Making k-means faster

Madhur Gupta

May 2, 2015

Abstract

We have a look at two algorithms to accelerate k-means clustering using *cached sufficient statistics* data structures: **kd-trees** and **metric trees** for low dimension and high dimension data respectively.

1 Naive k-means algorithm

K-means algorithm partitions the data into k-subsets such that points belonging to the same subset belong to the same cluster(center). Denoting number of data points by R, number of dimensions by M and number of centers by k, centroids after the i-th iteration by $C^{(i)}$ naive k-means algorithm can be summarized as

Initialization: Choose k points randomly to be the initial centers.
Repeat

1. For each point x, find the center in $C^{(i)}$which is closest to x. Associate x with this center.
2. Compute $C^{(i+l)}$ by taking, for each center, the center of mass of points associated with this center.

During each iteration of the algorithm, nearest center query is done k-times for each of the R times in M dimensions, time taken in one iteration is $O(kMR)$

2 Low Dimension Data

2.1 Multi Resolution kd Trees [1]

We define a new specialization of kd-trees: **multi resolution kd trees**. Multi-resolution kd trees are binary kd trees where each node contains information about all points contained in a hyper-rectangle h associated with that node. The hyper- rectangle is stored at the node as two M-length boundary vectors h^{max} and h^{min}. Further each internal node has a "split dimension" and a "split value" similar to a kd tree. Actual points are contained in the leaf nodes.
We further define the distance between two points $d(x, y)$ as Euclidean distance between them. For a point x and hyperplane h, $d(x, h)$ is defined as the distance between x and $closest(x, h)$ where $closest(x, h)$ is x if $x \in h$, otherwise $closest(x, h)$ is on the boundary of h. This boundary point can be found by clipping each coordinate of x, to lie within h

2.2 Algorithm

Definition. Given a hyper-rectangle h, and two centers c^1 and c^2 such that $d(c^l, h) < d(c^2, h)$, we say that c^1 dominates c^2 with respect to h if every point in h is closer to c^1 than it is to c^2 .

Lemma. Given two centers c^l, c^2, and a hyper- rectangle h such that $d(c', h) < d(c^2, h)$, the decision problem "does c^1 dominate c^2 with respect to h?" can be answered in O(M) time.
Proof Consider the vector $\vec{v} = c^2 - c^1$ and look at the extreme point in h in direction of $\vec{v}$. If this is closer to c^1 than to c^2, then any point in h will be closer to c^1 than to c^2 and c^1 will dominate h(else it will not). This extreme point can be found out by maximising the inner product $\langle v, p \rangle$ such that $p \in h$ which can be done in O(M) time.
We now present a recursive algorithm to update the $C^(i)$ in each iteration.

Initialization: h is a hyper-rectangle containing all points. C^0 is a set of k randomly chosen centroids.
Recurse: **Update**(h, C)

1. If h is a leaf: For each data point in h, find the closest center to it and update that center's counters. Return.

2. Compute $d(c, h)$ for all centers c. If there exists one center c with shortest distance:
   If for all other centers c', c dominates c' with respect to h:
   Update $c's$ counters using the data in h. Return.

3. Call Update(h_l, C) Update(h_r, C) on child nodes h_l and h_r

The algorithm tries to prune the tree(using Step 2) as much as possible so that number of distance comparisons are reduced for the pruned sub-tree. If we need to go down till leaf nodes, then this algorithm is slower than the naive k-means algorithm due to the computational overhead.

Blacklisting Algorithm: Making algorithm faster If in any iteration we find two centers c^1 and c^2 such that c^1 dominates c^2 with respect to h, then we can eliminate c^2 from possible set of centroids to check in $h's$ children also, as points in any subset of h will always be nearer to c^1 than to c^2.

3 High Dimension Data

3.1 Metric Trees [2]

In order to deal efficiently with high dimension data, we introduce a new data type **metric trees** where the only assumption they make about the metric is the satisfaction of *triangle inequality*. Further each node has two fields n_{pivot} and n_{radius} such that all the points represented by the node are within n_{radius} distance of the n_{pivot} element of the node. Like usual kd-trees, union of elements of mutually exclusive child nodes make up elements of parent node.

3.2 Anchors Hierarchy

The usual top-down way of structuring the metric tree would involved finding the n_{pivot}(f_1 and f_2) for children in the following way: f_1 will be the datapoint in n farthest from n_{pivot} and f_2 will be datapoint farthest from f_1. However, this is a slow method and we will show a new way of structuring metric tree which is similar to concept of clustering.
Algorithm: Maintain a list of anchors(each anchor a^i having a a^i_{pivot} and associated list of points with each anchor such that the all points in the list are nearest to corresponding anchor's a^i_{pivot} than any other anchors pivot element. We can say that the anchor *owns* the list of points. Each anchor i has a radius which the distance of a^i_{pivot} to the farthest point in its list. Further, the list of associated points is kept in decreasing order of distance to pivot element for reducing computations later.
At each iteration, we try to add a new anchor a^{new}. The pivot element a^{new}_{pivot} is farthest element from pivot element in the anchor with largest radius. Now, with this pivot element, we try to steal points from existing anchors' list by comparing distance of each point from a^{new}_{pivot} and existing anchor's pivot.
Middle Out Building: For R points, we initially find $\sqrt{R}$ anchors. Trying to keep the radius of nodes minimum, we merge similar nodes(try to minimise radius of parent node at each step) and build the tree upwards on these $\sqrt{R}$ nodes. Later, we recursively call Anchor hierarchy function these $\sqrt{R}$ nodes.

3.3 Algorithm

We present algorithm for efficient k-means clustering where each pass of the algorithm recurses over the nodes of the metric tree. At each iteration we maintain the invariant that $Cands$ is a subset of C which contain the possible owners(nearest centroids) of points in n

Initialization: node n - root node, C = set of centroids, $Cands$ - full set of centroids
Repeat:$KMeansStep(noden, CentroidSetC, CentroidSetCands)$

1. Reduce $Cands$: We consider the point c^* which is closed to n_{pivot} and then try pruning the set of $Cands$ by comparing distance $d(x, n_{pivot})$ to $d(c^*, n_{pivot})$ for all elements x in $Cands$

2. Update statistic of centroids in pruned $Cands$:

   - In case there is only one point left in $Cands$ then we know that this is nearest centroid to all points in n. Hence its weight is updated according to all points in n
   - In n is leaf node, then we iterate through all points and update the weights of the nearest possible centroid to the point.
   - call $KMeansStep$ on each of child nodes.

The algorithm works better based on the empirical observation that the set $Cands$ will keep reducing with each iteration. In this case, the number of distance operation calls will be less compared to naive k-means algorithm.

4 Results

The first algorithm gives multiple fold speed-up to run k-means on low dimension data by traversing each node(with extra statistics) of the pruned tree. It however gives bad results on high dimension(> 8) data.
The second algorithm gave similar speed-ups on low dimension data. On high dimension data, the speed up turned out to be dependent on the structure of data: if there is intrinsic structure in the data then we get good speed-ups, however in case there is no structure to the data i.e. data is uniformly distributed then we are not able to prune the tree leading to no acceleration in algorithm.

References

[1] Dan Pelleg, Andrew Moore. 1999. Accelerating Exact k-means Algorithms with Geometric Reasoning

[2] Andrew W. Moore . 2000. The Anchors Hierarchy: Using the 'triangle Inequality to Survive High Dimensional Data