Accelerated Clustering Through Locality-Sensitive Hashing

by

Shaunak Kishore

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2012

Author .....
Department of Electrical Engineering and Computer Science
May 25, 2012

Certified by .....
Jonathan A. Kelner
Assistant Professor
Thesis Supervisor

Accepted by .....
Dennis Freeman
Chairman, Department Committee on Graduate Theses

Accelerated Clustering Through Locality-Sensitive Hashing

by

Shaunak Kishore

Submitted to the Department of Electrical Engineering and Computer Science
on May 25, 2012, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

We obtain improved running times for two algorithms for clustering data: the expectation-maximization (EM) algorithm and Lloyd's algorithm. The EM algorithm is a heuristic for finding a mixture of k normal distributions in $\mathbb{R}^d$ that maximizes the probability of drawing n given data points. Lloyd's algorithm is a special case of this algorithm in which the covariance matrix of each normally-distributed component is required to be the identity.

We consider versions of these algorithms where the number of mixture components is inferred by assuming a Dirichlet process as a generative model. The separation probability of this process, α, is typically a small constant. We speed up each iteration of the EM algorithm from $O(nd^2k)$ to $O(ndk\log^3(k/\alpha))+nd^2)$ time and each iteration of Lloyd's algorithm from $O(ndk)$ to $O(nd(k/\alpha)^{0.39})$ time.

Thesis Supervisor: Jonathan A. Kelner
Title: Assistant Professor

1. Introduction

Clustering is one of the most fundamental problems in machine learning. Given a set of data points in $\mathbb{R}^d$, a clustering algorithm partitions the data into a natural set of groups, or clusters. The measure by which we judge this partition often depends on the application. Clustering can be used for data mining [4], character and face recognition [5], and pattern detection [3].

In this paper, we present accelerated versions of two widely-used clustering algorithms: the EM algorithm and Lloyd's algorithm. The EM algorithm models the data as a set of samples drawn from a mixture of normal distributions. The algorithm chooses an initial set of parameters for this mixture and iteratively updates these parameters, converging to a mixture that locally maximizes the likelihood of the data [2]. Lloyd's algorithm, also known as the k-means algorithm, is a special case of the EM algorithm where the covariance matrices of the normal distributions are all required to be the identity [9, 7].

These algorithms do not scale well for large data sizes. Many authors have suggested methods for improving their runtimes such as subsampling the data [10], assuming diagonal covariance matrices [10], and using kd-trees [7]. Unfortunately, these approaches come with drawbacks. Aggressively subsampling the data and assuming diagonal covariance matrices exacerbate the quality of the clustering produced. On the other hand, kd-trees do not offer substantial speed-ups for high-dimensional data.

We show how to speed the EM algorithm and Lloyd's algorithm using the locality-sensitive hash functions introduced in [6, 1]. After each iteration, our algorithms produce a mixture with the same expected log-likelihood as the original algorithms, so the quality of the clustering does not suffer. Our runtimes scale well with the dimension. In addition, our techniques can easily be combined with subsampling to obtain further improvements in runtime.

2. Locality-sensitive hashing

The basic idea of locality-sensitive hashing is to construct a family of hash functions for points in $\mathbb{R}^d$ such that the probability that two points collide is a decreasing function of the distance between them. Locality-sensitive hashing can be used to answer approximate nearest-neighbor queries in high-dimensional spaces in sublinear time [6]. In this regime, geometric data structures like kd-trees do not work well.

Our algorithms use the locality-sensitive hash functions based on p-stable distributions from [1]. We review a special case of these hash functions where $p = 2$:

Definition 1 (2-stable hash family). A hash function $h_{\vec{a},b} : \mathbb{R}^d \to \mathbb{Z}$ is chosen from a 2-stable hash family by sampling $\vec{a}$ from a unit normal distribution in $\mathbb{R}^d$ and sampling b from the uniform distribution over $[0, 1]$. The function maps the point $\vec{x}$ to

$$h_{\vec{a},b}(\vec{x}) = \lfloor \vec{a} \cdot \vec{x} + b \rfloor$$

Let x_1 and x_2 be two points in $\mathbb{R}^d$ and let r equal $||x_1 - x_2||$. Because $\vec{a}$ is normally distributed in $\mathbb{R}^d$, $\vec{a} \cdot (\vec{x_1} - \vec{x_2})$ is normally distributed in $\mathbb{R}$ with standard deviation equal to r. Therefore, the probability that $h_{\vec{a},b}$ maps x_1 and x_2 to the same value is solely a function of r. If we denote this collision probability by $cp(r)$, we can evaluate it as an integral:

$$\begin{aligned} cp(r) &= Pr\left[h_{\vec{a},b}(\vec{x_1}) = h_{\vec{a},b}(\vec{x_2}) \text{ given that } ||x_1 - x_2|| = r\right] \\ &= \int_0^1 \frac{1}{r\sqrt{\pi}} e^{-t^2/r^2}(1 - t)dt \\ &= \frac{r}{\sqrt{\pi}}\left(e^{-1/r^2} - 1\right) + \operatorname{erf}(1/r) \end{aligned}$$

Our analysis of this collision probability is substantially different from the original analysis in [1]. The authors of that paper prove that $cp(1 + \epsilon)$ is significantly less than $cp(1)$, which they use to obtain an algorithm for approximate nearest-neighbor search. Instead, we prove two lemmas about $cp(r)$:

Lemma 1. On the interval $[0, 1]$, the values of the function $-\frac{\partial}{\partial r} cp(r)$ are bounded between 0.3 and 0.6.

Lemma 2. Suppose that t is a real number in $[0, 1]$. On the interval $[0, 1]$, the values of the function

$$(1 - t)\frac{\log cp(0.5r)}{\log cp(0.5)} - \min\left(r^2 - t, 0\right)$$

are non-negative and strictly less than 0.39.

Lemma 1 can be viewed as a generalization of the original authors' analysis. It shows that the value of $cp(r)$ is sensitive to changes in r at all points in the interval $[0, 1]$, not just at $r = 1$. Lemma 2 bounds the multiplicative difference between $cp(r)$ and the probability density function of a normal distribution. We leave the proofs of these lemmas to the appendix.

3. Expectation-maximization

A mixture of k normal distributions in $\mathbb{R}^d$ is defined as a linear combination

$$p(x) = \sum_{s=1}^{k} p(x|s)p(s)$$

where each component's probability density function $p(x|s)$ is defined by a mean μ_s and a covariance matrix C_s as follows:

$$p(x|s) = \frac{1}{\sqrt{\pi^d |C_s|}} e^{-(x-\mu_s)^T C_s^{-1}(x-\mu_s)}$$

The posterior probability that a point x came from the sth component of the mixture is simply $p(x|s)p(s)p(x)^{-1}$.

Given n data points $x_1, x_2, \ldots x_n$ in $\mathbb{R}^d$, the expectation-maximization algorithm is a heuristic for choosing a k-component mixture defined by $(p(s), \mu_s, C_s)$ and an assignment of points to these components that maximizes the posterior probability of drawing the data.

The expectation-maximization algorithm is a heuristic for choosing parameters $(p(s), \mu_s, C_s)$ which maximize the likelihood of drawing n given data points $x_1, x_2, \ldots x_n$. The algorithm updates these parameters iteratively. Under some lenient assumptions, it is guaranteed to converge to a setting of the parameters that locally maximizes the likelihood [10].

In the version of the algorithm that we consider, the number of mixture components is not specified initially. Instead, we assume that the mixture is generated by a Dirichlet process with a given separation probability α, which is typically a small constant [8]. Our version of the algorithm also assigns each data point x_i to a single component s_i, not a probability distribution over components.

Each iteration of the algorithm thus consists of three steps:

1. Each point x_i is assigned to a random component s_i. The point is assigned to an existing component s with probability proportional to $(1 - \alpha)p(x_i|s)p(s)$, and to a new component with probability proportional to α.

2. For each component s, if S is the set of points assigned to it, then its probability $p(s)$, mean μ_s, and covariance C_s are set as follows:

$$p(s) = \frac{|S|}{n}$$
$$\mu_s = \frac{1}{|S|} \sum_{x \in S} x$$
$$C_s = \frac{1}{|S|} \sum_{x \in S} xx^T$$

3. The eigenvalues of each covariance matrix C_s are forced into a fixed range. (This step ensures that these matrices are well-conditioned, which is necessary for convergence.)

We briefly analyze the runtime of this algorithm. Sampling the component that each point x_i is assigned to requires evaluating k products of the form $(x_i - \mu_s)^T C_s (x_i - \mu_s)$, which takes $O(d^2 k)$ time. Thus, step 1 takes $O(nd^2 k)$ time, which dominates the $O(nd^2)$ time needed for the other steps.

3.1 Accelerating expectation-maximization

In this section, we show how to perform the sampling step of the above algorithm in $O(ndk \log^3(k/\alpha) + nd^2)$ time using locality-sensitive hashing.

Our algorithm assumes that the eigenvalues of each covariance matrix C_s is bounded between $\pi^{-1/2}$ and some value B. The lower bound implies that the normalization factor

$$N_s = \frac{1}{\sqrt{\pi^d |C_s|}}$$

of each component s is at most 1, which means that the units of $p(x|s)$ are the same as the units for α. We can embed each data point x_i and each center μ_s in $\mathbb{R}^{d+1}$ so that the last coordinate accounts for this normalization factor; this coordinate is 0 for the data points and $\log^{1/2} 1/N_s$ for the centers. For the rest of this section, we assume that we have performed this embedding.

For each mixture component s, we maintain a set of m locality-sensitive hash functions $h_1, h_2, \ldots h_m$. Each of these functions is parametrized by a vector $\vec{a}$ a scalar b, like a function drawn from a 2-stable hash family, but the distribution over the parameter $\vec{a}$ is different: the vector is drawn from the normal distribution over $\mathbb{R}^d$ with covariance matrix $\log(k/\alpha)C_s$.

Suppose that x is a data point. We claim that with this distribution over $\vec{a}$ and b, the probability that $h_{\vec{a},b}(x)$ equals $h_{\vec{a},b}(\mu_s)$ is monotonic in the conditional probability $p(x|s)$. In fact, if we define r as follows:

$$r = \sqrt{(x_i - \mu_s)^T C_s (x_i - \mu_s)}$$

then the collision probability is

$$Pr[h_{\vec{a},b}(x) = h_{\vec{a},b}(\mu_s)] = cp(r/\log^{1/2}(k/\alpha))$$

and the conditional probability is e^{-r^2}. Since both of these functions are strictly decreasing in r, the claim holds.

Let f be the fraction of the m hash functions on which x and μ_s collide. Note

that the expected value of f is $cp(r/\log^{1/2}(k/\alpha))$. Our algorithm uses the value of f to compute an estimate r' for r and an estimate p'_s for $p(x|s)$, as follows:

1. If $f \geq (1 - 0.5/\log(k/\alpha))cp(1)$, then we set

$$r' = \log^{1/2}(k/\alpha)cp^{Inv}(f) \text{ and } p'_s = e^{-(r')^2+1}$$

2. Otherwise, we set

$$r' = \log^{1/2}(k/\alpha) \text{ and } p'_s = e^{-(r')^2+1} = e\alpha/k$$

To assign x to a component in our algorithm, we first compute the estimates p'_s for every component s. Using these estimates for the conditionals, we assign x either to an existing component or a new component. Finally, if x is assigned to an existing component s, we compute $p(x|s)$ and reject the assignment with probability $\max(1 - p(x|s)/p'_s, 0)$. We continue to sample components s based on the estimates p'_s until we accept an assignment.

For each of the n data points, it takes $O(dkm)$ time to evaluate the m hash functions for each component. It takes $O(d^2)$ time to compute $p(x|s)$ for the component that x is assigned to, and an additional $O(d^2)$ time each time the sample is rejected.

To prove that our algorithm samples components with the correct probability, we need to prove that p'_s overestimates $p(x|s)$ with high probability. To obtain our stated runtimes, we also need to show that if m is $\Theta(\log^3(k/\alpha))$, then we only reject an expected constant number of assignments for each point. The main theorem of this section implies both of these facts.

Theorem 1. If we take m to be $\Theta(\log^3(k/\alpha))$, then

$$e\max(p(x|s), \alpha/k) \geq p'_s \geq p(x|s)$$

with high probability in k.

Proof. In the following analysis, we assume that all high-probability events occur. We split the proof into two cases:

1. $r \leq \log^{1/2}(k/\alpha)$. In this case, $E[f] \geq cp(1) \approx 0.49$, and Lemma 1 applies.

   By the Chernoff bound and our choice of m, f is within a $1 \pm 0.5/\log(k/\alpha)$ factor of $E[f]$ with high probability in k. In particular, this bound implies that f is at least $(1 - 0.5/\log(k/\alpha))cp(1)$, so r' equals $cp^{Inv}(f)$.

   The lemma implies that r' is correct to within a $1 \pm 1/\log(k/\alpha)$ factor, because the derivative of $cp(r)$ on $[0, 1]$ only varies by a factor of 2. Therefore, with high probability, $|(r')^2 - r^2|$ is at most 1, and p'_s satisfies

   $$ep(x|s) \geq p'_s \geq p(x|s)$$

2. $r > \log^{1/2}(k/\alpha)$. In this case, $E[f] < cp(1)$, and $p(x|s)$ is less than α/k.

   The Chernoff bound implies that f is less than $(1 + 0.5/\log(k/\alpha))cp(1)$ with high probability in k.

   If f is at least $(1 - 0.5/\log(k/\alpha))cp(1)$, then we compute our estimate p'_s for $p(x|s)$ as in Case 1, getting a value which is at most $e\alpha/k$. If f is less than this cutoff, we simply return the estimate $e\alpha/k$. In either event, p'_s satisfies

   $$e\alpha/k \geq p'_s \geq p(x|s)$$

In both cases, the desired bounds hold with high probability. □

Theorem 1 immediately shows that, with high probability, p'_s is an overestimate for $p(x|s)$, so the distribution that we sample from is correct. We now use it to show that the rejection probability is not too large.

Let S_{small} be the set of components s for which $p(x|s) < \alpha/k$ and let S_{large} be the set of the remaining components. We define

$$P_{small} = \sum_{s \in S_{small}} (1 - \alpha)p(x|s)p(s) \text{ and } P'_{small} = \sum_{s \in S_{small}} (1 - \alpha)p'_s p(s)$$

$$P_{large} = \sum_{s \in S_{large}} (1-\alpha)p(x|s)p(s) \text{ and } P'_{large} = \sum_{s \in S_{large}} (1-\alpha)p'_s p(s)$$

We write the rejection probability in terms of four expressions:

$$Pr[rejection] = 1 - \frac{P_{small} + P_{large} + \alpha}{P'_{small} + P'_{large} + \alpha}$$

By Theorem 1, we know that with high probability, P'_{large} is at most eP_{large}. Since there are only k components, the theorem also implies that with high probability, P'_{small} is at most $e\alpha$. Because P_{small} is non-negative, we have

$$Pr[rejection] \leq 1 - \frac{P_{large} + \alpha}{e\alpha + eP_{large} + \alpha} \leq e/(1+e)$$

so we only reject a constant number of assignments for each point in expectation.

4. Accelerating Lloyd's algorithm

In this section, we give an accelerated version of Lloyd's algorithm. Although our methods are still based on locality sensitive hashing, our analysis for Lloyd's algorithm is substantially different from the our analysis for expectation-maximization. Our improved runtimes for expectation-maximization come from dimensionality reduction, but our improved runtimes for Lloyd's algorithm come from only computing posterior probabilities for a fraction of the components.

In our modified version of Lloyd's, the probability that each point is assigned to a given cluster is correct, but these probabilities are not independent. This correlation may slow down convergence. However, the expected value of the log-likelihood of the final assignment is correct, and we expect that our algorithm will converge quickly on real data.

Lloyd's algorithm performs the same clustering task as the expectation-maximization algorithm, except that the covariance matrices of the normally-distributed components are all equal to the identity. Because of this simplification, the sampling step of Lloyd's algorithm only takes $O(ndk)$ time [7]. To evaluate the conditional proba-

bilities $p(x|s)$ for each data point x, we only need to compute the distance from x to the center μ_s of each component s.

As before, we assume a Dirichlet process with separation probability α as a generative model for the data. We show how to perform the sampling step of Lloyd's algorithm in $O(nd(k/\alpha)^{0.39})$ time.

Our modified Lloyd's algorithm requires several preprocessing steps at the beginning of each iteration:

1. We embed each data point x_i and each center μ_s in $\mathbb{R}^{d+1}$. The first d coordinates of x'_i equal the first d coordinates of x_i; the last coordinate is 0. The first d coordinates of μ'_s equal the first d coordinates of μ_s; the last coordinate is $\log^{1/2}((1-\alpha)p(s)/\sqrt{\pi^d})$.

2. For each i in $\{0, 1, \ldots \ln(1/\alpha)\}$, we choose m_i hash functions $g_1, g_2, \ldots g_{m_i}$. Each of these functions g_j is the concatenation of l_i locality-sensitive hash functions $h_{j1}, h_{j2}, \ldots h_{jl_i}$:

$$g_j(x) = (h_{j1}(x), h_{j2}(x), \ldots h_{jl(i)}(x))$$

   The hash functions h_{jk} are parametrized by a vector $\vec{a}$ and a scalar b like a 2-stable hash function, except that $\vec{a}$ is drawn from the normal distribution in $\mathbb{R}^{d+1}$ centered at the origin with covariance matrix $4\log(k/\alpha)I_{d+1}$.

3. For each i and each function g_j at level i, we build a hash table keyed by the hash function g_j. We insert each component s into each table at the cell indexed by $g_j(\mu'_s)$. In addition, for each cell of each table at level i, we add a dummy component s_{new} with probability $e^i\alpha$.

To assign a data point x to a component in our algorithm, we follow the following procedure:

1. We start at level 0.

2. Suppose the current level is i. We choose a random function g_j from the m_i functions at this level and hash the transformed point x' by that function.

3. If the list of components stored in that cell of g_j's hash table is non-empty, we select a random component s from it. Otherwise, we resample the function g_j and try this step again.

4. If this component is s_{new}, we return it. Otherwise, we compute an acceptance probability for this component s based only on the level i and the distance r between x' and μ'_s:

$$Pr[acceptance] = \frac{e^{-r^2+i}}{m_i cp(0.5r/\log^{1/2}(k/\alpha))^{l_i}}$$

5. We accept s based on this acceptance probability, returning it for certain if this probability is greater than 1. If we reject the sample, we resample g_j and return to step 3.

6. After sampling g_j $O(m_i \ln k)$ times at level i, we move to level $i+1$ and go back to step 2. After level $\ln(1/\alpha)$, we move back to level 0.

To analyze this algorithm, we first check that it samples components with the correct probability. Let r be the distance between the transformed data point x' and the transformed center μ'_s. We have

$$e^{-r^2} = \frac{1}{\sqrt{\pi^d}} e^{-||x-\mu_s||^2 - \log((1-\alpha)p(s))} = (1-\alpha)p(x|s)p(s)$$

Therefore, after the transformation, all factors of the posterior probability are included in the conditional probability of drawing x' from a unit normal distribution centered at μ'_s.

We want to show that the probability that we return component s is proportional to e^{-r^2}. By concatenating multiple 2-stable hash functions h_{jk} to get the hash function g_j, we amplify the effect of distance on collision probability, so that the probability that x' and μ'_s collide under g_j is

$$Pr[g_j(x') = g_j(\mu'_s)] = cp(0.5r/\log^{1/2}(k/\alpha))^{l_i}$$

so as long as our initial acceptance probabilities are less than 1, the probability of sampling each component is correct.

We claim that with high probability in k, all of our initial acceptance probabilities will be less than 1. Our proof of this fact relies on ignoring *negligible components.* We define a component s to be negligible if the distance between x' and μ'_s is greater than $\log^{1/2}(k/\alpha)$. Assume that we choose parameters m_i and l_i such that the following conditions hold for each data point x:

1. The acceptance probability for a non-negligible component μ'_s changes by only a constant factor between adjacent levels.

2. At level i, the acceptance probabilities for any two non-negligible components s_1 and s_2 differ by at most a factor of m_i.

3. An expected $O(1)$ components in each cell that x' hashes to are negligible.

If $i+1$ is the smallest level at which the acceptance probability for some component s becomes greater than 1, then at level i, the acceptance probability for s is constant. However, this means that all but $O(1)$ components in every cell that x' hashes to have an acceptance probability of $\Omega(1/m_i)$. Since we sample $m_i \ln k$ components before moving to the next level, with high probability in k, we never make it to level $i+1$.

In addition to showing that the probability of sampling each component is correct, this analysis also shows that we do not return to level 0.

Finally, we need to choose parameters m and l_i for which the above conditions hold. In addition, to obtain our runtime bounds, we need $l_i m_i$ to be $o((k/\alpha)^{0.39})$, since in the worst case, for each data point x_i, we evaluate $\tilde{O}(l_i m_i)$ hash functions at each of $O(\log \alpha)$ levels.

We will choose l_i such that the collision probability $cp(0.5r/\log^{1/2}(k/\alpha))^{l_i}$ is a good multiplicative approximation to the density function $\min(e^{-r^2+i}, 1)$ of a normal distribution boosted by the factor e^i. We will choose m_i to be the multiplicative difference between these two functions.

In particular, if we set l_i to be

$$l_i = \frac{\log(k/\alpha) - i}{-\log cp(0.5)}$$

then we have

$$\begin{aligned}\log\left(\frac{e^{-r^2+i}}{cp(0.5r/\log^{1/2}(k/\alpha))^{l_i}}\right) &= -r^2 + i + \left(\frac{\log(k/\alpha) - i}{cp(0.5)}\right) cp(0.5r/\log^{1/2}(k/\alpha)) \\ &= -\log(k/\alpha)\left((1-t)\frac{\log cp(0.5r')}{cp(0.5)} - (r')^2 + t\right)\end{aligned}$$

where r' is $r/\log^{1/2}(k/\alpha)$ and t is $i/\log(k/\alpha)$. Applying Lemma 2 to this expression, we find that if r is greater than $i^{1/2}$ and $\log^{1/2}(k/\alpha)$, then

$$(k/\alpha)^{-0.39} \leq \frac{e^{-r^2+i}}{cp(0.5r/\log^{1/2}(k/\alpha))^{l_i}} \leq 1$$

so if we take m_i to be $(k/\alpha)^{0.39}$, then the acceptance probabilities for any two non-negligible component are within a factor of m_i. (In fact, the lemma implies that setting m_i to be $(k/\alpha)^{0.35-\epsilon}$ is sufficient, for some positive ϵ.)

With these choices of l_i and m_i, the first condition holds trivially - since l_i changes by the acceptance probability of any non-negligible component changes by at most a factor of e between rounds. We have already ensured that the second condition holds with our choice of m_i. Finally, the third condition holds because the probability of a collision between x' and the center μ'_s of some negligible component s is at most

$$cp(0.5r/\log^{1/2}(k/\alpha))^{l_i} \leq cp(0.5)^{\frac{\log(k/\alpha)-i}{-\log cp(0.5)}} \leq e^{\ln\alpha - \log(k/\alpha)} = 1/k$$

There are only k components, so the expected number of negligible components that collide with x' is at most 1. This bound concludes the analysis of our modified version of Lloyd's algorithm.

Appendix A. Proofs of Lemma 1 and Lemma 2

In this section, we prove Lemma 1 and Lemma 2. These lemmas follow from straightforward computations. Recall that if the locality-sensitive hash function h is drawn from a 2-stable hash family, then the collision probability $cp(r)$ of two points at distance r is given by

$$cp(r) = \frac{r}{\sqrt{\pi}}\left(e^{-1/r^2} - 1\right) + \operatorname{erf}(1/r)$$

We use this expression for $cp(r)$ in the following proofs.

Lemma 1. On the interval $[0, 1]$, the values of the function $-\frac{\partial}{\partial r}cp(r)$ are bounded between 0.3 and 0.6.

Proof. We take the derivative of $cp(r)$ with respect to r:

$$-\frac{\partial}{\partial r}cp(r) = \frac{1}{\sqrt{\pi}}\left(1 - e^{-1/r^2}\right)$$

Note that this derivative is non-negative and decreasing on $[0, \infty]$. To prove the lemma, we need only evaluate $-\frac{\partial}{\partial r}cp(r)$ at 0 and 1, where its values are approximately 0.56 and 0.36, respectively. Since both of these values fall in the interval $[0.3, 0.6]$, the claim holds. □

Lemma 2. Suppose that t is a real number in $[0, 1]$. On the interval $[0, 1]$, the values of the function

$$(1-t)\frac{\log cp(0.5r)}{\log cp(0.5)} - \min\left(r^2 - t, 0\right)$$

are non-negative and strictly less than 0.39.

Proof. We first claim that $\log cp(r)$ is concave on the interval $[0, 0.5]$. By explicit computation, we find the second derivative of this function:

$$\frac{\partial^2}{\partial r^2}\log cp(r) = \frac{2\sqrt{\pi}e^{1/r^2}\operatorname{erf}(1/r) - \left(e^{1/r^2} - 1\right)r\left(\left(e^{1/r^2} - 1\right)r^2 + 2\right)}{r^3\left(\left(e^{1/r^2} - 1\right)r - \sqrt{\pi}e^{1/r^2}\operatorname{erf}(1/r)\right)^2}$$

It is easy to check that $r^3(e^{1/r^2}-1)$ is decreasing on $(0, 0.5]$, so this expression is at least $0.5^3(e^{1/0.5^2}-1) > 6.6$ on this interval. Also, on this interval, $e^{1/r^2}-1$ is at least $0.98e^{1/r^2}$.

Because $\mathrm{erf}(1/r)$ is always less than 1, the numerator of our expression for $\frac{\partial^2}{\partial r^2}\log cp(r)$ is bounded above by $(2\sqrt{\pi} - 0.98 \cdot 6.6)e^{1/r^2}$, which is negative. So $\log cp(r)$ is indeed concave on the interval $[0, 0.5]$.

Note that $cp(0.5)$ is negative, so the function $(1-t)\frac{\log cp(0.5r)}{\log cp(0.5)}$ is convex for r in $[0, 1]$. This function agrees with the line $(1-t)r$ when r is 0 or 1, so we have

$$(1-t)\frac{\log cp(0.5r)}{\log cp(0.5)} \leq (1-t)r$$

for all r in $[0, 1]$.

Let $f(r) = (1-t)r - \min(r^2 - t, 0)$. We now claim that $f(r)$ is non-negative and strictly less than 0.39 for r in $[0, 1]$. By our work above, this claim implies the lemma. Because $f(0) = f(1) = 0$, f is non-negative on $[0, 1]$ by convexity. We only need to prove the upper bound on f.

At the point where f achieves its maximum, f' must be 0 or undefined. If $t \leq 0.25$, then f' has a zero at $r = 0.5$; otherwise, f' has no zeroes. The value of $f(0.5)$ is $0.5(1-t) - 0.25 + t = 0.5t + 0.25$, which is at most 0.375 for these values of t.

The derivative f' is always undefined at $r = \sqrt{t}$. The value of $f(\sqrt{t})$ is $(1-t)\sqrt{t}$, which achieves its maximum value of $0.3849\ldots$ when $t = \frac{1}{3}$. For all t in $[0, 1]$, $f(r)$ is strictly less than 0.39, as claimed. □

Bibliography

[1] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing based on p-stable distributions. In the *20th ACM Symposium on Computational Geometry*, 1999

[2] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. In the *Journal of the Royal Statistical Society*, Series B (Methodological) 39(1):1-39, 1977.

[3] R. Duda and P. Hart *Pattern Classification and Scene Analysis.* New York: John Wiley & Sons, 1973.

[4] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining.* Published by the AAAI/MIT Press, 1996.

[5] Z. Ghahramani and G. Hinton. The EM Algorithm for mixtures of Factor Analyzers Listed as Technical Report CRG-TR-96-1.

[6] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In the *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, 1999

[7] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R .Silverman, and A. Wu. An efficient k-means clustering algorithm: analysis and implementation. In the *IEEE Transactions on Pattern Analysis and Machine Intelligence*, July 2002.

[8] K. Kurihara, M. Welling, and N. Vlassis. Accelerated variational Dirichlet process mixtures. In *NIPS*, Volume 19, 2006.

[9] S. Lloyd. Least Squares Quantization in PCM. In the *IEEE Transactions on Information Theory*, vol. 28, 129-137, 1982.

[10] J. Verbeek, J. Nunnink, and N. Vlassis Accelerated EM-based clustering of large data sets. In *Data Mining and Knowledge Discovery*, 3-2006, 291-307.