Clustering Faster and Better with Projected Data

Alibek Zhakubayev and Greg Hamerly

Abstract

The K-means clustering algorithm can take a lot of time to converge, especially for large datasets in high dimension and a large number of clusters. By applying several enhancements it is possible to improve the performance without significantly changing the quality of the clustering. In this paper we first find a good clustering in a reduced-dimension version of the dataset, before fine-tuning the clustering in the original dimension. This saves time because accelerated K-means algorithms are fastest in low dimension, and the initial low-dimensional clustering bring us close to a good solution for the original data. We use random projection to reduce the dimension, as it is fast and maintains the cluster properties we want to preserve. In our experiments, we see that this approach significantly reduces the time needed for clustering a dataset and in most cases produces better results.

1 Introduction

K-means is a very popular clustering algorithm. It splits data points into k disjoint clusters, where k is provided by the user. K-means is used in many different applications. It starts with randomly selecting k points from the dataset. These points are used as centers of the clusters. Then, it calculates the distances between each point and the k cluster centers. Each point is assigned to its closest cluster center. Then the centers move to the average location of their assigned points. These two steps (point assignment, moving centers) repeat until no points switch clusters.

The number of K-means iterations varies depending on the data and the initialization. If the number of iterations is high, the clustering process can take a lot of time. Therefore, many research papers proposed different enhancements for the K-means clustering algorithm that will improve the performance of the model without losing the quality of prediction. For example, multiple researchers proposed methods that use the triangle inequality to omit unnecessary distance calculations [7, 8]. They have shown that the K-means clustering algorithm with their enhancements produces the same results, but significantly faster.

Dimensionality reduction algorithms map the data from m dimensions down to to $n < m$ dimensions. If the projection is good enough, reduced data can be used in many different applications. For example, we may seek a 2-dimensional projection for visualization purposes. The theorem of Johnson and Lindenstrauss tells us that it is possible to project high dimensional data to much lower dimensions in a way that preserves the point-point squared distances with high probability [5]. It shows us that in many scenarios we can reduce the dataset dimension without losing much information.

Random projection is one technique that can reduce the dimension of the dataset time efficiently [4, 12]. If we represent the dataset as $r \times n$ matrix, where r is the number of rows and n is the original dimension, we can reduce the dimension by multiplying the matrix by a randomly constructed $n \times m$ matrix. As a result, we will have a dataset that has m dimensions.

In this paper, we are using the random projection dimension reduction technique to speed up the K-means clustering process. By reducing the dataset and running K-means clustering on the dataset with lower dimensions we are getting a set of centers that we use for clustering in higher dimensions. Then, we limit the number of iterations of the K-means algorithm on higher dimensions. We claim that assuming correct parameters this technique can cluster the data quickly without losing in quality. This is based on ideas first proposed by Dasgupta [5].

2 Related Works

There are many published proposals for different acceleration techniques for clustering. We are interested in accelerating Lloyd's k-means algorithm, specifically for high-dimensional data. Our algorithm uses Hamerly's acceleration, which is derived from Elkan's algorithm. We describe both of those algorithms in detail. Then, we describe the Theorem of Johnson and Lindenstrauss that can be used to prove the quality of the Random Projection algorithm.

2.1 Elkan's Algorithm

Lloyd's k-means algorithm does many unnecessary distance calculations between points and centers. Elkan uses the triangle inequality to identify and skip those calculations[7]. We say that a distance function d obeys the triangle inequality if for any three points x, y, z, it is always the case that $d(x, y) \leq d(x, z) + d(z, y)$. The triangle inequality alone does not directly give the information we need to skip distance calculations. However, Elkan showed two lemmas derived from the triangle inequality and those lemmas allow the algorithm to skip unnecessary distance calculations. Assume that x is a point and b and c are centers. We can skip calculating $d(x, c)$ if we know that $d(x, c) \leq d(x, b)$, because c cannot possibly be the assigned center for x.

The lemmas are, for point x and centers b and c:

1. if $d(b, c) \geq 2d(x, b)$ then $d(x, c) \geq d(x, b)$, and
2. $d(x, c) \geq \max(0, d(x, b) - d(x, c))$.

Elkan stores several bounds so that these lemmas can avoid distance calculations:

- one upper bound per point on the distance between that point and its closest center,
- k lower bounds per point on the distance between that point and each of the k centers, and
- $O(k^2)$ inter-center distances.

By using this knowledge and derived lemmas he produced a set of rules where the algorithm can skip many distance calculations. For detailed information about the algorithm, see [7].

Elkan tested his algorithm on six datasets with three different values of k (3, 20, and 100). The results showed that Elkan's algorithm can significantly speed up Lloyd's k-means algorithm. When the number of clusters k increases, we observe greater speedup, because the algorithm is more effective at reducing the (relative) number of distance calculations required.

2.2 Hamerly's Algorithm

This algorithm is a modification of the Elkan algorithm[8]. One of the major differences is in how many lower bounds are used, and what they represent. Elkan stores a lower bound for each point-center combination. Hamerly stores a lower bound only between each point and its second closest center. In addition, Hamerly's algorithm uses the same upper bounds as Elkan's algorithm.

While using one bound is not always more effective at reducing the number of distance calculations, it can result in faster computations, especially in low dimension. One of the main contributions of the algorithm is the use of a single lower bound. The author claims that it can skip 80% or more executions of the innermost loop. Another improvement of Hamerly's algorithm is in space complexity. Since it uses fewer lower bounds, it can use far less memory when k is large.

The algorithm outperforms Elkan's algorithm when the dimension is low. More specifically, when the dimension of the dataset is less than 50, the Hamerly algorithm tends to be more efficient.

The condition that Hamerly uses is $u(i) \leq \max(l(i), s(a(i))/2)$, where $u(i)$ is the upper bound on point x_i, $l(i)$ is the lower bound, $a(i)$ is the index of the assigned center, and $s(j)$ is the distance from cluster center j to the other center closest to j.

2.3 Random Projection

Random Projection is a dimension reduction technique. Dasgupta [4] used the technique to perform several experiments on real and synthetic data. Random projection multiplies the dataset with dimension d with a matrix filled with random values of size $d \times d'$, where $d' < d$, to get a dataset with lower dimension. There are two steps to create a random matrix. First, we initialize the matrix with Gaussian distribution $N(0, 1)$. Then, we make rows of the matrix orthogonal by applying Gram-Schmidt algorithm. Other random matrix construction methods are also possible and useful, such as sparse and binary matrices [1, 11].

2.4 The theorem of Johnson and Lindenstrauss

Dasgupta et al. provided a proof of the theorem of Johnson and Lindenstrauss [5]. The theorem shows us that we can map high dimensional dataset with n points to $O(\log n/\varepsilon^2)$-dimensional space so that the distance between each pair of points has an error of no more than ε, with high probability.

For mapping a dataset the authors used a random projection described in the previous subsection.

This theorem shows us that we can reduce the dimension of the data without losing much information, especially the distance information that we need for clustering with distance-based algorithms like k-means.

3 Methodology

In this section, we present the experimental setup used to speed up the k-means algorithm. First, we will provide an overview of our experiments. The dimensionality reduction algorithm that was used is then presented. Then, we discuss about the comparisons that we are making. Following that, we provide the experiment's evaluation metric that we use. Finally, we discuss used datasets.

3.1 Overview

The k-means method iteratively selects new centroids. It is completed when no points are moved to a different center after one iteration. We compute the distance between each point and each new center on each iteration. The points in the same group are then used to locate a new cluster center. The number of iterations can be large, and computing the distances might be time-consuming. The algorithm may then take a large amount of time to complete, depending on the dataset and initialization.

We propose a k-means algorithm that uses a dimensionality reduction algorithm along with another k-means model to initialize the points. First, we reduce the dimensionality of the dataset. Then, we use labels from the reduced data as an input to the k-means algorithm. Computing the distance between the points and the centers should take less time. We expect the first k-means method to converge much faster than the standard k-means approach. Depending on the dataset, the Theorem of Johnson and Lindenstrauss tells us that points that are close in higher dimensions should be within one epsilon in lower dimensions. Then, following the first k-means, we presume that many points will be correctly grouped. Finally, we run the algorithm on the entire dataset using the labels from the first k-means. We also limit the number of iteration on the second k-means to 40, because the initial assignments help us to get good results faster. We can set this parameter and limit the number of iterations to a different number. However, our experiments showed that choosing 40 as a limit works well and we are able to get assignments faster and their quality is better.

Algorithm 1 shows a pseudocode of the speeding up process. We first get initialization of the centers using the k-means++ algorithm. Then, we get the dataset with lower number of dimensions using the Random Projection. Then, we get the centers and assignments by applying Hamerly's version of k-means algorithm. Finally, we run the algorithm again on higher dimensions and limit the number of iterations. We limit the number of iterations, because the initial assignments are good enough and we are able to get the better assignments without waiting the convergence of the algorithm.

Algorithm 1: k-means speed up using random projection

```
1 k-meansDataProjection (data, d', k);
  Input  : data – dataset with n points in d dimensions
           d' – reduced number of dimensions
           k – number of clusters
  Output: centers - cluster centers
2 init = Kpp(data, k) // get k center assignments using k-means++
3 redData = RandomProjection(data, d, d') // project data to lower dimension
4 a1, c1 = HamerlyKMeans(redData, init) // reduced dimension
5 assignment, centers = HamerlyKMeans(data, a1, maxiterations) // original dimension
```

3.2 Dimension reduction technique

In this experiment, we use random projection to reduce the dimensionality of the dataset.

Random Projection is a dimensionality reduction approach that is extremely quick. Let N be the number of data points and M be the number of features. The result is an $N \times M$ matrix. This matrix is multiplied by a random $M \times P$ matrix, where P is the desired number of dimensions. As a result, we have an $N \times P$ dataset. Each value in the random matrix has a float number between 0 and 1. Because we only perform one matrix multiplication and random matrix initialization, this approach is reasonably fast. The disadvantage of this strategy is that the reduced dataset may not accurately represent the initial dataset.

Depending on the dataset we decide how many features we want to lower the dimension. This parameter should be set by the user. The higher the clusterability of the dataset, to lower the dimensions we can cluster our data. In order to prove it, we generated several synthetic datasets with different clusterability. We get a value of how clusterable the data is using Hopkins statistics [9, 3]. Then, we compare the significance in the difference between two predictions that use different dimensions.

3.3 Comparisons

We compare our implementation of the k-means to Lloyd's implementations with an enhancements that were proposed by Hamerly[8]. Hamerly algorithm is described in section 2.2. The main idea is that it uses a triangle inequality to skip unnecessary computations and only store one lower bound.

In all experiment runs we first initialize the algorithm with the k-means++ technique [2]. The k-means is very dependent on the initialization. The quality of the solutions can differ significantly for two different initializations. Therefore, we have used one of the most widely spread algorithm to get initial assignments.

3.4 Metrics

Our experiment's metrics are time and distortion. We compare how long it takes to run the k-means algorithm. It is measured in seconds. Distortion is the second metric. It shows how well an algorithm clustered the data. The data points should be as close to the centers as possible. The sum of the square distances between each point and its center is the distortion.

3.5 Datasets

The datasets that were used: KDDCUP04Bio, CIFAR-10, CIFAR-100[10] and MNIST[6]. Table 1 contains information about the datasets that were used in the experiments. Even though the k in KDD CUP 04 dataset is 2000, we have used this dataset with different k values.

Name	k	n	d
KDD CUP 04	2000	145751	74
CIFAR-10	10	50000	3072
CIFAR-100	100	50000	3072
MNIST	10	60000	784

Table 1: Information about datasets

3.6 Experimental Conditions

C++ language was used to implement all parts of the experiment. We run our code on the Intel(R) Xeon(R) CPU $E5 - 2695\ v4$ @ 2.10GHz machines.

We used Kodiak server which is a high performance computer cluster provided by the Baylor University. We are using 8 nodes to run each part of the experiment. This way we are making sure that nothing else is running on the machine. Therefore, the time comparisons are fair. The github link to the project is https://github.com/alibeklfc/K-means

4 Results

4.1 Clusterability

The number of dimensions to which we are projecting data is the most important decision that the user will make. Incorrect setup can significantly reduce the model's performance. The first k-means can produce incorrect assignments if the parameter is set too low. By reducing dimension we introduce noise to a dataset. Then, by reducing too much an algorithm can produce assignments that are far from local minimum in original dimensions. As a result, running only 40 iterations in higher dimensions will most likely be insufficient. Therefore, correctly setting the number of reduced dimensions is critical.

We claim that the clusterability of a dataset allows us to predict the parameter. The Hopkins statistic is the metric for clusterability that we employ. Equation 1 is a formula of Hopkins statistic. We select random m points from our dataset, and we also generate random m points in our space. Then, we compute the nearest neighbors for all this points. u is a list of the distances between the randomly generated points and their closest neighbors. Similarly, w is an array of distances between sample and randomly generated points [3].

The higher the Hopkins statistic, the better the data clusterability. We assume that clusterability is related to the reduced dimension parameter we must select. We created 12 synthetic datasets to demonstrate our point. The Hopkins statistic values in the datasets are different. We use our enhancements to run the K-means algorithm, which reduces the dimensions to 5, 20, and 40. The experiment was carried out a hundred times in per parameter value. The distortion values are gathered and compared.

$$H = \frac{\sum_{i=1}^{m} u_i^d}{\sum_{i=1}^{m} u_i^d + \sum_{i=1}^{m} w_i^d} \tag{1}$$

We used a t-test to get the significance of the difference between different setups. The common practice is that if the p-value from the t-test is below 0.05, the difference is significant. Otherwise, it is not significant. Each data point in the graph represents a p-value that we get by comparing distortion values between two groups. The size of each group is 100.

Figure 1 shows a line chart with the Hopkins statistic on the x-axis and the t-test between two groups on the y-axis. The horizontal black line represents a threshold. When lines are above the black line, the difference in distortion between the two groups is insignificant. Otherwise, the difference is significant. The red line represents a t-test between experiments when we reduced the dataset into the 40 and 20 dimensions. Only at point 0.65, the red line is below the threshold. This tells us that for all except one dataset there was no significant difference in cluster assignment quality between the two methods. Therefore, it might be

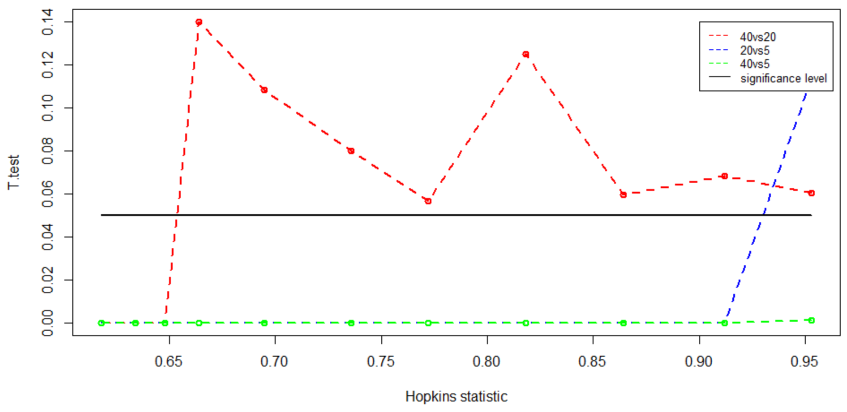

Figure 1: Hopkins Clusterability vs t-test(the significance between differences in distortions). The black line is the significance threshold. The red, blue, and green lines represent a significance in the difference between the two different methods. If the line is below the black line, the difference is significant.

reasonable to choose 20 dimensions, because that might save some time. The blue line represents a t-test for the datasets when we reduced the dimension into the 40 and 5 dimensions. We can see that only when the clusterability is very high, there is no difference. Therefore, there will be a significant difference in the quality of the results. Finally, the green line shows a t-test for 40 and 5 dimensions. The line is horizontal when the p value is 0. It means that the difference is significant for all datasets.

This figure shows us that the choice of parameter is very important because incorrect setup might lead to poor clustering. There is a connection between the clusterability of the dataset and the number of dimensions that we can choose. For the datasets with higher clusterability, it is a reasonable choice to select a lower reduced dimension parameter. Otherwise, if the clusterability is low, higher reduced dimension usage is preferred.

Even though clusterability is important information, the overall complexity of the K-means is another feature that should be taken into account.

4.2 Experiments

As mentioned, four different experiments were executed to test the performance of our enhancements. The metrics used are time and distortion. The time is measured in seconds, and it measures the total time required to get the output of the K-Means. When we measure time for our algorithm, it is a total time of the dimensionality reduction and two K-Means models.

Table 2 shows the results of the experiments. Four different datasets are used: KDD CUP 04, CIFAR-10, CIFAR-100, and MNIST. The table also shows the algorithm that was used. We reduce each dataset to 5, 20, and 40 dimensions. We then compare the results with the Hamerly algorithm. We run each algorithm for each dataset 100 times. Then, we averaged the results and put average distortion and time values into the table. The results that are better than the other ones are out in a bold style.

As shown in the subsection above, by reducing into the higher dimension we expect the distortion value to be higher. KDD CUP 04 dataset ($k = 2000$ and $k = 1000$) proves that claim. Using our algorithm when we reduce the dataset into the 5 dimensions, the distortion results are worse in comparison with the Hamerly algorithm. However, when we reduce the dataset into the 20 and 40 dimensions, the distortion values are better than the Hamerly algorithm. The average time for all of our algorithms is also better.

The results show that overall complexity of the model is an important factor. Considering KDD CUP 04 dataset, we see that the difference in distortion is not better when k is 10 and 100. Then, increasing k to

Dataset	Algorithm	k	Avg Distortion	Multiplier of distortion	Avg Time	Speed up	T.test distortion	T.test time
MNIST	Hamerly	10	2555694		27.11			
MNIST	Reduce to 5	10	**2554452**	0.9995	**20.69**	1.31	0.10	0.00
MNIST	Reduce to 20	10	**2554283**	0.9994	**21.32**	1.27	0.12	0.00
MNIST	Reduce to 40	10	**2555447**	0.9999	**21.12**	1.28	0.84	0.00
CIFAR-10	Hamerly	10	**7901887**		94.50			
CIFAR-10	Reduce to 5	10	7904669	1.0003	**64.24**	1.47	0.05	0.00
CIFAR-10	Reduce to 20	10	7903508	1.0002	**61.11**	1.54	0.28	0.00
CIFAR-10	Reduce to 40	10	7903755	1.0002	**62.50**	1.51	0.02	0.00
CIFAR-100	Hamerly	100	6422940		2544.63			
CIFAR-100	Reduce to 5	100	**6417810**	0.9992	**1254.80**	2.02	0.00	0.00
CIFAR-100	Reduce to 20	100	**6416476**	0.9989	**1165.39**	2.18	0.00	0.00
CIFAR-100	Reduce to 40	100	**6417299**	0.9991	**1118.11**	2.27	0.00	0.00
KDD CUP 04	Hamerly	10	**2423367**		5.64			
KDD CUP 04	Reduce to 5	10	2425568	1.0001	**3.61**	1.56	0.93	0.00
KDD CUP 04	Reduce to 20	10	2424068	1.0000	**3.41**	1.65	0.97	0.00
KDD CUP 04	Reduce to 40	10	2423586	1.0000	**4.64**	1.21	0.19	0.00
KDD CUP 04	Hamerly	100	**1111354**		205.23			
KDD CUP 04	Reduce to 5	100	1113860	1.0002	**98.08**	2.09	0.05	0.00
KDD CUP 04	Reduce to 20	100	1114225	1.0002	**121.41**	1.69	0.04	0.00
KDD CUP 04	Reduce to 40	100	1112577	1.0001	**161.90**	1.26	0.00	0.00
KDD CUP 04	Hamerly	1000	659591		2360.47			
KDD CUP 04	Reduce to 5	1000	660936	1.0002	**1300.69**	1.81	0.00	0.00
KDD CUP 04	Reduce to 20	1000	**659294**	0.9995	**1620.25**	1.45	0.00	0.00
KDD CUP 04	Reduce to 40	1000	**658562**	0.9984	**2165.76**	1.09	0.00	0.00
KDD CUP 04	Hamerly	2000	561566		3890.97			
KDD CUP 04	Reduce to 5	2000	562944	1.0002	**2632.96**	1.47	0.00	0.00
KDD CUP 04	Reduce to 20	2000	**560229**	0.9997	**3046.14**	1.27	0.00	0.00
KDD CUP 04	Reduce to 40	2000	**559634**	0.9996	**3733.67**	1.04	0.00	0.00

Table 2: Results of the experiment

1000 and 2000 show that the distortion values are significantly better in comparison with Hamerly algorithm. Also, the time required to get the clusters is still significantly faster.

CIFAR-10 and CIFAR-100 datasets contain image pixels that are flattened. Two datasets contain a different number of classes. Therefore, the number of clusters in both experiments differs. For the CIFAR-10 dataset, our algorithms need less time in comparison with the Hamerly algorithm. However, the distortion values are worse for all three algorithms.

Our algorithms for the CIFAR-100 dataset produce better average distortion and time than the Hamerly algorithm. Also, the average time for all three algorithms is approximately half. Even though for the CIFAR-100 dataset our algorithm produced better clusters, on the CIFAR-10 dataset the algorithm produced a worse solution.

Finally, we run our algorithms and classic Hamerly algorithm for the MNIST dataset 100 times. As shown in the table 2, both distortion and time average values are better in our algorithm.

5 Conclusion

Clustering is one of the biggest and important problems in data science. There are many different applications that use different clustering algorithms. Therefore, producing good clusters is important for the performance of the application. K-means is one of the most widely used clustering algorithms. The user of the algorithm should specify the number of output clusters, and the algorithm will always converge. Also, the complexity of the K-means might be very high. In this scenario, a lot of time might be needed for the algorithm to converge. Since time is one of the most important metrics in many applications, many research works have been published that proposed enhancements to the Lloyd's K-means algorithm that allow it to converge faster.

In this paper, we used Hamerly's implementation of the algorithm for the K-means. We clustered faster and better with projected data. By reducing the dimensionality of the dataset, we decrease the complexity of the algorithm. Then, K-means on projected data will converge faster. Then, we use the assignments that we got from the projected k-means and use it as an input to the k-means on initial data. We assume that the assignments that we received from the projected k-means are good. Therefore, we will not run the algorithm until it converges and only runs it for some number of iterations.

We project the data using a random projection algorithm. Its main advantage is efficiency. Our hypothesis is that projecting the data, running k-means in lower dimensions, and then using obtained assignments in higher dimensions can produce clusters better and faster.

We also analyzed the influence of clusterability on the parameter selection of the algorithm. In particular, the number of dimensions we should project our data. We found that the more clusterable the data is, to lower dimensions we can project it.

This work shows that projecting data to lower dimensions can improve the performance of the K-means algorithm. We were able to get better output clusters and did it faster. Even though the results are encouraging, for one dataset our algorithm performed worse in comparison with Hamerly's implementation of the K-means.

Projecting the data into lower dimensions and running in k-means on projected data can produce cluster assignments that can be used in original dimensions. If we initialize the dataset using this method, we can set a limit to the number of iterations needed. This algorithm can output clusters faster and better.

References

[1] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003.

[2] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[3] Amit Banerjee and Rajesh N Dave. Validating clusters using the hopkins statistic. In *2004 IEEE International conference on fuzzy systems (IEEE Cat. No. 04CH37542)*, volume 1, pages 149–153. IEEE, 2004.

[4] Sanjoy Dasgupta. Experiments with random projection. *arXiv preprint arXiv:1301.3849*, 2013.

[5] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2003.

[6] Li Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[7] Charles Elkan. Using the triangle inequality to accelerate k-means. In *Proceedings of the 20th international conference on Machine Learning (ICML-03)*, pages 147–153, 2003.

[8] Greg Hamerly. Making k-means even faster. In *Proceedings of the 2010 SIAM international conference on data mining*, pages 130–140. SIAM, 2010.

[9] Brian Hopkins and John Gordon Skellam. A new method for determining the type of distribution of plant individuals. *Annals of Botany*, 18(2):213–227, 1954.

[10] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[11] Ping Li, Trevor J Hastie, and Kenneth W Church. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 287–296, 2006.

[12] Santosh S Vempala. *The random projection method*, volume 65. American Mathematical Soc., 2005.