COMP 495 Research Report

*Andrew Li and David Stotts*

ABSTRACT

INTRODUCTION

WORK DONE

The first step was to collect submission and problem data from previous sections of the introductory programming course (in this case COMP 110). These data were stored in .bson files, and in order to manipulate them as desired in JavaScript they were converted to .json files. The mongorestore1 program creates a mongo database from a .bson file, and from there the mongoexport2 program was utilized to export the database into an array of .json objects that could be programmatically iterated through. Submission and problem data files without a suffix are from the spring semester of 2018, and every other submission data filename has a suffix corresponding to the semester it corresponds to (fall 2019, spring 2020, etc.).

Once I imported the data into the correct format (arrays of .json objects), I manipulated using functions defined in the processor.js file. Defined functions were called in the bottom of the file and ran in the console using node.js. Once the raw .json arrays of submission data were exported from mongodb, they were sorted into their respective problems and stored in the SubmissionsByProblem directory. Problem data were sorted in the same way.

Next, I determined that problems with only one input would best fit our use case for simple problems to analyze, as the correctness of each submission for a one-input problem was most likely to be binary rather than include partial credit. For each set of submission data, I included only submissions for single-input problems, and then sorted submissions by correctness. These data are found in the OneInputSubmissions directory. I sorted these single-input problems by total number of submissions, and selected two problems, the FizzBuzz array problem and For Loop Output problem. These two problems had high submission counts, and had at least twice the number of incorrect submissions as correct submissions. I eventually only worked on FizzBuzz array problem submissions.

In order to analyze each submission programmatically, facts need to be collected from submission source code before metadata can be passed into the unsupervised learning algorithm. I created the pcan.js module to collect these facts. Such general facts include:

* Number of for loop initializations
* Number of for loop variable initializations
* Number of for loop increment statements
* Number of accumulator initializations
* Number of alert statements

Beyond the general facts pcan.js also collects problem-specific facts, in this case facts specific to the FizzBuzz array problem. Pcan.js does this by first converting the submissions source code string to an abstract syntax tree with acorn.js3, and then uses the acorn-walk package to traverse the tree conditionally based on the desired fact. Afterwards, pcan.js returns a fact object that is attached to each submission and outputted into the submissionsWithFacts.json file. A processor.js function (createFactArray()) then converts that fact object into an array, where each element is the fact in insertion order without the labels, as well as an array of submission IDs also in insertion order. This is necessary in order to feed the fact data into the selected kmeans4 package.

Finally, the kmeans.js file runs the kmeans method from the imported ml-kmeans package on the 2d fact array eight times, with 2 through 9 clusters, inclusive. The output objects are stored in the kmeansResult.json file. Clustering results still need to be sorted through and interpreted, but each result does converge in between 2 and 5 iterations, a relatively low iteration count.

RESULTS AND DISCUSSION

CONCLUSION

REFERENCES

1 <https://stackoverflow.com/questions/6770498/how-to-import-bson-file-format-on-mongodb>

2 <https://stackoverflow.com/questions/8991292/dump-mongo-collection-into-json-format>

3 <https://www.npmjs.com/package/acorn>

4 <https://www.npmjs.com/package/ml-kmeans>