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 compu- tations 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 frame- work 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. Al- though, 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 set's descriptions also should be defined based on the data features, the problem becomes even more complicated. Scalability is- sues 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 instances 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 dimensionality of would also produce similar issues. Reuse is the process to decide what history information the program should reuse and how to reuse the pre-computed information. Challenges in this field are 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 will apply to 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 gives will give the motivation behind the work while section 4 will formalizes the framework and Section 5 presents the challenges and solutions we explored in details. We present the experiment results in Section 6. Related work is discussed in Section 7.

2. BACKGROUND

Computation reuse across executions benefits programs with long computation time most, especially the converging algorithms that require multiple iterations to compute the desired results. Such a class of programs is essentially NP hard problems, which have no generic polynomial time algorithms. 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 is a highly lucrative area of research. Some of the motivations for this work may be listed as follows:

- Frequently Used: These algorithms are used frequently to tackle real world problems and thus even small improvements can have a significant impact in performance improvement.
- These algorithms also have wide areas of real world application ranging from machine learning to big data problems.
- 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 are 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 frame- work 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 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 reduce the dimensionality (feature count) of the data set drastically while preserving the meaning 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.
- **Normalization:** An important basis of calculations for similarity is that the data is represented by similar metrics, without which any such computations would essentially be meaningless. Thus an important aspect of our framework is normalization of data into a common representation which may thus be used as the basis of "similarity" calculations.
- **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 highest 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 is not offset by the run time of the algorithm used to find the best match from historical data.
- **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 realized 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.

- **Normalization**: The next aim was to represent the data selected in the above step in such a way that it may be compared to another data set. Thus comparison data sets needed to be normalized using a common metric.

- **Similarity Metric Calculation**: Once the data set had been minimized while keeping it's 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 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 we will make use of the reduced data set to estimate the best possible match and then proceed to generate the "Similarity Metric" so we may select the best possible match for historical reuse.

5.1 Feature Set Reduction And Normalization

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 step 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 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 normalized data set are arranged in descending order of variance. 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 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 5.1, we proceed to calculate the probability based similarity metric. We achieve this using Welch's Test for non-parameterized data for null hypothesis testing.
- Our assumed null hypothesis for any compared data sets is that both data sets are exactly similar to each other. The Welch test thus gives us a probability metric that states that the difference in data sets is due to chance. Thus higher the probability of the difference in data sets being up to chance, the better is the probability that the two sets would be similar.
- We take individual dimensions from the two data-sets and run Welch 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

Once we have computed the above mentioned values for our data-set, we store it in memory as objects. Each historical data objects holds its original data, PCA metadata (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 proceed to use the histogram made by the PCA data points for quick distance comparison. We take the highest populated top-k bins and do a distance computation with points in similar bins in other data sets. 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 pre- computed and stored.) We now use the data set with the highest probabilistic similarity to out data set for computation reuse.

6. FINAL ALGORITHM FOR HISTORY REUSE IN K-MEANS

The final algorithm maybe divided into two major categories: Training and Run Time.

6.1 TRAINING DATA SETS:

1. PCA: For all data sets compute PCA and project data set points on thus computed eigen vectors.
2. Histogram Generation: Categorize PCA data for all historical data sets into bin-wise data for histograms and for use in distance computation.
3. Centroid Computation: Generate centroids for all history data sets using k-means++
4. Relative Ranking: Use Welch Test to rank all history data sets wrt to each other and store in decreasing order of similarity. (Note: Will be used in run time historical reuse data set computation.)

6.2 COMPUTING BEST MATCH FOR CURRENT DATA SET (Run Time)

1. PCA: Compute PCA for current Data set and project data set points on thus

computed eigen vectors.

2. Histogram Generation: Categorize PCA data for all historical data sets into bin-wise data for histograms and for use in distance computation.
3. Find Best Match (Screening): Done on PCA Data
    a. Take top-k most populated bins for current data set and select corresponding top-k bins from all data sets.
    b. Find ED for these data points between current data set and all historical data sets.
    c. Choose Historical Data Set with minimal distance as initial "Best Match".
4. Find Best Match (Similarity Metric => Probabilistic Score): Done on PCA Data
    a. Find "Similarity Metric" between current data set and above chosen "best match" data set using Welch's Test for all dimensions of both data sets.
    b. Similarly find "Similarity Metric" for top 3 relatively ranked data sets for current best match.
    c. Choose Data Set with highest Probability Score.
5. Use "Initial Centroids" (pre-computed in 6.1.3) from Best Match Historic "Data Set" and generate "Labels" by running K-Means initialization on PCA data for "Current Data Set".
6. Use above generated labels for k-means computation with actual data for Current Data Set.

7. OTHER METHODS USED:

ALGORITHMS	REASONS FOR FAILURE
1. Using Histograms on Real Data: • In this method I had initially used bin wise reduction on initial data and then used PCA for dimension reduction of these reduced data points for the calculation of eigen vectors and eigen values only. • I had then proceeded to run Student's T-Test for relative ranking in in training Run on real data. • For Run Time computation I had used only the distance between the eigen vectors for initial screening, choosing the data set with smallest difference in eigen vectors as initial Best Match data set. • I had then proceeded to Use Student T-Test on real data for computation of the probabilistic	1. Data was not normalized thus computation would not be correct. 2. Student T-Test worked only with Gaussian Distributions. Garbage value of non Gaussian Data. 3. Eigen Vectors of two data sets may be orthogonal yet PDF may be close enough such as to generate similar clusters. 4. Use of labels for direct initialization was flawed in the sense that if the data points were jumbled they would produce the wrong order of labels thus still providing a bad match. 5. Bin Wise distribution of data points was an expensive

metric using all dimensions for the computation of the same. • Python Numpy Libraries had been used for Student T-Test computation. • Centroid Computation during training run was done on real data of historical data set instead of PCA data. • Labels for Best Match historic data set were used as-is for current data set.	operation and computation overhead increased with increase in dimensionality of data. 6. Student T-Test had to compute for multiple dimensions of data and was a time expensive computation.
2. Custom CPP Implementation for Welch's Test • Implemented Custom Cpp implementation for Welch's Test to over come overhead created by using python libraries for it and calling python script from Cpp. • Removed the distribution of data into bins for data point reduction as it had a big overhead in computation and CPP Welch Test Libraries scaled well for larger data sets. • Changed to use of Welch's Test as compared to Student T-Test as Welch's Test works with non Gaussian distributed data as well as compared to Student T-Test which makes assumptions of data distribution being Gaussian in nature. • For Run Time computation I had used only the distance between the eigen vectors for initial screening, choosing the data set with smallest difference in eigen vectors as initial Best Match data set. • Centroid Computation during training run was done on real data of historical data set instead of PCA data. • Labels for Best Match historic data set were used as-is for current	1. Eigen Vectors of two data sets may be orthogonal yet PDF may be close enough such as to generate similar clusters. 2. Use of labels for direct initialization was flawed in the sense that if the data points were jumbled they would produce the wrong order of labels thus still providing a bad match. 3. Welch's Test still had to be computed for all data sets and was a time intensive calculation. 4. Centroid Computation was still done on non-normalized data and thus may not have translated well to another non-normalized data set. Restricted that data sets be similar in data type etc.

data set.	
3. Random Sampling for Order comparison. • Changed to use of PCA data for most computations. • Projected Historical data set points on Current Data Set eigen Vectors. Used distance between data points of Current Data Set and historical data sets by randomly sampling 10% of data sets against each other. Chose Data Set with minimum distance. • Welch's Test was still used across all dimensions of actual data Similarity Metric Computation. • Labels were used as-is for real data.	1. Use of labels for direct initialization was flawed in the sense that if the data points were jumbled they would produce the wrong order of labels thus still providing a bad match. 2. Random Sampling was a bad way to judge the order of the labels that would be generated by k-means for data set. 3. Eigen Vectors of two data sets may be orthogonal yet PDF may be close enough such as to generate similar clusters.
4. Use Of PCA Data for Welch's Test and PCA data for centroid computation in Training Run: • Used PCA data in training run for Centroid computation of historical data. • Used PCA data for Welch's Test based "Similarity Metric" computation. • Still used labels from best match historical data set as initialization for current data set. • Stopped projecting historical data sets data on current data set eigen vectors, instead used self projection which could be done offline. • Used Random Sampling for initial estimation of best match data set.	1. Use of labels for direct initialization was flawed in the sense that if the data points were jumbled they would produce the wrong order of labels thus still providing a bad match. 2. Random Sampling was a bad way to judge the order of the labels that would be generated by k-means for data set.

8. 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.

8.1 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.

8.2 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 highest concentration of the points while reducing the total number of points to a minimum.

8.3 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 processable format.
- Welch Test Implementation: No off the shelf Welch's Test Implementation was available for use and thus had to implement the tricky Welch's test algorithm in CPP.

9. 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 initialization 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 aver- age 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 compact- ness 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 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.

- Table 1: K-Means++ Performance:

Data Set Dims	Data Set Size	K-means++ Performance	K-Index
3	10 (Road Network: Reduced points)	2,337	3
3	10000 (Road Network)	28,889	80
10	20000 (Kegg Network)	106,245	80

T

a

T

- Table 2: K-means Custom Labels Performance:

Data Set Dims	Data Set Size	K-means HR Performance	K-Index
3	10 (Road Network: Reduced points)	6432	3
3	10000 (Road Network)	18,398	80

10	20000 (Kegg Network)	85,171	80

- Table 3: Tests for Group datasets (**Gas sensor array under flow modulation Data Set**)

Dims	Points	K-Index	K-means	K-means++	History Reuse K-means
11	14850	40	311569	214605	157035
11	14850	80	546,579	412,379	268,517.5

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++ and the original K-means algorithm. We can see also from above that as the cluster count increases our performance increases significantly.

(Note: 1. All time shown above are in microseconds. 2. Time taken in training history data sets is not taken into consideration, as it is a process, which would ideally be carried out offline. 3. At present the all historical data sets are used for comparison. Restricting the number of historical sets maintained for the sets will significantly improve performance.(TODO))

10. CONCLUSIONS

In conclusion we would like to say that probabilistic met- ric 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 applica- tions 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.

11. 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 realize 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 dataset may hold different importance. He was also there to correct me when initially I had taken a wrong 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.

12. REFERENCES

- OpenCV: docs.opencv.org/
- Welch's Test: https://en.wikipedia.org/wiki/Welch's_t_test
- 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
- C++ Documentations: http://www.cplusplus.com/doc/
- Community help (OpenCV)
- Boost Libraries: http://www.boost.org/
- **3D Road Network (North Jutland, Denmark) Data Set**
- **KEGG Metabolic Relation Network (Directed) Data Set**
- **Gas sensor array under flow modulation Data Set**