TP2

Basic PageRank

Functions

These are the functions we created to compute the PageRank index.

Sum powers of a matrix

Computes the sum of n first powers of matrix m.

n.d is a damping factor applied to the powers of m.

Example : *sum.powers.matrix(m, 3, 2)* returns: $m + \frac{1}{2^2} * m^2 + \frac{1}{3^2} * m^3$

```r
sum.powers.matrix <- function(m, n) {
  powers <- c(1:n)
  res <- Reduce('+', lapply(powers, function(x) m %^% x))
  res[res > 1] <- 1

  return(res)
}
```

Remove auto-references

Removes all the auto-references in matrix m (ie puts the diagonal to 0).

```r
remove.autoreferences <- function(m) {
    res <- m
    diag(res) <- 0

    return(res)
}
```

PageRank iteration

Computes an iteration of the PageRank algorithm.

Parameters:

- refs: the references matrix
- n: the number of powers of refs to consider (ie the depth of references)
- d: the PageRank damping factor
- n.d: the damping factor for powers of refs (see function sum.powers.matrix)
- pr: the current PageRank values

```
#Cette fonction fait une itérations de l'algorithme page rank. Nous avons ajouter une petite modificati
# article non référencé) on remplace ensuite la valeur qui sera égale a Inf, par un 0
page.rank <- function(m, d, pr){
  denum <- (pr/colSums(m))
  denum[denum == Inf] <- 0
  (pr <- (1-d)/3 + (d * (m %*%denum)))
  #print(pr[450])
  return(pr)
}
```

corr.vm

We're also using some of the other functions used in TP1 in regards to item-item recommandations.

corr.vm Computes the Pearson correlations between the different rows of a matrix.

```
#Correlation entre la rangee v (v = index) et chaque colonne de la matrice m
corr.vm <- function(v,m) {
  # on centre les valeurs de la matrice m en fonction de la moyenne, on la renomme m.centre
  v.i <- rowMeans(m[,], na.rm=T)
  # on enleve les NA
  m[is.na(m)] <- 0
  m.centre <- m - v.i
  # on centre le vecteur v en fonction de sa moyenne
  v.index <- v
  v.index[is.na(v)] <- 0
  v.index <- v.index - mean(v, na.rm=T)
  # on retourne ensuite un vecteur correspondant entre le vecteur v et sa correlation avec chaque range
  return( (v.index%*%t(m.centre))/(sqrt(sum(v.index^2) * rowSums(m.centre^2))))
}
```

cosinus.vm

function that computes the Cos between all the different columns of a matrix

```
## Cosinus entre un vecteur v et chaque colonne dela matrice m
cosinus.vm <- function(v,m) {
  # On on met tous nos valeurs de NA a 0, sinon on va avoir des problemes de calculs avec des matrices
  m[is.na(m)] <- 0
  v[is.na(v)] <- 0 ;
  # On calcule le cosinus entre le vecteur V et les colonnes de la matrice m en utilisant la formule vu
  (v %*% m)/(sqrt(colSums(m^2)) * sqrt(sum(v^2)))
}
```

page.rank.until.stab

This function calls an iteration of page rank until our rankings are stabilized. Stabilized meaning that the mean error between our last two iterations of page rank is less then 0.0001

```r
# Cette fonction exécute l'algorithme PageRank jusqu'à ce que c'est valeurs soit stabilisés. On définie
page.rank.until.stab <- function(m, d, pr){
  pr.next <- page.rank(m, d, pr)
  while(abs(mean(pr.next-pr)) > 0.0001){
    pr <- pr.next
    pr.next <- page.rank(m,d,pr)
  }
  return (pr)
}
```

PageRank flow

Read source file and turn it into a matrix (for later work).

```r
# Read source file
data <- read.table("citeseer.rtable")

# Cast data to matrix
references <- as.matrix(data)
```

Example of PageRank computations with a simple reference matrix.

```r
m <- matrix(c(0,1,1,0,0,1,1,0,0),3)
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    1
## [2,]    1    0    0
## [3,]    1    1    0
```

Initial PageRank values:

```r
pr <- rep(1,3)
```

```
## [1] 1 1 1
```

Damping factor:

```r
d <- 0.85
```

3 PageRank iterations:

```r
pr <- page.rank(m, d,  pr)
print(pr)
```

```
##       [,1]
## [1,] 0.900
## [2,] 0.475
## [3,] 1.325
```

```r
pr <- page.rank(m, d,  pr)
print(pr)
```

```
##         [,1]
## [1,] 1.17625
## [2,] 0.43250
## [3,] 0.83625
```

```r
pr <- page.rank(m, d,  pr)
```

```
##           [,1]
## [1,] 0.7608125
## [2,] 0.5499063
## [3,] 0.9175312
```

Question 1

We start our first question by importing all our data from our citeseer dataset.

```r
m = as.matrix(read.table("citeseer.rtable", check.names=F))
library("expm")
library("dplyr")
d <- 0.85
pr <- rep(1, dim(m)[1])
```

We then calculate the domain of our search. We specify $S_{primary}$ as our primary domain being the references of our target article, which is 422908. We're calculating our extended domain, which is S' and defined as the domain of all our articles in S. Which can be described as $S' = S \wedge \sum_i S(S_i)$. We calculate S' by squaring our references matrix, which gives us the references of our references and then adding it to our previous matrix. All values for an article x that are higher then 0 are considered in our domain.

```r
# On calcule notre domaine S comme étant tout l
#es article sont référencés par notre origine. Pour faire cela,
#on regarde ceux qui on une valeur positive dans notre matrice référentielle
S <- which(m["422908",]==1)
# Pour le domaine S prime, nous voulons aussi rajouter les références des références.
#Pour faire cela, nous faisont que prendre la somme des deux première puissance de la
# matrice référentielle. C'est a dire, la matrice référentielle elle-même
#et la deuxième puissance.
m.prime <- sum.powers.matrix(m,2)

#on enleve les auto-references
diag(m.prime) <- 0
#et on prend les valeurs positives
S.prime <- which(m.prime["422908",]==1)

print(S)
```

```
## 110303    124 131548 147460 149673 155792  17094  19422 241538 311874
##     41     92    113    162    168    184    232    295    401    547
## 315693   3170 396568 466838 497542 522428  64835
##    557    560    723    809    859    889    961
```

```r
print(S.prime)
```

```
## 110303    124 131548 147460 149673 149820 155792  17094  19422 206738
##     41     92    113    162    168    169    184    232    295    312
## 225173  22638 241538 296098 311874 315693   3170 389559 396568 425638
##    363    369    401    511    547    557    560    710    723    755
## 426325 466838 497542 522428  64835  70445  96767
##    758    809    859    889    961    985   1081
```

We then continue with on with calculating the PageRank of all our articles. We do this globally and locally.

```r
#On calcule le pagerank de tout nos articles (pas très long)
pr <- page.rank.until.stab(m, d, pr)

pr.S <- rep(1, dim(m[S,S])[1])
#on calcule le page rank local du domaine S
pr.S <- page.rank.until.stab(m[S,S], d, pr.S)
#et apres pour S prime
pr.S.prime <- rep(1, dim(m[S.prime,S.prime])[1])

pr.S.prime <- page.rank.until.stab(m[S.prime, S.prime], d, pr.S.prime)
```

And then we extract from all the Page Ranks, the one that we are interested in, being the ones in the $S_{primary}$ domain and the S' domain. We only do this in the global PageRanks since all the data from the local PageRanks are to be used for recommendations.

```r
#on place les rankings PageRank de notre domaine dans un vecteur
S.rankings <- pr[S]
#pour le domaine S prime
S.prime.rankings <- pr[S.prime]
#On crée un dataframe avec nos données
S.dat <- data.frame(S.rankings, S)
S.dat$article = rownames(S.dat)
#même chose pour le domaine S prime
S.prime.dat <- data.frame(S.prime.rankings, S.prime)
S.prime.dat$article = rownames(S.prime.dat)
print.data.frame(S.prime.dat)
```

```
##        S.prime.rankings S.prime article
## 110303       0.06687686      41  110303
## 124          0.07502095      92     124
## 131548       0.08741421     113  131548
## 147460       0.07570849     162  147460
## 149673       0.06458252     168  149673
## 149820       0.07597222     169  149820
## 155792       0.09559443     184  155792
## 17094        0.08532668     232   17094
## 19422        0.06003019     295   19422
## 206738       0.10839077     312  206738
## 225173       0.05069370     363  225173
## 22638        0.09349323     369   22638
## 241538       0.05000000     401  241538
## 296098       0.11393336     511  296098
## 311874       0.05000000     547  311874
## 315693       0.05897222     557  315693
## 3170         0.08210018     560    3170
## 389559       0.05000000     710  389559
## 396568       0.05000000     723  396568
## 425638       0.05000000     755  425638
## 426325       0.08988252     758  426325
## 466838       0.06651787     809  466838
## 497542       0.06432166     859  497542
## 522428       0.05000000     889  522428
## 64835        0.07185830     961   64835
## 70445        0.08631711     985   70445
## 96767        0.08451993    1081   96767
```

```r
print.data.frame(S.dat)
```

```
##        S.rankings   S article
## 110303 0.06687686  41  110303
## 124    0.07502095  92     124
## 131548 0.08741421 113  131548
## 147460 0.07570849 162  147460
## 149673 0.06458252 168  149673
## 155792 0.09559443 184  155792
## 17094  0.08532668 232   17094
## 19422  0.06003019 295   19422
## 241538 0.05000000 401  241538
## 311874 0.05000000 547  311874
## 315693 0.05897222 557  315693
## 3170   0.08210018 560    3170
## 396568 0.05000000 723  396568
## 466838 0.06651787 809  466838
## 497542 0.06432166 859  497542
## 522428 0.05000000 889  522428
## 64835  0.07185830 961   64835
```

after sorting all the values, we can determine which ones we will recommend.

```r
#On crée un dataframe avec nos données
S.loc.dat <- data.frame(pr.S)
S.loc.dat$article <- rownames(pr.S)
#meme chose pour S prime (local)
S.prime.loc.dat <- data.frame(pr.S.prime)
S.prime.loc.dat$article <- rownames(pr.S.prime)
#on trie pour S et S prime
S.loc.best <- S.loc.dat %>% select(pr.S, article) %>% arrange(desc(pr.S, arr.ind=T))
S.prime.loc.best <- S.prime.loc.dat %>% select(pr.S.prime, article) %>% arrange(desc(pr.S.prime, arr.in

#on effectue un trie sur nos valeurs, on sort celles qui sont les plus hautes en premier
S.best <- S.dat %>% select(S.rankings, S, article) %>% arrange(desc(S.rankings, arr.ind=T))
#même chose pour le domaine S prime
S.prime.best <- S.prime.dat %>% select(S.prime.rankings, S.prime, article) %>% arrange(desc(S.prime.ran
print("Version Globales")
```

```
## [1] "Version Globales"
```

```
head(S.best)
```

```
##   S.rankings   S article
## 1 0.09559443 184  155792
## 2 0.08741421 113  131548
## 3 0.08532668 232   17094
## 4 0.08210018 560    3170
## 5 0.07570849 162  147460
## 6 0.07502095  92     124
```

```
head(S.prime.best)
```

```
##   S.prime.rankings S.prime article
## 1       0.11393336     511  296098
## 2       0.10839077     312  206738
## 3       0.09559443     184  155792
## 4       0.09349323     369   22638
## 5       0.08988252     758  426325
## 6       0.08741421     113  131548
```

```
print("Versions Locales")
```

```
## [1] "Versions Locales"
```

```
head(S.loc.best)
```

```
##        pr.S article
## 1 0.3622987  131548
## 2 0.3440795    3170
## 3 0.3226922     124
## 4 0.2736885  155792
## 5 0.1934471   17094
## 6 0.1851999  466838
```

```
head(S.prime.loc.best)
```

```
##   pr.S.prime article
## 1  0.6188051  131548
## 2  0.4713936    3170
## 3  0.4631686     124
## 4  0.4201475   17094
## 5  0.2952258   70445
## 6  0.2722679  426325
```

Question 2

For the second question, we will compute recommendations based on the same principles then the TP1. We will use and item-item approch and recommend articles based on the similarity of their references. To do that, we will simply compute the Pearson Correlation and the Cosine between all of their rows.

```
## on calcule nos coefficients de correlation et du cosinus
corr.ratings <- corr.vm(m["422908", ], m[S,])
cos.ratings <- cosinus.vm(m["422908",], t(m[S,]))

# on remplace les NaN par 0
corr.ratings[is.nan(corr.ratings)] <- 0
cos.ratings[is.nan(cos.ratings)] <- 0

# on prends nos etiquettes
labels.corr <- colnames(corr.ratings)
labels.cos <- colnames(cos.ratings)

# on cree nos dataframes
df.corr <- data.frame(corr = as.vector(corr.ratings), article = labels.corr)
df.cos <- data.frame(cos = as.vector(cos.ratings), article = labels.cos)
```

Having done that, we can sort all of our coefficients and then take our highest ones for recommendation.

```
#on trie
df.best.cos <- df.cos %>% select(cos, article) %>% arrange(desc(cos, arr.ind=T))
df.best.corr <- df.corr %>% select(corr, article) %>% arrange(desc(corr, arr.ind=T))

print.data.frame(df.best.cos)
```

```
##          cos article
## 1  0.4850713  149673
## 2  0.4850713  466838
## 3  0.4583492  155792
## 4  0.4338609  497542
## 5  0.3960590  147460
## 6  0.3960590   17094
## 7  0.3429972  131548
## 8  0.3253957     124
## 9  0.3253957    3170
## 10 0.2169305   64835
## 11 0.1714986  315693
## 12 0.1400280   19422
## 13 0.0000000  110303
## 14 0.0000000  241538
## 15 0.0000000  311874
## 16 0.0000000  396568
## 17 0.0000000  522428
```

```
print.data.frame(df.best.corr)
```

```
##            corr article
## 1   0.482159228  466838
## 2   0.482159228  149673
## 3   0.453336944  155792
## 4   0.429746082  497542
## 5   0.390922862   17094
## 6   0.390922862  147460
## 7   0.336156075  131548
## 8   0.320173397     124
## 9   0.320173397    3170
## 10  0.210600712   64835
## 11  0.167613946  315693
## 12  0.134714960   19422
## 13  0.000000000  241538
## 14  0.000000000  311874
## 15  0.000000000  396568
## 16  0.000000000  522428
## 17 -0.005396661  110303
```

Question 3

Here, we will use the cross-validation method to evaluate the predictions resulting from the item-item approach.

Functions used for this approach:

```
# Calcul du cosinus entre un vecteur et les colonnes d'une matrice.
# Méthode adaptée pour résoudre les problèmes liés aux colonnes remplies de 0.
cosinus.vm <- function(v,m) {
  # On on met tous nos valeurs de NA a 0, sinon on va avoir des problemes de calculs avec des matrices
  m[is.na(m)] <- 0
  v[is.na(v)] <- 0 ;
  # On calcule le cosinus entre le vecteur V et les colonnes de la matrice m en utilisant la formule vu
  denom <- sqrt(colSums(m^2)) * sqrt(sum(v^2))
  denom[denom == 0] <- 1
  res <- (v %*% m) / denom

  return(res)
}

# Retourne les indices des n-premières valeurs du vecteur (ordre décroissant).
max.nindex <- function(m, n=5) {
    i <- order(m, decreasing=TRUE)
    return(i[1:n])
}

# Prédiction d'une valeur dans la matrice de références (approche item-item).
predict.value <- function(refs, user, item)
{
    # Moyenne de chaque item
    items.avg <- colMeans(refs, na.rm=T)

    # Similarité entre l'item étudié et les autres items
    items.similarity <- cosinus.vm(refs[,item], refs)

    # Sélection des 20 premiers voisins en fonction de leur similarité
    # (la distance n'était pas pertinente ici, vu le nombre de 0 dans la matrice
    # de références)
    n.voisins <- 20 + 1
    most.similar <- max.nindex(items.similarity, n.voisins)[-item]
```

```r
    # Calcul du facteur de correction (inverse de la somme des similarités).
    # Prise en compte du cas où cette somme est nulle.
    correction.factor <- 1 / sum(items.similarity[most.similar])
    if (correction.factor == Inf)
    {
        correction.factor <- 1
    }

    # Prédiction de la valeur
    value <- items.avg[item] + correction.factor *
             sum(sapply(most.similar, function(x)
                items.similarity[x] * (refs[user, x] - items.avg[x])), na.rm=T)

    return(value)
}

# Calcul de la RMSE entre une matrice de résultats et une matrice cible
rmse <- function(results, target)
{
    res <- sqrt(sum(abs(results^2 - target^2)) / dim(target)[1]^2)

    return(res)
}
```

Creation of the test database:

```r
# Proportion de références de test (ici : 5%, les calculs sont vraiment longs
# pour 10%)
cross.validation.factor <- 0.05

# Nombre d'articles dans la base
nb.articles <- dim(m)[1]

# Sélection aléatoire des indices des références de la base de test
test.refs.row.indices <- sample(1:nb.articles, round(cross.validation.factor * nb.articles))
test.refs.col.indices <- sample(1:nb.articles, round(cross.validation.factor * nb.articles))

# Séparation de la base d'entraînement et de la base de test
m.test <- m[test.refs.row.indices, test.refs.col.indices]
m.training <- m
m.training[test.refs.row.indices, test.refs.col.indices] <- NA
```

Prediction of the values from the test database (the results matrix won't be desplayed, since it is very large):

```r
results <-sapply(test.refs.row.indices, function(x)
            sapply(test.refs.col.indices, function(y)
                predict.value(as.matrix(m.training), x, y)))
```

Computation of the RMSE of these predictions:

```r
cross.validation.rmse <- rmse(as.matrix(results), as.matrix(m.test))
```

```
## [1] 0.04741334
```

It is hard to evaluate the performance of the item-item approach in this situation, since the references matrix has a very low number of 1 values, compare to zeros.

Interpretation

Judging from the recommendation of all our possible scenarios. We can safely assume that all of our recommendations are admissible. Admissible being that they are related to the original article in question.

We can see that the article 155792 is one of the few that figures in all our recommendations. It is recommended 3rd by both our item-item approaches and is recommended by the two domains of the PageRanks (global and local) approaches. This articles is about logics and models of real time, while the original article is about symbolic model checking for real-time systems. This is a very good recommendation.

It is also a good time to mention the differences observe between the Local PageRanks and the Global PageRanks. The difference being that the computation of the scores for the local versions are done on only the domain specifically whilst the global version is done on all the articles of the matrix. Intuitively, it should yield similar results. Although in reality, it does not. We can see that the recommandations based on S and S' are almost exactly the same while the scores on the global version of S and S' are very different. This is due to the fact that the local version doesn't take into account the references of article that aren't necessarily in the domain. This means that the local version of PageRanks scores on the popularity of the article in the domain only, while the global version scores on the popularity versus all of the article in the database but we only look at the articles in the domain for recommendation.

Now, if we look at our top recommendation of our S' domain of the global, we can see that the recommendation is a little bit different. The article recommended is about minimal state graph generation. It is recommending this article because while not being popular in the domain, it is a very general topic and very useful article in all sorts of disciplines thus demonstrating an advantage of the global pagerank versus the local pagerank. While not being closely related to real-time systems, it might be relevant through the mathematical principles found in the article itself and may be a very good recommendation based on what the reader is actually trying to accomplish.

On the other hand, The top recommendations of our item-item approaches are still about real-time systems while being part of our domains S and by extent S'

Conclusion

Judging from our results, we can conclude that both methods yield quite powerful results. Though they are very different in nature..

Our item-item approach gives recommendations that are very close in terms of euclidian distance. They account for the references between the articles and is not subject to a restricted domain. They look at the reference matrix in its whole for articles that are close to each other and calculates the ones that are closer to each other. Judging from our results, we can safely say that this method can be used for recommendations purposes.

the Page Rank approach gives a much more personnalized approach. the page rank algorithm is inherently more axed toward the relations between the articles thus making it more personalized then the item-item approach. One of the key factors of the Page Rank is the establishment of the domain. In our experiment, we used S and S' as the domains for our algoriths. Both of these domains yielded great results (Results that are very similar to the item-item approach), but S' yielded more interesting results. The top recommendation for S' isn't closely related to the original article itself, but it explains one particular part of the article in better detail. This makes it a better recommendation by nature then most recommendations given by S and the item-item approaches.

Although we concluded that we had good recommendations, it is ultimately depending on the user. What we consider a good recommendation might be considered a bad recommendation by a user who is looking into writing his research paper while our recommendation is a perfect recommendation for someone who is looking for some light reading. It depends on what the user needs.