Assignment2

Tianyi Fang

September 14, 2017

Problem 1: Eigenvalues and eigenvectors

1. Explain why the power method returns the largest eigenvalue and its eigenvector.

Power method: Given a matrix A, the algorithm will produce a number lamda, which is the largest(absolute value) eigenvalue of A, and a non-zero vector v, the corresponding eigenvector of lamda, such that

$$Av = \lambda v$$

. When we want to get the initial vector v, it can be written as

$$A(c_1x_1 + c_2x_2 + ... + c_nx_n) = \lambda_1c_1x_1 + \lambda_2c_2x_2 + ... + \lambda_nc_nx_n$$

if we keep multiple A for both sides, and scale it to normalize, the first vector's coefficient will be close to 1. That is it becomes the dominant vector. And the cofficient is its Eigen value.

2. Explain why the norm of a symmetric matrix is the ratio of its largest and smallest eigenvalues.

For a symmetric matrix A, it follows A*A^T=I, thus, following the definition:

Following the definition of conditional number k:

$K = ||A||||A^{-1}||$

$||A|| = \max_{||v||=1}||A|| = \max_{||v||=1}\sqrt{v^TA^TAv} = \max_{||v||=1}\sqrt{v^TBv} = \sqrt{\lambda\max(B)} = \sqrt{\lambda\max(A^TA)}$

For symmetric matrix A, $A^T = A$, $A^{-1}A = I$, so $\sqrt{\lambda\max(A^TA)} = \lambda\max(A)$

Since $A^{-1}A = I$, $Av = \lambda v \Rightarrow A^{-1}Av = A^{-1}\lambda v \Rightarrow v = A^{-1}\lambda v \Rightarrow \frac{v}{\lambda} = A^{-1}v$,

$\frac{1}{\lambda\max(A)}$ *is the largest Eigen value of* A^{-1}

$k = ||A||||A^{-1}|| \geq ||AA^{-1}|| = \lambda\max(A) * \lambda\max(A^{-1}) = \frac{\lambda\max(A)}{\lambda\max(A^{-1})} = \frac{\max(A)}{\lambda\min(A)}$

Figure 1:

Problem 2: Implementing priority queues and sorting.

3.What is the index of the parent node of element i of the vector? What are the indices of the left and right child nodes of element i of the vector?

Parent(A,i): if i == 1 return NULL; else return i/2 Left child(A,i): if 2*i <= heap_size(A) return 2*i else return NULL Right child(A,i): if 2*i+1<= heap_size(A) return 2*i+1 else return NULL

4.Write a function make heap to return an empty heap.

```
#pre-allocate
make_heap <-function(n){
  lmax <- rep(NA, n)#a vector contains n NA
  heap_size <- sum(!is.na(lmax))#count for # of non-NA elements as heap_size
  return(list(lmax,heap_size))
}
#test
make_heap(5)
```

```
## [[1]]
## [1] NA NA NA NA NA
##
## [[2]]
## [1] 0
```

5. Write a function to return the maximum element of the heap.

```
find_max <- function(lmax){
  if(heap_size > 0){
    root <- lmax[1]
  }
  return(root)
}
```

6. Write a function to remove the maximum element of the heap.

```
#remove_max
remove_heap <- function(lmax,n){
  heap_size <- n
  lmax[1]<- lmax[heap_size]#swap the last leaf with the root
  lmax[length(lmax)] <- NA #set the leaf as NA,
  heap_size <- heap_size - 1
  i <- 1
  #for(i in 1:(heap_size)){
  #for loop is not correct, because it will stop at some i since the next nodes has already sorted
  while(i < heap_size){
    left <- 2*i
    right <- 2*i +1
    max_index <- i #compare keys and get the index
    if(left <= heap_size && lmax[left]> lmax[i]){
      max_index <- left
    }
    if(right<= heap_size && lmax[right]> lmax[max_index]){
      max_index <- right
    }
    if(max_index != i){ #swap parent and child if neccesary
      value_i <- lmax[i]
      lmax[i] <- lmax[max_index]
      lmax[max_index] <- value_i
    }else{break} #when hit to the node and cannot go upper anymore.
  }
  return(lmax)
```

```
}
lmax <- c(9,8,7,4,1,1,3,2,2)
n<-length(lmax)
remove_heap(lmax,n)
```

```
## [1]  8  2  7  4  1  1  3  2 NA
```

7. Write a function to insert a new element into the heap. For the last two, you can create additional helper functions if you want to.

```
insert_heap <- function(lmax,n,key){
  heap_size <- n
  lmax[heap_size + 1] <- key# add key to the leaf
  heap_size <- heap_size + 1
  i <- heap_size
  while(i > 1){

    if(lmax[i]>lmax[i%/%2]){
      #parent <- i%/%2 # get the index of its parent
      max <- lmax[i]
      lmax[i] <- lmax[i%/%2]
      lmax[i%/%2] <- max
      i <- i%/%2
    }else{break}

  }
  return(lmax)
}
lmax <- c(8,4,7,1,1,3,2,2)
n<-length(lmax)
insert_heap(lmax,n,9)
```

```
## [1] 9 8 7 4 1 3 2 2 1
```

8.Write 2 functions to sort a vector of number of numbers.

```
your_ran <- floor(rnorm(20,100,40))
your_heap <- make_heap(30)
sort_heap <- function(your_ran, your_heap){
  queue <- your_heap[[1]] #first element of the list
  heap_size <- your_heap[[2]] #second element of the list

  for(i in 1:length(your_ran)){
    queue <- insert_heap(queue, heap_size, your_ran[i])
    heap_size <- sum(!is.na(queue))
  }
  while(heap_size >0){
   your_heap <- remove_heap(queue, heap_size)
   heap_size <- heap_size - 1
  }
  return(your_heap)
  #after remove, only NA is left
}
sort_heap(your_ran, your_heap)
```

```
##  [1] 147 138 123 111 106 113  93 103 107  62  93  51  88  29  91  52  47
## [18] 101  65  62  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA
```

Problem3:The knapsack problem

1.Write a function that accepts as input two vectors of the same length, **w** and **v**, the ith elements of which give the weight and value of the ith object. The function also accepts a scalar **W_knapsack** giving the capacity of the knapsack. The function should return two objects, a scalar **V_knapsack**, giving the maximum value of the knapsack, and a vector **obj_count**, the ith element of which gives the number of objects of type i contained in the bag when it is filled optimally.

4.Problem Markov chains

1.P(S/2=j, S/1=i)=

$$P(S_2 = j, S_1 = i) = \pi_i^1 A_{ij}$$

2.P(S_2=j)=

$$\sum_{i=1}^{N} \pi_i^1 A_{ij}$$

3.Write

$$\pi^2$$

as a function of

$$\pi^1$$

and A.

$$\pi^2 = ((\pi^1)^T A)^T$$

4.How many summations and multiplications involve:

it's

$$O(N^2)$$

, because

$$\pi_i^1 A_(ij)$$

is a Nx1 * NxN, so it goes N(N-1) times.

5.

$$\pi^t = A^T \pi^{t-1} = A^T A^T \pi^{t-2} = ... = (A^T)^{t-1} \pi^1$$

Since its maxtrix*matrix, so the cost is

$$O(N^3)$$

####6. How many sequences are there?

$$S = (S_1, S_2, S_3, ..., S_T - 1, S_T)$$

From S1(1) to S2, there are N states it can go, since the probability is N. So from S1 to ST, there is T times, so the total sequence is

$$N^T$$

7.

for P(St=i)

$$N^{t-1} - 1$$

for P(St)

$$N^t - N$$