Technical report 1

Glizela Taino, Yixin Lin

June 3, 2016

Von Mises distribution

Intuitively, the Von Mises distribution[4] is a simple approximation for the normal distribution on a circle (known as the *wrapped normal distribution*). The Von Mises probability density function is defined by the following equation:

$$P(x) = \frac{e^{b\cos(x-a)}}{2\pi I_0(b)}$$

where $I_0(x)$ is the modified Bessel function of the first kind; the Von Mises cumulative density function has no closed form.

The mean $\mu = a$ (intuitively, the angle that the distribution clustered around), and the circular variance $\sigma^2 = 1 - \frac{I_1(b)}{I_0(b)}$ (intuitively, b is the "concentration" parameter). Therefore, as $b \to 0$, the distribution becomes uniform; as $b \to \infty$, the distribution becomes normal with $\sigma^2 = 1/b$.

Von Mises-Fisher distribution

THe Von-Mises Fisher distribution is the generalization of the Von Mises distribution to n-dimensional hyperspheres. It reduces to the Von-Mises distribution with $n = 2$. The probability density function is defined by the following equation:

$$f_p(\boldsymbol{x};\boldsymbol{\mu},\kappa) = C_p(\kappa)\exp(\kappa\boldsymbol{\mu}^T\boldsymbol{x})$$

where

$$C_p(\kappa) = \frac{\kappa^{p/2-1}}{(2\pi)^p I_{p/2-1}(\kappa)}$$

and intuitively is an approximation for the normal distribution on the hypersphere.

K-means clustering

k-means clustering is a simple and popular algorithm for the *clustering problem*, the task of grouping a set of observations so that a group is "similar" within itself and "dissimilar" to other groups. k-means partitions n observations into k clusters, with each observation belonging to the nearest mean of the cluster. This problem is NP-hard in general, but there are heuristics which guarantee convergence to a local optimum.

The standard heuristic (known as *Lloyd's algorithm*) is the following:

Algorithm 1 Lloyd's algorithm for k-means clustering

```
generate an initial set of $k$ means
while not converged do
    assign all data points to nearest Euclidean-distance mean
    calculate new means to as the centroids of the observations in the cluster
end while
```

There is a choice of initialization method. The *Forgy method* randomly picks k observations as initial means, while the *Random Partition method* randomly picks a cluster for each observation.

The Lloyd's algorithm is a heuristic, so it does not guarantee a global optimum. Furthermore, there exists sets of points in which it converges in exponential time. However, it has been shown to have a smoothed polynomial running time, and in practice converges quickly.

Spherical k-means clustering

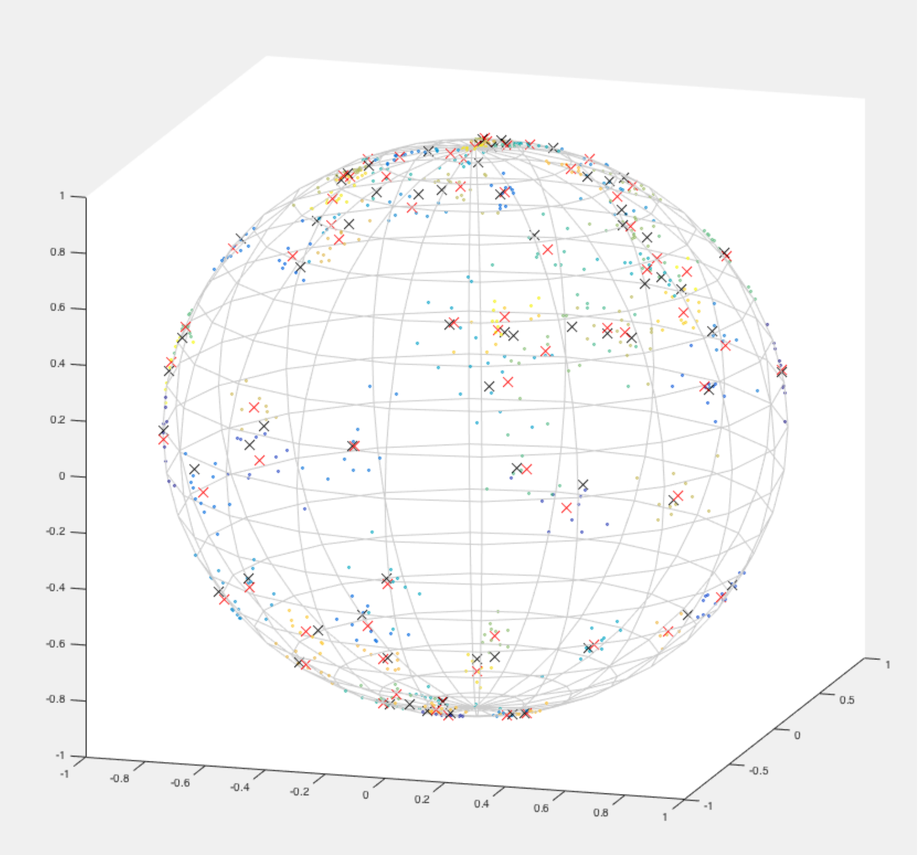
Spherical k-means clustering the same idea, but with points on a sphere. We investigated a MATLAB implementation by Nguyen[3, 2], which required a mean-and-norm-normalized dataset located on a hypersphere. Important aspects of this implementation include:

- When there exists an empty cluster, the largest cluster is split
- Use the dot product as "negative distance", which leverages the fact that observations are unit vectors on the hypersphere
- Use the normalized sum of observations as a centroid/mean, which leverages the fact that observations are unit vectors on the hypersphere. Note that this fails on pathological cases where the sum of observations is zero.

Our investigation

In order to test how well the spherical k-means clustering algorithm worked, we constructed a random data set using the Von Mises distribution random

Figure 1: Plot of 70 clusters and means on a 3-D sphere

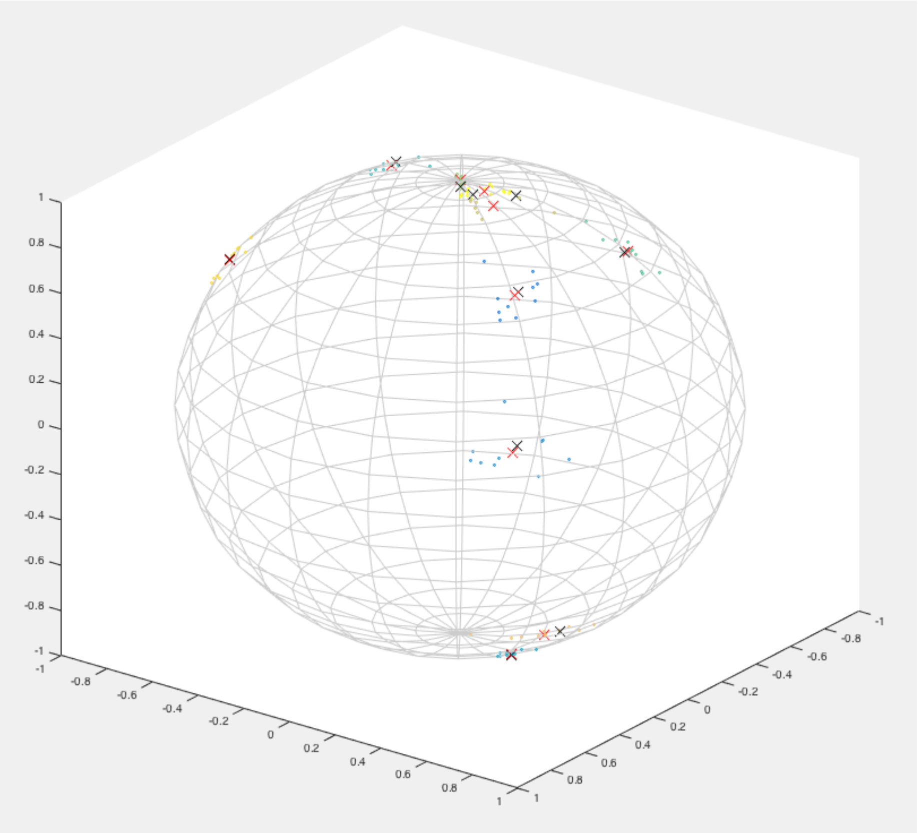


sampling function from the MATLAB Circular Statistics Toolbox[1]. We constructed 70 clusters of 10 points each, using random points on the sphere as means and with $\kappa = 100$, and then applied the spherical k-means clustering and visualized the results on the unit three dimensional sphere, color coded for each cluster and with estimated means labeled as red X's and true means labeled as black X's.

Karcher Mean

The Karcher mean is a geometric mean of various matrices and an extension of the well known geometric mean of two matrices. It is also referred to as the Riemannian geometric mean. In our investigation, we used it as a way to compute the average distance between two points on a hypersphere while obeying the curvature of the manifold. The process is stated in Algorithm 1. In

Figure 2: Plot of 10 clusters and means on a 3-D sphere



order to do this we must incorporate a Logarithm map and Exponential map on the manifold.

The Logarithm map is defined as a function that takes a point, ρ_2, on the manifold and maps its projection vector, γ, on the tangent plane at an origin point, T_{p_1}.

$$\rho_2 = Exp_{\rho_1}(\gamma) = cos(|\gamma|)\rho_1 + sin(|\gamma|)\frac{\gamma}{|\gamma|} \tag{1}$$

On the first iteration, ρ_1 is the origin point of the tangent space.

The Exponential map can be thought of as the inverse function of the Logarithm map. It is a function that takes in a vector on the tangent plane at origin point and maps it on the hypersphere. As the difference between the updating vectors on the tangent space begin to show little change, it is thought as the optimal mean between the two points on the manifold.

$$\gamma = Log_{\rho_1}(\rho_2) = \tilde{\rho}\frac{cos^{-1}(\langle\rho_1, \rho_2\rangle)}{\sqrt{\langle\tilde{\rho}, \tilde{\rho}\rangle}} \tag{2}$$

where $\tilde{\rho} = \rho_2 - \langle\rho_2, \rho_1\rangle\rho_1$. Karcher mean, as we have mentioned before, is calculated in a way to obey the curvature of the hypershere.

Spherical mean

Unlike the Karcher mean, the spherical mean does not go along the curvature of the hypersphere. In the assignment step of this algorithm, where we adjust the cluster centers, we find the mean by summing up all the vectors in a cluster and normalizing it back onto the manifold.

$$x_l = \frac{\sum_{i \in X_l} \rho_i}{||\sum_{i \in X_l} \rho_i||}, l = 1 \ldots K$$

where X_h are the center of each K number of clusters. This is the normalized gravity center of the new cluster.

Our Implementation

To find the best approach in calculating a cluster center, we put Karcher mean and Spherical K-mean to the test. We tested them in five types of situations in where the angles are either identical, orthogonal, opposite, two random angles, or multiple random angles as shown in Figure 3.

We found that Karcher mean and Spherical K-means produced the same results as long as the step size for Karcher mean was set to one. But both approaches failed in the case where the angles were opposite.

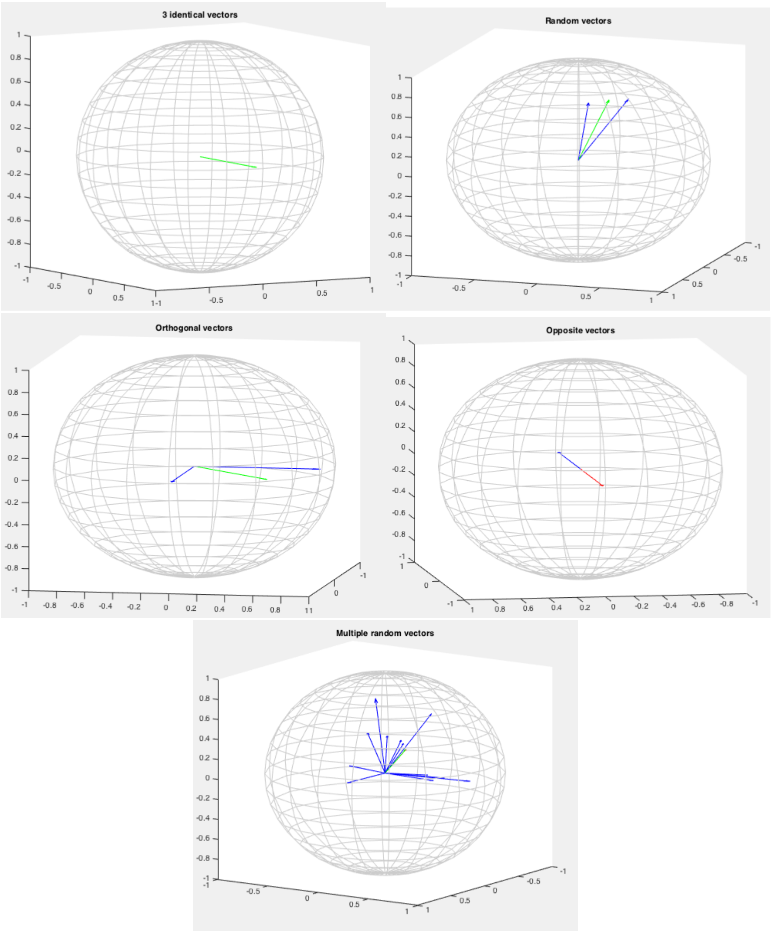


Figure 3: Different test cases comparing accuracy of Karcher mean and Spherical K-means.

Hierarchical clustering

For our lab, we want to develop the most efficient way to group data based based on a measure of dissimilarity. We explored the topic of hierarchical clustering which is used in data mining and statistics. It is visualized useing a dendogram to show the organization and relationship among clusters. Hierarchical clustering can be implemented in two ways: divisive and agglomerative. We will elaborate on these two variants in this paper to compare the its algorithmic complexity, or how much resources are needed when computed, and the different types of cluster linkage criterion. Further information on cluster analysis can be found in [**?**].

Divisive and Agglomerative

Divise clustering is also known as a "top down" method. This takes in a data set as one large cluster. Then as it moves down the hierarchy, it recusively splits until it reaches the leaves of the tree, or each observation is its own singleton cluster.

Agglomerative clustering can be thought to do the opposite of divisive clustering and is known as a "bottom up" strategy. It takes in a data set and it initially looks at each observation as its own singleton cluster. Then, based on its inkage criterion, it will merge recursively to form new clusters until all the data are in one large cluster. The dendrogram is used in both cases to record and visiualize each cluster merge and their distance to each other.

Cluster Dissimilarity Measurements

In forming clusters, a measurement must be made to determine which clusters split or merge in regards to divisive or agglomerative clustering, respectively. There are many different ways to measure dissimilarity but this paper will explore single linkage, complete linkage, and centroid linkage. Each of the measurements below refer to ways in measuring distances between k number of clusters $c_{i=1...k}$ in relationship with specific points $x_{i=1...k}$ in those clusters.

Single Linkage

Single linkage takes the distance between the closest observations, or minimum distance, in each cluster and merges the those with the shortest distance. This works well for data sets that desire to cluster together long chained data points. The equation is defined as

$$D(c_1, c_2) = \underset{x_1 \in c_1, x_2 \in c_2}{min} D(x_1, x_2)$$

Complete Linkage

Complete linkage is similar to single linkage. It takes the distance between the farthest observations, or the maximum distance, in each cluster. The clusters that have the shortest measured distance is merged together. This measurement is ideal for data that is sphereically organized relative to each other. The equation is defined as

$$D(c_1, c_2) = \underset{x_1 \in c_1, x_2 \in c_2}{max} D(x_1, x_2)$$

Centroid Linkage

Using centroid linkage involves taking the average of all points in a cluster and assigning that value as the mean cenroid. Then we find the centroids with the shortest distances and merge them together. Centroid linkage is defined as

$$D(c_1, c_2) = D((\frac{1}{|c_1|}\sum_{x \in c_1} \vec{x}), (\frac{1}{|c_2|}\sum_{x \in c_2} \vec{x}))$$

Our Implementation

We developed both hierarchical divisive and agglomerrative clustering functions to test which would organize data the fastest. In our particular application, instead of flat clustering, we had to cluster over the curved surface of a sphere. In order to do this, we utilized the SPKmeans function [**?**]. We also chose our linkage criterion to be based on centroid linkage.

Essentially, we developed the divisive clustering by following the intuition mentioned above. For our agglomerative clustering implementation, we did a slightly different variation hoping its run time would be slightly better than the traditional method. In our code, intuitively, we initiated each point as its own cluster with its own centroid. Then we recursively merged two clusters together based on centroid linkage to form a new cluster with its own cluster centroid everytime. This process stops when there is a single point to represent the entire data's average centroid.

Complexity determines how well the algorithm performs in real time as the problem size increases. We tested with a relatively small, three dimensional data set of 150 points. This way, we can later expand it to work with large data sets that are multidemnsional. Since the SPKmeans is the function that takes the longest to perform, it is our leading function to determine complexity. In the situtations of divisive and agglomerative, we input different cluster values into SPKmeans which determines how often the function gets called. In the divisive method, we asked SPKmeans to return two clusters everytime it is given a data set. This means the number of times we call SPKmeans is based on how many nodes we have which is $m = 2n - 1$ where n is the number of data points. For agglomertive, we asked SPKmeans to return the $n/2$ clusters. This means means we had to call SPKmeans at every level which is $l = ceiling(log_2 n)$.

Algorithm complexity analysis

Let n be the number of data points, k be the number of clusters (number of means), d be the dimensionality of the data, and i be the number of iterations until convergence. The runtime of the k-means algorithm (including spherical k-means) is $O(nkdi)$ (TODO: CITE).

Let us assume that the cost of initializing the means and splitting empty clusters is negligible, i.e. $O(1)$. In practice, the two are related: we find that explicitly picking good starting means for the agglomerate algorithm (e.g. randomly, or by maximizing distance between the means) makes the probability of empty clusters small, whereas it becomes excessive and unnecessary for the divisive algorithm. TODO: We explain this in a later section.

We will also assume that d and i are constants. Though in degenerate data sets i can be an exponential function of n, in practice i is a small constant on datasets with clustering structuring (CITE: WIKI?).

Let us also assume that we choose a branching factor of 2, i.e. a binary hierarchical tree structure. The following analysis does not depend on the branching factor, assuming it is constant.

Theorem. ***Agglomerative complexity***

$$T_{aggl}(n) = \Theta(n^2 di)$$

Proof. Our agglomerate algorithm has a recurrence relation of the following form:

$$T_{aggl}(n) = T(\frac{n}{2}) + nkdi$$

for some constant c, since we divide the problem size into two for every recursive call. But we choose $k = \frac{n}{2}$ each time, since the number of means we choose to cluster with is also half of n.

We use the master theorem (TODO: CITE) to solve this recurrence relation:

Master theorem variable	**Value**
a	1
b	2
$f(n)$	$\frac{n^2 di}{2}$
c	2
$log_b(a)$	0

This satisfies case 3 of the master theorem:

$$f(n) \in \Omega(n^c) \text{ s.t. } c > log_b(a)$$

since $f(n) \in \Omega(n^2)$ s.t. $c = 2 > log_b(a) = 0$, and

$$af(\frac{n}{b}) \leq k_0 f(n) \text{ for some } k_0 < 1 \text{ and sufficiently large } n$$

since when $k_0 = \frac{1}{4}$, $\left\{f(\frac{n}{2}) = (\frac{1}{4})\frac{n^2 di}{2}\right\} \leq \left\{k_0 f(n) = (\frac{1}{4})\frac{n^2 di}{2}\right\}$
and so, by the master theorem: $T_{aggl}(n) = \Theta(n^2 di)$. $\square$

Theorem. ***Divisive complexity***

$$T_{div}(n) = \Theta(n \log(n) di)$$

Proof. Our divisive algorithm has a recurrence relation of the following form:

$$T_{div}(n) = 2T(\frac{n}{2}) + nkdi$$

for some constant c, since we divide the problem size into two for every recursive call and we have two subproblems each time. But we choose $k = 2$ each time, since the number of means we choose to cluster with is always the branching factor.

We use the master theorem (TODO: CITE) to solve this recurrence relation:

Master theorem variable	**Value**
a	2
b	2
$f(n)$	$n(2)di$
c	1
$log_b(a)$	1

This satisfies case 2 of the master theorem:

$$f(n) \in \Theta(n^c \log_0^k n) \text{ s.t. } c = log_b(a)$$

by letting $k_0 = 0$, since $f(n \log^0(n)) = f(n) \in \Theta(n)$ s.t. $c = 1 = log_b(a) = 1$. and so, by the master theorem: $T_{div}(n) = \Theta(n \log(n)di)$. $\square$

References

[1] Phillip Berens. Matlab central file exchange: Circular statistics toolbox. *http://www.mathworks.com/matlabcentral/fileexchange/10676-circular-statistics-toolbox–directional-statistics-*. Accessed: 2016-05-18.

[2] Vinh Nguyen. Matlab central file exchange: The spherical k-means algorithm. *http://www.mathworks.com/matlabcentral/fileexchange/32987-the-spherical-k-means-algorithm*. Accessed: 2016-05-18.

[3] Vinh Nguyen. Gene clustering on the unit hypersphere with the spherical k-means algorithm: coping with extremely large number of local optima. In *World Congress in Computer Science, Computer Engineering, and Applied Computing (Hamid R. Arabnia and Youngsong Mun 14 July 2008 to 17 July 2008)*, pages 226–233. CSREA Press, 2008.

[4] Eric W. Weisstein. von mises distribution. *http://mathworld.wolfram.com/vonMisesDistribution.html*. Accessed: 2016-05-18.