CS454
AI Based Software Engineering, Autumn 2017

Coursework #2: Learning to Rank
Due by 23:59, 08 December 2017

1 Aim

Automated debugging and, in particular, fault localisation is a technique that aims to tell developer where the source of the observed failure is. In other words, if you provide test results as well as other inputs, the technique will tell you where the buggy line/method/class/file is. The aim of the second coursework is to implement a learning mechanism for automated debugging.

2 Fault Localisation

The aim of this coursework is essentially to replicate a state-of-the-art research on fault localisation, which has been published in the following paper:

- J. Sohn and S. Yoo. FLUCCS: Using code and change metrics to improve fault localisation. In Proceedings of the International Symposium on Software Testing and Analysis, ISSTA 2017, pages 273–283, 2017. (`https://dl.acm.org/citation.cfm?id=3092717`)

While technically it is possible to finish this coursework without consulting the paper, you are highly encouraged to read the paper before you start!

2.1 Dataset

The training dataset contains feature data for 156 real world faults. The data for each fault are stored in corresponding `csv` file (for example, `Math_106.csv` contains the data for fault 106 for Apache Commons Math project). In each `csv` file, you will find a header row with feature names, and rows of data: each row represents an individual method in the source code of the project. Features include 33 existing Spectrun Based Fault Localisation (SBFL)[1] scores and 8 additional code and change metrics. For more details and contexts, refer to the paper.

The last column in the `csv` file is titled `fault` and contains either 0 or 1. If it is 1, it means the method is faulty; 0, otherwise (i.e. method not faulty). Note that there can be multiple 1s (i.e. fixing the fault involves multiple methods). **Make sure that you do not use the fault column when training your model! That would be cheating.**

You can download the training datasets from the following url:
`http://coinse.kaist.ac.kr/assets/files/teaching/2017/cs454/fluccs_data.zip`

2.2 Goal & Approach

Your goal is to either train or learn a ranking model: given the `csv` file, produce a ranking of methods so that the faulty method comes at the top (or as close to the top as possible). For

[1] To understand the basics of SBFL, please refer to Section 3.2 of the following paper: W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa. *A survey on software fault localization.* IEEE Transactions on Software Engineering, 42(8):707740 August 2016. You do not have to read any other sections.

example, a trivial submission may just sort all methods using the `ochiai` column: this would only be as good as the Ochiai fault localisation metric is.

The idea is that, by combining all these features, we can do better than using a single feature. One possible approach, therefore, can be to use a GP based learning-to-rank approach. Your GP evolves an expression, which takes all the features for terminal values, and outputs a single score per method (i.e. row). Your fitness for each GP candidate expression is the rank of the faulty method, when methods are sorted according to the core produced by that expression.

The above GP approach is what has been used in the paper: any alternative approach will be welcome as far as it performs well. We have withhold 54 further faults, so that we can use them to evaluate your models.

2.3 Your Submission

In terms of implementation, you should submit the following three components:

- Training/learning algorithm: this is the algorithm that learns to rank methods so that the faulty one comes at the top. It would take all training datasets as standard input arguments: for example:

  ```
  > python train.py Math_1.csv Math_2.csv Lang_1.csv...
  ```

  Running `train.py` with training datasets should produce a model file. Note that you are allowed use external libraries, such as Deap (`https://github.com/DEAP/deap`) or Scikit-Learn (`http://scikit-learn.org/stable/`) for this coursework. However, make sure you explicitly state your dependencies so that we can execute your algorithms.

- Validation algorithm: this is the algorithm that applies your learnt model to unseen faults. It should print out the zero-based rank of the faulty method for each `csv` file used as input (for faults that involve multiple methods, print the highest rank). For example:

  ```
   > python validate.py [your model file] Math_7.csv Math_12.csv Lang_11.csv...
  0
  3
  2
  ...
  ```

  The output can be interpreted as: the highest rank of faulty methods in `Math_7.csv` is 0 (i.e. the top), the highest rank of faulty methods in `Math_12.csv` is 3 (i.e. the 4th place), etc.

- Your model file: you should store the best model you learnt into a file (choose whatever format that's convenient for you), so that we can evaluate the model using our withheld data.

2.4 Overfitting

You will want to avoid overfitting, because if your model performs very well for the training data but poorly for the withheld faults, it would be a failure. A common approach for avoiding overfitting is to perform cross-validation. A standard approach, which is called 10-fold cross validation, is as follows:

1. Divide your training data into 10 subgroups randomly.

2. Repeat 10 independent runs: in each run, use 9 subgroups as training set, and evaluate using the 1 remaining subgroup (i.e. measure how well your learnt model does against unseen faults).

Note that 10-fold cross validation is just a suggestion: you can use a different number of folds, or try another way of reducing overfitting. Regardless of the method you pick, choose a model that you think performs the best (for any arbitrary data), and submit that model.

3 Deliverables

Each person should submit the following deliverables by the submission deadline:

- **Implementation**: source code of your learn-to-rank algorithm (self-contained), as well as your best model stored in executable form.
- **Report**: include a written report that contains detailed descriptions of how you approached the problems. Describe the optimisation you have implemented in as much detail as possible. There is no page limit.

For ease of marking, follow the following directory structure, and submit a zip file containing the top level directory, through KLMS.

```
[your student number]
├── report.pdf ..... Your report documenting your implementation
└── rank ..... Your training & validation algorithm, as well as the best model
```

4 Grading Criteria

- Model Accuracy: Using your learnt model, we will evaluate the withheld faults and collect the rankings of the faulty methods. We will look at the distribution of the ranks of faulty methods (the lower the average rank is, the better), as well as the number of top-ranked faulty methods. This will account for 30% of your coursework marks.
- Implementation & Report Quality: As before, we will grade how good your implementation is. To get good marks here, adhere to good software engineering practice, in both your implementation and your report. Write good code, include test if possible, and make your report a good documentation. This will account for 30% of your coursework marks.
- Idea: Finally, we will evaluate how unique your main idea is. This will account for 40% of your coursework marks.