Low-Rank k-means Clustering

***Abstract*—This report discusses an approach to k-means clustering based on applying low-rank factorization to a semi-definite program. This approach works by first writing the k-means clustering problem as a rank-constraint SDP, from which we use low-rank factorization to derive a relaxed, nonlinear, nonconvex problem. The derived problem is then solved heuristically using the method of augmented Lagrangian. A simulation of the algorithm is discussed at the end of this report.**

I. Introduction

A problem commonly encountered in the fields of statistics and machine learning is that of clustering. The statement of the problem is, give a set of n data points in $\mathbb{R}^p$, how can the data be classified into k clusters $C_1, C_2, \ldots, C_k$ such that the sum of the squared Euclidean distance between each point and its cluster is minimized. More precisely, we have the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - m_j\|^2 \\ \text{where} \quad & m_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i \end{aligned} \tag{1}$$

where the optimization variable is the choice of all possible C_j's.

In this paper, we present an approximate algorithm to the above problem as outlined by Kulis in [2]. The paper is organized as the follows: In Section II, we give the formulation of our problem. In Section III, we describe the our approximate algorithm. In Section IV, we present the result of our simulation based on the approximate algorithm.

II. Problem Formulation

As shown by Peng in [1], the optimization problem 1 can be written as an SDP by following the steps below:

Let $A \in \mathbb{R}^{n \times k}$ be the assignment matrix defined by $A_{ij} = 1$ if $x_i \in C_j$, and $A_{ij} = 0$ otherwise. Using this notation, we can write the centroids as $m_j = \frac{\sum_{i=1}^{n} a_{ij} x_i}{\sum_{i=1}^{n} a_{ij}}$. Then the original problem becomes

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} \sum_{j=1}^{k} a_{ij} \|x_i - \frac{\sum_{l=1}^{n} a_{lj} x_l}{\sum_{l=1}^{n} a_{lj}}\|^2 \\ \text{subject to} \quad & \sum_{i=1}^{n} a_{ij} \geq 1 \\ & \sum_{j=1}^{k} a_{ij} = 1 \\ & a_{ij} \in \{0, 1\} \end{aligned}$$

where the first constraint ensures that every cluster has at least one element, and the second constraint ensures that every data point is assigned to exactly one cluster.

Next, let $K \in \mathbb{R}^{n \times n}$ be the Gram matrix formed by $K = XX^T$, where row i of X is the vector X_i, and $Z = A(A^TA)^{-1}A^T$. By expanding the terms in the above problem, we have

$$\begin{aligned} \text{minimize} \quad & Tr(K(I - Z)) \\ \text{subject to} \quad & Z1 = 1, Tr(Z) = k \\ & Z \geq 0, Z = Z^T, Z^2 = Z \end{aligned} \tag{2}$$

where $\geq$ refers to componentwise inequality. Clearly all feasible Z is a semi-definite matrix. Problem of this form is coined $0, 1$-SDP by Peng.

Peng showed that problem 2 can be relaxed to a normal SDP and obtain an approximate solution to the problem in 1. We however will take another approach outlined by Kulis in [2], which is based on the observation that the constraint $Z^2 = Z$ can be replaced by the requirement $rank(Z) = k$.

Motivated by this, we can attempt to factor Z into $Z = YY^T$, where $Y \in \mathbb{R}^{n \times k}$. This immediately gives the quadratic program

$$\begin{aligned} \text{minimize} \quad & -Tr(KYY^T) \\ \text{subject to} \quad & Tr(Z) = k \\ & YY^T1 = 1 \\ & YY^T \geq 0 \end{aligned} \tag{3}$$

The inequality constraint $YY^T \geq 0$ however massively slows down our algorithm described in Section III, so we will make one final relaxation to replace it with $Y \geq 0$ to obtain

$$\begin{aligned} \text{minimize} \quad & -Tr(KYY^T) \\ \text{subject to} \quad & Tr(YY^T) = k \\ & YY^T1 = 1 \\ & Y \geq 0 \end{aligned} \tag{4}$$

This is the program that we will solve in Section III.

III. Algorithm

We will solve our optimization problem using a heuristic approach based on augmented Lagrangian as described by Burer in [3]. Even though our problem is not convex (and so there is no theoretical guarantee that we can find the global minimum), the result by Burer suggests that non-convexity is not a huge issue for problems of this type, and the algorithm by Burer can find solution close to the global optimum.

Burer's algorithm considers problems of the form

$$\begin{aligned} \text{minimize} \quad & Tr(KYY^T) \\ \text{subject to} \quad & Tr(Y^TA_iY) = b_i \end{aligned}$$

Suppose at stage s, the algorithm has output Y_s, λ_s, t_s, at the next stage, we form the Lagrangian

$$\begin{aligned} & L(Y, \lambda_s, t_s) = Tr(KYY^T) + \\ & \sum_i \lambda_{i,s}[Tr(Y^TA_iY) - bi] + \frac{t_s}{2} \sum_i [Tr(Y^TA_iY) - bi]^2 \end{aligned} \tag{5}$$

At the next stage, the algorithm consists of three steps

1) Primal Update:
   $Y_{s+1} = \text{argmin } L(Y, \lambda_s, t_s)$ where this optimization subproblem is solved using $BFGS$.
2) Dual Update:
   Let $u = \sum_i [Tr(Y^T A_i Y) - b_i]^2$ be a measure of infeasibility.
   If $u_s \leq au$, set
   $$\lambda_{i,s+1} = \lambda_{i,s+1} + t_s[Tr(Y^T A_i Y) - bi]$$
   $$u_{s+1} = u$$
   $$t_{s+1} = t_s$$
   Otherwise set
   $$Y_{s+1} = Y_s$$
   $$u_{s+1} = u_s$$
   $$t_{s+1} = bt_s$$
   where $0 < a < 1, b > 1$ are some parameters to be tuned based on the problem.
3) Termination:
   We terminate if algorithm if the augmented Lagrangian is sufficiently close to the objective function. That is, we terminate if
   $$\frac{abs(L_{s+1} - Tr(KY_{s+1}Y'_{s+1})}{\max(abs(L_{s+1}), 1)} \leq \epsilon$$
   where ϵ is our error tolerance.

Before we proceed further, we note that the algorithm proposed by Burer only applies to problems with equality constraints. To solve this issue, we rewrite the constraint $Y \geq 0$ as $-Y + S = 0$, where $S \geq 0$.

Now, the augmented Lagrangian is

$$\begin{aligned} L(Y, S, \lambda_1, \lambda_c, \nu, t) &= -Tr(Y^T KY) + \lambda_1[Tr(Y^T Y) - k] \\ &+ \sum_i \lambda_{2,i}[Tr(Y^T 1 e_i Y) - 1] + \sum_{i,j} \nu_{ij}[S_{i,j} - Y_(i,j)] \\ &+ \frac{t}{2}[(Tr(Y^T Y) - k)^2 + \sum_i (Tr(Y^T 1 e_i Y) - 1)^2 \\ &+ \sum_{i,j} (S_{i,j} - Y_{i,j})^2] \quad (6) \end{aligned}$$

For primal update, we first minimize over S to get

$$\begin{aligned} Y &= \text{argmin}_{Y,S} L(Y, S, \lambda_{1,s}, \lambda_{2,s}, \nu_s, t_s) \\ &= \text{argmin}_Y - Tr(Y^T KY) + \lambda_1[Tr(Y^T Y) - k] \\ &+ \sum_i \lambda_{2,i}[Tr(Y^T 1 e_i Y) - 1] + \frac{t}{2}[(Tr(Y^T Y) - k)^2 \\ &+ \sum_i (Tr(Y^T 1 e_i Y) - 1)^2] + \sum_{i,j} P_{t_s}(Y_{ij}, \nu_{ij}) \quad (7) \end{aligned}$$

where $P_{t_s}(Y_{ij}, \nu_{ij}) = \frac{1}{2t_s}[\max(0, \nu_{ij} - tY_{ij})^2 - \nu_{ij}^2]$

With the revised primal update rule in Equation 7, we can then apply Burer's algorithm to our problem.

However, because our algorithm is a relaxed algorithm, after it terminates, a rounding procedure is required to extract a solution to the original problem. Empirically, we find that after the rounding procedure, there tends to be a reduction in the rank of Y, leaving us with $k' \leq k$ clusters.

IV. Simulation Result

In this section, we will discuss our simulation in Matlab and compare our algorithm with the algorithm described by Peng.

The implementation of our algorithm does not deviate significantly from what is described in Section III. First, we hot initiate our primal variable Y by running Matlab's built in k-means++ and uniformly initiate our dual variables. The initial step size is set to $t = 1.5$. Next, we use the augmented Lagrangian given by Equation 7 and its gradient with Matlab's built in BFGS function to update the primal variable. To update our dual variables and infeasibility, we use values of $a = 0.25$ and $b = \sqrt{10}$. Finally, for script termination, we use $\epsilon = 10^{-5}$. Once we find our optimal Y, we extract the clusters by treating Y as an indicator matrix. So we assign x_i to cluster j if Y_{ij} is the maximum in row i.

Figure 1, Figure 2, and Figure 3 show the results of low-rank k-means, Peng's Algorithm, k-means++, respectively.

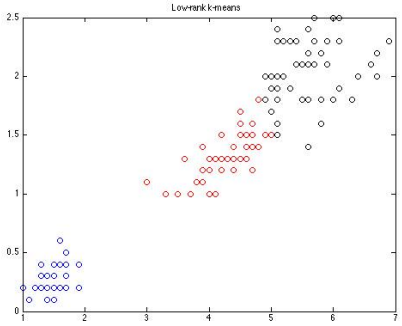

Fig. 1. A plot of the result of low-rank k-means on Matlab's fisheriris data. This is run with $k = 3$ clusters and has a total distance of $d = 55.3150$

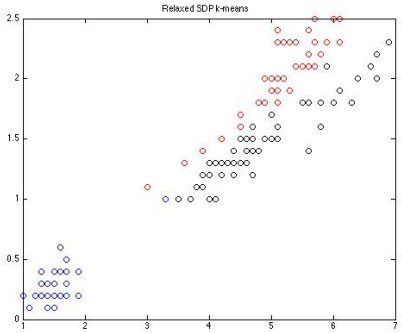

Fig. 2. A plot of the result of Peng's SDP k-means on Matlab's fisheriris data. This is run with $k = 3$ clusters and has a total distance of $d = 78.9800$

Even though low-rank k-means tend to perform better when k is low, it encounters stability issues once k gets large. There are two explanations for this. First, Matlab's line search function tends to fail and crash once the value of t gets large. This is possibly due to numerical instability from large numbers. Therefore, we could not set the tolerance of our low-rank k-means too low without crashing the program. There are two possible solutions to this- a different BFGS solver could work,

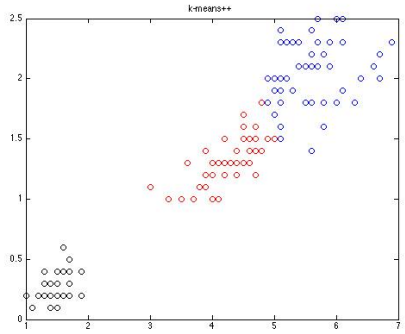

Fig. 3. A plot of the result of Matlab's built in k-means++ on Matlab's fisheriris data. This is run with $k = 3$ clusters and has a total distance of $d = 55.3357$

TABLE I
VARIOUS DISTANCE OF THE K-MEANS ALGORITHMS ON MATLAB'S FISHERIRIS DATA WITH VARIOUS NUMBER OF CLUSTERS. THE VALUES SHOWN ARE THE MINIMUM VALUE OBTAINED AFTER TEN SIMULATIONS

Num.	Low-rank	Spectral SDP	k-mean++
2	87.5880	87.5880	87.5880
3	55.3150	78.9800	55.3367
4	50.2968	79.4958	53.3481
5	42.0817	70.1657	43.8501

and change our Lagrangian to $\frac{1}{t\times\text{objective}} + \frac{1}{t\times\text{constraints}} + \frac{\text{penalty}}{2}$ instead might also alleviate this issue. Second, the rounding Y might lead to losses in rank, which reduces our number of clusters. Solving this issue requires a better rounding procedure, which is an interesting question to study.

Finally, even though low rank k-means perform decently well when k is small, it also tends to be much slower than the easy to implement and fast k-means++. The bottleneck of this algorithm is the time spent on evaluating $Y = \text{argmin}_Y L$. Again, a better BFGS algorithm might help, but evaluating the Lagrangian and its gradient requires computing $Tr(KYY^T)$ and KY, which needs around $O(zk)$ times, where z is the number of non-zero entries in K. So unless there is some sparsity pattern in K, scaling this algorithm to large problem might be difficult.

REFERENCES

[1] Peng, J. and Wei, J., *Approximating k-means-type clustering via semidefinite programming, SIAM J. Optim., 18 (2007), pp. 186205.*

[2] Kulis, B., Surendran, A. and Platt, J. (2007). Fast low-rank semidefinite programming for embedding and clustering. In Proceedings 11th international conference on AI and statistics (AISTATS).

[3] Burer, S. and Monteiro, R. *A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization, Springer-Verlag 2002*