Scalable Kernel k-Means via Centroid Approximation

Byung Kang
KAIST
Republic of Korea
byungkon@kaist.ac.kr

Woosang Lim
KAIST
Republic of Korea
quasar17@kaist.ac.kr

Kyomin Jung
KAIST
Republic of Korea
kyomin@kaist.edu

Abstract

Although kernel k-means is central for clustering complex data such as images and texts by implicit feature space embedding, its practicality is limited by the quadratic computational complexity. In this paper, we present a novel technique based on scalable centroid approximation that accelerates kernel k-means down to a sub-quadratic complexity. We prove near-optimality of our algorithm, and experimentally confirm its efficiency.

1 Introduction

As data representations grew complex, traditional clustering methods adopted similarity functions, known as *kernels*, to compactly represent and analyze the patterns in an embedded space [11, 2, 7, 13, 8]. One such approach is the design of a kernelized clustering algorithm known as *kernel k-means* [5]. Although it is useful over many domains, a major drawback of kernel k-means is its scalability. Because of its quadratic complexity $\Omega(n^2)$ with respect to the number of data points n, kernel k-means becomes quickly inefficient for large real-world data sets. In this paper, we present a novel centroid approximation algorithm called CATS (**C**entroid **A**pproximation **T**hrough **S**ampling) that accelerates kernel k-means algorithm to complexity $O(n^{1+\delta})$ for any small constant $\delta \in (0, 1)$.

2 Preliminaries

2.1 Kernel Methods

In many machine learning domains, non-linearity in data sets becomes troublesome. The *kernel method* deals with such non-linearity by mapping the data implicitly into some high dimensional space. We denote such high dimensional mapping by $\phi : \mathcal{X} \mapsto \mathcal{F}$, where $\mathcal{X} = \{\mathbf{x_1}, \cdots, \mathbf{x_n}\}$ is the set of data points and $\mathcal{F}$ is an unknown feature space to which the data are embedded. For example, $\mathcal{X} \subset \mathbb{R}^s$ and $\mathcal{F} \subset \mathbb{R}^t$ such that $s < t$.
A function $\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is called a kernel function if $\kappa(\mathbf{x_i}, \mathbf{x_j}) = \phi(\mathbf{x_i}) \cdot \phi(\mathbf{x_j})$ for some $\phi : \mathcal{X} \mapsto \mathcal{F}$, where $\mathbf{x} \cdot \mathbf{y}$ denotes the dot product. For convenience, we will simplify $\kappa(\mathbf{x_i}, \mathbf{x_j})$ as $\kappa(i, j)$. The gist of kernel methods is that as long as κ is symmetric and positive semi-definite, the algorithms can exploit the high-dimensional embeddings without explicitly having to know the embedded representation ϕ [10]. Some commonly used kernel functions that satisfy these properties include the Gaussian kernel ($\kappa(\mathbf{x_i}, \mathbf{x_j}) = \exp\left(-r\|\mathbf{x_i} - \mathbf{x_j}\|^2\right)$) and the Sigmoid kernel ($\kappa(\mathbf{x_i}, \mathbf{x_j}) = \tanh\left(c\mathbf{x_i} \cdot \mathbf{x_j} + \theta\right)$) [5].
While this property is advantageous for algorithm design, we show in later subsections that it is harmful to the computational complexity of kernel k-means.

2.2 Kernel k-Means

Dhillon et al [5] proposed a kernel k-means clustering algorithm to deal with non-linearity. Like the original k-means, kernel k-means takes the distortion as a measure of the clustering quality. Let k be a constant. Given a clustering $\bar{C} = \{C_1, \cdots, C_k\}$, the distortion of $\bar{C}$ is given as $D_{\bar{C}} = \sum_h D(C_h)$, where the individual distortions $D(C_h)$ are defined as the sum of squared distances between $\phi(\mathbf{x_i})$ ($\mathbf{x_i} \in C_h$) and the centroid $\mathbf{m_{C_h}} = \frac{1}{m}\sum_{i \in C_h} \phi(\mathbf{x_i})$ with $m = |C_h|$. Kernel k-means aims to find a partition $\bar{C} = \text{argmin}_{\bar{C}} D_{\bar{C}}$ in a manner similar to k-means. This optimization is an NP-hard problem [3], so kernel k-means uses an EM-based iterative method as in the original k-means. At each iteration of kernel k-means, each point i selects the cluster $\text{argmin}_C ||\phi(\mathbf{x_i}) - \mathbf{m_C}||_2^2$. Once all points have selected their clusters, each cluster C updates its $\mathbf{m}_C$. To compute $\text{argmin}_C ||\phi(\mathbf{x_i}) - \mathbf{m_C}||_2^2$, we expand the expression as

$$||\phi(\mathbf{x_i}) - \mathbf{m_C}||_2^2 = \kappa(i,i) - \frac{2}{m}\sum_{j \in C}\kappa(i,j) + \frac{1}{m^2}\sum_{j,r \in C}\kappa(j,r), \tag{1}$$

thereby representing $||\phi(\mathbf{x_i}) - \mathbf{m_{C*}}||_2^2$ without using ϕ.
The main computational bottleneck comes from the second and third terms of Eqn (1). For example, computing the second term in Eqn (1) requires $\Theta(n)$ operations per each point i, which leads to an overall cost of $\Theta(n^2)$ per iteration. This quadratic time complexity becomes problematic when we deal with realistic datasets, where the number of data points ranges in the order of 10^4 or higher.

3 Algorithm

To alleviate this matter, we exploit the fact that distance relations of points are preserved well even after a low-dimensional random projection [4, 1]. Thus, we propose to approximate the centroid of each cluster C in a random subspace of $\mathcal{F}$. To be more precise, we let for any $x \in \mathcal{F}$,

$$D_C(x) = \sum_{i \in C} \|\phi(\mathbf{x_i}) - x\|_2^2. \tag{2}$$

Note that $\mathbf{m_C} = \text{argmin}_{x \in \mathcal{F}} D_C(x)$. In our algorithm, we randomly sample l points into the set $S \subseteq C$, and define the approximate centroid to be $\tilde{\mathbf{m}}_C = \text{argmin}_{x \in \text{Span}(S)} D_C(x)$. We represent the approximate centroid as

$$\tilde{\mathbf{m}}_C = \sum_{i \in S} \alpha_i \phi(\mathbf{x_i}), \tag{3}$$

where α_i's are the coefficients that we optimize. With a slight abuse of notation, we denote $D_S \equiv D_C(\tilde{\mathbf{m}}_C)$ for simplicity. The values of α_i's are determined by minimizing the distortion $D_C(\sum \alpha_i \phi(\mathbf{x_i}))$. We express the distortion as a function of α_i's:

$$\begin{aligned} D_C\left(\sum \alpha_i \phi(\mathbf{x_i})\right) &= \sum_{i \in C}\left\{\kappa(i,i) - 2\sum_{j \in S}\alpha_j \kappa(i,j) + \sum_{j,t \in S}\alpha_j \alpha_t \kappa(j,t)\right\} \\ &= Tr(K) - 2\sum_{i \in C, j \in S}\alpha_j \kappa(i,j) + n\sum_{i,j \in S}\alpha_i \alpha_j \kappa(i,j), \end{aligned}$$

where K is the Gram matrix of the kernel function κ such that $K_{i,j} = \kappa(i,j)$. Using the fact that $D_C(\cdot)$ is quadratic and convex over the α_i's, we show that the optimal α_i's can be computed efficiently. By differentiating $D_C(\cdot)$ with respect to the α_i's and solving for zero, we obtain the minimizer $\boldsymbol{\alpha} = (\alpha_i)$ of $D_C(\cdot)$ as:

$$\boldsymbol{\alpha} = \frac{M^+ L \mathbf{e_1}}{n}, \tag{4}$$

where $L \in \mathbb{R}^{l \times n}$ is a matrix that contains the kernel values between points in S (rows) and C (columns), and $M \in \mathbb{R}^{l \times l}$ is a matrix containing the kernel values over $S \times S$. Here, $M^+ = V\Sigma^+ U^\top$, where $M = U\Sigma V^\top$ is the singular value decomposition of M and $\Sigma^+_{i,j} = 1/\Sigma_{i,j}$ if $\Sigma_{i,j}$ is nonzero, and zero otherwise.

One thing to be cautious about our algorithm is the convergence. Since we are approximating the distortion through random sampling, there is no guarantee that the distortion would decrease monotonically. This leads to the possibility of oscilation towards the end of the iteration. To overcome

this problem, we halt the iteration if the variance of the past w distortions is below a given small threshold. This overall procedure is outlined in Algorithm 1. In our experiments, the initialization of $\bar{C}$ is performed by randomly choosing k points, and gathering points around those initial k points.

Algorithm 1 CATS Kernel k-Means

Input: Set of points P, Number of clusters k, Window length w, Variance threshold θ
Output: A clustering $\bar{C} = \{C_1, \cdots, C_k\}$

1: Randomly initialize $\bar{C} = \{C_1, \cdots, C_k\}$
2: **while** the variance of the last w distortions $> \theta$ **do**
3: **for all** C_j **do**
4: $S_j \leftarrow$ Uniformly sample l points from C_j
5: Compute $\boldsymbol{\alpha}_j$ (Eqn (4)) and cache $\tilde{\mathbf{m}}_{\mathbf{C_j}} \cdot \tilde{\mathbf{m}}_{\mathbf{C_j}}$
6: **end for**
7: **for all** $i \in P$ **do**
8: Assign i to $C \leftarrow \text{argmin}_C \, ||\phi(i) - \tilde{\mathbf{m}}_{\mathbf{C}}||_2^2$
9: **end for**
10: **end while**

3.1 Analysis

In this section, we first present the per-iteration time complexity. The complexity of the original kernel k-means is $\Omega(n^2)$ per iteration, for example, from computing $\mathbf{m_j} \cdot \mathbf{m_j}$. In CATS kernel k-means, the complexity of computing $\tilde{\mathbf{m}}_{\mathbf{C_j}} \cdot \tilde{\mathbf{m}}_{\mathbf{C_j}}$ is $\Theta(l^2)$. Solving for $\boldsymbol{\alpha}$ as in Eqn (4) requires a SVD of M and multiplication of M^+ and L. This takes $O(l^3 + nl)$ time per cluster. For each point i, the complexity of computing a single distortion with respect to i (line 8) is $\Theta(l)$ since the kernel is evaluated against only the sampled points. This leads to the point-loop (lines 7 to 9) complexity of $\Theta(nl)$. Therefore, the final complexity is $O(l^3 + nl)$.

Now we present the performance bound, whose proof is given in Appendix A.

Theorem 1. *Let $D_C \equiv \min_{x \in \mathcal{F}} D_C(x)$. By taking l samples, $E[D_S] - D_C \leq \frac{1}{l} D_C$ for the CATS kernel k-means algorithm.*

For example, if $l = n^\delta, \delta \leq \frac{1}{2}$, we get the bound $E[D_S] - D_C \leq n^{-\delta} D_C$ in $O(n^{1+\delta})$ time per-iteration.

4 Experiments

We tested our algorithm on the UCI database [6] and the MNIST database [9], both of which consist of handwritten digit images. The UCI dataset has 10,992 16-dimensional points. For the MNIST data, we randomly selected 10,000 points out of the original 70,000 748-dimensional points. We compared our approach against the kernel k-means clustering with $k = 10$ and $l = \sqrt{n/k}$ using a sigmoid kernel $\tanh(0.0045(\mathbf{x_i} \cdot \mathbf{x_j}) + 0.11)$ as in [14]. The clustering quality measure we use is the *normalized mutual information* (NMI) [12]:

$$NMI(C_1, C_2) = \frac{\sum_{c_i \in C_1, c_j \in C_2} p(c_i \cap c_j) \log \frac{p(c_i \cap c_j)}{p(c_i)p(c_j)}}{\max(H(C_1), H(C_2))},$$

where C_1 and C_2 are equal-sized sets of clusters, and H is the entropy function. The stopping criterion compares the variance of the 10 most recent samples against the threshold of 2E-4 to determine whether or not to stop.

Because we use a randomized algorithm, we ran the algorithm 100 times on each initial clustering to get the mean and the standard deviations of distortion, time, and the NMI. To explore the effect of different initial clusterings, we ran the algorithm over three different initial clusterings each, whose results are denoted as MNIST1, ..., MNIST3, UCI1, ..., UCI3. We report these mean values in Table 1, along with the respective standard deviations. For fair comparison, we allowed the kernel k-means to stop early using the same criterion as ours, and the corresponding distortion and time are shown in the upper row of each line (KKM-v). The bottom row shows the values acquired when

we let the kernel k-means run until convergence (KKM-c). The numbers show that our algorithm closely matches kernel k-means in clustering quality, using much less time. Under the same stopping criterion, there are instances in which our algorithm outperforms kernel k-means (MNIST 1, UCI 3).

		MNIST 1	MNIST 2	MNIST 3	UCI 1	UCI 2	UCI 3
Distortion	KKM-v	19.1328	19.1288	19.1018	3.7353	3.5904	3.6888
	KKM-c	19.0967	19.1273	19.0913	3.7351	3.5904	3.5447
	CATS	19.1221	19.134	19.1291	3.7358	3.5904	3.6356
		(0.017)	(0.0026)	(0.04)	(0.002)	(7E-7)	(0.0004)
Time (sec.)	KKM-v	1232	1200	1575	1150	1253	1343
	KKM-c	3763	1854	3326	1750	1253	3476
	CATS	127	110	141	66	76	75
		(56)	(47)	(48)	(8.76)	(4.8)	(0.3)
NMI		0.88	0.91	0.86	0.99	0.99	0.91
		(0.046)	(0.019)	(0.06)	(0.008)	(0.0001)	(0.0003)

Table 1: Results for three MNIST and UCI initializations. Numbers in parentheses are standard deviations. The NMI measures how close our clusterings are to the ones produced by the converged kernel k-means.

We also provide the change of distortion of an instance of each data set in Figure 1 and Figure 2. In each figure, the red line denotes $D_C(\mathbf{m_C})$ over the clustering C computed by the kernel k-means. The blue line indicates $D_C(\tilde{\mathbf{m}}_\mathbf{C})$, and the green line indicates $D_C(\mathbf{m_C})$ over the clustering C computed by CATS. The green and red lines need to be compared to get a fair comparison of clustering quality, as they reflect the true performances of CATS and kernel k-means, respectively. Notice that while $D_C(\tilde{\mathbf{m}}_\mathbf{C})$ (blue) is estimated to be somewhat high, the actual distortion (green) closely follows that of kernel k-means (red). This effect is more pronounced in the UCI instance, where the true performance of our algorithm is almost equivalent to kernel k-means. We present more similar results in appendix B.

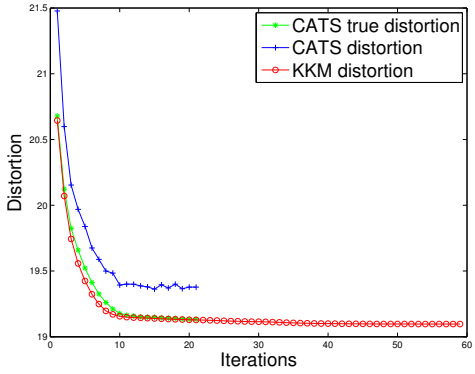

Figure 1: Distortion change of an MNIST instance over iterations.

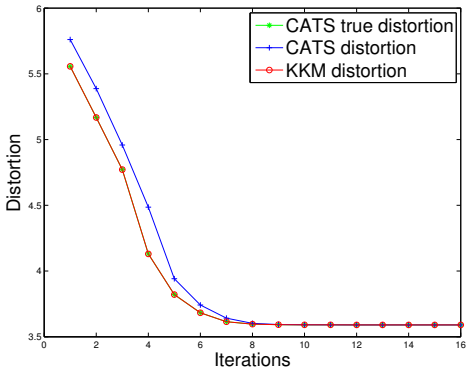

Figure 2: Distortion change of a UCI instance over iterations.

References

[1] D. Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66, 2003.

[2] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. Smola, and H.-P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21:47–56, 2005.

[3] A. Daniel, D. Amit, H. Pierre, and P. Preyas. NP-hardness of euclidean sum-of-squares clustering. *Machine Learning*, 75:245–248, 2009.

[4] S. Dasgupta and A. Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Structures & Algorithms*, 22, 2002.

[5] I. Dhillon, Y. Guan, and B. Kulis. Kernel k-means, spectral clustering and normalized cuts. In *Proceedings of KDD*, 2004.

[6] A. Frank and A. Asuncion. UCI machine learning repository. `http://archive.ics.uci.edu/ml`, 2010.

[7] K. Grauman and T. Darrell. Pyramid match hashing: Sub-linear time indexing over partial correspondences. In *Proceedings of CVPR*, 2007.

[8] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *Proceedings of ICCV*, 2009.

[9] Y. LeCun and C. Cortes. MNIST databse. `http://yann.lecun.com/exdb/mnist/`, 1998.

[10] J. Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society*, 209:441–458, 1909.

[11] J. Suzuki, Y. Sasaki, and E. Maeda. Kernels for structured natural language data. In *Proceedings of NIPS*, 2003.

[12] Y. Y. Yao. Information-theoretic measures for knowledge discovery and data mining. *Entropy Measures, Maximum Entropy and Emerging Applications*, pages 115–136, 2003.

[13] J. Zhang, M. Marszalek, S. Lazebnik, and C. Schmid. Local features and kernels for classification of texture and object categories: A comprehensive study. *International Journal on Computer Vision*, 73, 2007.

[14] R. Zhang and A. Rudnicky. A large scale clustering scheme for kernel k-means. In *Proceedings of ICPR*, 2002.

Appendix A: Proof of Theorem 1

for all possible S, the distortion with setting each $\alpha_i = 1/l,\ i \in S$ upper-bounds D_S since $\sum_{i \in S} \phi(\mathbf{x_i})/l \in \mathrm{Span}(S)$. We define the *sample average distortion* $\bar{D}_S$ as:

$$\bar{D}_S = \sum_{i \in C} \left\{ \kappa(i,i) - \frac{2}{l} \sum_{j \in S} \kappa(i,j) + \frac{1}{l^2} \sum_{j \in S, t \in S} \kappa(j,t) \right\}.$$

As mentioned, $\forall S D_S \leq \bar{D}_S$, and hence $E[D_S] \leq E[\bar{D}_S]$. It follows that $E[D_S] - D_C \leq E[\bar{D}_S] - D_C$. We prove the following useful Lemma.

Lemma 1. *Let $m = |C|$ and $l = |S|$. Then, if S is chosen uniformly at random without replacement,*

$$E\left[\bar{D}_S\right] = \left(1 + \frac{m-l}{l\,(m-1)}\right) D_C.$$

Proof. The distortion of a specific sampled instance S_t is as follows.

$$D_{S_t} = \sum_{i \in C} \kappa\,(i,i) - 2 \sum_{i \in C, j \in S_t} \alpha_j \kappa\,(i,j) + m \sum_{i,j \in S_t} \alpha_i \alpha_j \kappa\,(i,j)\,, \quad S_t \subseteq C$$

Now, we want to compute $E[D_S] = \sum_{S_t \subseteq C, |S_t|=l} D_{S_t} P(S_t)$, where P is a probability distribution over all possible subsets of C. For our random sampling scheme, $P(S_t)$ is $1/\binom{m}{l}$ for all S_t. Hence

$$E[D_S] = \frac{1}{\binom{m}{l}} \sum_{S_t \subseteq C, |S_t|=l} \left(\sum_{i \in C} \kappa(i,i) - 2 \sum_{i \in C, j \in S_t} \alpha_j \kappa(i,j) + m \sum_{i,j \in S_t} \alpha_i \alpha_j \kappa(i,j) \right). \quad (5)$$

For the second term of Eqn (5) that is linear in α_i's for each $(i,j) \in C \times C$, we encounter $\binom{m-1}{l-1}$ $k(i,j)$'s considering all possible $\binom{m}{l}$ cases for S_t. For the last quadratic term for each $(i,j) \in C \times C$, there are $\binom{m-2}{l-2}$ $\kappa(i,j)$, $i \neq j$ instances and $\binom{m-1}{l-1}$ instances of $\kappa(i,i)$'s. Hence, for $\alpha_i = 1/l$,

$$\begin{aligned} E[\bar{D}_S] &= \frac{1}{\binom{m}{l}} \sum_{S_t \subseteq C, |S_t|=l} \left(\sum_{i \in C} \kappa(i,i) - \frac{2}{l} \sum_{i \in C, j \in S_t} \kappa(i,j) + \frac{m}{l^2} \sum_{i,j \in S_t} \kappa(i,j) \right) \\ &= Tr(K) - 2 \frac{1}{l\binom{n}{l}} \sum_{i,j \in C} \binom{m-1}{l-1} \kappa(i,j) \\ &+ m \frac{1}{\binom{m}{l}} \left(\frac{1}{l^2} \sum_{i,j \in C} \binom{m-2}{l-2} \kappa(i,j) + \frac{1}{l^2} \sum_{j \in C} \binom{m-2}{l-1} \kappa(j,j) \right), \end{aligned}$$

where we used the equality $\frac{1}{l^2} \sum_{j \in C} \binom{m-1}{l-1} \kappa(j,j) = \frac{1}{l^2} \sum_{j \in C} \left(\binom{m-2}{l-2} + \binom{m-2}{l-1} \right) \kappa(j,j)$ in the last line. The above expression simplifies to:

$$E[\bar{D}_S] = Tr(K) \left(1 + \frac{m-l}{l(m-1)} \right) + \sum_{i,j \in C} \kappa(i,j) \left(-\frac{2}{m} + \frac{l-1}{l(m-1)} \right)$$

To facilitate the derivation, we use the following expanded form of D_C:

$$\begin{aligned} D_C &= \sum_{i \in C} \|\phi(\mathbf{x_i}) - \frac{1}{m} \sum_{j \in C} \phi(\mathbf{x_j})\|_2^2 \\ &= \sum_{i \in C} \left\{ \kappa(i,i) - \frac{2}{m} \sum_{j \in C} \kappa(i,j) + \frac{1}{m^2} \sum_{j,r \in C} \kappa(j,r) \right\} \\ &= Tr(K) - \frac{1}{m} \sum_{i,j \in C} \kappa(i,j). \end{aligned}$$

Finally, we subtract D_C from $E[\bar{D}_S]$ to obtain the desired result.

$$\begin{aligned} E[\bar{D}_S] - D_C &= \frac{m-l}{l(m-1)} \left(Tr(K) - \frac{1}{m} \sum_{i,j \in C} \kappa(i,j) \right) \\ &= \frac{m-l}{l(m-1)} D_C. \end{aligned}$$

□

The statement of Theorem 1 follows by combining Lemma 1 and the time complexity of our algorithm. The conclusion is valid even if we allow sampling S with replacement, by similar computation.

Appendix B: Extra Experimental Results

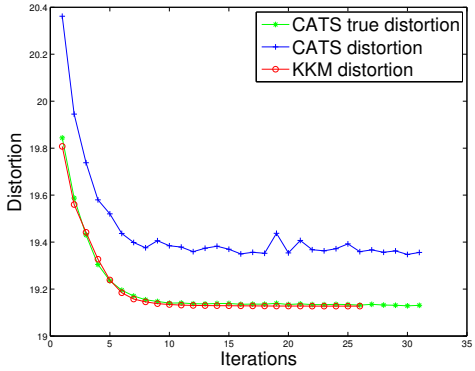

Figure 3: Distortion change of an MNIST instance.

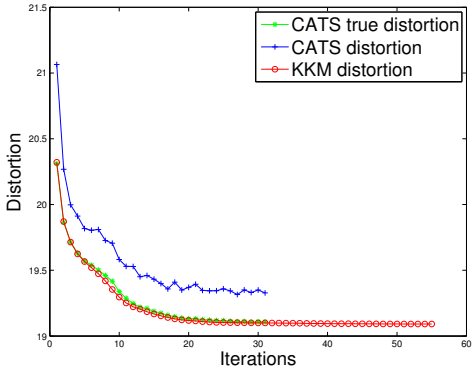

Figure 4: Distortion change of an MNIST instance.

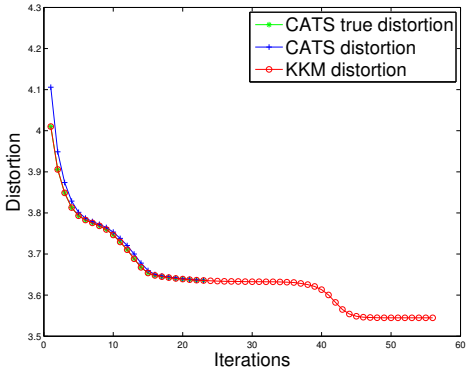

Figure 5: Distortion change of a UCI instance.

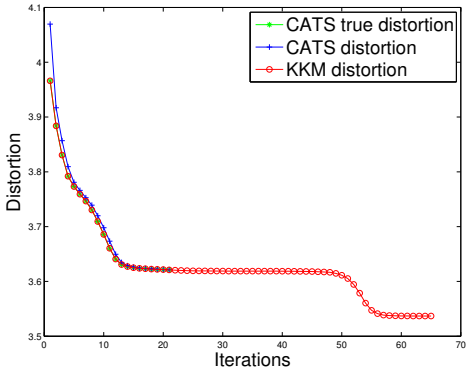

Figure 6: Distortion change of a UCI instance.