A Large Scale Clustering Scheme for Kernel K-Means

Rong Zhang and Alexander I. Rudnicky
School of Computer Science, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, USA
{rongz,air@cs.cmu.edu}

Abstract

Kernel functions can be viewed as a non-linear transformation that increases the separability of the input data by mapping them to a new high dimensional space. The incorporation of kernel function enables the K-Means algorithm to explore the inherent data pattern in the new space. However, the recent applications of kernel K-Means algorithm are confined to small corpora due to its expensive computation and storage cost. To overcome these obstacles, we propose a new clustering scheme which changes the clustering order from the sequence of samples to the sequence of kernels, and employs a disk-based strategy to control data. The new clustering scheme has been demonstrated to be very efficient for large corpus by our experiments on hand-written digits recognition, in which more than 90% of the running time was saved.

1. Introduction

A number of kernel-based learning methods have been proposed in recent years [1,2,3]. All of these methods employ kernel function to increase the separability of data. Generally speaking, kernel function implicitly defines a non-linear transformation that maps the data from their original space to a high dimensional space where the data are expected to be more separable. Consequently, the kernel methods may achieve better performance by working in the new space.

K-Means is an unsupervised learning algorithm that partitions the data set into a selected number of clusters under some optimization measures. For example, we often want to minimize the sum of squares of the Euclidean distance between the samples and the centroids. The assumption behind this measure is the belief that the data space consists of isolated elliptical regions. However, such assumption isn't always held on specific applications. To tackle this problem, one idea is to investigate other measures, e.g., the cosine similarity used in information retrieval. An alternate idea is to map the data to new space that satisfies the requirement of the optimization measure. In this case the kernel function is a good choice.

Usually the extension from K-Means to kernel K-Means is simply realised by expressing the distance in the form of kernel function [4,5]. However, such implementation suffers serious problems, such as the high clustering cost due to the repeated calculations of kernel values, or insufficient memory to store the kernel matrix, that make it unsuitable for large corpora. We propose an efficient large-scale clustering scheme to break through this limitation. Different from the previous algorithm in which the clustering is implemented on the sequence of samples, our scheme changes it to the sequence of kernels. This enables us to use a disk-based data management strategy that theoretically extends the storage space to the entire disk, and minimizes the number of I/O operations as well.

2. Kernel Function

Sometimes it isn't sufficient for a given learning machine to work in the input space because the assumption behind the machine doesn't match the real pattern of the data. For example, SVM and Perceptron require the data are linearly separable, while K-Means with Euclidean distance expects the data distribute into elliptical regions. When the assumption isn't held, we may apply some kind of transformation to the data, mapping them to a new space where the learning machine can be used. Kernel function provides us a means to define the transformation.

Suppose we are given a set of samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$, where $\mathbf{x}_i \in R^D$, and a mapping function ϕ that maps $\mathbf{x}_i$ from the input space R^D to a new space Q. The kernel function is defined as the dot product in the new space Q:

$$H(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) \tag{2.1}$$

An important fact about kernel function is that it is constructed without knowing the concrete form of ϕ [6]. Namely, the transformation is defined implicitly. Three commonly used kernel functions are listed below:

Polynomial $$H(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^d \tag{2.2}$$

Radial $$H(\mathbf{x}_i, \mathbf{x}_j) = \exp(-r\|\mathbf{x}_i - \mathbf{x}_j\|^2) \tag{2.3}$$

Neural $$H(\mathbf{x}_i, \mathbf{x}_j) = tanh(a\mathbf{x}_i \cdot \mathbf{x}_j + b) \tag{2.4}$$

The main weaknesses of kernel function include: First, some properties of the new space are lost, e.g. its dimensionality and value range, due to the lack of the explicit form for ϕ. Second, the determination of the appropriate kernel form for a given data set has to be realised through experiments. In addition, the computation and storage cost are increased by a wide margin.

3. From K-Means to Kernel K-Means

Suppose the data set has N samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$. K-Means algorithm aims to partition the N samples into K clusters, $C_1, C_2, \ldots, C_K$, and then returns the centre of each cluster, $\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_K$, as the representatives of the data set. Thus a N-point data set is compressed to a K-point "code book". The batch mode K-Means clustering algorithm using Euclidean distance works as follows:

1. Select K initial centres: $\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_K$.
2. Assign each sample $\mathbf{x}_i$ $(1 \le i \le N)$ to the closest centre, forming K clusters. Namely, compute the value of indicator function $\delta(\mathbf{x}_i, C_k)$ $(1 \le k \le K)$.

$$\delta(\mathbf{x}_i, C_k) = \begin{cases} 1 & D(\mathbf{x}_i, \mathbf{m}_k) < D(\mathbf{x}_i, \mathbf{m}_j) \text{ for all } j \ne k \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

3. Compute the new centre $\mathbf{m}_k$ for each cluster C_k

$$\mathbf{m}_k = \frac{1}{|C_k|} \sum_{i=1}^{N} \delta(\mathbf{x}_i, C_k)\mathbf{x}_i \tag{3.2}$$

where $|C_k|$ is the number of samples in C_k.

$$|C_k| = \sum_{i=1}^{N} \delta(\mathbf{x}_i, C_k) \tag{3.3}$$

4. Repeat step 2 and 3 until converge.
5. Return $\mathbf{m}_k$ $(1 \le k \le K)$.

In (3.1), $D(\mathbf{x}_i, \mathbf{m}_k)$ is the Euclidean distance satisfying:

$$D^2(\mathbf{x}_i, \mathbf{m}_k) = \|\mathbf{x}_i - \mathbf{m}_k\|^2 \tag{3.4}$$

The key issue extending traditional K-Means to kernel K-Means is the computation of distance in the new space. Let $\mathbf{u}_i = \phi(\mathbf{x}_i)$ denoting $\mathbf{x}_i$'s transformation. The Euclidean distance between $\mathbf{u}_i$ and $\mathbf{u}_j$ is written as:

$$\begin{aligned} D^2(\mathbf{u}_i, \mathbf{u}_j) &= \|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\|^2 \\ &= \phi^2(\mathbf{x}_i) - 2\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j) + \phi^2(\mathbf{x}_j) \\ &= H(\mathbf{x}_i, \mathbf{x}_i) - 2H(\mathbf{x}_i, \mathbf{x}_j) + H(\mathbf{x}_j, \mathbf{x}_j) \end{aligned} \tag{3.5}$$

Let $\mathbf{z}_k$ be the cluster centre in transformed space that

$$\mathbf{z}_k = \frac{1}{|C_k|} \sum_{i=1}^{N} \delta(\mathbf{u}_i, C_k)\mathbf{u}_i \tag{3.6}$$

where $\delta(\mathbf{u}_i, C_k)$ is the indicator function. The distance between $\mathbf{u}_i$ and $\mathbf{z}_k$ is expressed as

$$\begin{aligned} D^2(\mathbf{u}_i, \mathbf{z}_k) &= \left\| \mathbf{u}_i - \frac{1}{|C_k|} \sum_{j=1}^{N} \delta(\mathbf{u}_j, C_k)\mathbf{u}_j \right\|^2 \\ &= H(\mathbf{x}_i, \mathbf{x}_i) + f(\mathbf{x}_i, C_k) + g(C_k) \end{aligned} \tag{3.7}$$

where

$$f(\mathbf{x}_i, C_k) = -\frac{2}{|C_k|} \sum_{j=1}^{N} \delta(\mathbf{u}_j, C_k) H(\mathbf{x}_i, \mathbf{x}_j) \tag{3.8}$$

$$g(C_k) = \frac{1}{|C_k|^2} \sum_{j=1}^{N} \sum_{l=1}^{N} \delta(\mathbf{u}_j, C_k)\delta(\mathbf{u}_l, C_k) H(\mathbf{x}_j, \mathbf{x}_l) \tag{3.9}$$

By applying (3.7) to the traditional K-Means, we obtain the kernel-based K-Means algorithm:

1. Assign $\delta(\mathbf{x}_i, C_k)$ $(1 \le i \le N, 1 \le k \le K)$ with initial value, forming K initial clusters $C_1, C_2, \ldots, C_K$.
2. For each cluster C_k, compute $|C_k|$ and $g(C_k)$.
3. For each training sample $\mathbf{x}_i$ and cluster C_k, compute $f(\mathbf{x}_i, C_k)$. And then assign $\mathbf{x}_i$ to the closest cluster:

$$\delta(\mathbf{x}_i, C_k) = \begin{cases} 1 & f(\mathbf{x}_i, C_k) + g(C_k) < f(\mathbf{x}_i, C_j) + g(C_j) \\ & \text{for all } j \ne k \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

4. Repeat step 2 and 3 until converge.
5. For each cluster C_k, select the sample that is closest to the centre as the representative of C_k.

$$\mathbf{m}_k = \underset{\mathbf{x}_i \text{ that } \delta(\mathbf{x}_i, C_k)=1}{\arg\min} D(\phi(\mathbf{x}_i), \mathbf{z}_k) \tag{3.11}$$

Note in (3.10), the factor $H(\mathbf{x}_i, \mathbf{x}_i)$ is ignored because it doesn't contribute to determine the closest cluster. The main difference between kernel K-Means and its traditional version exists in step 5. Since the cluster centre in the transformed space can't be expressed explicitly, we have to choose a pseudo centre instead.

4. Large Scale Clustering Scheme

A careful analysis will find the kernel K-Means algorithm described in last section has serious problem on large corpus. Note in step 2 and 3, the kernel $H(\mathbf{x}_i, \mathbf{x}_j)$ repeatedly occur in the calculations of $g(C_k)$ and $f(\mathbf{x}_i, C_k)$. It would be an expensive cost to compute $H(\mathbf{x}_i, \mathbf{x}_j)$ again and again, especially when the data set is large or the dimensionality is high. Therefore, we should separate the computation of kernels from the clustering. That is, compute the kernel matrix $\mathbf{H}$ that contains all of the kernels before Step 1:

$$\mathbf{H} = \begin{bmatrix} h_{1,1} & h_{1,2} & \ldots & h_{1,N} \\ h_{2,1} & h_{2,2} & \ldots & h_{2,N} \\ \ldots & \ldots & \ldots & \ldots \\ h_{N,1} & h_{N,2} & \ldots & h_{N,N} \end{bmatrix} \tag{4.1}$$

where $h_{i,j}$ denotes $H(\mathbf{x}_i, \mathbf{x}_j)$.

Nevertheless, this will cause another problem: the kernel matrix $\mathbf{H}$ may be too large to be stored in memory. Say the data set has 30,000 samples. The number of kernel values that need be determined is about 450,000,000 (given the symmetry of $\mathbf{H}$) which would exceed the memory capability of a common PC. Moreover, the training corpus for practical application is usually much larger.

Some solutions have been proposed to speed up the clustering on large corpus [7,8]. All of these methods are based on the knowledge of the concrete form of sample and some properties of the data set. For example, some techniques need to know the dimensionality and value range of the data that are unavailable for kernel K-Means.

We solve the large scale clustering in a different fashion. First we change the clustering order from the sequence of sample to the sequence of kernel that enables us to take an efficient way handling the kernel matrix $\mathbf{H}$. Moreover, we use the disk space to make up the insufficiency of memory. Therefore, the size of $\mathbf{H}$ can be theoretically extended to as large as the entire disk. We split $\mathbf{H}$ into blocks which size is determined according to the I/O capability and the affordable memory. For example, assuming $\mathbf{H}$ has N^2 kernels and the memory can store S^2 of them, $\mathbf{H}$ is then split into $\frac{1}{2}N^2/S^2$ blocks. These blocks are moved into memory successively and processed there. Note the block is moved as a whole, so the number of I/O operations to move the entire $\mathbf{H}$ is equivalent to the block number which is the minimum value we can have.

The new clustering scheme is described in the C-style pseudo-code.

```
1. Compute kernel matrix $\mathbf{H}$ and store every necessary block $\mathbf{B}$ to the disk.
2. Assign $\delta(\mathbf{x}_i, C_k)$ with initial value.
3. For each cluster $C_k$, compute $|C_k|$ and let $g(C_k)=0$. For each training sample $\mathbf{x}_i$ and cluster $C_k$, let $f(\mathbf{x}_i, C_k)=0$.
4. Read the next block $\mathbf{B}$ from disk into memory.
5. For every kernel $h_{u,v}$ $(1 \le u, v \le N)$ in $\mathbf{B}$:
    • Check the clusters that $\mathbf{x}_u$ and $\mathbf{x}_v$ belong to:
        $\theta_u = k_1$ if $\delta(\mathbf{x}_u, C_{k_1}) = 1$    (4.2)
        $\theta_v = k_2$ if $\delta(\mathbf{x}_v, C_{k_2}) = 1$    (4.3)
      where $\theta_u$ and $\theta_v$ are the variables denoting the clusters.
    • If $u < v$, ignore this kernel. (Only the kernel below or on the diagonal of $\mathbf{H}$ need be processed given the symmetry of $\mathbf{H}$.)
    • If $u = v$ ($h_{u,v}$ is on the diagonal of $\mathbf{H}$, so $\mathbf{x}_u = \mathbf{x}_v$ and $\theta_u = \theta_v$) then
        $f(\mathbf{x}_u, \theta_v) = f(\mathbf{x}_u, \theta_v) - 2 * h_{u,v} / |C_{\theta_v}|$    (4.4)
        $g(\theta_u) = g(\theta_u) + h_{u,v} \big/ |C_{\theta_u}|^2$    (4.5)
    • If $u > v$ then
        $f(\mathbf{x}_u, \theta_v) = f(\mathbf{x}_u, \theta_v) - 2 * h_{u,v} / |C_{\theta_v}|$    (4.6)
        $f(\mathbf{x}_v, \theta_u) = f(\mathbf{x}_v, \theta_u) - 2 * h_{u,v} / |C_{\theta_u}|$    (4.7)
      And if $\theta_u = \theta_v$ then
        $g(\theta_u) = g(\theta_u) + 2 * h_{u,v} \big/ |C_{\theta_u}|^2$    (4.8)
6. Repeat step 4 and 5 until every block has been processed.
7. For each training sample $\mathbf{x}_i$ and cluster $C_k$, compute
    $\delta(\mathbf{x}_i, C_k) = \begin{cases} 1 & f(\mathbf{x}_i, C_k) + g(C_k) < f(\mathbf{x}_i, C_j) + g(C_j) \\ & \text{for all } j \ne k \\ 0 & \text{otherwise} \end{cases}$    (4.9)
8. Repeat step 3 to 7 until converge.
```

This clustering scheme assumes that all of the variables $f(\mathbf{x}_i, C_k)$ can be maintained in memory. If this assumption doesn't hold in the case that the corpora are too large, one can also split them into blocks as we did for the kernel matrix. A very good point of this scheme is that, the computations in the step 1, 4, 5 and 6 are only dependent on the size of $\mathbf{H}$. When we double the cluster number, the clustering time won't increase too much.

5. Experiments

5.1. K-Means Classifier

In order to compare the performance of kernel K-Means with traditional K-Means, we choose the pattern recognition problem for our experiments because the classification error rate can give us a straightforward metrics evaluating the performance.

The K-Means based classifier works as follows. In the training phase, each class is built a K-point "code book" using K-Means algorithm. In the test phase, the distance between a new sample $\mathbf{x}$ and class Γ_γ is computed as:

$$D(\mathbf{x}, \Gamma_\gamma) = \frac{1}{L} \sum_{l=1}^{L} D(\mathbf{x}, \mathbf{m}_l^\gamma) \tag{5.1}$$

where $\mathbf{m}_l^\gamma$ is $\mathbf{x}$'s l-th nearest neighborhood in the "code book", and L is the smoothing factor that $1 \le L \le K$. The new sample $\mathbf{x}$ is assigned to class γ^* if

$$\gamma^* = \underset{\gamma}{arg\,min}\, D(\mathbf{x}, \Gamma_\gamma) \tag{5.2}$$

The K-Means classifier is similar to the K-Nearest Neighbourhood except that it uses the small size "code book" for generalization instead of the entire data set.

5.2. Data Set

Our experiments use the MNIST as the training and test set [9]. MNIST is a 10-class database designed for handwritten digit recognition. The training set has 60000 samples in which the size of each class varies from 5421 (character '5') to 6742 (character '1'). The test set has 10000 samples in which each class equally has 1000 samples. Every sample consists of 28 by 28 pixels which value is inside the interval [0,255]. In our experiments, the value was normalised to [0,1].

We further divided the training set into two parts, the pure training set and the cross-validation set, with the ratio of 2:1. The form of kernel function and its parameters are determined on the validation set. Through a series of preliminary experiments, we found the neural kernel function (see formula 2.4) with the parameter value $a = 0.0045$ and $b = 0.11$ works well on MNIST. We used them for the following experiments.

5.3. Experimental Results

Our first experiment was to compare the performance of kernel K-Means with traditional K-Means as classifier. Table 5.1 presents their misclassifications on MNIST.

In this experiment the number of clusters *K* is set to 512 that is also determined by preliminary experiments on cross-validation set. The parameter *L* is the smoothing factor in (5.1). The result shows that the kernel K-Means works consistently better than its traditional version. When *L*=3, the best case for both, kernel K-Means achieves 10.8% reduction on the misclassification. This supports the observation that a good kernel function can make data more separable by mapping them to a new space.

L	Misclassification	
	K-Means	Kernel K-Means
1	497	483
3	471	420
5	476	433
8	511	471
16	630	560
32	784	695
64	1003	889
128	1401	1206

Table 5.1 Performance of K-Means and kernel K-Means

The second experiment was to demonstrate the efficiency of the new clustering scheme. We ran traditional K-Means and the new clustering scheme on the 40,000-sample pure training set, and compare the time they used. Both algorithms were compelled to finish 20 iterations. The block size in the new scheme was set to 4000 by 4000 which resulted in 55 blocks. The experiment was run on a P-3 PC. Table 5.2 presents the running time in minutes.

Table 5.2 shows most of the time was spent on the computation of kernels. If we use the standard kernel K-Means to do clustering, in which the kernel matrix is repeatedly computed in every iteration, the total running time would be more than 184*20 = 3680 minutes. So our new scheme saved at least 90% of the running time.

The time spent on the I/O operations also took a large portion, but it is the minimised value one can achieve. As we analysed before, the change to the number of cluster doesn't affect the new scheme too much. The time for the traditional K-Means was doubled when *K* is changed from 512 to 1024, while it increased only about 11.7% for the new scheme.

Operations	*K*=512		*K*=1024	
	K-Means	Scheme	K-Means	Scheme
Compute Kernel	-	184	-	184
I/O	-	73	-	73
Clustering	112	76	223	115
Total	112	333	223	372

Table 5.2 Running Time of K-Means and New Scheme

6. Summary

We provide a comprehensive description for kernel K-Means algorithm, as well as a new large-scale clustering scheme. All of the previous solutions to speed up the training on large corpus are dependent on some properties that are unavailable for kernel method. Our scheme solves the problem by clustering on the kernels instead of the samples, and using disk-based data management strategy. The effectiveness and efficiency of the new scheme are demonstrated by our experiments on hand-written digits recognition.

This research was sponsored in part by the Space and Naval Warfare Systems Center, San Diego, under Grant No. N66001-99-1-8905. The content of the information in this publication does not necessarily reflect the position or the policy of the US Government, and no official endorsement should be inferred.

References

[1] Christopher Burges. "A Tutorial on Support Vector Machines for Pattern Recognition". Data Mining and Knowledge Discovery, 2(2), 1998.

[2] Thilo-Thomas Friess, Nello Cristianini, and Colin Campbell. "The Kernel-Adatron Algorithm: a Fast and Simple Learning Procedure for Support Vector Machines". Proc. of the 5th Intl. Conference on Machine Learning, 1998.

[3] J.A.K Suykens and J. Vandewalle. "Least Squares Support Vector Machine Classifiers". Neural Processing Letters, Vol. 9, No. 3, 1999.

[4] Klaus-Robert Muller, Sebastian Mika, et al. "An Introduction to Kernel-Based Learning Algorithms". IEEE Trans. On Neural Networks, vol. 12, No. 2, March, 2001.

[5] Mark Girolami. "Mercer Kernel Based Clustering in Feature Space". To Appear in IEEE Trans. On Neural Networks.

[6] Nello Cristianini and John Shawe-Taylor. "Support Vector Machines and Other Kernel Based Learning Methods". Cambridge University Press, 2000.

[7] Cutting Douglas R., Jan O. Pedersen, et al. "Scatter/Gather: A Cluster Based Approach to Browsing Large Document Collections". In SIGIR'92, pp. 318-329.

[8] Dan Pelleg and Andrew Moore. "Accelerating Exact K-Means Algorithm with Geometric Reasoning". Conference on Knowledge Discovery in Databases 1999, (KDD99).

[9] MNIST is available on http://www.research.att.com/~yann/exdb/mnist/index.html.