STAT 598G Spring 2011
Sergey Kirshner
Homework 4 Solution (Written Exercises)

April 25, 2011

This homework is due on **Thursday, April 28, at 10:30am**. Late homeworks will not be accepted. The homework can be either turned in at the beginning of class or submitted through BlackBoard Vista (under assignments). Please do not email your homeworks.

Please show all your work. Also, please acknowledge all of the help that you received while working on the assignment (according to the guidelines outlined in the syllabus).

- Reading: notes (optimization, conjugate gradients, k-means, mixtures of Gaussians, EM algorithm, linked lists), handouts (k-nearest neighbor)
- Exercises:

1. (2 pts) The complexity of k-means algorithm for n points with Euclidean distance in $\mathbb{R}^d$ with T iterations is $\mathcal{O}(kndT)$ (dominated by the evaluation of the distances between the points and the current cluster means). How can the k-means algorithm be sped up beyond $\Theta(knT)$ in the case of $d = 1$?

   Solution: Consider a single iteration of the k-means algorithm. Each iteration consists of two steps. In the first step, each of n points has its membership reestimated first by computing the distance between each point and each of k means $\Theta(knd)$, and then finding the closest mean to each point $\Theta(nk)$, totaling in $\Theta(knd)$ time. In the second step, the means are reestimated in $\Theta(nd + kd)$ time. As it is assumed that $k < n$, the computation in each step is dominated by the distance estimation in the membership reestimation step.

   One possible approach is to speed up on $\Theta(knd)$ for estimating distances between the means and the points. Notice that in 1-d, the means can be ordered. Once ordered ($O(k \log k)$), to find the cluster mean closest to a point, it is enough to perform a binary search on the sorted array of means, $O(\log k)$ per point, resulting in time $O(n \log k + k \log k) = O(n \log k)$ for $k < n$. This is asymptotically smaller than $\Theta(nk)$. The resulting k-means in 1-d would have complexity of $O(n \log k)$ for the class membership reestimation, and $O(n)$ for the mean reestimation, $O(nT \log k)$ total for T iterations.

   It should be notes that approaches which sort the *data* points instead of the cluster means are unlikely to yield significant benefit, as such a sort would have $O(n \log n)$ run-time complexity, with the term $\log n$ likely large enough to outgrow $T \log k$.

2. (3 pts) Derive the EM algorithm for a mixture of two univariate log-normal distributions with unknown mixture weights, and both parameters for each component. Write down your pseudocode. Explain how you would initialize the parameters, and how you would decide on whether the algorithm has converged.

Solution: A pdf for a mixture of two log-normal densities is

$$f\left(x|\boldsymbol{\theta}\right)=\frac{1-\pi}{\sqrt{2\pi\sigma_0^2}x}\exp\left\{-\frac{\left(\ln x-\mu_0\right)^2}{2\sigma_0^2}\right\}+\frac{\pi}{\sqrt{2\pi\sigma_1^2}x}\exp\left\{-\frac{\left(\ln x-\mu_1\right)^2}{2\sigma_1^2}\right\}$$

with parameters $\boldsymbol{\theta}=\left(\pi,\mu_0,\sigma_0^2,\mu_1,\sigma_1^2\right)$, $\pi\in[0,1]$, $\mu_0,\mu_1\in\mathbb{R}$, $\sigma_0^2,\sigma_1^2\in(0,\infty)$.
Suppose $\boldsymbol{X}=(x_1,\ldots,x_n)$ is a data set of iid samples from f for $i=1,\ldots,n$. Alternatively, each point x_i can be thought of as generated as (c_i,x_i) with $c_i\sim\mathsf{Bern}\left(\pi\right)$ and $x_i|\ c_i\sim\mathsf{LogNormal}\left(\mu_{c_i},\sigma_{c_i}^2\right)$ iid for $i=1,\ldots,n$. Let $\boldsymbol{C}=(c_1,\ldots,c_n)$. Algorithm 1 outlines the optimization procedure. Assume current set of parameters $\boldsymbol{\theta}^{(t)}$, the EM iteration finds $\boldsymbol{\theta}$ maximizing the expected complete log-likelihood[1]

$$\begin{aligned}Q\left(\boldsymbol{\theta};\boldsymbol{\theta}^{(t)}\right)=E_{P\left(\boldsymbol{C}|\boldsymbol{X},\boldsymbol{\theta}^{(t)}\right)}\ln P\left(\boldsymbol{X},\boldsymbol{C}|\boldsymbol{\theta}\right)&=-\sum_{i=1}^n\ln x_i-\frac{1}{2}\ln\left(2\pi\right)-\frac{\ln\sigma_1^2}{2}\sum_{i=1}^n\gamma_{i1}-\frac{\ln\sigma_0^2}{2}\sum_{i=1}^n\gamma_{i0}\\&+\ln\pi\sum_{i=1}^n\gamma_{i1}+\ln\left(1-\pi\right)\sum_{i=1}^n\gamma_{i0}-\frac{1}{2\sigma_1^2}\sum_{i=1}^n\gamma_{i1}\left(\ln x_i-\mu_1\right)^2-\frac{1}{2\sigma_0^2}\sum_{i=1}^n\gamma_{i1}\left(\ln x_i-\mu_0\right)^2\end{aligned}$$

where

$$\begin{aligned}\gamma_{i1}=P\left(C_i=1|x_i,\boldsymbol{\theta}^t\right)&=\frac{P\left(C_i=1|\boldsymbol{\pi}^{(t)}\right)f\left(x_i|\mu_1^{(t)},\left(\sigma_1^2\right)^{(t)}\right)}{P\left(C_i=1|\boldsymbol{\pi}^{(t)}\right)f\left(x_i|\mu_1^{(t)},\left(\sigma_1^2\right)^{(t)}\right)+P\left(C_i=0|\boldsymbol{\pi}^{(t)}\right)f\left(x_i|\mu_0^{(t)},\left(\sigma_0^2\right)^{(t)}\right)}\\&=\frac{1}{1+\frac{1-\pi^{(t)}}{\pi^{(t)}}\sqrt{\frac{\left(\sigma_1^2\right)^{(t)}}{\left(\sigma_0^2\right)^{(t)}}}\exp\left\{-\frac{1}{2\left(\sigma_0^2\right)^{(t)}}\left(\ln x_i-\mu_0^{(t)}\right)^2+\frac{1}{2\left(\sigma_1^2\right)^{(t)}}\left(\ln x_i-\mu_1^{(t)}\right)^2\right\}}.\end{aligned}\tag{1}$$

and $\gamma_{i0}=1-\gamma_{i1}$. Differentiating of $Q\left(\boldsymbol{\theta};\boldsymbol{\theta}^{(t)}\right)$ with respect to the parameters yields:

$$\begin{aligned}\frac{\partial Q}{\partial\pi}&=\frac{1}{\pi}\sum_{i=1}^n\gamma_{i1}-\frac{1}{1-\pi}\sum_{i=1}^n\gamma_{i0}=0\Longrightarrow\hat{\pi}=\frac{1}{n}\sum_{i=1}^n\gamma_{i1};\\\frac{\partial Q}{\partial\mu_j}&=\frac{1}{\sigma_j^2}\sum_{i=1}^n\gamma_{ij}\left(\ln x_i-\mu_j\right)=0\Longrightarrow\hat{\mu}_j=\frac{\sum_{i=1}^n\gamma_{ij}\ln x_i}{\sum_{i=1}^n\gamma_{ij}};\\\frac{\partial Q}{\partial\sigma_j^2}&=-\frac{1}{2\sigma_j^2}\sum_{i=1}^n\gamma_{ij}+\frac{1}{\sigma_j^4}\sum_{i=1}^n\gamma_{ij}\left(\ln x_i-\mu_j\right)^2=0\Longrightarrow\hat{\sigma}_j^2=\frac{\sum_{i=1}^n\gamma_{ij}\left(\ln x_i-\hat{\mu}_j\right)^2}{\sum_{i=1}^n\gamma_{ij}}.\end{aligned}$$

To initialize the parameters, one can for example perform k-means on the $\ln x_i$s, and then compute the means $\mu_j^{(0)}$ and variances $\left(\sigma_j^2\right)^{(0)}$ for each cluster, and set $\pi^{(0)}$ as a proportion of the data points in cluster for component 1. The iterations may stop when the log-likelihood changes less than a predefined threshold, e.g., $\frac{1}{n}\left(l\left(\boldsymbol{\theta}^{(t+1)}\right)-l\left(\boldsymbol{\theta}^{(t)}\right)\right)\leq\epsilon$.

3. (3 pts) Assume matrices $\mathbf{X},\mathbf{Y}\in\mathbb{R}^{n\times n}$ are both sparse, with n_X and n_Y denoting the number of non-zero entries in $\mathbf{X}$ and $\mathbf{Y}$, respectively, and $n_X,n_Y\in\mathcal{O}\left(n\right)$. Assume $\mathbf{X}$ is represented by a $n_X\times3$ array A with each entry $x_{ij}\neq0$ of X is recorded as a row (i,j,x_{ij}) of A. Assume $\mathbf{Y}$ is represented similarly by a 2-d array B.
Describe $\mathcal{O}\left(n^2\right)$ worst-case complexity algorithm for finding $\mathbf{XY}$. Prove the complexity of your algorithm. Write down your algorithm as a pseudocode, making clear of all inputs, outputs, and assumptions.

[1]Not to confuse the mixing probability with the constant $\pi=3.14\ldots$ in $-\frac{1}{2}\ln\left(2\pi\right)$.

Algorithm 1 Expectation-Maximization for a Mixture of Two Log-Normals

$t = 0$, initialize $\boldsymbol{\theta}^{(0)}$ ▷ e.g., by k-means on $(\ln x_1, \ldots, \ln x_n)$
repeat ▷ iterating, iteration t
  Compute γ_{i1} from Equation (1), and $\gamma_{i0} = 1 - \gamma_{i1}$ ▷ E-step:
  $\pi^{(t+1)} = \frac{\sum_{i=1}^{n} \gamma_{i1}}{n}$ ▷ M-step: updating π
  $\mu_j^{(t+1)} = \frac{\sum_{i=1}^{n} \gamma_{ij} \ln x_i}{\sum_{i=1}^{n} \gamma_{ij}},\ j = 0, 1$ ▷ M-step: updating μ_0, μ_1
  $\left(\sigma_j^2\right)^{(t+1)} = \frac{\sum_{i=1}^{n} \gamma_{ij} (\ln x_i - \hat{\mu}_j)^2}{\sum_{i=1}^{n} \gamma_{ij}}$ ▷ M-step: updating σ_0^2, σ_1^2
  $t = t + 1$
until convergence ▷ e.g., $l\left(\boldsymbol{\theta}^{(t+1)}\right) - l\left(\boldsymbol{\theta}^{(t)}\right) \leq n\epsilon$

Algorithm 2 Sparse Matrix Multiplication

function SPARSEMATRIXMULTIPLICATION(A, B)
  INPUTS: An $n_X \times 3$ array A, an $n_Y \times 3$ array B representing non-zero entries of an $n \times n$ sparse matrices X and Y, respectively
  OUTPUT: An $n \times n$ matrix (array) $Z = XY$
  Initialize all entries $Z(i, j)$, $i, j = 1, \ldots, n$ to 0
  for $l = 1, \ldots, n_X$ **do** ▷ non-zero x_{ik}, $i = A(l, 1)$, $k = A(l, 2)$
    for $m = 1, \ldots, n_Y$ **do** ▷ non-zero $y_{k'j}$, $k' = B(m, 1)$, $j = B(m, 2)$
      if $A(l, 2) = B(m, 1)$ **then** ▷ product $x_{ik}y_{k'j}$ participates only if $k = k'$
        $Z(A(l, 1), B(m, 2)) = Z(A(l, 1), B(m, 2)) + A(l, 3) \times B(m, 3)$ ▷ $z_{ij} \leftarrow z_{ij} + x_{ik}y_{kj}$
      end if
    end for
  end for
  return Z
end function

Solution: Suppose $Z = XY$, and denote by x_{ij}, y_{ij}, and z_{ij} the entry in the i-th row and j-th column for matrices X, Y, and Z, respectively, $i = 1, \ldots, n$, $j = 1, \ldots, n$. Then $z_{ij} = \sum_{k=1}^{n} x_{ik}y_{kj}$. Note that the only contributing pairs of entries $x_{ik}y_{kj}$ are the ones where both x_{ik} and y_{kj} are not zero. Since there are $n_X \in \mathcal{O}\left(n^2\right)$ non-zero elements x_{ik} of X and $n_Y \in \mathcal{O}\left(n^2\right)$ non-zero elements $y_{k'j}$ of Y, there are $\mathcal{O}\left(n^2\right)$ non-zero products $x_{ik}y_{kj}$. Note than the number of non-zero products $x_{ik}y_{kj}$ would usually be significantly less than $n_X n_Y$ as not all pairs x_{ik} and $y_{k'j}$ would have matching column indices k and row indices k'. However, the observation about the number of non-zero product pairs is sufficient to come up with the $\mathcal{O}\left(n^2\right)$ algorithm for computing $Z = XY$, Algorithm 2. Note that even though both X and Y are sparse, the matrix Z is not necessarily sparse.

4. (2 pts) Assume you are given two sparse vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d$ in the linked list representation. Write down the pseudocode for computing the angle between these two vectors. Suppose that $\boldsymbol{u}$ and $\boldsymbol{v}$ contains $d_u \in o(d)$ and $d_v \in o(d)$ non-zero components. What is the complexity for computing the angle? (Assume that arccos can be computed in $\Theta(1)$ time.)

Solution: Denote by $\theta(\boldsymbol{u}, \boldsymbol{v})$ the angle between $\boldsymbol{u}$ and $\boldsymbol{v}$. Let u_i and v_i denote the i-th components of vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, respectively. Then $\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \|\boldsymbol{u}\| \, |\boldsymbol{v}| \cos\theta(\boldsymbol{u}, \boldsymbol{v})$, so

$$\theta(\boldsymbol{u}, \boldsymbol{v}) = \arccos \frac{\langle \boldsymbol{u}, \boldsymbol{v} \rangle}{\sqrt{\langle \boldsymbol{v}, \boldsymbol{v} \rangle \langle \boldsymbol{u}, \boldsymbol{u} \rangle}}.$$

Since $\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \sum_{i=1}^{d} u_i v_i$, and since $u_i v_i \neq 0$ if and only if both $u_i \neq 0$ and $v_i \neq 0$, for sparse $\boldsymbol{u}$ and $\boldsymbol{v}$, the complexity of the computation of the inner product is $\mathcal{O}(\min\{d_u, d_v\})$. Then $\langle \boldsymbol{u}, \boldsymbol{u} \rangle$ and

Algorithm 3 Angle Between Vectors

function SparseAngle(`uhead`, `vhead`)
  INPUTS: Pointers to linked lists for sparse vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, respectively
  OUTPUT: Angle θ between $\boldsymbol{u}$ and $\boldsymbol{v}$
  $\theta = \arccos \frac{\sqrt{\texttt{SparseInnerProduct}(\boldsymbol{u},\boldsymbol{u}) \times \texttt{SparseInnerProduct}(\boldsymbol{v},\boldsymbol{v})}}{\texttt{SparseInnerProduct}(\boldsymbol{u},\boldsymbol{v})}$
  return θ
end function

function SparseInnerProduct(`uhead`, `vhead`)
  INPUTS: Pointers to linked lists for sparse vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, respectively
  OUTPUT: $ip = \langle \boldsymbol{u}, \boldsymbol{v} \rangle$
  $ip \leftarrow 0$ ▷ initialize inner product to 0
  `ucurrent` $\leftarrow$ `uhead` ▷ suppose $i =$ `uhead` $\rightarrow$ `value` is the current index for $\boldsymbol{u}$
  `vcurrent` $\leftarrow$ `vhead` ▷ suppose $j =$ `vhead` $\rightarrow$ `value` is the current index for $\boldsymbol{v}$
  while `ucurrent` $\neq$ `nil` $\wedge$ `vcurrent` $\neq$ `nil` **do** ▷ still non-zero components left in both vectors
    if `ucurrent` $\rightarrow$ `index` $<$ `vcurrent` $\rightarrow$ `ucurrent` **then** ▷ $i < j$
      `ucurrent` $\leftarrow$ (`ucurrent` $\rightarrow$ `next`) ▷ considering next non-zero component with index i
    else if `ucurrent` $\rightarrow$ `index` $>$ `vcurrent` $\rightarrow$ `ucurrent` **then** ▷ $i > j$
      `vcurrent` $\leftarrow$ (`vcurrent` $\rightarrow$ `next`) ▷ considering next non-zero component with index j
    else ▷ $i = j$
      $ip \leftarrow ip +$ `ucurrent` $\rightarrow$ `value` $\times$ `vcurrent` $\rightarrow$ `value` ▷ adding the contribution $u_i v_i$
    end if
  end while
  return ip
end function

$\langle \boldsymbol{v}, \boldsymbol{v} \rangle$ have $\mathcal{O}(d_u)$ and $\mathcal{O}(d_v)$ computational complexity, respectively. Therefore, the complexity of computation of the angle $\theta(\boldsymbol{u}, \boldsymbol{v})$ is $\mathcal{O}(d_u + d_v + \min\{d_u, d_v\}) = \mathcal{O}(d_u + d_v) = o(d)$.

We are assuming that each `node` contains three fields: `next`, a location of the next `node` in the list; `index`, a component index for a non-zero component stored in this `node`; and `value` of the above non-zero component. We are assuming that `head` denotes the location of the first `node` in the linked list, and that `node`s in the list are arranged sequentially in the order of the corresponding `index` values corresponding to non-zero components of a vector. Pseudocode is shown in Algorithm 3.