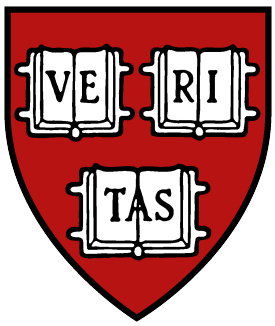
Turning Big Data Into Small Data: Hardware Aware Approximate Clustering With Randomized SVD and Coresets

Turning Big Data into Small Data

Hardware Aware Approximate Clustering with Randomized SVD and Coresets



Tarik Adnan Moon

A thesis submitted to the Department of Applied Mathematics in partial fulfillment of the requirements for a Bachelor of Arts degree with Honors

Department of Applied Mathematics
Harvard University

Cambridge
Massachusetts

1 April 2015

"In God we Trust, all others must bring data"

William Edwards Deming (1900-1993)

Abstract

Organizing data into groups using unsupervised learning algorithms such as k-means clustering and GMM are some of the most widely used techniques in data exploration and data mining. As these clustering algorithms are iterative by nature, for big datasets it is increasingly challenging to find clusters quickly. The iterative nature of k-means makes it inherently difficult to optimize such algorithms for modern hardware, especially as pushing data through the memory hierarchy is the main bottleneck in modern systems. Therefore, performing on-the-fly unsupervised learning is particularly challenging.

In this thesis, we address this challenge by presenting an ensemble of algorithms to provide hardware-aware clustering along with a road-map for hardware-aware machine learning algorithms. We move beyond simple yet aggressive parallelization useful only for the embarrassingly parallel parts of the algorithms by employing data reduction, re-factoring of the algorithm, as well as, parallelization through SIMD commands of a general purpose processor. We find that careful engineering employing the SIMD instructions available by the processor and hand-tuning reduces response time by about 4 times. Further, by reducing both data dimensionality and data-points by PCA and then coreset-based sampling we get a very good representative sample of the dataset.

Running clustering on the reduced dataset, we achieve a significant speedup. This data reduction technique reduces data dimensionality and data-points, effectively reducing the cost of the k-means algorithm by reducing the number of iteration and the total amount of computations. Last but not least, using we can save pre-computed data to compute cluster variations on the fly. Compared to the state of the art using k-means++, our approach offers comparable accuracy while running about 14 times faster, by moving less data fewer times through the memory hierarchy.

Acknowledgements

I would like to express my deepest appreciation for Stratos Idreos, who has guided and mentored me over the last one year on this project. His inspiration and feedback have been invaluable for carrying out the research. Without his persistent help this thesis would not have been possible.

I would also like to thanks the members of the DASlab and especially Manos Athanassoulis and Abdul Wasay for their help. They have helped me tremendously with countless reviews and with the directions of the research.

In addition, I would thank Chris Rycroft for helping me understand the randomized SVD and other numerical techniques through AM 205 lectures and the final project.

Finally, I would thanks my family, classmates, and some very special friends who have made this journey to finishing this thesis possible.

Contents

Dedicated to my family

Chapter 1

Introduction

In the age of Big Data, we accumulate more than 2.5 exabytes of data everyday [1]. This massive volume of data outgrows our ability to extract information from it. We collect increasingly complicated, diverse and voluminous data that needs to be processed and classified to extract meaningful information. This journey from raw data to knowledge extraction is a complicated one and requires multitude of smart and computationally expensive machine learning algorithms. Especially, in industry often the data is raw and unlabeled. A very common way to gather more insight about those datasets is to perform some unsupervised learning on the dataset. Classes of machine learning algorithms such as clustering, regression, and classification are some of the most widely applied unsupervised and supervised ones. The challenge in this knowledge discovery process often lies in the computational complexity and movement of data through the memory walls due to the iterative nature of these algorithms. This is especially true when the dataset is massive.

Figure 1.1: Segmentation using clustering (left), MNIST handwritten digit dataset (right)

1.1 Clustering as Unsupervised Learning

Clustering is a part of the class of machine learning algorithms known as unsupervised learning. Accordingly to Murphy, "The goal [of unsupervised learning] is to discover "interesting structure" in the data; this is sometimes called knowledge discovery" [2]. Unsupervised learning solves some fundamental pattern recognition problems such as finding clusters, discovering graph structures, and matrix completion.

Clustering is widely used as an organic way to infer meaning from data where similar elements are grouped together based on a metric such as euclidean distance between the points. Different versions of distance based clustering techniques such as k-means [3], hierarchical clustering [4], canopy clustering [5] have been extensively used in learning and classification for various fields. All these methods start with some initial cluster centers and assign those to each of the points. Based on the update the cluster center changes at each iteration and so does the cluster assignments. The algorithm stops when the total distance from the cluster centers to the assigned points is minimized or no update is made to cluster assignments. In the recent years, Gaussian Mixture Model (GMM) has also been popular for clustering. GMM is a probabilistic model that is trained on the dataset using expectation maximization

(EM) algorithm. Then we assign each point to the cluster which has the highest probability of having that point.

However, as both of these fundamental clustering algorithms are iterative by nature, they need to go through the data numerous time. Therefore, these algorithms are inherently slow in a modern system. In this thesis, we propose an ensemble of algorithms to alleviate these problems and improve the performance of clustering.

1.2 Clustering Algorithms and Modern Systems

The advent of modern hardware having multiple cores, deeper memory hierarchies and diverse storage presents new challenges and opportunities to rethink the implementation and design of existing algorithms. Modern hardware has untapped potential that can be harnessed to improve the performance of various machine learning algorithms. Increasingly, we are using multi-core, multi-processor systems, but this induces the challenges such as managing the memory hierarchy efficiently. Especially with the introduction of increasingly large amount of disk and main memory (RAM) compared to the CPU cache size, and widening gap in their I/O speed, it is more important than ever to consider the reduction of data movement through the memory walls.

Due to the iterative nature of the machine learning algorithms such as k-means or GMM, their implementations are often not optimized for these systems. Especially given with the advent of multi-cores, multi-processor systems it is important to consider how much data we read and if we can reduce the amount of data to read, how we can do so. So in this thesis, we incorporated both hardware conscious optimization and data reduction techniques to both reduce the number of iterations in our

algorithms and to make the whole algorithm aware of the memory walls in a modern system.

1.3 Hardware Aware Clustering

There has been work on parallelizing the clustering algorithms with multi-core processors [6], GPU computing [7–10], MapReduce [11], and similar techniques, but because clustering is iterative and each iteration is dependent on the previous one it is hard to scale it using those techniques. In the light of recent advancements in hardware architectures, there are new opportunities to work on a system that leverages both probabilistic algorithms and new hardware architectures to design a better clustering algorithm. In this thesis, we concentrate on a single CPU machine and show how modern hardware features such as SIMD instructions, cache awareness coupled with data reduction techniques can dramatically improve the performance of clustering algorithms.

1.4 Sampling and Data Reduction

Optimizing an algorithm only for specific hardware platform is not going to provide optimal results. So leveraging some of the current probabilistic methods to connect them with the philosophy of leveraging the modern hardware architecture provides the best result. Due to the memory walls moving large amount of data from th disk to CPU is very slow. Therefore, we need to make sure that some of the most computationally expensive parts, such as the iterations, of the algorithms is minimized. We use some sampling techniques to reduce the amount of data hence both reduce the number of iterations and data movement. Moreover, we use an approximate singular

value decomposition (SVD) algorithm [12] to find a low rank approximation of the data. While the reduced dataset will not provide the most accurate result, in this thesis, we show through experiments and theoretical results that the clusters are good enough for many practical applications– especially given the original dataset is often very noisy.

1.5 Contributions

As we increasingly rely on ML algorithms to extract knowledge from growing amount of data, we test the limits of what hardware can provide. Popular ML algorithms such as k-means clustering and GMM based clustering need to iterate over the dataset multiple times in order to achieve high accuracy in their operation. Combining this fundamental algorithmic design of clustering with the hardware trends leading to the memory wall, there is a growing miss-match between what hardware is optimized for and what ML algorithms need.

In this thesis, we address this widening gap, by presenting a set of principles, which used in concert, radically change the operation of iterative ML algorithms. Specifically:

- Using a carefully designed subset of the data we can initialize the iterative algorithms (e.g., initial seeds of cluster centers for k-means) avoiding iterative passes on main-memory by having samples that fit in the CPU cache.
- Careful re-organization of the data accesses during each iteration on full data, allows for reading strictly once, leading to 2x speedup.
- Aggressively using SIMD in embarassing parallel operations (such as distance calculations) further boosts performance by about 2x.

- Using randomized SVD and smart adaptive sampling we reduce the size of the dataset. Then we take into account the memory hierarchy to optimize clustering for the smaller dataset. Due to the reduction of data points and dimensions, the learning process is 10 - 100 times faster compared to the state of the art techniques.

- We can save the dataset with reduced with the aforementioned ensemble data reduction algorithms. That reduced dataset can be used in subsequent runs of clustering algorithms with different parameter values, e.g., number of cluster centers. This allows for significant speedup when the goal is to find the best number of clusters or latent features.

Chapter 2

Background and Related Work

Over half a century old and showing no sign of aging, clustering is one of the most frequently and widely used algorithms in machine learning. There has been an extensive amount of work done in this topic [13]. In this chapter, we will discuss some background regarding clustering and will highlight some of the literature related to our work.

2.1 Clustering and Its Applications

Unsupervised learning such as clustering has been used to solve problems ranging from business analytics to computer vision. For example, in business, clustering is used to group similar products [14]. Products generally have different attributes such as price, quality, type, brand, and so on. Using all these attributes as a "feature vector", we can partition the set of products into different groups with clustering. Moreover, users can be clustered similarly using different user attributes.

Other uses of clustering are found in computer vision segmentation [15] and image recognition. In Figure 1.1, we see how an image is segmented based on its color with k-means. This technique is applied in computer vision for solving many problems including face detection and object detection for automated driving.

Especially when we have a massive dataset without labels (which is the case for most of the modern big datasets), the first step is to find the structure of the dataset using some cluster or pattern finding algorithms.

While k-means and its variants are distance based algorithm, Gaussian Mixture Model (GMM) has gained popularity in the recent years due to its robustness and superior cluster results. GMM is a probabilistic model that is trains a mixture of multivariate Gaussian (normal) distributions using the dataset.

GMM is used in both supervised and unsupervised learning. For example, if the dataset is labeled then each of the Gaussian of the mixture represents each of the labels. On the other hand, if the data is not labeled then the model uses a user specified number of features, k and fits the data to a mixture of k Gaussians. In that case, the number of clusters (or features), k is used specified. Those features are also known as the latent features of the data, because often the user does not know what those features represent. After training GMM, we can very easily find the probability of any point belonging to one of the k clusters. As a result, GMM is widely used in clustering and latent feature detection for unsupervised learning.

2.2 Clustering Methods

K-means is a popular clustering algorithm, as a result, a number of optimizations has recently been proposed including k-means++ [16], canopy clustering [5], coreset [17–19]. K-means++ uses a smart initialization technique that initializes the cluster

center carefully and as a result converges faster than other algorithms. Canopy clustering is another technique that takes into account the topology of the dataset. As a result, it is possible to reduce the amount of work by partitioning the problem. Finally, coreset clustering first uses a probabilistic method to produce a smart sample that performs stratified sampling called coreset. Then the algorithm does clustering on the reduced dataset. Some of these algorithms depend on probabilistic techniques, but none of these sampling based approaches are cognizant of the memory hierarchy. Keeping in mind the memory hierarchy, we sample efficiently and show that hardware conscious design can lead to significant gain in performance.

2.3 Implementing Clustering Algorithms

Clustering algorithms partition the data-points of a dataset into a number of classes based on their similarities or proximity. In general, clustering algorithms can be divided into two main classes: 1) k-means and similar distance based algorithms, and 2) GMM and similar probabilistic clustering algorithms.

2.3.1 K-means Clustering and Other Variants

K-means clustering is the most popular clustering algorithm due to its simplicity and speed. However, finding exact clustering is NP-hard [20]. There has been some work [21] on finding fixed pass, approximate methods that provides theoretical constant factor approximations to the optimal. However, in practice they do not perform well for a large amount of data because cluster center is found using a recursive approach. This involves non-trivial amount of data movement and hence the speed is slower than the heuristics bases algorithms. Therefore, heuristics based variant of k-means clustering is the most popular choice for clustering.

While these variant have significantly improved k-means, but the principle is the same. K-means clustering partitions the n datapoints of a dataset into k clusters, where each observation belongs to nearest cluster center. Cluster centers are defined by the mean (center of gravity) of the points belonging to the cluster.

If we have n datapoints $x_1, \cdots, x_n$ and given cluster number k. Then the objective of the k-means algorithm is to find $S = \{S_1, \cdots, S_k\}$ such that the following objective function is minimized:

$$\sum_{i=1}^{k} \sum_{x \in S_i} d(x, c_i)$$

where c_i are the centers of cluster S_i and for any two points a, b, the distance metric $d(a, b)$ is defined as the square of the euclidean distance between them.

The algorithm generally starts with some initial assignments to the cluster centers. And then performs several iterations consisting of assignment and update steps until convergence.

Assignment. In this step, we partition the whole set into clusters S_i– we assign a point to the cluster center closest to it. Mathematically on t-th iteration:

$$S_i^{(t)} = \{x : d(x, c_i) \leq d(x, c_j) \forall j, 1 \leq j \leq k\}$$

Update. In the update step, we calculate the new cluster center by calculating the centroid of the datapoints in a cluster:

$$c_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x \in S_i^{(t)}} x$$

Most of the variants of k-means algorithms (i.e. other distance based clustering algorithms) generally differ from in terms of initialization or the distance metric.

2.3.2 Gaussian Mixture Model (GMM)

Gaussian mixture model(GMM) is another algorithm for finding clusters and it uses a probabilistic model instead of the distance metrics used by other clustering algorithms. For clustering, GMM is considered to be "soft cluster assignment" algorithm compared to the fixed cluster assignment of k-means clustering. This is because for each point we get a probability of that belonging to a certain cluster, which is not a fixed assignments as the ones in k-means.

GMM is an algorithm from statistical inference. With GMM, we fit the dataset $\mathcal{D}$ to a mixture (weighted sum) k normal distribution $\mathcal{N}(x_i; \mu_j, \Sigma_j)$, where μ_j, Σ_j are the respective mean and covariance matrix of the respective normal distributions. We can write the model as

$$p(x_i|\mathcal{D}) = \sum_{j=1}^{k} \pi_j \mathcal{N}(x_i; \mu_j, \Sigma_j)$$

where π_j are the weight for the respective normal distribution.

Expectation Maximization (EM) Process to fit GMM. EM is a process from the statistical inference to train a probabilistic model based on the log likelihood function. For GMM we start with an initial guess of the mean vectors and covariance matrices. The initial mean vector is generally determined by k-means clustering. At each step of EM mean and co-variance are updated until the log-likelihood function converges.

While more complex and more computationally expensive than the k-means clustering GMM has gained popularity due to being more robust and better in general, yielding more accurate results. Instead of assigning hard cluster number we can calculate the probability of the point belonging to a certain cluster. To get a hard cluster

assignment similar to the classical, distance based clustering, we assign each point to the cluster which has the higher probability of having that point.

2.4 Reducing Size and Dimension

For data with large number of dimensions, a common practice is to reduce the dimensions by standard techniques such as SVD. Currently the state of the art is the probabilistic SVD algorithm [12] that speeds up the SVD process using a randomized matrix approximation of the original matrix. By incorporating similar SVD techniques in the system to reduce the dimensionality of data, we make our algorithm faster.

2.4.1 PCA via Randomized SVD

PCA analysis is very similar to SVD finding. Looking in the literature for fast and efficient SVD algorithm, we find that randomized SVD is a very fast algorithm with a tight and reasonable error bound.

For a real $m \times n$ matrix A, if we perform q power iterations then the error bound for the ℓ-truncated SVD satisfies [12] :

$$E(A - U\Sigma_\ell V^*) \leq \sigma_{\ell+1} + \left(1 + 4\sqrt{\frac{2\min(m,n)}{\ell - 1}}\right)^{1/(2q+1)} \sigma_{\ell+1} \tag{2.1}$$

where $\sigma_{\ell+1}$ is the $\ell + 1$-th left value and the matrix A is approximated by the matrix product

$$A_\ell = U\Sigma_\ell V^* \approx A$$

Discussion on the Error Bound. One of the first, concern about using the Randomized SVD compared to other deterministic truncated SVD calculation process is the error. We know the following bound for any rank ℓ-approximation matrix X of data matrix A [22]:

$$\min_{rank(X)\leq\ell} |A - X| = \sigma_{\ell+1}$$

Therefore, it is impossible reduce the error term below $\sigma_{\ell+1}$. In equation 2.1, we see that the error is bounded above by the singular value $\sigma_{\ell+1}$ and another additive term which can be made arbitrarily small by increasing the value of q. This technique is known as the power iteration. In practice, $q = 1$ or 2 is enough to make the error term small enough for the most practical purposes [12].

Algorithm. Here we briefly present the randomized SVD calculation and the low rank matrix approximation algorithm. The randomized SVD algorithm [12] has two parts. The first part is randomized range finder. Which goes as follows:

Given an $m \times n$ matrix A and integer ℓ this algorithm computes $m \times \ell$ orthonormal matrix Q which approximates the range of A

1. Draw an $n \times \ell$ standard normal random matrix Ω.
2. Form $Y = (AA^T)^q\Omega$
3. Find Q with traditional QR factorization i.e. $Y = QR$ and the column of Q form an orthonormal basis for the range of Y.

And the second part is finding the SVD of the matrix A after projecting it on Q. i.e. we find the range Q and then form $M = Q^T \cdot A$. Then find the SVD of M.

For finding PCA,

1. Center the data by subtracting row-wise means.

2. Find V with randomized SVD.

3. Project back the data with $A_\ell = A \cdot V^T$.

2.4.2 Uniform Random Sampling and Coreset

Various forms of sampling are used by clustering algorithms in many different ways. For example the most widely used way of initializing the cluster center is to randomly sample k points as the cluster centers from the datapoints. However, because k-means is sensitive to the initialization of the cluster centers, the execution time of k-means using this technique is unpredictable.

In this thesis, instead of just using random samples for initialization, we aim at finding a random sample of cluster centers that represent those of the actual dataset. We propose an algorithm with random sampling to initialize the cluster centers. Moreover, propose to use another algorithm with a smart adaptive sampling technique called, coreset [17–19] to find a representative dataset. Coreset construction is a stratified sampling technique that performs a k-class stratified sampling on the data. The basic assumption is that many of the points are within a close proximity of the cluster center and the outliers are sparsely distributed around. Coreset construction samples from those close to centers and far from the center data-points separately from each of the perceived clusters. This stratified sampling gives us a representative sample of the dataset. Compared to the strong [17] and weak coresets [18], our coreset building algorithm differs due to its reduced computational complexity. We do not use any weight on the samples or any convergence criterion. Therefore our proposed coreset construction algorithm is a constant pass algorithm.

From the classic random sampling to more successful k-means++, all these algorithms run on the full dataset and depends on a good initialization. In this thesis, we propose an algorithm leveraging both dimensionality reduction with fast SVD and sampling with coreset to reduce the data and hence reduce the movement of data through the memory walls.

2.5 Machine Learning Systems

Systems specifically designed for machine learning, such as MLBase [23], SystemML [24] aims at improving both the current implementations and abstraction learning algorithms. For both systems, the main objective is to make machine learning process easier by abstracting out the implementation details, rather than making the algorithms run the fastest possible. More specifically, SystemML uses MapReduce to parallelize many of the machine learning algorithms. So, instead of writing low-level, complex map-reduce jobs, user can perform ML algorithms in a higher-level language. Similarly, MLBase makes machine learning easier by a system that can parse simple commands and run complex ML algorithms under the hood to infer information from the dataset. MLBase is a much higher level language than the SystemML and has functions such as `doClassify()` that classifies the data using some ML algorithms. Moreover, it uses traditional hardware system instead of the distributed computing techniques used by SystemML. While, these systems aim at providing a higher-level abstraction to facilitate the end-user running machine learning tasks in a distributed environment, we present hardware-conscious strategies to improve the performance of a specific class of iterative machine learning algorithms.

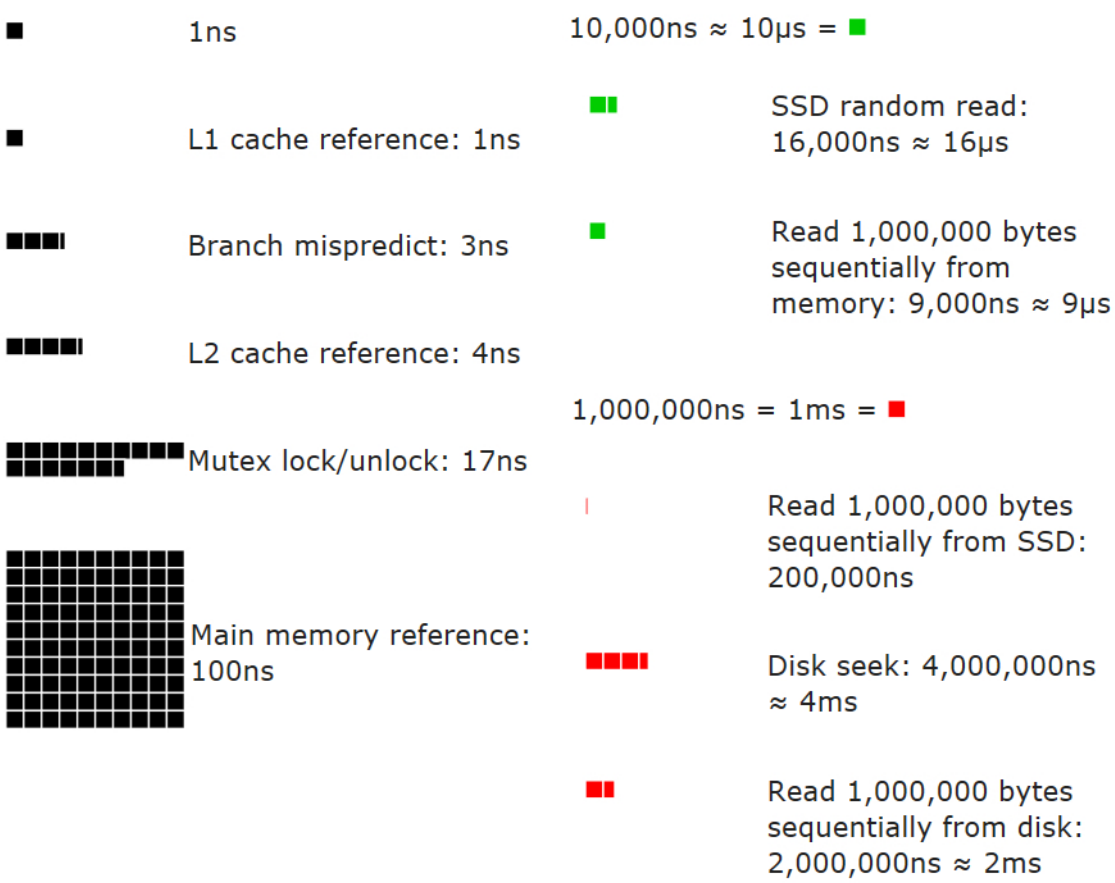


Figure 2.1: Comparison of memory latency between CPU caches, main memory (RAM), and disk [25, 26]

2.6 Hardware Conscious Clustering

There has been a recent trend in hardware conscious design for ML algorithms. Due to the nature of the modern hardware, being cache-conscious and memory hierarchy conscious are extremely important. For distributed systems MapReduce has been used to cluster a large amount of data [11]. However, it is comparatively slow due to the overhead due to communication between nodes.

Moreover, leveraging parallelism is another paradigm that has been used extensively. Because many of the ML algorithms are iterative, parallelizing them, in most cases, entails parallelizing the complex mathematical computations in each iteration. For

example: GPU computing has been used extensively [7–10] to make clustering parallel, but again, the only embarrassingly parallel part of the algorithm is the calculations done in each iteration.

While these approaches aim at scaling the algorithm by sheer computation power, they neither reduce actual computation nor improve them by better execution following the hardware aware algorithm design. As we see in Figure 2.1, it takes 1ns to read data from L1 cache vs 4×10^6ns to read it from the disk. Even referencing data in the main memory is 100 times slower than referencing in the L1 cache. This non-trivial difference in speed asks for better algorithm designs and implementations to reduce the amount of I/O and data movement though the memory walls (i.e. moving memory to CPU from disk). Therefore, in this thesis, we propose an algorithm that performs faster due to data movement reduction in the proposed algorithm.

2.7 Fast Parameter Tuning

In practice, we need to run the same unsupervised learning process with different parameters to select the best parameter for a model. For example, for unlabeled data selecting the right k is often impossible without any prior knowledge about the data [27]. In those cases, there are many heuristics such as running a v-fold cross validation and choosing the right cluster numbers [27]. In this method, the dataset is divided randomly into v equally sized parts. Then one of them is set aside as a test set and the other $v - 1$ as the training set. Then clustering is computed on the training set and then the cluster centers are assigned to the test set. The best model would be the one with the lowest sum of distance from the center. Similarly for finding the best number of features, k for GMM a similar cross validation technique is used.

Especially, when a complex ensemble model consisting of multiple models is trained and clustering is used as the first step of the model, the whole clustering process is repeatedly executed. However, we can avoid these multiple runs of training on the full dataset by saving the precomputed data reduced by SVD and coresets. In this thesis, we propose an algorithm that uses the precomputed reduced data saved in memory to train our model for different parameters and avoid a huge amount of repeated computations.

Chapter 3

Designing Hardware-conscious Clustering Algorithms

Data movement through the memory walls [28] is the main bottle-neck in executing algorithms on a modern system. Hardware optimization techniques such as using SIMD instructions, making the algorithm cache and branching conscious can improve the overall performance. However, for a massive, high dimensional dataset we first need to use some pre-processing such as sampling and SVD to make the data small enough so that we can apply those hardware conscious design principles. Moreover, with the reduced dataset, we perform fewer iterations and fewer computations, hence reducing the movement of data.

3.1 Reducing Data Movement

As we want to reduce the data movement, we need to minimize the number of passes over our data. First we need to make sure that during each iteration, we read the dataset exactly once. Generally, we can assure this by simple memoization techniques

and a careful implementation, in general. We can also force the CPU to cache some of the most used data and hence reduce the movement. While parallelization seems to be a natural solution for scaling up ML algorithm, as ML algorithms such as k-means clustering, GMM are iterative, we cannot parallelize them easily because each iteration is dependent on the result of the previous one.

Therefore, to minimize the passes over the data we use several probabilistic sampling techniques first. First, we use uniform random sampling to find the initial cluster centers for k-means. While this provides good result for our dataset, there is no guarantee for it to perform well. Therefore, we can use a modified weak coreset (which is a better representative sample). Because we are dealing with fewer datapoints, our algorithm converges much faster with less number of iterations. In the same spirit, we use the probabilistic SVD algorithm described in Section 2.4.1 to find a low rank approximation of the data matrix that further reduces the data dimensionality. Reducing dimension with SVD especially works very well when we have a sparse dataset. Then we use aforementioned sampling techniques to reduce the number of data points. This two step process can reduce the data to a great extent while giving us a very good approximation.

Data reduction techniques are much less expensive compared to the full iteration on the big dataset, hence, in order to reduce data movement the fundamental principle is to start by reducing the overall data size with minimal loss of information. This paradigm is especially important because by reducing the dataset we can make sure that even a very large dataset would fit in memory after reduction step and hence improve the overall performance of the algorithm.

3.2 Vectorized Execution Leveraging SIMD

Vectorized execution is another necessary technique for utilizing the most of a single core. Using SIMD instruction sets we can pack multiple computations in a single vector computation. For most of the iterative algorithms some very fundamental operations are performed numerous time during each iteration. In the case of k-means clustering calculating distance between points is the most used operation. Therefore, we vectorize the distance calculation operation using SIMD instruction set, hence reduce the computation. Similarly, for most of the iterative algorithms we can vectorize some of the most executed operations and hence speed the the calculation.

3.3 Saving Pre-Processed Data

If we need to perform the same algorithm many times for parameter tuning, we can often reuse many of the computed results. For example, we save the output of the truncated SVD for during the first pass of our optimized k-means clustering. As a result, when we need to perform clustering for a different cluster number (which is the case for many data exploration or ensemble model testing tasks), we can perform clustering on the fly using the pre-processed data saved in memory. Similar technique works for GMM and can be generalized to other unsupervised learning algorithms. Hence pointing out he reusable bits of an algorithm can drastically improve the performance of parameter tuning.

Chapter 4

Optimizing Clustering

In this chapter, we put together all the techniques we discussed and design an algorithm for fast clustering. We also run several experiments and with experimental results show about 14x speedup on single run. We show further performance improvements with our parameter tuning algorithm.

4.1 Dataset and Experimental Setup

In this thesis, we use the MNIST image dataset [29]. This dataset consists of 60,000 images with 28×28 pixel resolution. We model each of these images as a point in a $28 \times 28 = 784$ dimensional space.

Our experiments are run on a machine with Intel(R) Xeon E7-4820, 2.00GHz processors. While we do not use multi-core architecture, we use the SSE (SIMD) capability for vectorized operations.

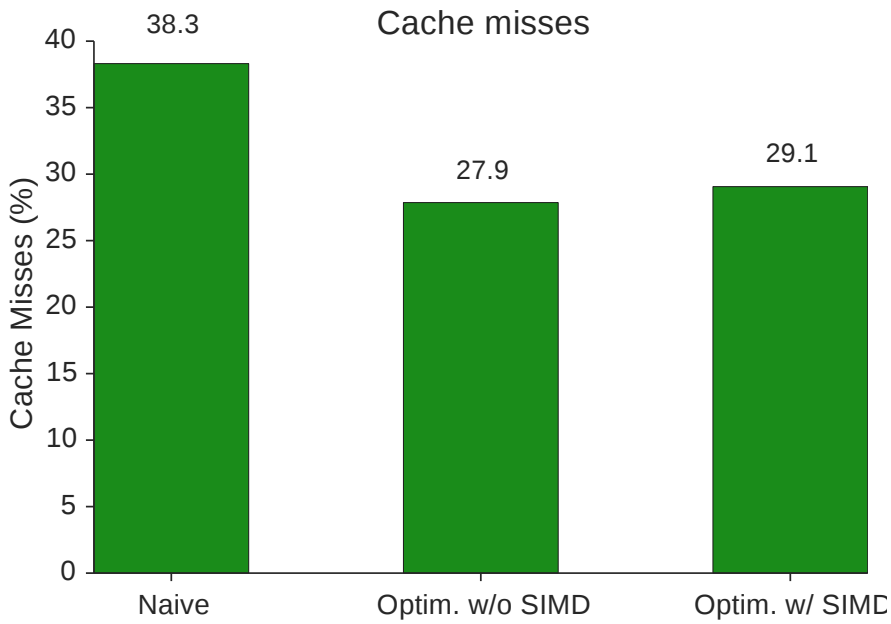


Figure 4.1: Percentage of cache misses from the perf test report

4.2 Hardware Awareness

As modern hardware is becoming more and more complex, it is becoming increasingly important to make an algorithm hardware conscious. Although clustering is an iterative algorithm, we can use parallelism for a single iteration. As each iteration of clustering is embarrassingly parallel, it has already been parallelized with both CPU [6] and GPU [7–10]. In this thesis, we leverage SIMD instructions to optimize the algorithm.

According to the perf test we find that most the algorithm spends the majority of its time calculating distances between the points and cluster centers. We also find that we have about 38.31% cache misses (Figure 4.1). To reduce the number of I/O and cache misses, we make sure that at each iteration the whole dataset is read exactly once. Moreover, we prefetch the cluster centers, which are used for calculating the distance for each point. These hardware aware implementation reduces the cache

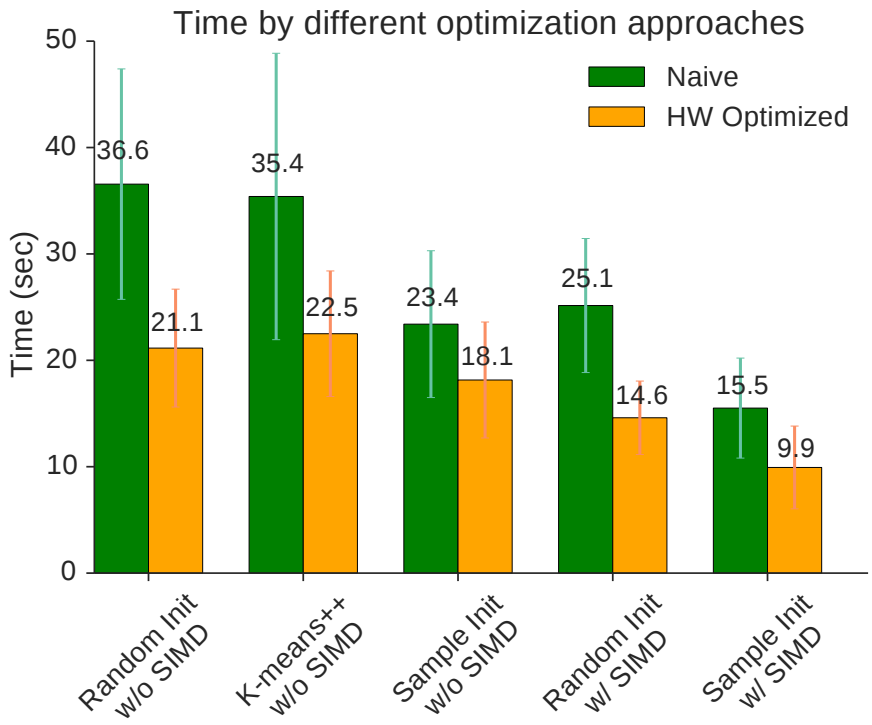


Figure 4.2: Time by different optimization approaches. This graph verifies that hardware optimization techniques significantly improve the performance. For the sample initialized (Sample Init) algorithm we consider the average of execution time for 15%-25% samples. These algorithms are run 20 times with different random seeds.

misses (27.9%) and I/O significantly and reduces the average clustering time to 21.1 sec from 36.6 sec (Figure 4.2).

```
#define sqr(x) ((x)*(x))

float calc_distance(int dim, float *p1, float *p2)
{
  float distance_sq_sum = 0;

  for (int ii = 0; ii < dim; ++ii)
    distance_sq_sum += sqr(p1[ii] - p2[ii]);

  return distance_sq_sum;
}
```

Code Snippet 4.1: Distance calculation without SIMD optimization

```
float vsum(const float *a, int num)
{
    float sum;
    __m128 vsum = _mm_set1_ps(0.0f);

    for (int i = 0; i < num; i += 4)
    {
        __m128 v = _mm_load_ps(&a[i]);
        vsum = _mm_add_ps(vsum, v);
    }
    vsum = _mm_hadd_ps(vsum, vsum);
    vsum = _mm_hadd_ps(vsum, vsum);
    _mm_store_ss(&sum, vsum);

    return sum;
}

float vdist(const float *a, const float *b, float *store,
                        int num)
{
    __m128 dist = _mm_set1_ps(0.0f);

    for (int i = 0; i < num; i += 4)
    {
        __m128 v1 = _mm_load_ps(&a[i]);
        __m128 v2 = _mm_load_ps(&b[i]);
        dist = _mm_sub_ps(v1, v2);
        dist = _mm_mul_ps(dist, dist);
        _mm_store_ps(&store[i], dist);
    }

    return vsum(store, num);
}
```

Code Snippet 4.2: Distance calculation with SIMD optimization

Then we make sure that even on a single core our operations are vectorized. Therefore, we use SIMD instruction `_mm_sub_sp()` to find the difference between two four dimensional vector with one instruction. Then we use `_mm_prod_sp()` to find the square of distance for the four component of the vector. Finally, we add them using

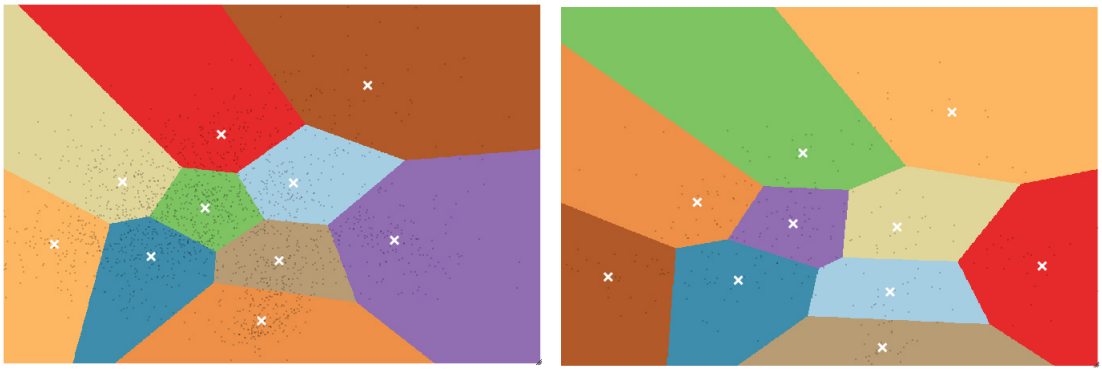


Figure 4.3: Clusters for full dataset (left) and uniform random samples (right)

another sequence of SIMD instructions. Thus we use half of the instructions compared to the non-SIMD version. This speeds up our algorithm significantly reducing the execution time from 21.15 sec to 14.6 sec (Figure 4.2). While we should get at least 2 times speedup, in practice we do not get that because of the slight increase in cache misses and extra memory allocation used for the SIMD distance calculation (Figure 4.1).

4.3 Uniform Random Sampling

While Hardware Optimization is a good technique for making k-means faster, still for a large dataset it is prohibitive to perform exceedingly large amount of iterations due to the time required for the distance metric to converge. Therefore, we aim at reducing the total amount of data. So the first approach we take is uniform random sampling to find the initial cluster centers.

For random sampling, we sample a fraction, α datapoints uniformly. Then we perform clustering on them. After that we use the cluster centers of the sampled data as initialization for the cluster centers for the whole dataset. Finally, we cluster the whole dataset again. Note that this is different from the typical cluster initialization

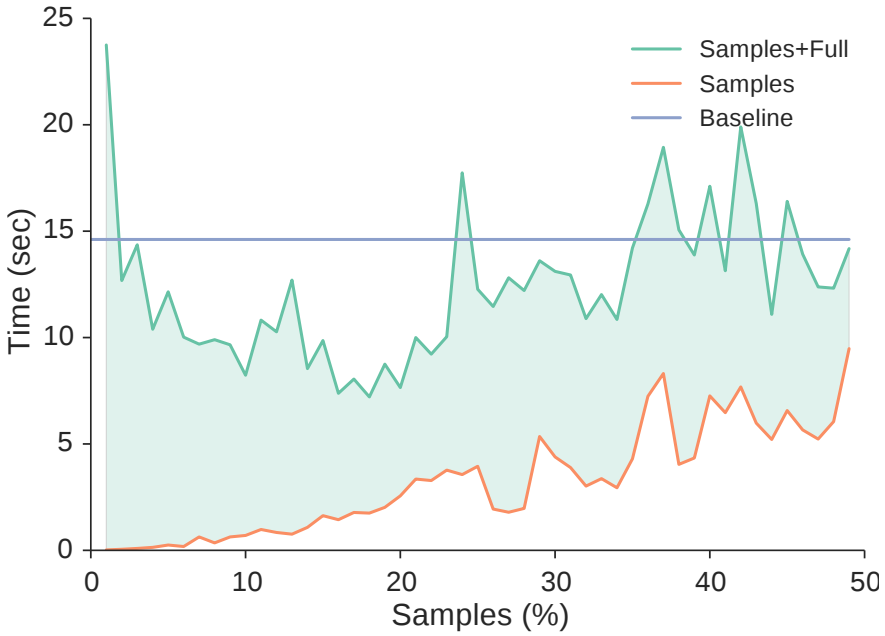


Figure 4.4: Time for clustering the sample and then the full dataset using the cluster centers of the sample data as a seed

technique where the cluster centers are chosen randomly. Instead, here we use the cluster centers of the sampled data as our cluster center.

Here the rationale is that if we sample enough datapoints, our sample is going to be a good representation of the final dataset with a high probability (illustrated in Figure 4.3). Therefore the cluster centers of the sampled data is going to provide a good seed for initial cluster centers. More importantly, the full run of k-means on a small sample takes the time of a few iterations of the k-means run on the full dataset. Careful initialization using this algorithm saves many iterations on the full dataset and the algorithm converges faster when run on the full dataset.

As we increase α i.e. the number of samples, we see that we are getting better and better approximation for the cluster centers (Figure 4.4). With the increasing number of samples, however, we need more time to cluster them. Experimentally, we find that sampling 15%-25% data is good enough for getting a good approximation.

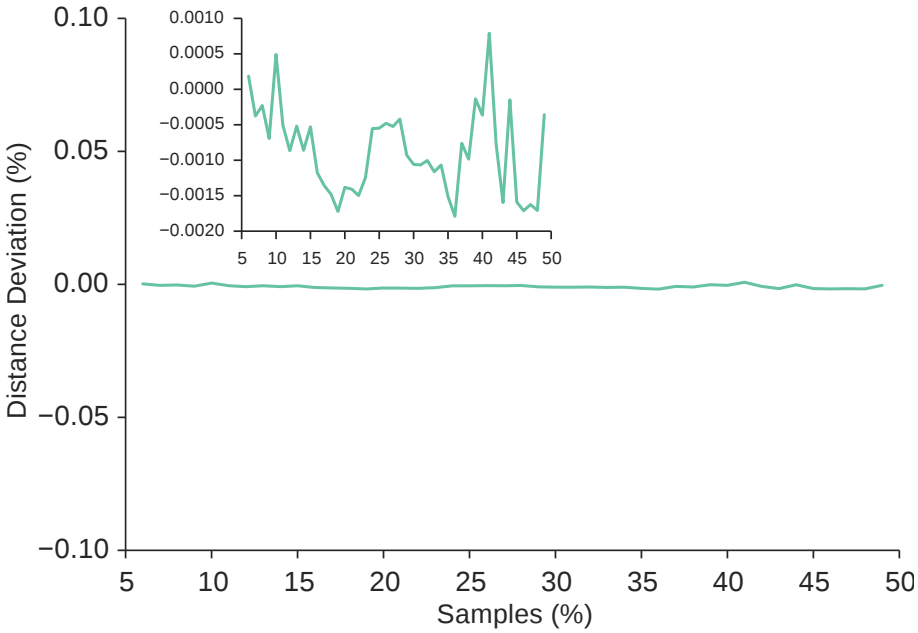


Figure 4.5: Distance deviation (from baseline–kmeans++) vs percentage of samples

Uniform random sampling provides very good clustering results for our current dataset. As we can see in Figure 4.5, the objective value (the total distance from the cluster centers to points in the respective clusters) are very close to the baseline. In fact, from the zoomed in version, we see that for most cases we have negative percentage error, which means that our objective metric is further minimized with random sample initialized cluster centers. However, due to the probabilistic nature of the algorithm, there is no absolute guarantee of converging faster. Therefore, dimensionality reduction and smarter sampling techniques are necessary to get superior results.

4.4 Making the Dataset Smaller

As uniform random sampling does not guarantee faster convergence, it is necessary to find a more representative sample for better clustering. However, it is computationally expensive to run a superior sampling technique on a high dimensional dataset. So first

we reduce the dimension of the dataset using a low dimensional projection using the randomized PCA algorithm discussed in Section 2.4.1. Due to using random matrix for calculating a truncated SVD, this is much faster than typical SVD algorithms. When we have a lower dimensional representation of the dataset, then we construct a coreset, which provides a representative of the dataset. The ensemble of these algorithms turns our large dataset into a smaller dataset, which is much better suited for an iterative algorithm such as k-means clustering.

4.4.1 Dimensionality Reduction

Reducing the dimension of a dataset is a necessary technique when dealing with a massive dataset. Especially, if the dataset is sparse and high dimensional projecting them on a low dimensional space could save a massive amount of calculation.

Here we use a randomized PCA algorithm to get a truncated SVD rank 100 approximation of the 60000×784 matrix from the MNIST dataset. We save this result as a precomputed set for future calculation (to be discussed in Section 4.7). The significance of this method is that we turned our high dimensional data matrix to a low dimensional matrix. Therefore it is much less computationally intensive to calculate the result.

4.4.2 Adaptive Sampling via Coreset

As uniform random sampling does not always provide the best representation of the original dataset, we use a smart adaptive sampling technique, Coreset, to sample the dataset and get a smaller representative sample of the dataset. Therefore we have a much smaller dataset and we pass less data during the iterations of our clustering

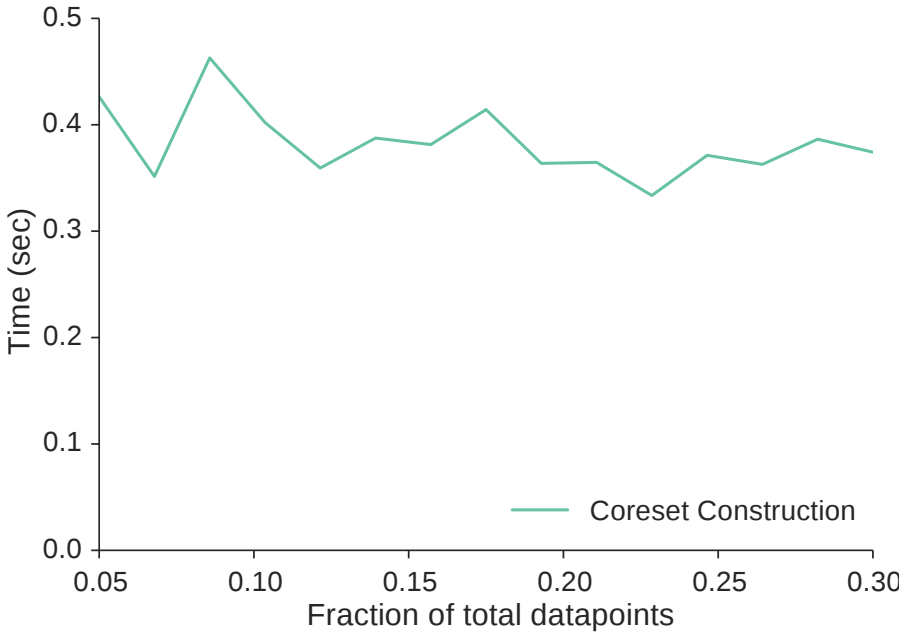


Figure 4.6: Time vs α graph for coreset construction

algorithm. Moreover, k-means converge much faster with a small number of data points.

Our coreset algorithm differs from the typical coreset used for probabilistic models. This is a constant pass algorithm that does not assign any additional weight to the points. Moreover, it is not an iterative algorithm because there is no metric convergence criterion for the algorithm to stop.

4.4.3 Implementation and Performance

For implementing the randomized PCA, we used Eigen 3 library [30] that has optimized linear algebra operations. Moreover, they use SIMD which is in line with our hardware aware approach to clustering. We implemented the coreset in C and as this is not iterative, this is equivalent to a few iterations of the k-means clusterings. In Figure 4.6, we see that constructing a weak coreset is very fast compared to computing the whole cluster. Moreover, as the number of passes through data

Data: PCA transformed images A (N images), sample size α, sample size for initial clustering β

Define:

- $d(a, b)$ as the square of the euclidean distance between two points and
- $cost(S, a) = \sum_{s \in S} d(s, a)$, where S is a point set.

Result: Coreset C

Uniformly sample βN images;

Run an approximate kmean clustering on the sample;

One more step of kmeans on the full dataset to find centers closest to the rest;

for $C_i \in \mathcal{C}$ **do**

  Partition C_i into two sets, C_i^{in}, C_i^{out};

  For each cluster center c_i, all the points that are in the ball $b(c_i, r_i)$, where $r_i = \sqrt{\frac{cost(C_i, c_i)}{|C_i|}}$ we put them in C_i^{in} ;

  Put the rest in C_i^{out}

end

for $C_i^{in} \in \mathcal{C}^{in}$ **do**

  Randomly sample $\alpha |C_i^{in}|$ images.

end

for $C_i^{out} \in \mathcal{C}^{out}$ **do**

  Resorvior Sample with probability of choosing a point $q \in C_i^{out}$, $p_q = \frac{d(q, c_i)}{cost(C_i^{out}, c_i)}$;

end

Merge two sampled sets as C and return C

Algorithm 1: Modified Weak Coreset Building Algorithm

does not depend on how much we are sampling, we see that coreset construction is independent of the number of samples.

Both of the algorithms run much faster than the total time k-means clustering takes. The randomized SVD on average takes about 1.97 sec for a 100 rank approximation. For the coreset construction, the execution time is even smaller than the total time

PCA takes (about 0.4 sec on average). While weak coreset construction is computationally inexpensive by nature, in this case, it is much less expensive because we run the algorithm of the rank 100 approximation of the dataset instead of the full dataset.

4.5 Optimized Clustering with SVD and Coreset

From the performance results it is clear that it is possible to leverage the ensemble of these algorithms to reduce both the dimension and size of the dataset. As a result, clustering becomes easier than taking the naive approach. In fact, for a rank 100 approximated coreset of size 20% (12,000 data points) of the full dataset, on average we can perform the full clustering in 0.2 sec using our random sample initialized algorithm.

Implementation and Performance

As our final goal is to find good clustering (i.e. good cluster assignments), we are going to compare the performance of the approximate model compared with the clusters we found with the naive model. For this purpose we are using two score indexes traditionally used for cluster analysis 1) Adjusted Rand Score 2) Adjusted Mutual Information (AMI). The first score is more of an accuracy measure and the later is a probabilistic measure based on the entropy and the information gain. We use python scikit library's [31] build in functions for these metrics. Note that because this is unsupervised learning and we don't generally have the ground truth, the cluster labels are permutation invariant. We discuss these cluster analysis metrics in details in Appendix A.

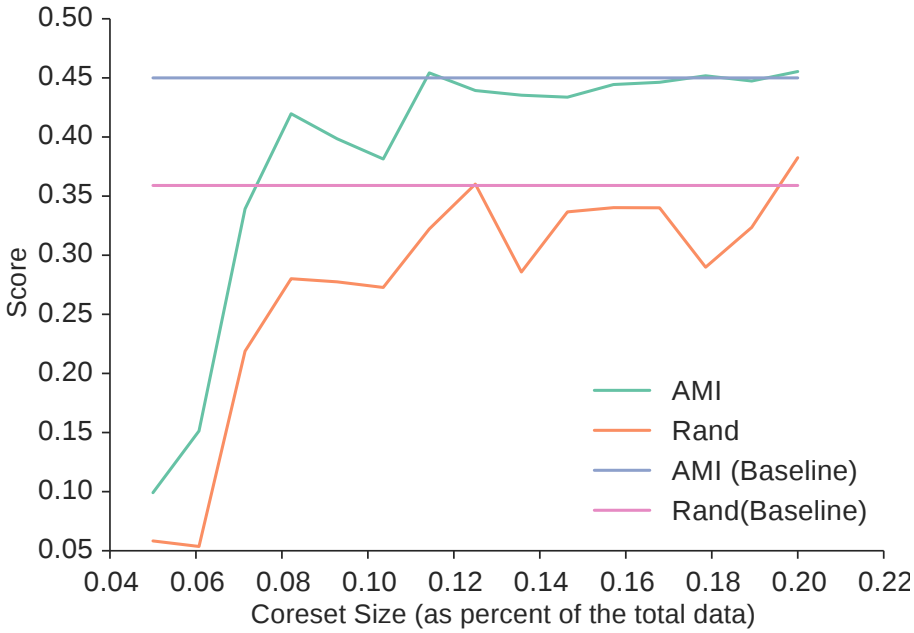


Figure 4.7: Cluster Quality Score vs the fraction of samples, α for k-means clustering on the coreset

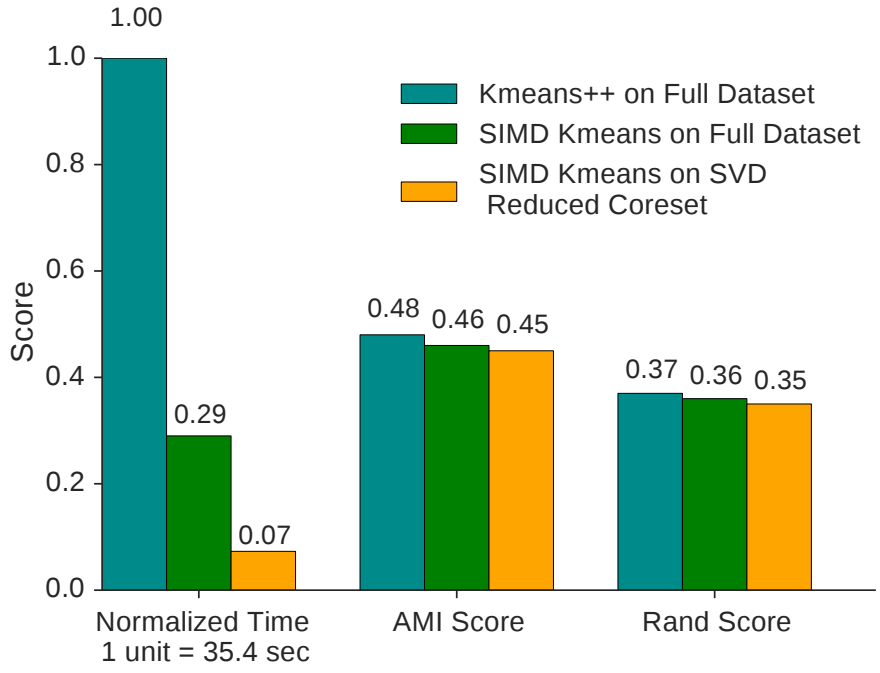


Figure 4.8: Clustering quality scores and (scaled) time graph for three approaches for k-means clustering

As we can see from this figure, the ensemble of SVD and coreset dramatically decreases the amount of computation by smart dimensionality reduction. Then we can compare the scores our coreset achieve as we increase α. From Figure 4.7, it is clear that with even 15% sized coreset, we can get a very good cluster assignments.

Now we put all the results together and show how our ensemble algorithm for approximate cluster outperforms all other clustering algorithms while providing clusters very close to the state of the art (k-means++) algorithm (Figure 4.8).

4.6 Clustering via Optimized GMM

In this section, we demonstrate the effectiveness of our ensemble algorithm on optimized GMM. We implemented GMM with Armadillo [32] linear algebra library, which has optimized vector and matrix operations. Then similar to clustering, we use our ensemble data reduction algorithm to perform SVD and then we construct the coreset. For implementing the algorithm, we used our sample initialized k-means clustering to initialize the mean vector and perform at most 50 iteration or stop if it converges before.

Figure 4.9 shows that GMM is much slower than k-means even on the reduced coreset. In fact, it takes much more time to train on the full dataset (197.8 sec). Therefore, our ensemble data reduction algorithm is even more effective in this case compared to the k-means clustering.

While much slower than the k-means clustering it is more robust and provides much better clustering performance compared to k-means clustering. In Figure 4.10, we show how even with 13%-20% sized coreset the algorithm performs as good as it does on the whole reduced dataset.

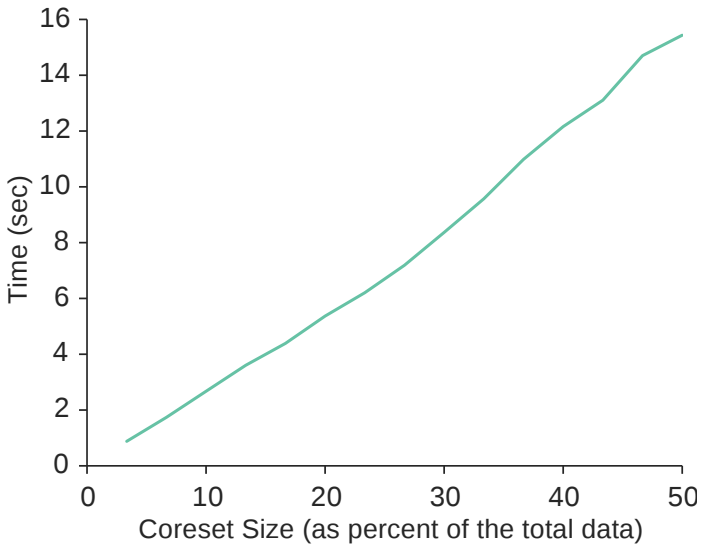


Figure 4.9: Time for calculating for different size of the coreset (as a percentage of the full SVD reduced dataset)

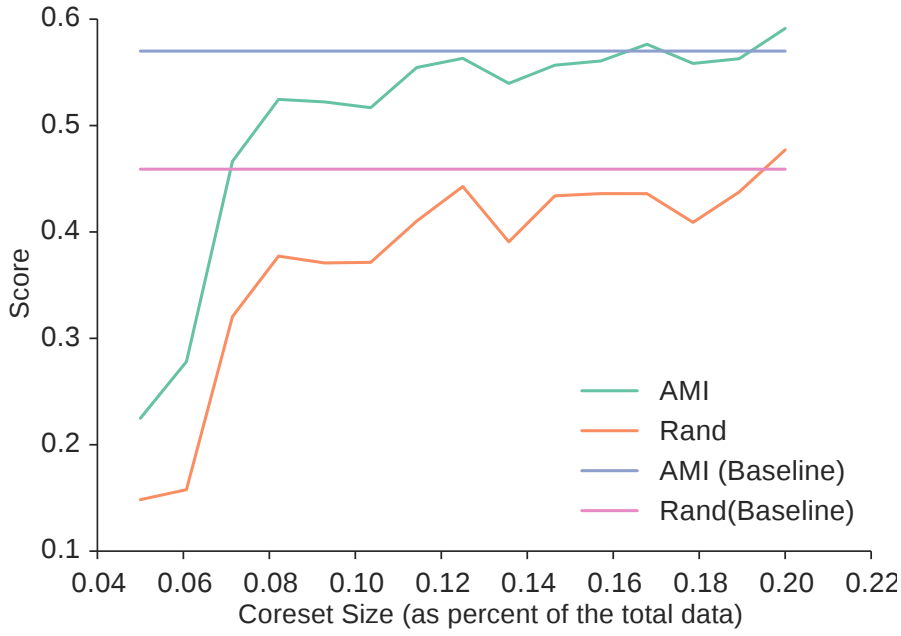


Figure 4.10: Cluster Quality Score vs the fraction of samples, α for GMM clustering on the SVD reduced coreset

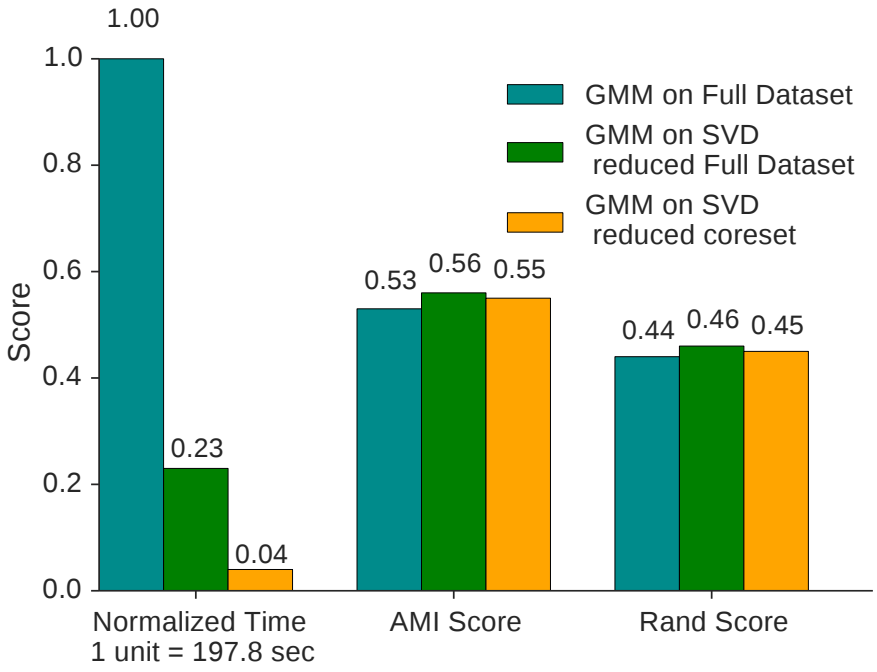


Figure 4.11: Cluster quality scores and (scaled) time graph for three approaches for GMM clustering

Moreover, we show that in fact the AMI and Rand score is better for the SVD reduced dataset (Figure 4.11). This is because on a larger dataset the co-variance matrix did not fully converge within our 50 iteration limit.

4.7 Preprocessed Data for Fast Parameter Tuning

In practice, while performing clustering on a dataset, a researcher has to find a good cluster number. While in the MNIST dataset we clearly have 10 classes, for general unsupervised learning process the researcher does not know the correct cluster number. This number is often found using a parameter search approach and by running clustering for different cluster numbers, k such that some metric (such as Rand score) is optimized. However, this involves running clustering many times which is very time consuming. However, we propose a way of reducing the total computation. For this, we first find a rank 100 approximation using the randomized PCA and save the

Figure 4.12: Time taken by the system for different number of parameters in the parameter search space

dataset in memory. Then we can run coreset and kmeans on the reduced dataset very efficiently. As the time for our ensemble clustering technique mostly is bound by the the randomized PCA computation time, we just run the less expensive coreset construction and k-means on the reduced dataset for different k's. This reduces the total computation time drastically and hence speeds up the data exploration process.

Here in Figure 4.12 we show how parameter search for even 10 parameters (different number of clusters) can become very expensive. However, with our approach we perform the same parameter search in 27.97 sec compared to 1393.5 sec for the full run from start. For GMM we get a very similar level of speedup. This demonstrates that our ensemble data reduction algorithm coupled with hardware optimization techniques outperform the state of the art algorithms.

Chapter 5

Conclusion

While we cannot make an iterative algorithm such as k-means fundamentally non iterative, it is possible to reduce its computation time dramatically by hardware-aware algorithm design and an ensemble of data reduction techniques. In our example with MNIST dataset, the runtime is reduced from 36.1 sec to 2.6 sec; a massive gain in latency. Moreover, for parameter search problems we can reduce the total computation time by pre-computing and saving intermediate results. For just 10 parameters in the search space we reduce the computation time from 1393.5 sec to 27.97 sec. In particular, we address using this technique, the problem of finding the number of clusters k, significantly reducing the execution time of the parameter search problem and making the learning process significantly faster.

Future Directions. While we have not used any multi-core, multi-processor system for parallelization, it is possible to use those techniques along with the techniques described in this thesis to further accelerate the algorithm. It will be interesting to optimize the clustering algorithms keeping non-uniform memory access (NUMA) in mind.

Moreover, using the eigenspace updating algorithm [33] we can design an algorithm for real time clustering that supports updates such as adding, deleting, and changing data-points. Such an algorithm would be very useful for clustering streaming data or data from a real-time data-feed.

Appendix A

Measuring the Quality of Clusterings

The question of the quality of a clustering is a non-trivial one to answer. Especially, given that for unsupervised learning we often do not know the ground truth, determining the best clustering out of many clusterings is a challenging, if not ambiguous, problem to solve. However, there are a few heuristics and even an information theoretic framework to determine the quality of the clusters [27, 34, 35]. Here we discuss these approaches in details.

Approach with Cross Validation

As discussed in Section 2.7 briefly, running a v-fold cross validation is a common practice for choosing the best cluster number or deciding whether a clustering is better than the other [27]. In this method, the dataset is divided randomly into v equally sized parts. Then one of them is set aside as a test set and the other $v-1$ as the training set. Then clustering is computed on the training set and then the

cluster centers are assigned to the test set. The best model would be the one with the lowest sum of distance from the cluster centers and their respective assigned points. In general, we can use this technique with any function we are trying to minimize and test how well the clustering in the training set does in terms of minimizing the function in the test set.

Index for Cluster Analysis

There are a few indices for finding the comparative cluster quality [34, 35]. However, in this case we need to consider a set of labels for the dataset as the ground truth. Therefore, if we don't have labeled dataset these scores would not be the best measurement of cluster quality, rather they would measure the difference between the new clustering and the clustering, perceived as the ground truth. In both cases, the adjusted scores are defined as

$$\text{Adjusted Index} = \frac{\text{Index} - \text{Expected Index}}{\text{Max Index} - \text{Adjusted Index}}$$

Adjusted Rand Index (ARI)

Adjusted Rand Score is an adjusted index based on Rand measure. Rand Index is a clustering measure index based on set matching. It measures the similarity of two clustering based on the co-occurrence of elements in different clusters [34, 35].

For two partitions, $X = \{X_1, \cdots, X_r\}, Y = \{Y_1, \cdots, Y_s\}$ of a dataset $\mathcal{D}$ with n points we define the contingency table as follows:

X/Y	Y_1	Y_2	$\cdots$	Y_s	Sum
X_1	n_{11}	n_{12}	$\cdots$	n_{1s}	a_1
X_2	n_{21}	n_{22}	$\cdots$	n_{2s}	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
X_r	n_{r1}	n_{r2}	$\cdots$	n_{rs}	a_r
Sum	b_1	b_2	$\cdots$	b_s	

Here n_{ij} is the number of datapoints common between X_i, Y_j i.e.

$$n_{ij} = |X_i \cap Y_j|$$

Finally, ARI is defined as

$$\text{ARI} = \frac{\sum_{ij} \binom{n_{ij}}{2} - \left(\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right) / \binom{n}{2}}{\frac{1}{2}\left(\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right) - \left(\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right) / \binom{n}{2}}$$

Properties and Caveats. The value of ARI is always between -1 and 1. However, it is not a proper metric [35]. Moreover, ARI assumes a generalized hyper-geometric distribution as null. Therefore, many statisticians question the usefulness of this index with this strong assumption [34].

Adjusted Mutual Information (AMI)

AMI is an information theoretic measure [35]. It uses the mutual information between the clusterings and the entropy of each of the clusters to find the score for two different clusters. AMI is defined as follows:

$$\text{AMI}(U, V) = \frac{MI(U, V) - E[MI(U, V)]}{\max\{H(U), H(V)\} - E[MI(U, V)]}$$

Where $MI(U, V)$ is defined as the mutual information of two cluster partitions U, V, $H(X)$ is the entropy of the cluster partition U and $E[MI(U, V)]$ is the expected value of the mutual informaton $MI(U, V)$.

Properties and Caveats. The value of AMI lies in $[0, 1]$. While is also not a proper metric, it provides an information theoretic framework for calculating the similarity between two clusterings. Moreover, AMI does not have any strong assumptions and therefore AMI is a more robust cluster index compared with ARI.

Bibliography

[1] IBM. What is Big Data? *IBM White Paper*, 2013. URL http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html.

[2] Kevin P Murphy. Machine learning: a probabilistic perspective (adaptive computation and machine learning series), 2012.

[3] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. An Efficient k-Means Clustering Algorithm: Analysis and Implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002. doi: 10.1109/TPAMI.2002.1017616. URL http://doi.ieeecomputersociety.org/10.1109/TPAMI.2002.1017616.

[4] Joe H Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

[5] Andrew McCallum, Kamal Nigam, and Lyle H Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 169–178, 2000. doi: 10.1145/347090.347123. URL http://doi.acm.org/10.1145/347090.347123.

[6] Jinlan Tian, Lin Zhu, Suqin Zhang, and Lu Liu. Improvement and parallelism of k-means clustering algorithm. *Tsinghua Science & Technology*, 10(3):277–281, 2005.

[7] Reza Farivar, Daniel Rebolledo, Ellick Chan, and Roy H Campbell. A Parallel Implementation of K-Means Clustering on GPUs. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pages 340–345, 2008.

[8] Ren Wu, Bin Zhang, and Meichun Hsu. Clustering billions of data points using gpus. In *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*, pages 1–6. ACM, 2009.

[9] NSLP Kumar, Sanjiv Satoor, and Ian Buck. Fast parallel expectation maximization for gaussian mixture models on gpus using cuda. In *High Performance Computing and Communications, 2009. HPCC'09. 11th IEEE International Conference on*, pages 103–109. IEEE, 2009.

[10] You Li, Kaiyong Zhao, Xiaowen Chu, and Jiming Liu. Speeding up K-Means Algorithm by GPUs. In *Proceedings of the IEEE International Conference on Computer and Information Technology (CIT)*, pages 115–122, 2010. doi: 10.1109/CIT.2010.60. URL http://dx.doi.org/10.1109/CIT.2010.60.

[11] Weizhong Zhao, Huifang Ma, and Qing He. Parallel k-means clustering based on mapreduce. In *Cloud Computing*, pages 674–679. Springer, 2009.

[12] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Review*, 53(2):217–288, 2011. doi: 10.1137/090771806. URL http://dx.doi.org/10.1137/090771806.

[13] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666, 2010.

[14] Lyle H Ungar and Dean P Foster. Clustering methods for collaborative filtering. In *AAAI workshop on recommendation systems*, volume 1, 1998.

[15] Thrasyvoulos N Pappas. An adaptive clustering algorithm for image segmentation. *Signal Processing, IEEE Transactions on*, 40(4):901–914, 1992.

[16] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODS)*, pages 1027–1035, 2007. URL http://dl.acm.org/citation.cfm?id=1283383.1283494.

[17] Dan Feldman, Matthew Faulkner, and Andreas Krause. Scalable Training of Mixture Models via Coresets. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, pages 2142–2150, 2011. URL http://papers.nips.cc/paper/4363-scalable-training-of-mixture-models-via-coresets.

[18] Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A PTAS for k-means clustering based on weak coresets. In *Proceedings of the ACM Symposium on Computational Geometry*, pages 11–18, 2007. doi: 10.1145/1247069.1247072. URL http://doi.acm.org/10.1145/1247069.1247072.

[19] Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 291–300, 2004. doi: 10.1145/1007352.1007400. URL http://doi.acm.org/10.1145/1007352.1007400.

[20] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi R Varadarajan. The planar k-means problem is NP-hard. *Theoretical Computer Science*, 442:13–21, 2012. doi: 10.1016/j.tcs.2010.05.034. URL http://dx.doi.org/10.1016/j.tcs.2010.05.034.

[21] Amit Kumar, Yogish Sabharwal, and Sandeep Sen. A Simple Linear Time $(1+\epsilon)$-Approximation Algorithm for k-Means Clustering in Any Dimensions. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 454–462, 2004. doi: 10.1109/FOCS.2004.7. URL http://dx.doi.org/10.1109/FOCS.2004.7.

[22] Leon Mirsky. Symmetric gauge functions and unitarily invariant norms. 1960.

[23] Tim Kraska, Ameet Talwalkar, John C Duchi, Rean Griffith, Michael J Franklin, and Michael I Jordan. MLbase: A Distributed Machine-learning System. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, 2013. URL http://www.cidrdb.org/cidr2013/Papers/CIDR13_Paper118.pdf.

[24] Amol Ghoting, Rajasekar Krishnamurthy, Edwin P D Pednault, Berthold Reinwald, Vikas Sindhwani, Shirish Tatikonda, Yuanyuan Tian, and Shivakumar

Vaithyanathan. SystemML: Declarative machine learning on MapReduce. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 231–242, 2011. doi: 10.1109/ICDE.2011.5767930. URL http://dx.doi.org/10.1109/ICDE.2011.5767930.

[25] Peter Norvig. Teach yourself programming in ten years. URL http://norvig.com/21-days.html#answers.

[26] Scott Collin. Latency numbers every programmer should know. URL http://www.eecs.berkeley.edu/~rcs/research/interactive_latency.html.

[27] Duc Truong Pham, Stefan S Dimov, and CD Nguyen. Selection of k in k-means clustering. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 219(1):103–119, 2005.

[28] Wm A Wulf and Sally A McKee. Hitting the memory wall: implications of the obvious. *ACM SIGARCH computer architecture news*, 23(1):20–24, 1995.

[29] Yann Lecun and Corinna Cortes. The MNIST database of handwritten digits. URL http://yann.lecun.com/exdb/mnist/.

[30] Gael Guennebaud, Benoit Jacob, et al. Eigen v3, 2010. URL http://eigen.tuxfamily.org.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[32] Conrad Sanderson. Armadillo: An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments. Technical report, NICTA, 2010. URL http://arma.sourceforge.net/.

[33] Shivkumar Chandrasekaran, BS Manjunath, Yuan-Fang Wang, Jay Winkeler, and Henry Zhang. An eigenspace update algorithm for image analysis. *Graphical Models and Image Processing*, 59(5):321–332, 1997.

[34] Silke Wagner and Dorothea Wagner. *Comparing clusterings: an overview*. Universität Karlsruhe, Fakultät für Informatik Karlsruhe, 2007.

[35] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1073–1080. ACM, 2009.