccuranceTime xs questionNum userNum
        otherwise -> 0
-------------------------------------------------------------------------------------------------------------
timeTakenOnQuestion anonList questionNum userNum = (firstOccuranceTime anonList questionNum userNum) -
(firstCorrectOccuranceTime anonList questionNum userNum)
---Calculates the time taken by a user on a given question to go from the first compile to a pass
-------------------------------------------------------------------------------------------------------------
totalTimeTaken anonList userNum qList = case qList of
---Calculates the total time taken by user to complete all passed questions
```

```
---Gives Int

---working

        x::xs -> if userQuestionCorrect anonList x userNum == True

                then (timeTakenOnQuestion anonList x userNum) + (totalTimeTaken anonList userNum xs)

                else (totalTimeTaken anonList userNum xs)


        otherwise -> 0
```

```
averageTimePerQuestion userNum = (round((totalTimeTaken userQuestionTimePass userNum
listOfQuestions)/(userUniqueTrueOccurances userQuestionTimePass listOfQuestions userNum)))//1000

---Calculates the average time spent per question by a user

---Gives Int
```

```
globalTimeSum uList = case uList of

        x::xs -> ((averageTimePerQuestion x) + globalTimeSum xs)

        otherwise -> 0

---Adds up all the average times of evert user

---Gives Int

---working
```

```
globalAverageTimePerQuestion uList = ---(globalTimeSum uList)

                -1243609//117

---Gets the overall average time taken on each question by each user

---gives int

---working

---replaced globalTimeSum with the exprimental value to increase optimization
```

```
userTimeScore userNum = roundN 3 (toFloat(averageTimePerQuestion userNum))/(toFloat(globalAverageTimePerQuestion
listOfUsers))
```

```
userQuestionCorrect anonList questionNum userNum = case anonList of
```

```
---Checks if a user ever got a certain question correct
---Gives a Boolean Value
---working
        (x1, x2, x3, x4)::xs -> if x1 == userNum
                        then
                          if questionNum == x2
                            then
                              if x4 == True
                                then True
                                else userQuestionCorrect xs questionNum userNum
                            else userQuestionCorrect xs questionNum userNum
                        else userQuestionCorrect xs questionNum userNum
         otherwise -> False
-----------------------------------------------------------------------------------------------------------------
userUniqueTrueOccurances anonList questionsList userNum = case questionsList of
---Gives the number of questions a given user has passed
---Gives Int
---working
        x::xs -> if userQuestionCorrect anonList x userNum == True
                then 1 + (userUniqueTrueOccurances anonList xs userNum)
                else userUniqueTrueOccurances anonList xs userNum
        otherwise -> 0
-----------------------------------------------------------------------------------------------------------------
listOfCompletedQuestions anonList questionsList userNum = case questionsList of
---Gives a list of all the questions a user has passed
---Gives List Int
---working
        x::xs -> if userQuestionCorrect anonList x userNum == True
                then [x] ++ (listOfCompletedQuestions anonList xs userNum)
                else listOfCompletedQuestions anonList xs userNum
```

```
        otherwise -> []

-----------------------------------------------------------------------------------------------------------------

userScore userNum = (((2.75*(e^(roundN 3 ((userUniqueTrueOccurances userQuestionTimePass listOfQuestions userNum)/40))))-
(e^(avgEaseOfQuestion userNum (listOfCompletedQuestions userQuestionTimePass listOfQuestions userNum)))- userTimeScore
userNum))

-----------------------------------------------------------------------------------------------------------------

---The userTimeScore check is to check for people who copy/pasted answers.

trueUserScore userNum = if (userUniqueTrueOccurances userQuestionTimePass [1..103] userNum) < 2

                then 0

                else

                  if round (userTimeScore userNum*10) == 0

                   then 1

                   else if userScore userNum > 7

                        then 7

                        else floor(userScore userNum)

-----------------------------------------------------------------------------------------------------------------

numUsersPassedQuestion questionNum listOfUsers = case listOfUsers of

---Determines the number of users who passed a given question

---Gives Int

---working

        (x)::xs -> if userQuestionCorrect userQuestionTimePass questionNum x == True

                then 1 + (numUsersPassedQuestion questionNum xs)

                else (numUsersPassedQuestion questionNum xs)

        otherwise -> 0

-----------------------------------------------------------------------------------------------------------------

numPercentPassedQuestion questionNum = (((numUsersPassedQuestion questionNum listOfUsers)/(117.0)))

---Calculates the percent of users who passed a given question

-----------------------------------------------------------------------------------------------------------------

usersList anonList = case anonList of

---Creates a list of all users

---Gives List Int
```

```
            (x1, x2, x3, x4)::(y1, y2, y3, y4)::xs -> if y1 /= x1
                                        then [x1, y1] ++ usersList ((y1, y2, y3, y4)::xs)
                                        else usersList ((y1, y2, y3, y4)::xs)
            otherwise -> []
```

```
unique list = Set.toList(Set.fromList(list))
```

----Removes duplicate values from given list, sorting the list in the process

---Gives List

```
roundN n x =
  let
    p =
      10 ^ (toFloat n)
  in
    (toFloat << ceiling <| x * p) / p
```

```
totalEaseOfQuestion userNum userQuestionsList = case userQuestionsList of
```

---Calculates the sum of the percents of each passed question

```
            x::xs -> (numPercentPassedQuestion x) + totalEaseOfQuestion userNum xs
            otherwise -> 0
```

----Calculates the average 'ease' of a users completed questions, lower number is better

```
avgEaseOfQuestion userNum userQuestionsList = roundN 3 ((totalEaseOfQuestion userNum
userQuestionsList)/(userUniqueTrueOccurances userQuestionTimePass listOfQuestions userNum))
```

----VARIABLE CONSTANTS

```
currentUser = 239
currentUser2 = 45
currentQuestion = 2
```

```
----FIXED CONSTANTS

----DO NOT CHANGE THESE

listOfUsers = unique(usersList userQuestionTimePass)

numberOfUsers = List.length listOfUsers

listOfQuestions = [1..103]

createLabel usersList move1 move2 move3 move4 move5 = case usersList of

        x::xs -> if x < 35

                then(text (("User: ") ++ toString(x) ++ "  =  "  ++ toString(trueUserScore x) )|> size 15 |> filled blue |> move(-600,
move1)) :: createLabel xs (move1 - 25) move2 move3 move4 move5

                else

                  if x < 84

                   then (text (("User: ") ++ toString(x) ++ "  =  "  ++ toString(trueUserScore x) )|> size 15 |> filled blue |> move(-400,
move2)) :: createLabel xs move1 (move2 - 25) move3 move4 move5

                  else

                    if x < 201

                     then (text (("User: ") ++ toString(x) ++ "  =  "  ++ toString(trueUserScore x)) |> size 15 |> filled blue |> move(-200,
move3)) :: createLabel xs move1 move2 (move3 - 25) move4 move5

                     else

                       if x < 238

                        then (text (("User: ") ++ toString(x) ++ "  =  "  ++ toString(trueUserScore x)) |> size 15 |> filled blue |> move(0,
move4)) :: createLabel xs move1 move2 move3 (move4 - 25)  move5

                        else (text (("User: ") ++ toString(x) ++ "  =  "  ++ toString(trueUserScore x)) |> size 15 |> filled blue |>
move(200, move5)) :: createLabel xs move1 move2 move3 move4 (move5 - 25)

        [] -> []

-------------------------------------------------------------------------------------------------------------------

frame = group ((createLabel listOfUsers (300) (300) (300) (300) (300)) ++ [text "Made by Kumail Naqvi 400068946" |> size 15 |>
filled red |> move (200, -300)])
```