Project documentation

Coverage

The code solves all four parts.

Usage

There are 2 Perl scripts to execute:

The Usage is also documented in the files.

CreateQuestionSheet.pl

Solves 1a.

Used to Score the exams and return useful information about them.

Usage: *perl ScoreExam.pl [master_file_path] [studentFiles]*

[master_file_path] the path to the master file

[studentFiles] A list of student files. Supports file pattern like *-short_exam_master_*.txt

Example:

*perl ScoreExams.pl FHNW_entrance_exam_master_file.txt SampleResponses/**

*perl ScoreExams.pl exam_master_file.txt *-short_exam_master_*.txt*

ScoreExams.pl

Solves 1b and part 2 – 4.

Used to create a question sheet from a master file.

Usage: *perl CreateQuestionSheet.pl [master_file_path] [debug]*

[master_file_path] the path to the master file

[debug] optional flag. When set then the created sheet will be answered randomly.

Example:

perl CreateQuestionSheet.pl FHNW_entrance_exam_master_file.txt

perl CreateQuestionSheet.pl short_exam_master_file.txt debug

Used Cpan Modules

- Time::Moment
- Text::Levenshtein
- List::Util

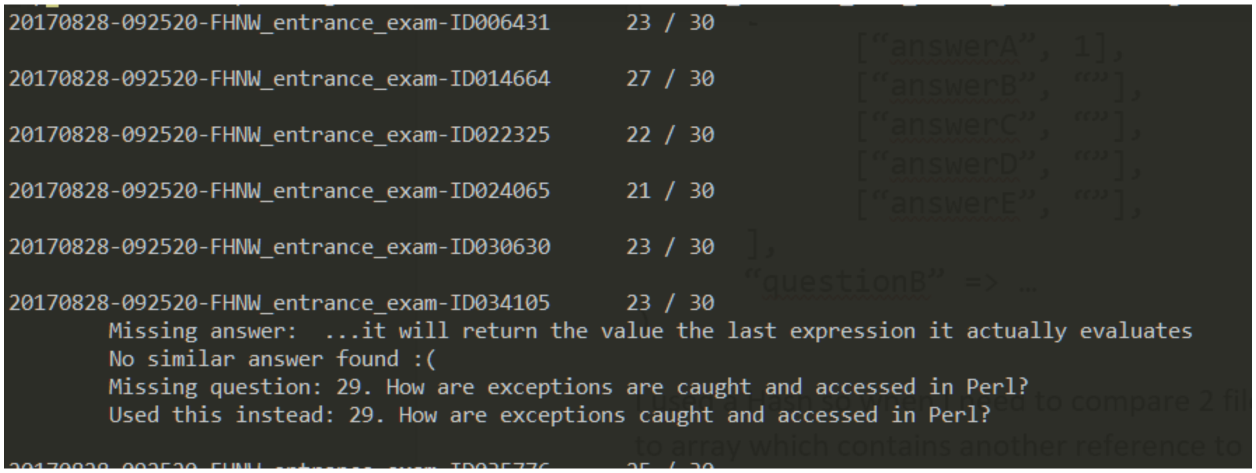
Part 1a

To solve 1a I made a **Parser** module which returns a hash with all the information I need form the file. The Parser will return a Hash which looks like follow:

```
(
      “questionA” =>
      [
            [“answerA”, 1],
            [“answerB”, “”],
            [“answerC”, “”],
            [“answerD”, “”],
            [“answerE”, “”],
      ],
      “questionB” => …
)
```

I used a Hash so when I need to compare 2 files it is easy to match 2 questions. The value is an reference to array which contains another reference to an array, in this array is the text of the answer and a Boolean if it is marked with an x or not.

Part 2a – 4



```
20170828-092520-FHNW_entrance_exam-ID006431      23 / 30

20170828-092520-FHNW_entrance_exam-ID014664      27 / 30

20170828-092520-FHNW_entrance_exam-ID022325      22 / 30

20170828-092520-FHNW_entrance_exam-ID024065      21 / 30

20170828-092520-FHNW_entrance_exam-ID030630      23 / 30

20170828-092520-FHNW_entrance_exam-ID034105      23 / 30
        Missing answer:  ...it will return the value the last expression it actually evaluates
        No similar answer found :(
        Missing question: 29. How are exceptions are caught and accessed in Perl?
        Used this instead: 29. How are exceptions caught and accessed in Perl?
```

In this part I use the parser written in Part 1a to match the master file with n student files. When I check if a question form the master file exists in the student Hash then I do 3 steps:

- Check if the key exists in the Student Hash
- If not, then trim and lower case the question and all key in the student hash and search for a identical match.
- If these also fails, normalize the question and the student keys and then calculate the Levenshtein distance with the question and all students key. If the minimum distance divided with the length of the normalizes question is smaller than 0.1 then we found a match.
- When nothing matches, we will ignore the question and print out an error.

When a question was found, a similar algorithm is used for the matching of the answer. But because we don't have a hash we skip the first step.

Now we have found which answer in the masterfile correspond to which answer in the student file. After I matched the answers I will keep track of which one was marked with a x and which one was marked correctly. So now it is easy to find out if the question was answered correctly I check if the question was marked correctly and only one was marked.

The statistic Hash

To solve part 3 and 4 I saved important information in a Hash which looks as follows:

```
@statistics =
(
	{
		score => int,
		answeredQuestions => int,
		file => sting,
		answerBinaryMatrix => \@int
	}
)
```

Score: The score reached by this student.

AnsweredQuestions: How many question the student has marked with one or more x's.

File: The file name.

AnswerBinaryMatrix: I stored the answers of the student in a special array with integers. So for every answer I save a single int value. These value needs to be interpreted in binary.

When we have for example 3 Questions the student has answered the Question as follows:

Q0: A0 (correct answered)
Q1: A0 and A2
Q2: A1
Q stands for question and A for answer the number behind it is the id of the question.

The array would save the following ints in bit notation.

	A2	A1	A0	Correct
Q0	0	0	1	1
Q1	1	0	1	0
Q2	0	1	0	0

The created Array would look like this (3, 10, 4).

Now it is easy to find out if 2 students have answered the question the same, we only need to look if the number are the same. And it is also easy to find out if the student has answered the question correctly we only need to check if the number is odd. But I like to write the check as follows: `($a | 1) == $a` because it is clearer that I check the first bit.

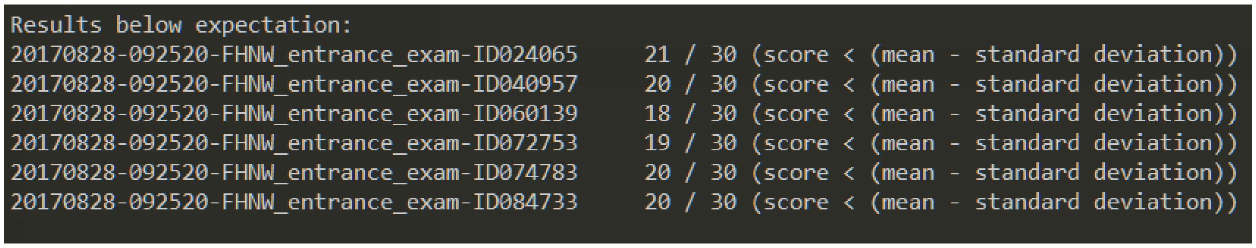
So, when we have 2 students:

@S1 = (3, 10, 4)
@S2 = (3, 10, 5)

S1 has 1 question answered correctly and S2 hast 2 correct.

S1 and S2 booth answered question 0 and 1 the same.

Identifying below-expectation results



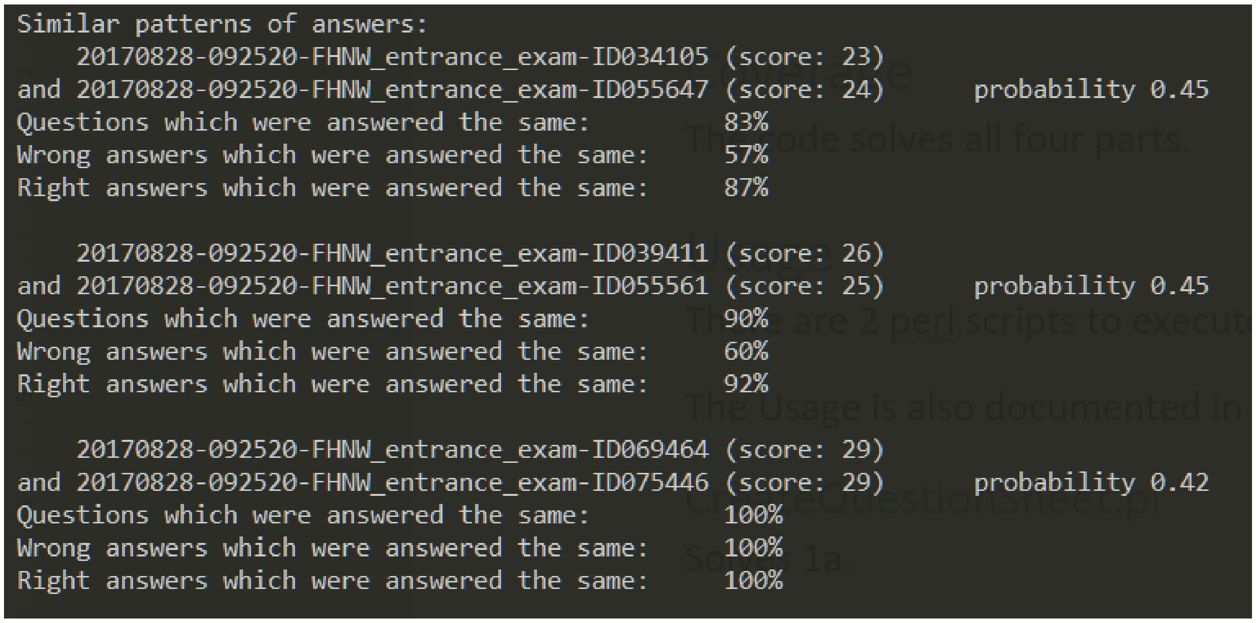
```
Results below expectation:
20170828-092520-FHNW_entrance_exam-ID024065     21 / 30 (score < (mean - standard deviation))
20170828-092520-FHNW_entrance_exam-ID040957     20 / 30 (score < (mean - standard deviation))
20170828-092520-FHNW_entrance_exam-ID060139     18 / 30 (score < (mean - standard deviation))
20170828-092520-FHNW_entrance_exam-ID072753     19 / 30 (score < (mean - standard deviation))
20170828-092520-FHNW_entrance_exam-ID074783     20 / 30 (score < (mean - standard deviation))
20170828-092520-FHNW_entrance_exam-ID084733     20 / 30 (score < (mean - standard deviation))
```

The code for solving this problem is written in the **Statistics** module.

I choose the following three criteria's for finding below-expectation results:

- The score is under 50% of the total points.
- The score is below the mean - standard deviation.
  - Standard deviation: $\sqrt{\sum_{i=0}^{d} \frac{(xi - \bar{x})^2}{d-1}}$
    - $\bar{x} = mean\ of\ the\ scores$
    - $xi = \ the\ i^{th}\ score$
    - $d \ = number\ of\ scores$
- The student has answered less than 50% of the questions.

Identifying possible academic misconduct



```
Similar patterns of answers:
    20170828-092520-FHNW_entrance_exam-ID034105 (score: 23)
and 20170828-092520-FHNW_entrance_exam-ID055647 (score: 24)      probability 0.45
Questions which were answered the same:         83%
Wrong answers which were answered the same:     57%
Right answers which were answered the same:     87%

    20170828-092520-FHNW_entrance_exam-ID039411 (score: 26)
and 20170828-092520-FHNW_entrance_exam-ID055561 (score: 25)      probability 0.45
Questions which were answered the same:         90%
Wrong answers which were answered the same:     60%
Right answers which were answered the same:     92%

    20170828-092520-FHNW_entrance_exam-ID069464 (score: 29)
and 20170828-092520-FHNW_entrance_exam-ID075446 (score: 29)      probability 0.42
Questions which were answered the same:         100%
Wrong answers which were answered the same:     100%
Right answers which were answered the same:     100%
```

The code for solving this problem is written in the **MisconductDetector** module.

To Identify the possible academic misconduct, I choose to use the following algorithm:

Compare all students with each other.

I used this two nested for loop to achieve this:

```
for(my $i = 0; $i < $length - 1; $i++)
{
    for(my $k = $i+1; $k < $length; $k++)
    {
        #ignore if $i == $k
        #compare student $i with student $k
    }
}
```

Which produce a comparison that looks like this:

	Student0	Student1	Student2	Student3	Student4
Student0	Ignore	x	x	x	x
Student1		Ignore	x	x	x
Student2			Ignore	x	x
Student3				Ignore	x
Student4					Ignore

I also filtered out all students which reached the maximum score.

Compare Student a with Student b

When *sameWrongPercantage* is larger than 0.5 and *sameRightPercantage* is lager 0.5 then mark them as a possible academic misconduct.

To calculate the probability, I used this formula: $\text{samePercantage} * \left(1 - \frac{\min(scoreA, scoreB)}{n} * 0.6\right)$

I used the min of the score to lower the probability. Because it is more likely when the two students answered a lot of question correct that they have a lot of similar answers. The 0.6 is there to dampen down the value so the percentage don't get small.

n: Number of questions.

same: Count how many questions were answered the same.

sameRight: Count how many questions were answered the same and correct.

sameWrong: Count how many questions were answered the same and incorrect.

aWrong: Count how many questions A has answered incorrect.

bWrong: Count how many questions B has answered incorrect.

aRight: Count how many questions A has answered correct.

bRight: Count how many questions B has answered correct.

samePercentage: $\frac{same}{n}$

sameWrongPercentage: $\min\left(\frac{sameWrong}{aWrong}, \frac{sameWrong}{bWrong}\right)$

sameRightPercentage: $\min\left(\frac{sameRight}{aRight}, \frac{sameRight}{bRight}\right)$

scoreA: The score of Student b.

scoreB: The score of student b.