SDS 385 Ex 08:
Sparse Matrix Factorization

November 10, 2016

Jennifer Starling

Sparse Matrix Factorization

Algorithm Details

I implemented the rank 1 sparse matrix factorization algorithm detailed on pages 519-520 of Witten, Hastie & Tibshirani, 2009, A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis. The outline of this algorithm is as follows.

Optimization Problem:

$$\underset{u \in R^N, v \in R^p}{\text{argmin}} = ||\mathbf{X} - \mathbf{d}\mathbf{u}\mathbf{v}^{\mathbf{T}}||_{\mathbf{F}}^{\mathbf{2}}$$

subject to

$$||u||_2^2 = 1, ||v||_2^2 = 1,$$
$$||u||_1 \leq \lambda_u, ||v||_1 \leq \lambda_v,$$

where F indicates the squared Frobenius norm of a matrix (sum of squared elements).
This problem is equivalent to:

maximiz$_{u,v} u^T X v$ subject to the same constraints as above.

The equivalence proof is in the appendix of the paper.

Algorithm:

1. Initialize v to some random vector where the l2 norm equals 1.

2. Let $u = \frac{S_{\theta_u}(Xv)}{||S_{\theta_u}(Xv)||_2}$

3. Let $v = \frac{S_{\theta_v}(X^T u)}{||S_{\theta_v}(X^T u)||_2}$

Iterate these steps until convergence is reached.
Convergence criteria: $sum(abs(v.old - v.new)^2 < \epsilon$

Notes:
- $S_{\theta_v}(a) = sign(a)(|a| - \theta_v)_+$ is the same soft-thresholding operator used in previous exercises 6, 7 and 8.
- θ_u and θ_v are found using binary searches within each iteration.

My Implementation

See R code appendix for my implementation of this function.

Reasonableness Checks

I made several reasonableness checks to ensure that my function was behaving in a sensible way. These included:
1. Comparison with the R package PMA written by the authors of the paper, and
2. Generating a matrix of random values X, and verifying that as the λ_u and λ_v penalties get closer to zero, the resulting u and v vectors are more sparse.

Example: For $\lambda_u = \lambda_v$, the table below shows the number of nonzero elements of v and u for a handful of lambda values.

```
  lambdas nonzero.u nonzero.v
1     2.0         5         4
2     1.5         3         4
3     1.0         1         1
4     0.5         1         1
5     0.0         1         1
```

Application to Marketing

First, I transformed the data using the square root transform. I selected this transform because the matrix X contains count data, so is likely Poisson. The Anscombe transform also could be appropriate here.

After transforming the data, I calculated the means for each category as a quick way to eyeball what categories might be interesting (as higher counts correspond to more tweets for a specific category). This quick visualization yielded a few potential interesting categories. The means are below in decreasing order. (Note: not a rigorous analysis, just a quick initial investigation of the data.)

```
                 colmeans
chatter          1.9094277
photo_sharing    1.3681041
current_events   1.0501033
health_nutrition 1.0167170
travel           0.9549531
cooking          0.9548596
sports_fandom    0.9266684
politics         0.9127590
food             0.8753729
shopping         0.8478124
college_uni      0.7991081
personal_fitness 0.7801828
tv_film          0.6993931
news             0.6925771
uncategorized    0.6505390
religion         0.6451401
family           0.6420915
online_gaming    0.6414276
parenting        0.6044721
fashion          0.6030449
automotive       0.5741795
outdoors         0.5594291
school           0.5515864
music            0.5246168
sports_playing   0.4988884
beauty           0.4791737
computers        0.4741713
art              0.4619582
home_and_garden  0.4469229
dating           0.4349158
eco              0.4313205
crafts           0.4241252
business         0.3661789
small_business   0.2972368
adult            0.1598987
spam             0.0063218
```

I began with $\lambda_u = \lambda_v = 5$. This yielded a very un-sparse matrix v, where the only uninteresting (zero-ed out) categories were 'adult' and 'spam'.

I then tried $\lambda_u = \lambda_v = 2$. In this case, interesting categories shown below.

```
             cols          v
1         chatter 0.29399338
2   photo_sharing 0.05593838
3         tv_film 0.06838591
4            food 0.06990010
5        shopping 0.00104017
6 health_nutrition 0.84543834
7         cooking 0.08204693
8        outdoors 0.22349908
9 personal_fitness 0.35975760
```

These are fairly consistent with the initial means investigation, as a sanity check.

Finally, I tried $\lambda_u = \lambda_v = 2$.

```
             cols          v
1         chatter 0.21975824
2 health_nutrition 0.93825570
3        outdoors 0.09066449
4 personal_fitness 0.25132150
```

Two comments on these results:

• The number of categories to select as interesting would depend on the marketing strategy and budget. This would drive the ultimate selection of the lambda penalties. Lower lambda penalties yield fewer selected categories.

• I included 'chatter' on these lists of interesting categories, but if 'chatter' is uninteresting to the marketing campaign (which seems likely), it could be removed prior to the analysis.

Appendix: R Code

```
#Big Data Exercise 9
#November 10 ,2016
#Jennifer Starling

#References paper:
# Witten, Tibshirani, Hastie.  2009. A penalized matrix decomposition...

#################################
###   REQUIRED FUNCTIONS:     ###
#################################

#l1 penalty; Soft thresholding operator.
soft <- function(x,theta){
  return(sign(x)*pmax(0, abs(x)-theta))
}

#l1 norm of a vector.
l1norm <- function(vec){
  a <- sum(abs(vec))
  return(a)
}

#l2 norm of a vector.
l2norm <- function(vec){
  a <- sqrt(sum(vec^2))
  return(a)
}

#Binary search function
#(Source: Witten, Hastie & Tibshirani R package PMA: https://cran.r-project.org/
    web/packages/PMA/)
#(For finding theta for soft-thresholding for each iteration)
BinarySearch <- function(argu,sumabs){

  l2n = function(vec) {return(sqrt(sum(vec^2)))}
  soft = function(x,theta) { return(sign(x)*pmax(0, abs(x)-theta))}

  if(l2n(argu)==0 || sum(abs(argu/l2n(argu)))<=sumabs) return(0)
  lam1 <- 0
  lam2 <- max(abs(argu))-1e-5
  iter <- 1
  while(iter < 150){
    su <- soft(argu,(lam1+lam2)/2)
    if(sum(abs(su/l2n(su)))<sumabs){
      lam2 <- (lam1+lam2)/2
    } else {
      lam1 <- (lam1+lam2)/2
    }
    if((lam2-lam1)<1e-6) return((lam1+lam2)/2)
    iter <- iter+1
  }
  warning("Didn't quite converge")
  return((lam1+lam2)/2)
}

#######################################################
```

```r
###   PENALIZED MATRIX DECOMPOSITION FUNCTION:     ###
######################################################


#-----------------------------------------------------------------------
#Sparse Matrix Factorization (Penalized Matrix Decomposition) Function
#For a single factor, ie K=1 (rank-1 approximation to original matrix)
#Inputs:
#   X = matrix to be factorized
#   lambdaU = the u penalty (c1)
#   lambdaV = the v penalty (c2)
#      *If lambda1 = lambda2 = 0, function returns the non-sparse Rank 1 SVD of X.
#   maxiter = maximum number of iterations allowed
#   tol = tolerance level for convergence check
#Output: List object, including the following:
#   Xsp = sparse matrix factorization of X.
#   U, D, V = the decomposed elements of X, where X = U * D * t(V)

sparse.matrix.factorization.rank1 = function(X, lambdaU=1, lambdaV=1, maxiter=20,
    tol=1E-6){

    #1. Housekeeping parameters.
    i=1                     #Initialize iterator.
    converged <- 0          #Indicator for whether convergence met.
    p = ncol(X)             #Number of columns of X matrix.

    #2. Initializations
    v.old = rnorm(p)        #Initialize v.old to a random vector. (To get iterations
        started.)
    v = rep(sqrt(1/p),p) #Initialize v to meet constraint l2norm(v) = 1.

    #Iterate until convergence.
    for (i in 1:maxiter){

        #1. Update u.

        #First, calculate theta for sign function.
        u.arg = X %*% v      #Argument to go into sign function: Xv
        u.theta = BinarySearch(u.arg,lambdaU)

        #Second, update u.
        u = matrix( soft(u.arg,u.theta) / l2norm(soft(u.arg,u.theta)), ncol=1)

        #-------------------------
        #2. Update v.

        #First, calculate theta for sign function.
        v.arg = t(X) %*% u
        v.theta = BinarySearch(v.arg,lambdaV)

        #Second, update v.
        v = matrix( soft(v.arg,v.theta) / l2norm(soft(v.arg,v.theta)), ncol=1)

        #-------------------------
        #3. Convergence check steps.

        #Exit loop if converged.
        if(sum(abs(v.old - v)) < tol){
            converged=1
```

```
            break
        }

        #If not converged, update v.old for next iteration.
        v.old = v
    }

    #Set d value.
    d = as.numeric(t(u) %*% (X %*% v))

    #Reconstruct sparse X matrix.
    Xsp = d * tcrossprod(u,v)

    #Return function results.
    return(list(Xsp=Xsp,u=u,d=d,v=v,lambdaU=lambdaU,lambdaV=lambdaV,converged=converged,iter=i))
}
#----------------------------------------------------------------------

#Simulate a matrix to chek that results behaving sensibly.

X = matrix(rnorm(20),nrow=5,ncol=4)
n = nrow(X)
p = ncol(X)

#Paper notes that if you want u and v to be equally sparse, set a constant c,
#and let lambdaU = c*sqrt(n), and let lambdaV = c * sqrt(p)

c = 2
lambdaU = c*sqrt(n)
lambdaV = c*sqrt(p)

test2 = sparse.matrix.factorization.rank1(X,lambdaU,lambdaV,maxiter=20,tol=1E-6)

#Just a few random tests with various lambda values.
#Confirm that u and v getting more sparse as lambdas decrease.
test2 = sparse.matrix.factorization.rank1(X,lambdaU=2,lambdaV=2,maxiter=20,tol=1E-6)
test1.5 = sparse.matrix.factorization.rank1(X,lambdaU=1.5,lambdaV=1.5,maxiter=20,tol=1E-6)
test1 = sparse.matrix.factorization.rank1(X,lambdaU=1,lambdaV=1,maxiter=20,tol=1E-6)
test.5 = sparse.matrix.factorization.rank1(X,lambdaU=.5,lambdaV=.5,maxiter=20,tol=1E-6)
test0 = sparse.matrix.factorization.rank1(X,lambdaU=0,lambdaV=0,maxiter=20,tol=1E-6)

#Number of non-sparse u and v in each test.
lambdas = c(2,1.5,1,.5,0)
nonzero.u = c(sum(test2$u!=0),sum(test1.5$u!=0),sum(test1$u!=0),sum(test.5$u!=0),sum(test0$u!=0))
nonzero.v = c(sum(test2$v!=0),sum(test1.5$v!=0),sum(test1$v!=0),sum(test.5$v!=0),sum(test0$v!=0))

cbind.data.frame(lambdas=lambdas,nonzero.u=nonzero.u,nonzero.v=nonzero.v)

print("lambdaU = lambdaV = 2")
test2$u
test2$v
```

```r
print("lambdaU = lambdaV = 1.5")
test1.5$u
test1.5$v

print("lambdaU = lambdaV = 1")
test1$u
test1$v

print("lambdaU = lambdaV = .5")
test.5$u

print("lambdaU = lambdaV = 0")
test0$u

###########################################
###   APPLICATION TO MARKETING          ###
###########################################

#Read in marketing data.
data = read.csv('/Users/jennstarling/UTAustin/2016_Fall_SDS 385_Big_Data/Course Data/social_marketing.csv',header=T)
X.counts = as.matrix(data[,-1]) #Gets rid of ID column.

#Square Root or Anscombe-transform data, because it is count data.
#Chose square root in this case.
X = sqrt(X.counts)

#Eyeball data to see which categories might be interesting.
cbind(colmeans=sort(colMeans(X),decreasing=T))

#Try various lambda values to see which categories are interesting.
result1 = sparse.matrix.factorization.rank1(X, lambdaU=5, lambdaV=5, maxiter=20, tol=1E-6)
cbind.data.frame(colnames(X),result1$v)

result2 = sparse.matrix.factorization.rank1(X, lambdaU=2, lambdaV=2, maxiter=20, tol=1E-6)
cbind.data.frame(colnames(X),result2$v) #Show all.
cbind.data.frame(cols=colnames(X)[which(result2$v!=0)],v=result2$v[which(result2$v!=0)])
    #show non-zero only.

result3 = sparse.matrix.factorization.rank1(X, lambdaU=1.5, lambdaV=1.5, maxiter=20, tol=1E-6)
cbind.data.frame(colnames(X),result3$v)
cbind.data.frame(cols=colnames(X)[which(result3$v!=0)],v=result3$v[which(result3$v!=0)])
    #show non-zero only.
```