Feature reduction of Darshan counters using evolutionary algorithms

Neeraj Rajesh, Anthony Kougkas, Xian-He Sun
Illinois Institute of Technology
Chicago, IL, USA
nrajesh@hawk.iit.edu,{akougkas,sun}@iit.edu

Quincey Koziol, Suren Byna, Houjun Tang, Jean Luca Bez
Lawrence Berkeley Laboratory
Berkeley, CA, USA
{koziol,sbyna,htang4,jlbez}@lbl.gov

1 EXTENDED ABSTRACT

Feature reduction is an integral part of data preparation in machine learning. It helps denoise the data and makes it easier to fit the model. Predicting the performance of an application using Darshan counters can be tricky due to the large amount of data available, with not all of them being pertinent to predicting the I/O performance. There exist methods for feature reduction, the most common being Recursive Feature Elimination (RFE) [1]. The RFE method aims to correlate the features to a specific data point. We aim to get a subset of features that are able to distinguish between the different applications. Then compare the effectiveness of the subset by creating a model to predict I/O performance and compare that with a similar model created with all the features and with a subset of features got using RFE implemented on Scikit Learn [2].

Currently, we have a variety of profiling tools like Darshan [3] and Recorder [5] that can give us an idea about the I/O of an application [4]. These counters while useful are not all necessary to distinguish applications. Additionally, in order to maintain a set of varied applications to compare new applications, fewer counters also implies smaller size and computational requirements. In this effort, we use evolutionary algorithms, like genetic algorithms, to select a subset of Darshan counters which can be used to distinguish between a varied set of applications. We then compare how well the subset of the counters would correlate with I/O performance of the application.

2 METHODOLOGY

(1) **Data Preparation**
- Darshan is highly configurable and provides a lot of data. This causes different datasets to not have the same information uniformly across them and, as such, need to be processed.
  - A Darshan profile can have a number of different I/O modules, POSIX, MPI-I/O, HDF5 (H5F + H5D), STDIO, and more. We have limited it to POSIX and MPI-I/O due to the ubiquitous availability of those counters.
  - The Darshan profile of a job is a collection of *record IDs (RID)* equal to the number of files accessed by the application. Certain RIDs would have certain counters missing, as the counters only follow certain patterns and increment as needed. This makes it important to unify the counters across RIDs and fill in the missing values. We filled the missing values with 0 as they wouldn't affect the cosine similarity used.
  - We focus mainly on RIDs, from a varied set of jobs, to avoid aggregating RIDs in jobs.

(2) **Creating the Search Algorithm**
- Search State: A search state in this algorithm is any subset of the Darshan counters encompassing every permutation from 1 to all features.
- Population Generation: A random selection of a subset of the Darshan counters. These counters were in their generic regex forms.
- Population mutation: A random removal and addition of Darshan counters.

(3) **Fitness Function**
- Prior to measuring the fitness of the application of the Darshan counters, the counters need to be normalized. To do so, we perform column-wise min-max normalization.
- We then calculate the loss of data when using the subset, compared to using all the counters. Here we have a minimization function where the best fitness is 0 and the worst fitness is 1.
- A consequence of the above fitness function is that when taking a subset of the counters, you can effectively choose a subset of counters that can create higher similarity between the respective applications. In these situations, we set the fitness to 1.

$$fitness(pop) = \begin{cases} \Sigma_{i=0}^{n} \frac{Sim^i_{all} - Sim^i_{pop}}{n}, & \text{if } \forall_i Sim^i_{all} >= Sim^i_{pop} \\ 1, & \text{otherwise} \end{cases}$$

$$Sim^i_{all} = sim(app_i, counters_{all})$$
$$Sim^i_{pop} = sim(app_i, pop)$$

- We attempt to create a min-max aspect of the fitness function, in addition to the above fitness, by minimizing the number of counters selected for the population. Here, in order to minimize the loss in similarity between the applications, the search function will pick more counters, acting as the maximizing force to the counter selection. To do that, we add a minimizing fitness where we try to minimize the number of counters picked, picking lesser counters would cause an increase in the loss in similarity between applications.

(4) **Prediction I/O Performance**
- We need to quantify the effectiveness of our subset of features. We first calculate the throughput of the application from the Darshan trace, after which we perform the following methods for verification

  - PCA Analysis: We calculate the correlation of the counters to the throughput of the application. The acts as a litmus test for the effectiveness of our subset of counters.
  - Model effectiveness: We create 3 models, one with all the features in the dataset, one using a subset of features got using the Reduced Feature Elimination method, and one using the subset of features got from our evolutionary algorithm using a dataset different from the one used for feature reduction and measure the root-mean-square error of the 3 models on a separate test dataset.
- Our subset of features had a fitness of approximately 0.150

3 LIMITATIONS

- The set of applications must be representative of as many applications as possible, because with more contextual information about applications, the more accurate the feature reduction would be.
- This method also is limited by issues plagued by such evolutionary algorithms.
- The feature reduced model may perform worse with an application that is drastically different from the trained applications due to the fewer features available in the feature reduced model.

4 OBSERVATIONS

a) **Data Preparation**: When preparing the data, Darshan counters are not uniform across records, and they need to be properly combined without skewing their fitness.

b) **Search Algorithm creation**: The fitness function needs to be properly bounded as per the quirks of the metrics used for fitness. In our case, we used the cosine similarity, so there are invalid states that need to be penalized where the similarity between applications can increase beyond their actual value based on the subset of features used as the subset no longer captures the differences between the applications. In the subset, it is possible some features, residual features, may neither positively nor negatively affect the similarity between the applications and can still be carried over generations.

c) **I/O Performance Prediction**: We observe that the feature reduced model is capable of effectively predicting the I/O performance of the application. The loss of the GA feature reduced model is lower than the RFE feature reduced model, which has a lower loss compared to the model with all the features. We see that features which help distinguish applications also help to better fit the I/O performance model. The residual features do not affect the performance of the model, and, since there is an overall reduction in noise in the data, it is now easier for the model to fit the data better.

5 CONCLUSION

We have presented a methodology to perform feature reduction using genetic algorithms and have highlighted the quirks in doing so. We then test the effectiveness of the subset of features by performing a PCA analysis on a separate dataset used for the feature reduction, where we notice that a majority of the features selected correlated well with the I/O performance, with the other features being residual features. We then confirm our hypothesis by using our subset of features to train a random forest regression model, and we observe that the feature reduced model is capable of predicting the I/O performance well. Due to the reduction in noise in the feature reduced dataset, we also notice that the feature reduced model has a better fit model compared to the model trained with all the features.

REFERENCES

[1] Burcu F Darst, Kristen C Malecki, and Corinne D Engelman. 2018. Using recursive feature elimination in random forest to account for correlated variables in high dimensional data. *BMC genetics* 19, 1 (2018), 1–6.

[2] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.

[3] Shane Snyder, Philip Carns, Kevin Harms, Robert Ross, Glenn K Lockwood, and Nicholas J Wright. 2016. Modular hpc i/o characterization with darshan. In *2016 5th workshop on extreme-scale programming tools (ESPT)*. IEEE, 9–17.

[4] Shane Snyder, Philip Carns, Robert Latham, Misbah Mubarak, Robert Ross, Christopher Carothers, Babak Behzad, Huong Vu Thanh Luu, and Surendra Byna. 2015. Techniques for modeling large-scale HPC I/O workloads. In *Proceedings of the 6th International Workshop on Performance Modeling, Benchmarking, and Simulation of High Performance Computing Systems*. 1–11.

[5] Chen Wang, Jinghan Sun, Marc Snir, Kathryn Mohror, and Elsa Gonsiorowski. 2020. Recorder 2.0: Efficient parallel I/O tracing and analysis. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 1–8.