A systematical study on computation reuse across program executions on different input data

[Extended Abstract]

Shanil Puri*
Graduate Student, NCSU
Raleigh, North Carolina
spuri3@ncsu.edu

ABSTRACT

In this data explosion era, there are myriad programs which are executed repeatedly on large sets of data, consuming high amounts of energy and resources. Critical process reused on different data-sets will no doubt help reduce the time and computations. In this paper we will show introduce a new probabilistic model of measuring data similarities. Based on these data similarities we will show how critical computations may be re-used across data-sets, saving energy and improving performance.

This intuitive idea has only been explored spotted across history. This work is the first one that will systematically explored this topic. Some of the key questions that this work will try and answer can be given as follows:

- Is there significant potential of improvement?
- What are the difficulties for a general frame- work?
- How to address these difficulties to make this framework generally accessible?
- Introduce the term History Reuse.
- Introduce the concept of probability based similarity between data-sets.

1. INTRODUCTION

In this study we give an empirical study, showing that effectively reuse could enhance program performance. Although, the idea of computation reuse is simple, there are many difficulties need to be solved to achieve efficient and effective reuse. In these explorations, we found out challenges in 4 aspects: data features, similarity definition, scalability, and reuse. Data features, is the description of data-sets. Similarity definition (distance between two data-sets) is the way we would like to describe similarity between two data-sets. Data feature and distance definition are used together for reuse instance selection from a program specific database. The selection of history record is the most essential part of computation reuse because different record will lead to dramatically different computation amount. Because the most important properties of the input data may vary on different programs, it is difficult to decide a universal data feature and data distance definition. Since the method used to calculate distances between two data-setâĂŹs descriptions also should be defined based on the data features, the problem becomes even more complicated. Scalability issues are related with number of instances in database, size of the target data-set, and dimension of data-set. Goal of computation reuse is to save computation time and energy. In order to select a suitable history record, some extra computation is inevitably introduced into the computation. The dilemma is the trade off between the amount of computation reuse and the introduced overhead. For example, increase the number of in- stances in database will increase the probability of finding a good history record. However, it will also increase the introduced selection overhead. The size of target data-set and the dimension of data-set have similar problem. Reuse is the process to decide what history information the program should reuse and how to reuse the pre-calculated information. Challenges in this field is mainly caused by the differences among programs and algorithms. For a specific program or algorithm, it might be a trivial solution while for another selecting historical data may be a complex problem. But to build a general framework, this becomes much more complicated. In this paper, we investigate multiple solutions to address each of the challenges, and come up a framework, which we applied on several important programs and get performance improvements on them. The paper is organized in the following sections: Section 2 gives background and definitions of terms we use in this paper. Section 3 formalizes the framework and Section 4 presents the challenges and solutions we explored in details. We present the experiment results in Section 5. Related work is discussed in Section 6.

*

2. BACKGROUND

Computation reuse across executions benefits pro- grams with long computation time most, especially the converging algorithms that require multiple iterations to compute the desired results. Another benefit may be improved accuracy. Most of the converging algorithms belong in the NP class of problems and as such do not have optimal solution. Through appropriate history result reuse, numbers of iterations of computations could be saved while improving the accuracy of the algorithms. The key point of computation reuse on different data is finding suitable history information to reuse. Thus the above problem stated problem essentially boils down to introducing a probabilistic method of calculating data-set similarities quickly and accurately. With databases containing a number of instances. data-set feature is a key component of each instance, and history computation result of the program on this data-set is the value. Then, through computing distance from current data-set to each instance, our framework could select the instance which has the highest probability to provide most computation reuse.

3. MOTIVATION

Optimization of converging Algorithms such as K-means clustering is a highly beneficial objective. Algorithms of this category are used frequently and have a multitude of applications in real world machine learning and big data processing. In this age of data explosion this, thus a highly important research topic. Some of the motivations are:

- These algorithms are used frequently and thus even small improvements can have a big impact in performance improvement.
- These algorithms also have wide areas of application.
- These algorithms are ideal suited for such optimizations as their speed of convergence and accuracy is directly dependent on the starting points for the algorithm.

4. OBJECTIVES:

The main objectives of this paper may be listed as follows:

- Design and implement a framework capable of efficiently and accurately calculating similarity between data-sets and choosing the most optimal historical data-set for critical computations reuse.
- Introduce a new metric to compare previously computed input data-sets to the current data-set. The metric must be an accurate representation for comparison and ensure that its calculations is time effective.i.e. The time taken for metric calculation and data-set selection, should not nullify any improvements in run time that may be seen using this algorithm.
- Improve accuracy for converging algorithms. Since most algorithms are essentially NP class problems, their solution is almost never optimal. Yet their solutions are dependent upon the starting state of the algorithms. We believe that the use of historical computations from closely matched historical data will help with ensuring that most optimal starting states for the algorithms may be found.

5. FRAMEWORK

Our main objective for this the development of the framework is to provide a meaningful metric for comparing data-sets in a time efficient and accurate manner. For this purpose we have proposed a framework which aims to alienate the following four major hurdles in computation reuse:

- Data Features: Every data set is categorized by a set of features in the form of columns, assuming the data-set is represented as a 2D array. In such a case not all features for the data-set hold equal importance in the data-set categorization. Thus the first aim of our architecture was to formulate a method to ensure the selection of only the most important data-set features. This served a two fold purpose: 1. It allowed us to ensure that we could redeuce the dimensionality (feature count) of the data-set drastically while preserving the meanining of the data-set, which provided us with a way to reduce computations. 2. Extra storage required by the historical data-sets was greatly reduced.
- Similarity Definition: Another important feature requirement for our problem statement was to come up with a uniform metric for feature set comparison. For this we propose a Probability based metric for similarity between data-sets. By this metric we can make a quick yet accurate assessment regarding the degree of similarity in between data-sets. Obviously the best choice of historical data-set would be the one with the highes probability of being similar to the current data-set.
- Scalability: Another important requirement of the problem statement is scalability. In general converging algorithms run on large data-sets and thus it is extremely important that the time improvements seen by the reuse of critical computations are not offset by the algorithm to find the best match of historical critical computations.
- History Reuse: The last part of the framework will aim to ensure that the selected computations are the best match possible for the current data-set. This will thus ensure that at any point we are using the best possible starting state for our algorithm. This is also responsible for the computation of the best possible way to use the historical computations with our current data-set.

Keeping the above framework requirements in mind we finally settle on the framework components as follows:

- Feature Set Reduction: The first aim of the framework is to reduce the feature set of the provided and historical data to the minimum while still maintaining the essence of the data-set. Along with the feature set reduction, we also realised that for optimal performance of the algorithm, we would also need to minimize the number of data-points in each data-set. This too had the major restriction of requiring that the points chosen in essence represent the complete meaning of the data-set.

- Similarity Metric Calculation: Once the data set had been minimized while keeping its meaning intact, we now needed to design a probabilistic metric for the measure of similarity for the data-set. For this purpose we decided to use the above important Feature Set Reduction computation to our advantage. Our above computations essentially give us the a compressed data-set which in essence holds the same meaning as the original data-set. We then proceed to compute the probabilistic similarity on a per-feature basis of the current data-set with the historical data-set.

- History Computation Storage: The last part of the framework will aim to ensure that the selected computations are the best match possible for the current data-set. This will thus ensure that at any point we are using the best possible starting state for our algorithm. This is also responsible for the computation of the best possible way to use the historical computations with our current data-set.

- Matcher and Selector: In this part of the algorithm we use all the above calculations for finding the best match for the current data-set. We set about by, first computing and finding the closest match with the eigen values and eigen vectors for the PCA computed for the current data-set with historical data-sets. We choose the closest match that we find in out data-set and then use step 2 mentioned above to get a genuine probability metric for the two data-sets. We then calculate the probabilistic metric for the top three relative ranked data-sets for the historical data-set which we found with the current data-set and use the data-set for which we get maximum probabilistic metric.

5.1 Feature Set Reduction

The aim of this section of the framework is to compress the data to the highest extent possible while still maintain the meaning of the data-set. For this purpose we decided to take a leaf out of the Image processing/statistics books. This part essentially consists of two major parts:

- Dimension Reduction: In this part of the algorithm we essentially reduce the total dimensions of the data-set to the minimal possible without loosing the meaning of the data-set. This is achieved by the use of the principal of Principal Component Analysis (PCA). PCA is a method which takes in a data-set and then proceeds to return a modified data-set such that the dimensions in the updated, normalised data-set are arranged in descending order of importance. This way we can take the top few dimensions for our computations.

- Point Count Reduction: In this part of the algorithm we essentially aim at selecting the optimal sample set of data-points from the data-set for our computations. This is essentially achieved by dividing all the points into a buckets. We then proceed to pick the top-k highest populated buckets to get the highest density range of the data-set, while at the same time greatly reducing the total no of data-points that would be needed to be computed in our data-set.

It is important to note that this part of the algorithm works on a single data-set at a time.

5.2 Similarity Metric Calculation

Once we have evaluated the reduced data-set using steps mentioned in section 2.1, we proceed to calculate out similarity metric. We achieve this calculation using another statistical method T-Test based probability calculation. We take individual dimensions from the two data-sets and run T-Test on them to get a probabilistic measure of their similarity on a per dimension and a cumulative sum across dimensions. We then rank the data-sets with respect to each other based on the computed probabilistic metrics. Now this is an important aspect of our computation because this is the algorithm that we used for our probabilistic metric calculation, which is in turn used for ranking all data-sets relative to each other.

5.3 History Storage Architecture

One we have computed the the above mentioned values for our data-set, we store it in memory as objects. Each historical data objects holds its original data, PCA meta-data (eigen values and eigen vectors etc), bucket-wise histogram and the reduced data-set. In addition to this each object also stores a score of the probabilistic similarity it holds with each of the other historical data-sets and maintains them in non-increasing order of probability. This is done for quick access of data sets for computations when we are comparing real time data with historical data-sets.

5.4 Matcher and Selector

This is by far the most time critical part of the framework. It needs to be quick because this is the function that is responsible for matching the current data-set with the sets in the historical data-set and find the best match for computation reuse selection in real time. Since, historical data may be large, we need a quick way to find the closest match from the historical data-set. Our framework goes about doing this in the following two passes:

- In the first pass we do the PCA computation for our current set. We then use the eigen values and vectors from our current set calculations to scan over the

historical data and find the data-set with the closes match for eigen values and eigen vectors. This part of the algorithm runs linearly without much time delay. Thus we can afford to do this kind of matching with all the historical data-sets and get the closest match.

- We now use this historical data-set along with the current data-set to compute the probabilistic metric of similarity between the data-sets and then proceed to compute the same metric for the top three closest matches to the historical data-sets (this is already precomputed and stored.) We now use the data-set with the highest probabilistic similarity to out data-set for computation reuse.

6. CHALLENGES

Each part of the framework of implementations brought with it, its own set of challenges. While T-Test was an easy method to calculate the probability of similarity between two data-sets it had the shortcomings of being slow and thus could not be used for comparing all data-set with the current data set in real time. Also it could be run only on a single dimension at a time. This clearly was not a very scalable method. Some of the other challenges and their solutions are listed below.

6.1 Reducing Data-set dimension size:

The first major challenge was to reduce the data-set sample size while maintaining its essence. For this purpose we chose to use Principle Component Analysis. This is a method often used in Image Processing to reduce the dimensions of the points in such a way that it ensured that the meaning of the image would be maintained. We argued that since and image is essentially a set of multiple points with multiple dimensions, the matrix representation of the image used for PCA was essentially similar to the input data-sets that our typical converging algorithms would see. We thus argued, that since it worked for the image dimension reduction it would work for our data-set as well. Our argument was proved correct.

6.2 Reducing Data-points:

Once we had reduced the dimensions, we needed to implement a method of reducing the total number of points to represent the data set in the most minimalistic way possible. For this purpose we decided to use the range method. In this case we divided each dimension range across a dimension into a constant number of bins each having a certain range. Once this was done we divided the points within the bins depending on the dimension values. We then proceeded to build a histogram for the bins and chose the top few bins across all dimensions. This essentially gave us the highes concentration of the points while reducing the total number of points to a minimum.

6.3 Other Challenges:

- OpenCV Implementation for dimension reduction: Since we had chosen to use PCA for dimension reduction and C++ as our language of choice for this project (being run time critical as this was) we had to use the OpenCV implementation of PCA for our computations. OpenCV being a library primarily meant for image processing was not ideally suited to work on normal data. Thus one of the bigger challenges was representing our data in an OpenCV procesable format.

- Python Integration with C++: We had chosen the T-Test library for our probability based similarity metric calculation. The problem was that there were no good C++ based implementation for the T-Test library and we had to use the Python based SciPy library for statistics. Also due to the run time critical nature of the problem at hand we could not do multiple read writes even to main memory to interface C++ functions with the Python functions. Thus we chose to use embedded python with C++. This still being in its nascent stage documentation was hard to come by. Also sufficient example of implementation for the same were not present. This, thus presented us with a challenge in our choice of implementation.

7. RESULTS

In the implementation of our project we have been able to successfully demonstrate that our above proposed algorithm has been able to calculate our probabilistic metric efficiently and that it is an efficient way of categorizing data-set similarity. We have been able to effectively demonstrate that as the size of the data set increases, the use of our algorithm optimizes the k-means performance to be faster (at an average of 50 percent lesser time taken) compared to the k-means++ algorithm, while maintaining similar compactness as achieved by the k-means++ algorithm. In our benchmarking tests we have used 3 different size data-sets (as shown in tables) with different dimensions to benchmark our initialisation algorithm with the K-Means++ algorithm. We will show in our observations, that in general our algorithm has been able to reduce the K-means convergence time to less than half of the k-means++ algorithm in the average case, while reducing convergence time to just 1/10th of the time taken by the K-Means++ algorithm. We have also been successfully been able to demonstrate that we have been able to maintain compactness similar to the compactness achieved by the K-Means++ algorithm (though we have not been able to achieve an improvement in this area.) In the below two tables we we compare the run time taken by both the algorithms. The first table (Table 1) is used to display the run time for k-means++ algorithm for data-sets of various sizes and with varying K-Index.

The second table (Table 2) holds the values for time taken by our algorithm, which provides custom labels as starting points for the k-means algorithm, for the same data-sets and K-Indexes which were used for the K-Means++ algorithm representation.

Data Set Dims	Data Set Size	K-means (Cust Labels) Performance	K-Index
3	10	2337	3
3	10000	2922598	40
10	20000	6386345	40
3	10000	5481493	80
10	20000	6727399	80

Table 1: K-Means++ Performance

Data Set Dimensions	Data Set Size	K-means (Cust Labels) Performance	K-Index
3	10	6432	3
3	10000	482393	40
10	20000	3815759	40
3	10000	741193	80
10	20000	6727399	80

Table 2: History-Reuse K-means Performance

In the above table you will see that the performance of our algorithm in terms of time taken is way better compared to, the K-means++ algorithm.

(Note: 1. For timing the run time of the algorithms, we have timed the execution time taken by the program to cluster 4 new data sets. Thus each time represented above is time taken for clustering 4 data-sets. 2. All time shown above are in micro seconds. 3. Time taken in training history data-sets is not taken into consideration as it is a process which would ideally be carried out in the background.)

8. REMAINING ISSUES AND POSSIBLE SOLUTIONS

- Detailed benchmarking with varying input and accuracy requirements for the algorithm remains to be implemented.
- A way to normalise data so as to get better compactness.

9. LESSONS AND EXPERIENCES:

This was my first piece of research work and felt I had the tremendous opportunity to learn a lot from this. I was able to go outside my field of expertise and learn from multiple different fields including Statistics, Mathematics and Image processing. This, thus clearly helped me expand my horizons and knowledge. I also realised my passion for research and the countless hours I was able to spend on doing something for the simple reason that I actually enjoyed what I was doing. I also got a very clear idea about the fact that research is HARD. It takes a lot of time and perseverance and it is easy to get frustrated midway when you cant seem to be able to reach a satisfiable solutions. I hit multiple roadblocks along the way both in terms of idea and unfeasible implementation of the same. At such times I learnt that patience and hard work is the only solution.

10. NOVEL IDEAS

Some ideas to pursue in a similar field may be as follows:

- Store historical cluster data as graphs and then once closest match data-set is found use the graph for that data-set and update with new data-set points using the clustering points as single source shortest path algorithms. This may be a way to compute the K-means clustering algorithm in polynomial time.
- Use multiple data-set to choose different centroids instead of using a single-closest match data-set. This would ensure that all centroids chosen are the most optimal possible.

11. CONCLUSIONS

In conclusion we would like to say that probabilistic metric for data-set comparison is one of the best metrics for this purpose. It tells us the exact degree to which we can expect the computation reuse to be successful for any historical data-set. It also represents a generic way of ranking data-sets based on similarity and may have many more applications than just in history-reuse computation. We have also seen based on the above results that the above mentioned algorithm is handy only when data-set sizes are large. For smaller data-sets, any improvements achieved in converging algorithms, will be offset in the time taken to compute the closest matching historical data-set. I am also confident that the research presented in this paper is novel and will go a long way in helping improving converging algorithms.

12. ACKNOWLEDGMENTS

This paper was a collective work of multiple people. At this point I will like to thank Dr. Xipeng Shen, for the contributions. He was critical in helping me realise the problem statement and all the implicit and explicit details of the problem. He was critical in helping me understand how different dimensions in a data-set may hold different importance. He was also there to correct me when initially I had taken a wrogn approach to the problem with histogram based PDF functions. I would also like to mention the contribution of Tao Wang, who started this project and was critical in making me understand what pitfalls he had faced and the reasons certain thought trains had not been able to come to a correct conclusion.

13. REFERENCES

- OpenCV: docs.opencv.org/
- T-Test: https://en.wikipedia.org/wiki/Student

- : K-means++: https://en.wikipedia.org/wiki/K-means
- : K-means||: http://www.cse.buffalo.edu/faculty/miller/Courses/CSE633/Chandramohan-Fall-2012-CSE633.pdf
- Python References: https://docs.python.org/2/extending/extending.html
- C++ Documentations: http://www.cplusplus.com/doc/
- Community help (OpenCV):
- Community help (Python-C++Interface): http://stackoverflow.com/questions/[illegible]/python-c-funtion-call-from-cpp
- Community help (T-Test): http://stackoverflow.com/questions/30023932/pca-computation-using-opencv-use-return-value http://stackoverflow.com/questions/30090926/python-runtime-error-on-running-t-test

APPENDIX

A. HEADINGS IN APPENDICES

A.1 Introduction

A.2 Motivation

A.3 Objectives

A.4 Framework

A.4.1 Feature Set Reduction

A.4.2 Similarity Metric Calculation

A.4.3 History Storage Architecture

A.4.4 Matcher and Selector

A.5 Challenges

A.5.1 Reducing Data-set dimension size:

A.5.2 Reducing Data-points:

A.5.3 Other Challenges:

A.6 Results

A.7 Remaining Issues and Possible solutions

A.8 Lessons and Experiences

A.9 Conclusion

A.10 Acknowledgement

A.11 References