History Reuse K-Means: A new initialization methodology for the classical K-Means for Consistent Speedup

Shanil Puri Dept. of Computer Science
North Carolina State University
890 Oval Dr, Raleigh
North Carolina, USA
spuri3@ncsu.edu

Xipeng Shen Dept. of Computer Science
North Carolina State University
890 Oval Dr, Raleigh
North Carolina, USA
xshen5@ncsu.edu

ABSTRACT

In this data explosion era, there are myriad programs that 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.

This thesis explores how Computation Reuse can be implemented for augmenting the performance of some time consuming data analytics algorithms. Specifically, we developed a general framework for efficiently finding similar datasets and effectively reusing history. We demonstrated the large benefits (up to 20X in average case and upto 200X in best case) on K-means and SGD-based SVM algorithm.

1. INTRODUCTION

In this data explosion era, there are myriad programs that 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 work we will introduce a new probabilistic model of measuring data similarities. Based on these data similarities we will show how critical computations may be reused across data sets, saving energy and improving performance.

Our work will introduce a general architecture for comparing two or more data sets and get a probabilistic measure of similarity. We reason that if two data sets are similar, computation reuse from previous iteration of an algorithm for the similar data set will provide a good speed up in the execution of the algorithm for the current data set. We first define the architecture to compute the above-mentioned metric of similarity and then proceed to validate our above-mentioned stipulation for the K-Means and the SGD based SVM algorithms.

In these explorations, we found 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 metric we use to describe similarity between 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.

Since the method used to calculate distances between two data sets, should also be defined based on data features, the problem becomes even more complex. 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. The dilemma is the trade-off between the amount of computation reuse and the introduced overhead.

Reuse, thus is the process to decide what history information the program should reuse and how to reuse the pre-computed information. In this study we introduce the concept of Historical Computation reuse. This is an Intuitive Idea whose exploration for the family of algorithms mentioned above is spotted at best.

Computation Reuse is the process of deciding what history information the program should reuse and how to best reuse the pre-computed information. While the idea is simple and intuitive in nature, it promises big gains if correctly implemented with a wide variety of uses. In this study we give an empirical study, showing that effectively computation 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 our explorations, we found out challenges in 4 aspects: data feature abstraction, similarity definition, scalability, and reuse. *Data Features*, is the description of data sets. The major challenged we faced in *Data Feature Abstraction* was to reduce dimensionality (for the sake of optimization), while preserving the integrity of data. *Similarity definition* (distance between two data-sets) is the way we would like to describe similarity between two data set. The challenge was to come up with a meaningful metric to give an accurate prediction of suitability for reuse. We used *Data feature abstraction* and *Similarity definition* together for reuse instance selection from a program specific database. The selection of history data set is the most essential part of computation reuse because different data sets will lead to dramatically different computation times. 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 sets descriptions also should be defined based on the data features, the problem becomes even more complicated. *Scalability* issues are related with number of data set instances in database, size of the target data set, and its dimensionality. 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 tradeoff 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 of deciding what history information the program should reuse and how to reuse this 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. Our approach of building a general framework to work as a plug and play device makes the process even more complicated. In this work, we investigate multiple solutions to address each of the challenges, and come up a framework, which we will apply to two of commonly used algorithms, namely: ***K-Means(Lloyd's Clustering algorithm)*** and the ***Stochastic Gradient Decent based Support Vector Machine*** for our explorations and providing proof of effectiveness of History Reuse and to validate the efficiency and the effectiveness of our algorithm. The paper is organized in the following sections: Section *2* gives motivation, background and definitions of terms we use in this paper. Section *3* will give formal definition to the challenges we faced and our approach for coming to a successful conclusion. Section *4* will formalize the framework and give the formal algorithm for our architecture. We will also discuss the other approaches we used to come to our conclusions and give reasons for their failures. We present the evaluations for our algorithm in Section *5*.Future work is discussed in Section *6*, while Section *7* provides conclusion derived from our explorations.

2. MOTIVATION AND BACKGROUND

2.1 Motivation

Computation reuse across executions benefits programs with long computation time most, especially converging algorithms that require multiple iterations to compute desired results. This class of problems as such has no generic polynomial time algorithms, thus making history reuse for predictive initialization a good candidate for optimization. Another benefit may be improved accuracy. Through appropriate history result reuse, numbers of iterations of computations could be saved while the accuracy gets improved. The key point of computation reuse on different data is finding suitable history information to reuse. Thus, the above 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 features 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 effective computation reuse. Optimization of converging Algorithms such as K-means clustering and SGD based SVM are highly beneficial. Algorithms of this category are used frequently and have a multitude of applications in real world machine learning and big data processing. 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.
- *Wide Area of Application* These algorithms also have wide areas of real world application ranging from machine learning to big data problems.
- *Prime Candidates:* 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 and thus can be used as proof for the benefits of history reuse easily.

2.2 Background and Related Work

For most part of the generation and the testing of our *History Reuse architecture* we use the classical *K-Means algorithm (Lloyd's Algorithm)* [18] . We then test the architecture with the *Stochastic Gradient Decent based Simple Vector Machines* [29] to prove the global viability of our algorithm. The classic K-Means algorithm (Lloyd's algorithm) consists of two steps. For an input of 'n' data points of 'd' dimensions and 'k' initial cluster centers, the assignment step assigns each point to its closest cluster, and the update step updates each of the k cluster centers with the centroid of the points assigned to that cluster. The algorithm repeats until all the cluster centers remain unchanged in a single iteration. Because of its simplicity and general applicability, the algorithm is one of the most widely used clustering algorithms in practice, and is identified as one of the top 10 data mining algorithms (Wu et al., 2008). However, when n , k, or d is large, the algorithm runs slow due to its linear dependence on n, k, and d. There have been a number of efforts trying to improve its speed. Some try to come up with better initial centers (e.g. K-Means++ [1] or parallel implementations [2]. This thesis will look to present more on this approach by exploring an avenue not much pursued before, namely historical data set cluster center reuse for K-Means Initialization. Prior efforts in this direction include: K-Means++(Arthur and Vassilvitskii, 2007; Bahmani et al., 2012), K-Means Initialization Methods for Improving Clustering by Simulated Annealing (Gabriela Trazzi Perim et al. 2008), an optimized initialization center K-Means clustering algorithm based on density (Xiaofeng Zhou et al. 2015). These prior methods, while having made a significant contribution, have failed to replace the Lloyd's algorithm which still remains the dominant choice in practice exemplified by the implementations in popular libraries, such as GraphLab (Low et al.), OpenCV, ml-pack (Curtin et al. 2013) and so on. Previous implementations that have tried to optimize the selection of the initial centroids based only on the current data set. For instance, the original K-Means Algorithm (Lloyd's algorithm) chooses k cluster centers randomly from the points available in the current data set. The K-Means++ [1] further optimizes this by taking steps to increase probability of getting good starting points by first choosing a random

centroid and then proceeding to choose the furthest possible centroid from last chosen centroid iteratively, doing this for each centroid computation. The approximation method for initialization [6], aims to approximate the selection of centroid, but this approach produces clustering results different from the results of the standard K-Means. The above algorithms as can be seen work only on the current data set at hand. No prior work has directly tried to systematically exploit historical data for computation, which forms the basis of this thesis. This work will introduce and formalize the History Reuse Architecture. We will then use *History Reuse* to introduce a new Initialization methodology for the K-Means algorithm: "Historical dataset center reuse for K means initialization", an enhanced K means implementation which aims to optimize the K means algorithm by aiming to choose the best possible starting points for the K-Means [18] for faster convergence. Since the only modification that are being made are in the initialization step of the algorithm it stands to reason that the algorithm would continue to uphold the same standards as the standard K-Means algorithm. We will also use our History Reuse Architecture and test it for SGD based SVM [29] to prove the global application of our generic architecture.

While history reuse has been prevalent and an area of great exploration as of late, this approach of utilizing offline computed training data for History Reuse is unique and as such has not been explored. History Reuse though has been seen in myriad other works including Yinyang K-Means [9] where in pre-computed geometrical information for distances for points from centroids is stored and reused in the multiple iterations of the algorithm in a single run. It makes use of this information for both initialization and re-labeling of points by reusing distance computation information. Similarly, the K-means optimized by Elkan [11] and by Drake and Hamerly [10] also computes the triangular inequality and uses it to optimize run time by minimizing computations of distance per iteration by reusing the precalculated distance bounds with changes in the limit of the distances maintained in history on a per iteration basis.

Compiler studies have also had a lot of work done in the area of history reuse for code optimization such as the "Automated Locality Optimization Based on the Reuse Distance of String Operations" [23] which aims to use call context on Cache hits for optimal use of L2/L3 caches. These and other works [14] [3] [4] [13] have often used history reuse in the optimization at the micro or program level.

Some of the other specific works in optimizing the K-Means algorithm [18], which is also a product of this thesis, have used myriad approaches for optimizing the K-Means algorithm. Some optimizations use approximation methodologies ([7]; [24]; [21]; [12]; [28]) while other try to speed up K-Means inherently, while trying to maintain the semantics of the original algorithm. An example of the latter is to speed up the algorithm using KD-Trees [20] [16] which show promise for smaller cluster sizes but do not perform as well for larger cluster sizes.

With our proposed solution we extend this approach to a macro level choosing to look at the problem at the data level as compared to the optimizations done at program level. Since our approach is program independent to a large extent, our algorithm will work as a generic framework for all data related optimizations and can be directly combined with any program level optimization to achieve further speedups. For instance, we may use our approach in conjunction with the above mentioned Yinyang K-Means implementation to optimize the initialization as well iteration time for a single run of the algorithm. This way we stand to gain the best of both worlds by combining both program level and data level optimizations.

2.3 Important Terminology

For the propose of our discussion we assume data to be represented in a *2-D matrix* where in each *row* represents a *single point* in a data set while the *columns* are used to represent the *feature set* of each point. The following are important terms for this work and are used through out the later chapters:

- **Data Source**: These are the actual sources of Data Set repositories from which we source our data for testing purposes.
- **Data Set**: Data on which the actual algorithm is run after dividing the data from the *Data Source*. Each *Data Set* is built up of multiple *Data Items or Data Points*.
  - *Current Data Set:* Represents the data set on which the current iteration of the algorithm is to be run.
  - *Historic Data Set:* Data Set present in the History Data Base for which results have been computed in previous iterations of the algorithm, making the data set one of the viable candidates for *History Reuse* for *Current Data Set*.
  - *Data Item / Data Point:* Single row in Data set. Represents a single data point in the larger *Data Set*.
  - *Dimensionality(d):* Number of columns used to represent a single *Data Item*.
- **History Data Base:** Data base for storing all data sets that may be candidates for *History Reuse* for *current data set* and for which final results have been calculated in previous iterations of the algorithm.
- **Cluster Count (k):** The total number of cluster in which the *Data Set* is to be clustered when using the *K-Means Algorithm*.
- **SGD based SVM**: Stochastic Gradient Decent Based Simple Vector Machine. [29]
- **PCA:** Principal Component Analysis. [26] [19]

3. CHALLENGES

History Reuse seems an intuitive solution to many problems. While the concept in itself is simple enough: reuse some computations for previous runs for current run of an algorithm, the implementation of the same provides quite a challenge. Some of the major challenges faced are: Data Feature Abstraction, Similarity definition, Scalability, and Reuse.

3.1 Data Feature Abstraction

Each *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 challenge faced by us was ensuring that we use the most important features only for computation of *History Reuse Data Set.* Use of too many features for the computation of our Historic Data Sets may affect efficiency, while on the flip side, use of too few data set features may result in the loss of the meaning of the data set itself thus invalidating its candidacy for History Reuse. This thus was one of the major challenges face by us in the computation of Historical Data Sets for Computation reuse.

3.2 Similarity Computation

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. Defining such a metric though poses its own challenges, namely: data sets used for comparison may have different scale (may not be normalized), may not have the same important features or may not be along the same axis of projection. These and more issues make the definition of a Similarity metric a difficult task. Failure to solve all the challenges mentioned above would lead to the failure of our *probabilistic similarity metric.* Following sub sections will shed a little more light on the challenges in :

3.2.1 Non Uniform Scale

Data sets present in the *History Data Base* and the current data set may not have the same scale, i.e. distributions and variances. This essentially means, any similarity computations between the two data set would essentially be meaningless as computation reuse across such data sets may be impossible. This thus presents the first challenge of normalizing the data sets on to the same plane to provide a common platform for similarity computation.

3.2.2 Different Feature Sets: Feature Set Abstraction

Another major challenge faced is that the most influential features present in *historical data sets* and the current data set may be very different. Since the class of algorithms targeted by our algorithm is often highly dependent upon the feature sets for final results, the matching of data sets based on only the most important features becomes imperative for good reuse results. This, thus presents the challenge of analyzing both the current data set and the historical data set for the extraction of the most important features. This must be done in real-time and must be both efficient and effective. as mentioned in *section 1*, we must also ensure the optimal use of the features to ensure meaning full comparisons.

3.2.3 Rotated Data

Often the axis of projection for current and historical data may often in different planes, and while the distributions may be similar for data, their being on different planes altogether makes similarity computation challenging. Thus one of the major challenges in computing similarity was to ensure both *historic and current data set* be in the same plane. This again must be done at run time, so as to ensure that both the historic and current data and the historic data are on the same plane (*plane* for current data is known only at run time.) This further poses the challenge of an efficient method for the planar normalization of the data sets being compared.

3.3 Efficient and effective comparison

The above challenges seen clearly pose the challenge of efficiency and affectivity. The loss of either of the two would essentially entail the failure of any proposed algorithm. Efficiency is desired since a lot of the above challenges must be solved in real-time as they require analysis of the data set used in the current iteration of the algorithm. Effective is desired since a non-effective data set may provide a bad historical data set for computation reuse. Our experimentation has shown us, use of badly matched historical data sets tend to cause severe punishment in terms of both run time and quality of results for our tested algorithms.

3.4 How to use the historical datasets

The last challenge faced is how to reuse computation from the selected history data set itself. For example in case of the K-Means algorithm we get both the labels, as well as cluster centroids for historical data sets. The use of labels though quicker in initialization may produce different results depending upon the order of the data points in the data set. Another example maybe, the data item count for historical data set may be different from the current data set and thus using labels directly may simple cause the failure of the algorithm to divide the data into the required number of clusters. On the flip side too few data items in historic data set may cause some data items in current data set to not being assigned any cluster at all. The reuse of computed centroids to regenerate labels on the other hand, while slower, will always ensure the correct label initialization irrespective of the order of the data points in the data set, or the count of the required data labels etc. Thus we can see that the correct use of Historical data is an equally challenging problem.

4. HISTORY REUSE ARCHITECTURE

4.1 Overview

For any framework to be successful globally in the computation of History Reuse needed to effectively and comprehensively solve all the challenges motioned above. In addition it also needed to be directly translatable and not be over dependent upon the algorithm in question. Keeping the above framework requirements in mind we finally settle on the framework components as follows:

4.1.1 Feature Set Reduction And Normalization

The aim of this section of the framework is to capture the most relevant features of the data set (features along which maximum variation is seen). 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 losing the meaning of the data set. This is achieved by the use of Principal Component Analysis (PCA) [26] [19]. 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 needing consideration in our data set.

4.1.2 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*** [27] for non-parameterized data for null hypothesis testing. In statistics, Welch's t-test (or unequal variances t-test) is a two-sample location test, and is used to test the hypothesis that two populations have equal means. Welch's t-test is an adaptation of Student's t-test, and is more reliable when the two samples have unequal variances and unequal sample sizes. In our algorithm, we use it to approximate the degree of similarity of means between our current data set and our historic data sets.

- 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 based on the evaluation of the difference in their means. 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 choose the Welch's test because it gives more accurate results as compared to the Student T-Test for data sets whose distribution in non-Gaussian and whose sample size may be different.

- 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.

4.1.3 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 object 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.

4.1.4 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 buckets and do a distance computation with points in similar buckets in other data sets. This gives us one candidate for History Reuse. Another candidate is calculated by computing distance as mentioned above but in this case instead of taking highest populated buckets from History Reuse candidates, we now use buckets having closest ranges to the selected buckets for current data set. This is done to estimate similarity in distribution. Doing this we now get another candidate for history reuse. 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 approximate closest match.

- We now use the above computed candidate historical data sets 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 (pre-computed and stored.) We now use the data set with the highest probabilistic similarity to select data set for computation reuse. This step gives us the final candidate for History Reuse.

4.1.5 Algorithm: HRu Architecture

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

- **Training Algorithm**: The training algorithm is used to prepare and store data so as to have highest availability of reuse components for all candidates. This part of the algorithm is carried out offline and thus does not affect the run time of the algorithm when run for most current data set.

- **Run Time Algorithm:** The run time algorithm is responsible for choosing the best match historical data set to be used for Computation Reuse. This part of the algorithm is online, thus it has a direct impact on the run time for the algorithm when run for current data set. This part of the algorithm must be quick and must ensure that the computation time required for historical data set selection not overshoot the time for benefits gained by such history reuse.

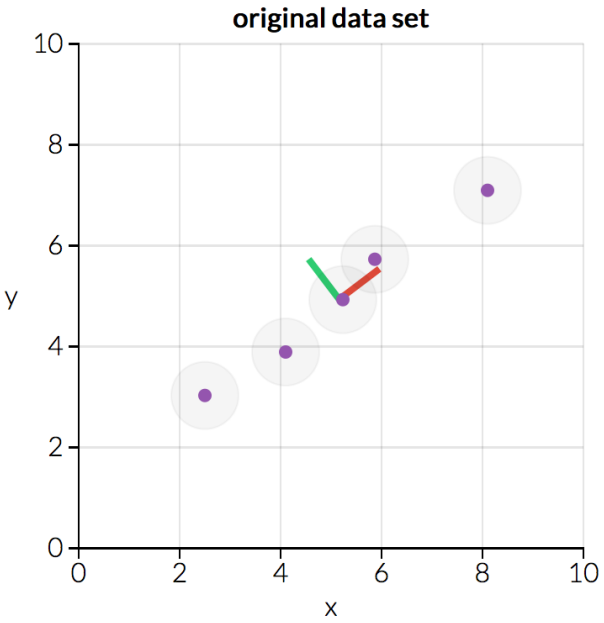

Figure 1: Example: Real Data with 2 Dimensions [19]

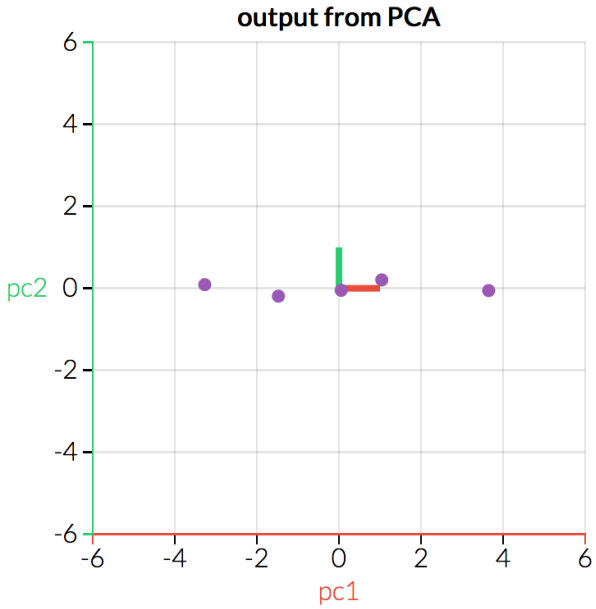

Figure 2: Example: Data Projected on Eigen Plane [19]

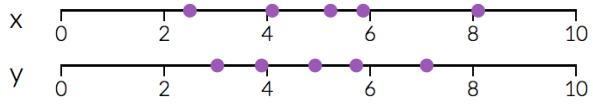

Figure 3: Example: Variance for Real Data on X and Y axis individually

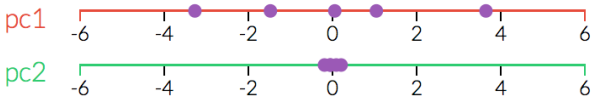

Figure 4: Example: Variance of Data on PCA axis individually

Both the training algorithm and the run time algorithm use some complex methodologies to ensure the best possible approach for historical data set selection. These are defined in the following sub sections.

4.1.6 *Principal Component Analysis (PCA)*

PCA [26] [5] [15] and project data set points on thus computed Eigen vectors. PCA essentially extracts the top n most influential *features* of a data set. For our algorithm we choose the **top 3 principal components.** PCA returns Eigen vectors for the three chosen principal axis (or components) for our data set based on maximum variance for each feature set (represented by a single column). We then proceed to project the data onto the new Eigen plane using the above computed Eigen vectors. This step reduces the dimensionality of our data into a fixed three-dimension space. **figure** 2 shows an example data distribution in a 2-D plane and **figure** 3 shows the same data projected into a 2-D Eigen plane. Since Eigen planes are always coplanar, this thus normalizes the data along the same axes.

We can also see from figure 3 that maximum variance of data is along the **pc1 axis**. This makes the **pc1 axis** the 1st principal component of the data. Using this property of PCA we can choose the most important *features* only from a data set while excluding the less important *features*. We see from **figure** 5, real data needs both the **x and y plane** to represent the features (spread) of the data. On the other hand in **figure ??** we can see that most of the variance for the data items has been condensed with negligible variance along the **pc2 axis**. All features of the data can now be abstracted onto the **pc1** plane. This method is the method we use for data abstraction while maintaining the features of the data set.

4.1.7 *Histogram Generation*

Once the dimensionality of the data has been reduced using PCA, we need to reduce the point count ensuring we use the most dense distribution areas for data set comparisons. We tackle this problem by the use of Histograms. Histogram is then generate on a per dimension *(di)* basis with each dimension having 16 or 32 buckets with ranges computed as follows :

$$
\begin{aligned}
&bucket\_count = 16 \; or \; 32 \\
&dim\_bucket\_range_{di} = max_{di} - min_{di} : \forall \; di \; in \; (1 \; .. \; 3) \\
&bucket\_size_{di} = dim\_bucket\_range_{di} \; / \; bucket\_count \; : \\
&\forall \; di \; in \; (1 \; .. \; 3) \\
&bucket\_range_j = [(bucket\_size * bucket_j + min_{di}), \\
&(bucketSize * (bucket_j + 1) + min_{di}))]; \\
&\forall \; j \; in \; (1 \; .. \; n)
\end{aligned}
\tag{1}
$$

This will give us the buckets with their ranges for all dimensions. We then proceed to generate *three* histograms. Each data point is put into a bucket based on its value for

that particular dimension. *E.g.:* let *three points* be represented in a 3-D Eigen plane by coordinates as follows:
$p1 = (1,\ 2,\ 3)$ $p2 = (-1,\ 0,\ 1)$ $p3 = (2,\ 3,\ 4)$
Now, let there exist 2 buckets per dimension with Ranges as follows:
$b_{11} = [-1,\ 2)$ $b_{12} = [2,\ 4)$ $b_{21} = [0,\ 2)$ $b_{22} = [2,\ 4)$ $b_{31} = [1,\ 3)$ $b_{32} = [3,\ 5)$

Using the above points and bucket ranges we will now generate **3 Histograms**(one per dimension) with **2 buckets per histogram.** The histograms may be represented as follows:

- Histogram for first dimension:
  $h\_points_1 = b_{11} : (p1,\ p2);\ b_{12} : (p3)$;
- Histogram for second dimension:
  $h\_points_2 = b_{21} : (p2);\ b_{22} : (p1,\ p3)$;
- Histogram for third dimension:
  $h\_points_3 = b_{31} : (p2);\ b_{32} : (p1,\ p3)$;

Thus we can see from above example how each item becomes a part of a bucket if its coordinate for the given dimension lies in the bucket range for that dimension.

4.1.8 Similarity Metric Computation

We now use the Welch's Test [27] [25] to compute the probabilistic metric for similarity and 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.) Welch test is computed for each dimension of the two sets being matched and the summation of the score for all three dimensions is as Similarity score for the two data set. Data sets are then ranked with each other based on the similarity score. Higher score means a better and match and vice versa.

4.2 Training Algorithm

This part of the algorithm is used to prepare and store data so as to have highest availability of reuse components for all candidates. We try and solve the challenges mentioned in 3 using the methods mentioned in 4.1. Let our history database consist of "n" data sets named: $HD_1, HD_2, ...HD_n$. The following Steps are done for each HD_i for (i in 1 to n) :

- **PCA:** Compute Eigen vectors for HD_i and project data on the Eigen vectors to normalize in Eigen plane and reduce dimensionality as shown in 4.1.6. We reduce all data into 3-D space for comparison purposes.
- **Histogram Generation:** Categorize PCA data for HD_i buckets for histograms as shown in 4.1.7. One histogram is generated per dimension of data, thus we have a total of 3 histograms for each data set. Each histogram comprises of a total of 32 buckets.
- **Relative Ranking:** Use Welch Test to rank all history data sets wrt to each other. Data sets are stored in decreasing order of similarity as shown in Algorithm 1. (Note: Will be used in run time historical reuse data set computation.)
- **Reuse Data for Data Set:** Run Algorithm of choice for Data Set and store *Computation Reuse Data.* E.g.: In case of the K-means algorithm we choose the *final centroids* computed by the K-Means Algorithm for current data set.

Algorithm 1 History Data Sets- Relative Ranking

```
1: procedure RankHistoryData(History_Database)
2:     relative_rank_map = ORDERED_MAP(Similarity_Score, Data_Set)
3:     score_Final = 0
4:     for each HD_i in History_Database do
5:         for each HD_j in History_Database != HD_i do
6:             score_Final = ComputeSimilarity(HD_i, HD_j)
7:             relative_rank_map.insert(score_Final, HD_j)
8:         end for
9:         (HD_i).Relative_Ranking = relative_rank_map
10:    end for
11: end procedure
```

Algorithm 2 Similarity Metric Computation

```
1: procedure ComputeSimilarity(D_S_Cur, D_S_Hist )
2:     score_Final = 0
3:     Data_CurDS = TOP_3_BUCKET(D_S_Cur)
4:     Data_HistDS = TOP_3_BUCKET(D_S_Hist)
5:     for each dim_i in Dimensions(D_S_Cur) do
6:         scr += CalcWelchScr(Data_CurDS.colAt(dim_i),
7:
8:         Data_HistDS.colAt(dim_i))
9:     end for
10:    return scr
11: end procedure
```

4.2.1 Computing Best Match for Current Data-set (Run Time)

This is the part of the algorithm that is actually responsible for the selection of the best match from the history data sets for the initialization for the current data set. This needs to be quick and accurate. Quick so that the overall overhead introduced by the selection process must not overshoot the total runtime improvements it may help us get, while the accuracy directly effects both the run time and the quality of clusters produced.

The two parts of the Matcher and Selector part of our algorithm may be defined as follows:

- **Screening:** This is the process where we select our initial candidates for the next step of our selection algorithm. This part of the algorithm essentially estimates the similarity in distribution for the current and corresponding historical data sets using the Histograms generated in the previous steps. Variance Similarity is estimated as follows:
  1. Take top-k most populated buckets for current data set and select corresponding top-k bins from all history data sets and top-k buckets with closest min and max to current data-set top-k bins.
  2. Find ED for these data points between current data set and all historical data sets.

3. Choose Historical Data Sets with minimal distance as initial "Best Match" for both top-k buckets by rank and top-k buckets by range.

Pseudo Code for an understanding of how our *screening algorithm* runs can be seen in Algorithm 3.

Algorithm 3 Screening Algorithm

```
1: procedure SCREENING–PRIMARYCANDIDATESELECTION
2:     least_distance_rank = MAX
3:     least_distance_range = MAX
4:     dim_distance_rank = 0
5:     dim_distance_range = 0
6:     HD_Rank_selected
7:     HD_Range_selected
8:     for each HD_x in History Database do
9:         for each Dim_i where i in 1 .. 3 do
10:            for each Bucket_j in Top 3 Buckets(Dim_i) in
    Decreasing Order of Item Count for current data set do
11:
12:                for each pt_cd and pt_HDi in
    points(Bucket_j) do
13:                    distance_rank = dist(pt_cd, pt_HDi);
14:                end for
15:                Bucket_k = Bucket in
    HD_i where range(bucket_k) ~ range(bucket_j)
16:                for each pt_cd in points(Bucket_j)
17: and pt_HDi in Bucket_k do
18:                    distance_range = dist(pt_cd, pt_HDi);
19:                end for
20:            end for
21:            dim_distance_rank = dim_distance_rank +
    distance_rank
22:            dim_distance_range = dim_distance_range +
    distance_range
23:        end for
24:        if dim_distance_rank ¡ least_distance_rank then
25:            least_distance_rank = dim_distance_rank;
26:            HD_Rank_selected = HD_x;
27:        end if
28:        if dim_distance_range ¡ least_distance_range then
29:            least_distance_range = dim_distance_range;
30:            HD_Range_selected = HD_x;
31:        end if
32:    end for
33:    return [HD_Rank_selected, HD_Range_selected]
34: end procedure
```

- **Find Best Match (Similarity Metric =¿ Probabilistic Score):** One we have found the best match candidates from out screening step, we simple calculate the Welch Test score for each dimension (dim_i) for each pair of current data set and chosen historic data sets.

  1. Find "Similarity Metric" as expalined in subsection 4.1.2 and between current data set and above chosen "best match" data sets using Welch's Test for all dimensions of all three data sets as shown in Algorithm 2.
  2. Similarly find "Similarity Metric" for top two relatively ranked data sets for current best match and top one for second best match.

Algorithm 4 Best Match Selection Algorithm

```
1: procedure BESTMATCHSELECTION
2:     primary_candidates =
   Screening–PrimaryCandidateSelection
3:     BM_score_final = INT_MIN
4:     BM_score_cur = CompSim(DS_cur,
5: primary_candidates[0])
6:     BM_data_set = primary_candidates[0]
7:     BM_score_cur = CompSim(DS_cur,
8: primary_candidates[1])
9:     if BM_score_cur > BM_score_final then
10:        BM_score_final = BM_score_cur
11:        BM_data_set = primary_candidate[1]
12:    end ifPC = primary_candidates;
13:    for each HD_x in TOP_3_rel_rank(PC[0]) do
14:        BM_score_cur = CompSim(DS_cur, HD_x)
15:        if BM_score_current > BM_score_final then
16:            BM_score_final = BM_score_cur
17:            best_match_data_set = HD_x
18:        end if
19:    end for
20:    for each HD_y in TOP_2_relative_rank
21: (PC[1]) do
22:        BM_score_cur =
23: CompSim(DS_cur, HD_y)
24:        if BM_score_cur > BM_score_final then
25:            BM_score_final = BM_score_cur
26:            BM_data_set = HD_y
27:        end if
28:    end forreturn BM_data_set
29: end procedure
```

3. Choose Data Set with highest Probability Score.

Pseudo Code for an understanding of how our *final selection algorithm* runs can be seen in Algorithm 3.

- **Reuse Computations:** In this section we use the computational data we had stored for history reuse in the training run for the historical data set selected. For instance, in the case of the K-Means algorithm, we now use the centroids from the chosen historical data set that were computed during the training run for historical data.

4.2.2 Discussion

Our final algorithm was reached at after a fair few trials and errors. Some of the major approaches used apart the latest approach for major challenges may be defined as below:

4.2.3 Computation on Real Data and Incorrect Screening algorithm:

Implementation

- In this method I had initially use bin wise reduction on initial (real) 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 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 metric using all dimensions for the computation of the same.
- Python Numpy Libraries had been used for Student T-Test computation.
- History Reuse component Computations during training run was done on real data of historical data set instead of PCA data.

Reasons for failure

- Data was not normalized thus computation would not be correct.
- Student T-Test worked only with Gaussian Distributions. Garbage value was returned for non Gaussian Data.
- Eigen Vectors of two data set may be orthogonal yet PDF (probability density function) may be close enough such as to generate similar clusters.
- 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.
- Bin Wise distribution of data points was an expensive operation and computation overhead increased with increase in dimensionality of data.
- Student T-Test had to compute for multiple dimensions of data and was a time expensive computation.

4.2.4 *Custom CPP Implementation for Welch's Test*

Some of the earlier seen issues were corrected in this section as I noticed that a lot of the run time improvement was overshadowed by the time taken for choosing the historical data set. I also noticed that the Student T-Test was not reliable for non-Gaussian data and failed completely in case of different data set sizes. This iteration was also mainly about ensuring quicker run time for the selection algorithm, so incremental updates were made to optimize the same. Some of the updates made may be enlisted as follows:

- 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 data set.

4.2.5 *Random Sampling for Order comparison.*

Experimentation on the approach showed non-reliable best match selection. Also I saw that often the best match might not even yield best results. One of the major reasons for this was the use of the incorrect use of historical data. For instance, in case of the K-Means algorithm the use of labels instead of the computed centroids lead to dependence on the order of the items in the historic data set. While the two data set may be similar and produce similar clusters, History Reuse with this method would still fail as initially the data points in current data set may get labeled incorrectly. I also realized that comparison of non-normalized data was in its very essence. I tried to correct the above issues with the following methodology:

- Changed implementation 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. This was done to try to use Historical data set with most point order similarity (and thus generated label order similarity) in conjunction with overall data set similarity.
- Welch's Test was still used across all dimensions of actual data Similarity Metric Computation.
- Labels were used as-is for real data.

This implementation though corrected some of the above mentioned issues, it in its turn generated new issues:

- 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.
- Random Sampling was a bad way to judge the order of the labels that would be generated by K-Means for data set.
- Eigen Vectors of two data set may be orthogonal yet PDF may be close enough such as to generate similar clusters.

4.2.6 *Use Of PCA Data for Welch's Test and PCA data for centroid computation in Training Run:*

Experimentation still showed both the history reuse to be inconsistent and the overhead of computing historical data set was still quite large. I also realized that while Eigen Vectors of two data set may be orthogonal yet PDF might be close enough such as to generate similar clusters. To correct the above issues I used the following methods:

- 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.

The use of Labels was still flawed. Also the use of random sampling was not a very effective methodology for screening as it would often lead to the selection of bad candidates.

5. EVALUATIONS

5.1 Methodology

We have used leave-one cross testing for all experimentations. Most data sets for experimentation are stock databases available on the UCI machine-learning repository while some source data sets are Microsoft released source data sets for machine learning. These data sources are then divided into data sets. Depending on the size of the data source [17] we may have any where between 10 to 43 data sets, where in all save one are treated as historic data sets. We have also allowed all algorithms to converge to the same epsilon error rate thus ensuring the run time comparison for similar quality output for algorithms. All experiments have been carried out on Octa core Intel Xeon CPU E5-2650 Running Ubuntu Linux 14.04 LTS with 16GB of RAM.

5.2 Experiments

To demonstrate the efficacy and efficiency of our program we evaluate our approach by testing it on various large real world data-sets and compare our algorithm to two most frequently used and vastly accepted K-Means initialization algorithms: standard K-Means (Random Select initialization also known as Lloyd's Algorithm) and K-Means++ initialization (Weighed Probability based initialization based on squared of distance from initial randomly selected center). Both the above algorithms are implemented in the OpenCV library, which has been used as the standard library for Lloyd's Algorithm implementation. We run all three algorithms on the same data sets with the same error rate for convergence. We have also compared the historical data-set selected by our algorithm to all other data-sets available in history to show the accuracy of our algorithm in selecting the best available data-set.

Consistent Selection of Best Match Historical Data-set from History: The experiments run, show that our algorithm is able to consistently able to select one of the top 5 data-sets for history reuse from available historical data-sets. By consistent, we mean that our algorithm is able to select one of the top 5 available data sets with hit rate of above seventy percent across data sets irrespective of the size and the cluster count. This is expected as we aim to compare data set based on similarity while the cluster count is not taken into consideration in the matching process. ***Table 1*** shows the hit rate for our algorithm. We can clearly see from the table that our algorithm is able to select one of the top 5 best data sets for historical reuse from the available data sets consistently irrespective of the cluster count and the total historically available data sets. This thus validates our theory for probabilistic selection for matching data sets using the **Welch's Test** for null hypothesis testing for similarity of data sets. We are also able to see clearly that our selection criteria continues to perform well across a wide range of cluster counts and thus validates our hypothesis that data-set similarity should be used as the primary measure to gauge quality of data-set for history reuse without the explicit need for taking into account the total number of clusters *(k)* the data-set is to be clustered into. We have chosen to limit the total number of clusters to 256 so as to have meaningful clusters for data sets of relatively small sizes. Columns 3, 4 and 5 show us how well our data-set selection algorithm works across various cluster indexes

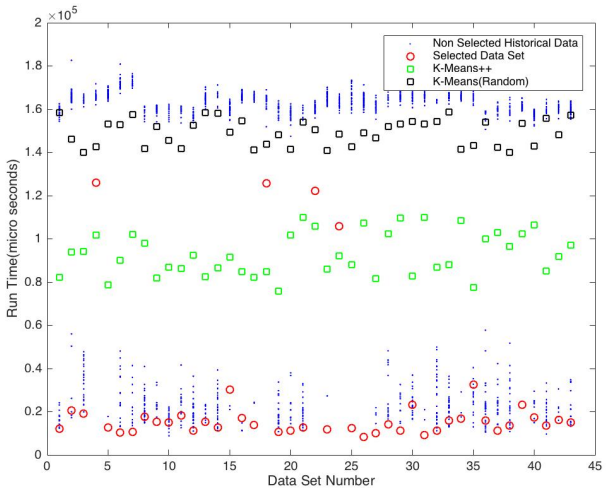

Figure 5: Hit-Rate (Road Network k=120)

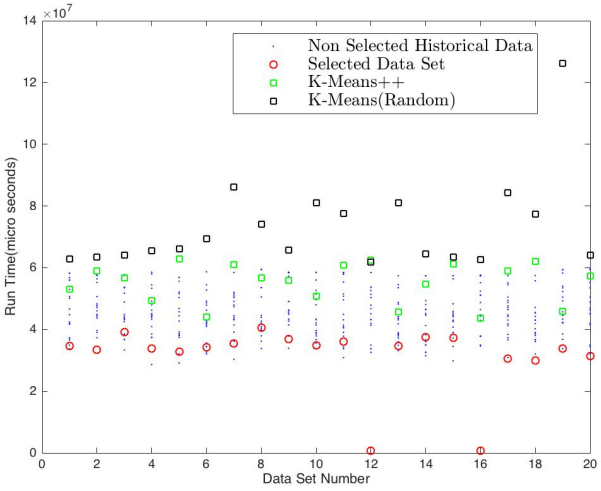

Figure 6: Hit-Rate (Caltec 101 for k=600)

Data-set Name	History Data-set cnt.	Cluster Count	Top 5 cnt	Hit Rate %
Road Network	43	40	42	97.67%
		80	41	95.34%
		120	43	100%
		240	43	100%
Kegg Network	10	40	8	80%
		80	9	90%
		120	8	80%
		240	8	80%
US Gas Sensor Data	36	40	28	77.7%
		80	29	80.5%
		120	29	80.5%
		240	28	77.7%
NotreDame	20	40	17	85%
		80	18	90%
		120	17	85%
		240	17	85%
Tiny	20	80	20	100%
		120	18	90%
		240	16	80%
		360	17	85%
		480	18	90%
		600	18	90%
Uk Bench	20	80	20	100%
		120	18	90%
		240	16	80%
		360	17	85%
		480	18	90%
		600	18	90%
Caltec 101	20	80	20	100%
		120	18	90%
		240	16	80%
		360	17	85%
		480	18	90%
		600	18	90%

Table 1: Selection Hit Rate K-Means History Reuse

and also how little variation is seen in the performance of the data-sets selected by our algorithm irrespective of the number of clusters *(k)*. We also show that our hit rate is a minimum of 70 percent (Some error is to be expected as the selection algorithm works on probabilistic model for making best guess.) Figures 6 and 7 show how our selected historical data set performs in comparison to all other data sets available for selection in our history database. We see from them, the accuracy of algorithm in consistently choosing one of the best match data sets from history with high accuracy (¿ 75%). We see that our algorithm chooses either the best match data set or data set close to best match. This proves the accuracy of our algorithm. **Consistent Scalable approach to Selection:** Our experiments also show that our method scales well with the increase in dimensionality of data *(d)*, the size of the data-set *(n)* and the cluster count *(k)*. ***Figures 8 and 9*** show our experimental results for various data sets with varying dimensionality and sizes. These figures clearly show that for any given data set the selection time is completely independent of the cluster count *(k)*. This is in complete contrast to both the *random K-Means (Lloyd's Algorithm)* and the K-Means++ algorithm where the initialization overhead is directly proportional to the cluster count *(k)* for the data set. Our experimental results also serve the purpose of showing us that our initialization methodology has an over head very small compared to the total run time of the algorithm even for the smallest of chosen cluster counts. This combined with the proven choosing of better initialization centroids (as seen in Table 2 and Table 3 which show an overall improvement in both run time as well as the number of iterations taken for the data to converge) gives us a win-win situation of a comprehensively better initialization methodology in comparison to both the previously mentioned methods.

Referencing the above mentioned tables and figures, we can also see that our selection mechanism takes only a very small percentage of the total run time for the algorithm for larger data sets (¡ 0.1%) and while it may be a significant part of the initialization for smaller data sets the quality improvement in the initialization centroids is sufficient to offset the run time for the algorithm to be a viable initialization methodology for all data sets of varying sizes. Though speed up using our methodology is seen in all cases, it is best suited for larger data sets, as in that case our overhead for history best match selection almost becomes negligible.

From ***Figure 10*** we can also see that even while retaining higher variances for historic data sets, no further speedup is seen in run time while the introduced overhead becomes a

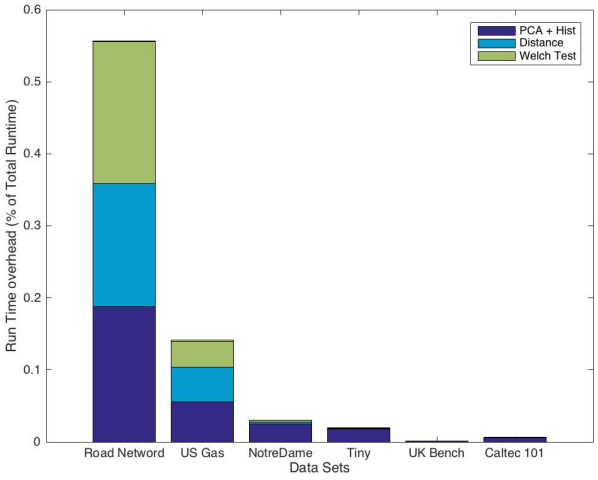

Figure 7: Best Match Selection Overhead (Maximum for Small k)

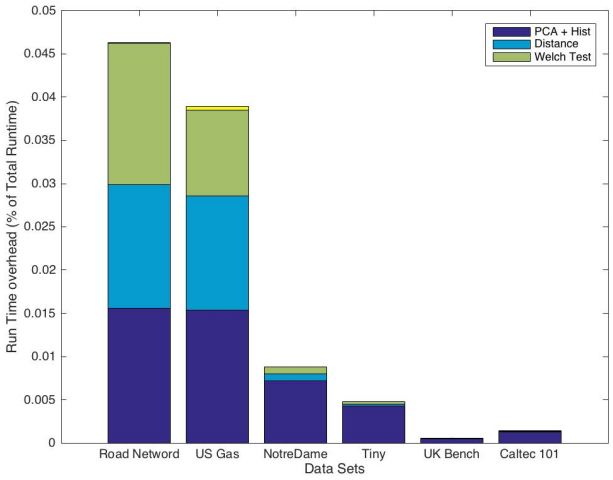

Figure 8: Best Match Selection Overhead (Minimum for Large k)

significant percentage for the total run time and may lead to further overheads as data set sizes increase.

Consistent Speed up for K-Means algorithm: The end requirement of any initialization algorithm is to ensure that we achieve a consistent speed up over previous approaches. While running our initialization algorithm for K-Means we were able to see consistent speed up in runtime of algorithm as compared to both K-Means and K-Means++ algorithm for all data sets across a wide range of cluster counts ranging from 40 to 600. This is though dependent on quality of historical data sets available. A bad match may lead to relatively bad starting points which in turn leads to an increase in time taken for convergence. Not withstanding the above argument we were able to see consistently good performance by our algorithm for real data sets. This is largely due to the normalization of the data sets using the PCA based approach and then using cluster information based on this normalized data rather than the actual data. This also removes the need for the data to have Gaussian distribution. This gives a higher chance of finding a better historical match, thus giving consistent speed up over various data sets with different dimensionality *(d)* and different cardinality *(n)* for varying cluster counts k.

Figures 13 and 14 shows the range of speed up across various data sets with varying degrees of freedom as well as different dimensionality for the same cluster counts. We can clearly see from the graphs that our algorithm consistently outperforms both the *random K-Means(Lloyd's Algorithm)* and the *K-Means++ algorithm* in run time performance.

Figures 11 and 12 depict the speed up seen by our algorithm over various cluster sizes for all data sets. We can see that the general trend is that speed up seen is in general higher for larger cluster count. This is particularly true because as the cluster count increases we see a speed up in both the initialization method as well as the actual run time because of the improvement seen in the initialization centroids.

Results seen in Figures 8 and 9 and Table 2 together also show us that the speed ups are consistent irrespective of the cluster counts *(k)* in terms of both the run time of the algorithms and the number of iterations taken for the convergence of the algorithm to similar error rate. This is again an intuitive results which confirms our hypothesis: a consistent selection methodology independent of the cluster index size would lead to sizable improvement in run time. This is so because run time improvement is seen in both the initialization overhead as well as the actual run time of the algorithm. As the cluster count *(k)* increases, the amount of initialization overhead for both the above mentioned initialization methodology becomes a significant part of the total run time for the algorithm.

Another interesting feature seen in the above box plots is the varying degrees of speed up seen for the various segments for same data sets. Looking at the speed up that *K-Means with History Reuse* achieves over the original *K-*

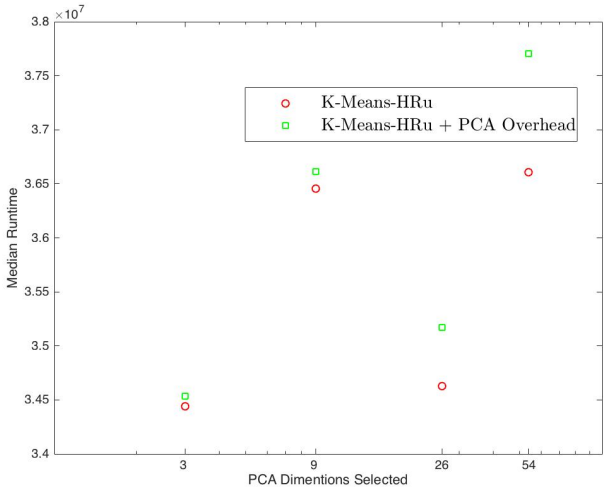

Figure 9: Performance for various PCA Dimension Count(Caltec 101; K = 600). Default PCA Dims: 3

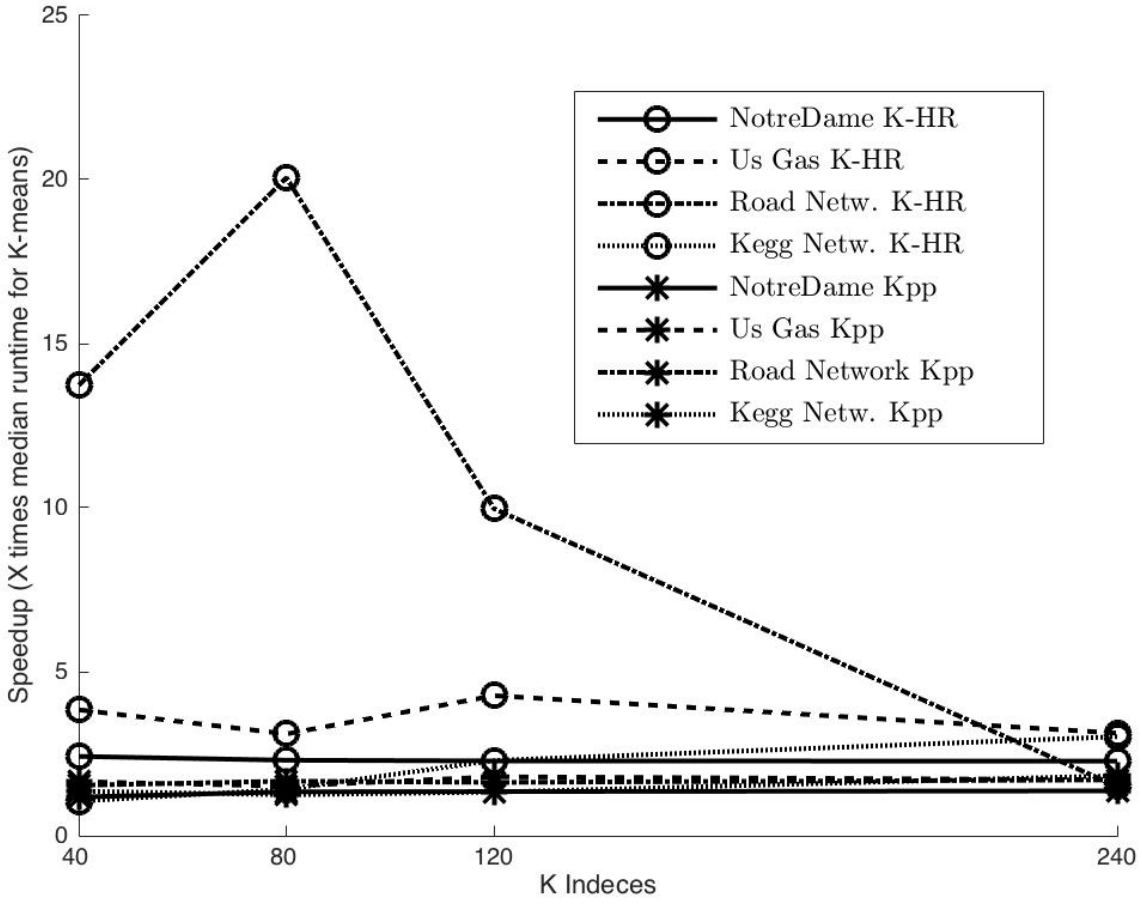

Figure 10: Median Speed up, Small Data (k=240)

Data-set	n	d	k	K-Means (Time μS)	K-Means++	K-Means-HRU
Road Net.	1.0E4	3	40	8.5E4	1.7X	6.7X
			80	1.6E5	2.4X	10.0X
			120	2.01E5	2.2X	10.0X
			240	2.4E5	1.8X	1.6X
Kegg Net.	2.0E4	10	40	1.9E5	2.5X	2X
			80	2.8E6	1.9X	2.2X
			120	3.9E5	1.6X	2.8X
			240	6.3E5	1.9X	3.2X
US Gas	1.45E4	13	40	3.2E5	2.4X	5.6X
			80	4.7E5	1.9X	4.4X
			120	5.2E5	1.9X	3.9X
			240	8.9E5	1.7X	4.3X
NotreDame	1.45E4	13	40	2.1E6	1.2X	2.4X
			80	3.8E6	1.3X	2.3X
			120	5.5E6	1.3X	2.3X
			240	1.02E6	1.4X	2.3X
Tiny	1.45E4	13	80	0.9E8	1.4X	2.8X
			120	1.3E8	1.6X	2.9X
			240	2.5E8	2.1X	3.8X
			360	3.2E8	2.0X	2.7X
			480	3.9E8	2.9X	4.1X
			600	4.3E8	2.9X	4.1X
Uk Bench	1.45E4	13	80	1.7E7	1.1X	2.7X
			120	2.3E7	1.2X	2.5X
			240	4.2E7	1.4X	2.5X
			360	4.7E7	1.25X	2.4X
			480	5.8E7	1.4X	2.1X
			600	6.8E7	1.3X	2.3X
Caltec 101	1.45E4	13	80	1.9E7	1.2X	2.6X
			120	2.2E7	1.1X	2.1X
			240	4.2E7	1.4X	2.3X
			360	5.3E7	1.4X	2.3X
			480	6.5E7	1.3X	2.2X
			600	7.3E7	1.3X	2.1X

Table 2: Runtime comparison: K-Means, K-Means++ and K-Means-HRU *(*All times in μS)*

Means (Lloyd's algorithm) we can see that the speed up is generally spread out over a larger range as compared to the performance of the *K-Means++ algorithm* as compared to *Lloyd's Algorithm*. This thus, brings to fore our above made assumption about the quality of history data set available in terms of the probability of similarity . Our experiments prove our intuitive hypothesis that probabilistically similar data sets will tend to be classed together into similar clusters, thus making them the best history match data sets. The larger variance seen in the box plots above is testament to this. Normalization using *PCA* of data sets, tends to mitigate this to a large extent thus providing us the ability to use history data not along same planes as well and thus providing a consistent speed up, but a better match of normalized data would provide even higher speed ups. We consistently saw that maximum speed ups are seen when the probability of similarity is higher. For segment with high probabilistic similarity we saw speed ups of up to and above ***100 X***.

6. FUTURE WORK

As we have seen in our experimentation the speedup in the run time for the algorithm is dependent of the quality (probability of similarity) of historically available data sets for selection. Better quality selection would lead to better speedups. Thus one future work could involve new methodology for maintaining best quality history data sets. Another possibility is to explore this History Reuse methodology for other algorithms. The general use of determining best match history data set is generic in nature (i.e. independent of the K-Means algorithm in itself) and thus can be extended to other algorithms such as the *SVD based SVM algorithm*. Only a minor change to the training step specific to K-Means is required in such a case. This thus can be used to prove the comprehensible usability of this algorithm for all algorithms in the same class. On the systems side the History reuse methodology can be used for memory data validation in Non Volatile Memory (NVM). This would prevent multiple expensive re-writes to NVM from main memory in case of already existing data.

Another area to explore may be the use of non uniform generation of hostogram for a better sample size selection in case of the screening algorithm. We chose not to explore this avenue in current work because our technique for unform buckets was showing significant performance gains. [8] shows an interesting way for generating non uniform histograms while still capturing sufficient desnity information such as to facilitate the selection of the density information more

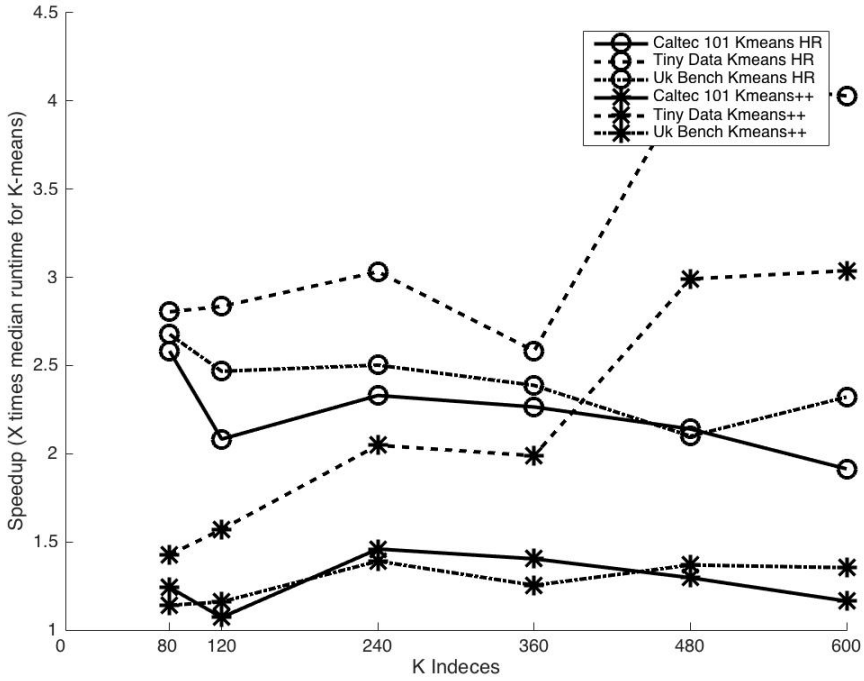

Figure 11: Median Speed up, Large Data (k=600)

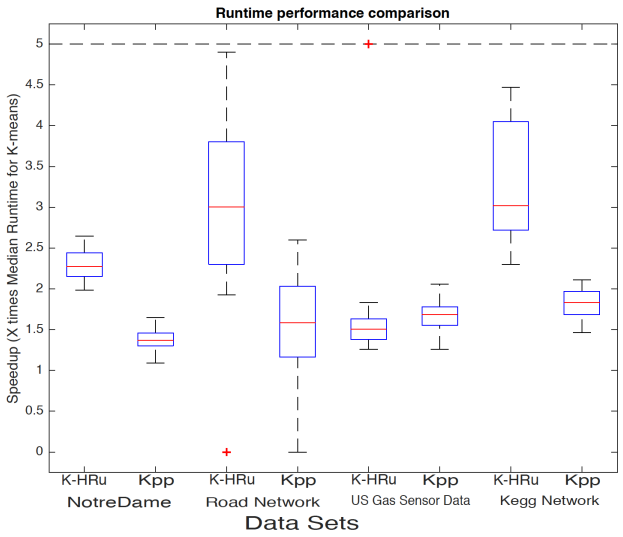

Figure 12: Median Speed up, Small Data (k=240)(*K-HRu: K-Means-History Reuse; Kpp: K-Means++)

succinctly. Integral Histogram [22] represent a new a way of capturing non uniform histograms in the cartesian space and my be extended to our use as well. Both these above mentioned methods may help with furhter improving the accuracy of the screening algorithm.

7. CONCLUSION

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

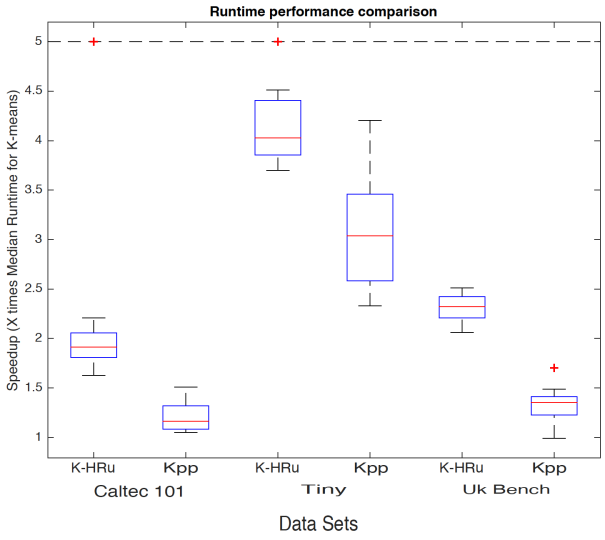

Figure 13: Median Speed up, Large Data (k=600)(*K-HRu: K-Means-History Reuse; Kpp: K-Means++)

Data-set	n	d	k	K-Means (Random)	K-Means++	K-Means-HRu
Road Net.	1.0E4	3	40	128	69	39
			80	119	52	27
			120	110	46	30
			240	102	79	82
US Gas	1.45E4	13	40	117	58	26
			80	111	59	32
			120	106	58	33
			240	100	55	37
NotreDame	1.45E4	13	40	162	142	69.5
			80	154	114.5	65
			120	132	101	60
			240	103	74	48.5
Tiny	1.45E4	13	80	195	164	108
			120	181.5	154.5	98
			240	168.5	120.5	87.5
			360	129.5	99.5	74
			480	119.5	87.5	67
			600	111.5	72	58
Uk Bench	1.45E4	13	80	274.5	191	85.5
			120	273.5	174.5	82
			240	251	121	75
			360	233	102	70.5
			480	216	79.5	63.5
			600	196	72	57.5
Caltec 101	1.45E4	13	80	219.5	175.5	84.5
			120	171	156.5	80.5
			240	178	116.5	74
			360	146.5	100	69
			480	137	90	61
			600	108	79	58.5

Table 3: Median Iteration count comparison for K-Means, K-Means++ and K-Means with History Reuse.

also seen based on the above results that the above mentioned algorithm is handy for all data sets irrespective if the dimensionality, cardinality or cluster count for the data set. Though the algorithm shows speedups for all data sets for all cluster indexes we see a substantially higher speedup for larger cluster counts as this results in a speedup in both, the run time of the algorithm as well as the initialization step for the algorithm. The initialization is sped up considerably by our approach and this plays a vital part in the final speedup for larger cluster counts as intialization becomes a significant amount of the run time for both the *Random K-Means* and the *K-Means++* algorithm. Since this can be used as a drop in replacement for any initialization method, it can be used with any flavour of the original K-Means Algorithm.t preserves the semantic of the original K-means. These appealing properties, plus its simplicity, make it a practical replacement of the standard K-means as long as viable historical data sets are available for reuse.

8. REFERENCES

[1] D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[2] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii. Scalable k-means++. *Proc. VLDB Endow.*, 5(7):622–633, Mar. 2012.

[3] R. Bodík, R. Gupta, and M. L. Soffa. Load-reuse analysis: Design and evaluation. In *Proceedings of the ACM SIGPLAN 1999 Conference on Programming Language Design and Implementation*, PLDI '99, pages 64–76, New York, NY, USA, 1999. ACM.

[4] R. Bodík, R. Gupta, and M. L. Soffa. Load-reuse analysis: Design and evaluation. *SIGPLAN Not.*, 34(5):64–76, May 1999.

[5] G. Bradski. Opencv library. *Dr. Dobb's Journal of Software Tools*, 2000.

[6] A. Czumaj and C. Sohler. Sublinear-time approximation algorithms for clustering via random sampling. *Random Structures & Algorithms*, 30(1-2):226–256, 2007.

[7] A. Czumaj and C. Sohler. Sublinear-time approximation algorithms for clustering via random sampling. *Random Struct. Algorithms*, 30(1-2):257–286, Jan. 2007.

[8] L. Devroye. Sample-based non-uniform random variate generation. In *Proceedings of the 18th Conference on Winter Simulation*, WSC '86, pages 260–265, New York, NY, USA, 1986. ACM.

[9] Y. Ding, Y. Zhao, X. Shen, M. Musuvathi, and T. Mytkowicz. Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup. In D. Blei and F. Bach, editors, *Proceedings*

of the 32nd International Conference on Machine Learning (ICML-15), pages 579–587. JMLR Workshop and Conference Proceedings, 2015.

[10] J. Drake and G. Hamerly. Accelerated k-means with adaptive distance bounds. In *in 5th NIPS Workshop on Optimization for Machine Learning. Dec 8th, 2012. Lake Tahoe*.

[11] C. Elkan. Using the triangle inequality to accelerate k-means, 2003.

[12] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. *SIGMOD Rec.*, 27(2):73–84, June 1998.

[13] J. Huang and D. J. Lilja. Exploiting basic block value locality with block reuse. In *High-Performance Computer Architecture, 1999. Proceedings. Fifth International Symposium On*, pages 106–114, Jan 1999.

[14] J. Huang and D. J. Lilja. Extending value reuse to basic blocks with compiler support. *IEEE Transactions on Computers*, 49(4):331–347, Apr 2000.

[15] Itseez. *The OpenCV Reference Manual*, 2.4.9.0 edition, April 2014.

[16] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):881–892, July 2002.

[17] M. Lichman. UCI machine learning repository, 2013.

[18] S. P. Lloyd. *Least squares quantization in PCM*, volume 28. 1982.

[19] Prinipal Component Analysis.

[20] D. Pelleg and A. Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '99, pages 277–281, New York, NY, USA, 1999. ACM.

[21] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2007.

[22] F. Porikli. Integral histogram: a fast way to extract histograms in cartesian spaces. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 829–836 vol. 1, June 2005.

[23] S. Rus, R. Ashok, and D. X. Li. Automated locality optimization based on the reuse distance of string operations. In *Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '11, pages 181–190, Washington, DC, USA, 2011. IEEE Computer Society.

[24] D. Sculley. Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 1177–1178, New York, NY, USA, 2010. ACM.

[25] J. Siek, L.-Q. Lee, and A. Lumsdaine. Boost random number library. http://www.boost.org/libs/graph/, June 2000.

[26] F. Song, Z. Guo, and D. Mei. Feature selection using principal component analysis. In *System Science, Engineering Design and Manufacturing Informatization (ICSEM), 2010 International Conference on*, volume 1, pages 27–30, Nov 2010.

[27] B. L. WELCH. The generalization of 'student's' problem when several different population varlances are involved. *Biometrika*, 34(1-2):28–35, 1947.

[28] G. Zeng. Fast approximate k-means via cluster closures. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 3037–3044, Washington, DC, USA, 2012. IEEE Computer Society.

[29] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 116–, New York, NY, USA, 2004. ACM.