HomeWork-3

Viveksinh Solanki

3/31/2020

```
### Extract top 10 and bottom 10 pairs by values ###

getTopOrBottom10 = function(m, top=TRUE){
  # Ranking pairs by distance/similarity values
  if(top==TRUE){
    o <- order(m, decreasing = TRUE)[1:10]
  }else{
    o <- order(m)[1:10]
  }
  pos <- arrayInd(o, dim(m), useNames = TRUE)

  # returns top/bottom values, if you want to return top/bottom indices use [2]
  # instead of [1]
  output_values <- list(values = m[o], position = pos)[1]

  return(output_values)
}

### Extract top 10 and bottom 10 pairs by values (removing similarities with itself)
getTopOrBottom10_removing100 = function(m, top=TRUE){
  # Ranking pairs by distance/similarity values
  if(top==TRUE){
    o <- order(m, decreasing = TRUE)[101:110]
  }else{
    o <- order(m)[101:110]
  }
  pos <- arrayInd(o, dim(m), useNames = TRUE)

  # returns top/bottom values, if you want to return top/bottom indices use [2]
  # instead of [1]
  output_values <- list(values = m[o], position = pos)[1]

  return(output_values)
}

## Read data into dataframes
sec_df <- read.csv('securities.csv')
fund_df <- read.csv('fundamentals.csv')

# To view files as table
#View(sec_df)
#View(fund_df)
```

```r
# Subset for year 2013
fund_df_year_2013 <- subset(fund_df, fund_df$For.Year == '2013')
#View(fund_df_year_2013)

# Remove missing values
fund_df_year_2013_processed <- na.omit(fund_df_year_2013)
#View(fund_df_year_2013_processed)

# Subset of 100 tickers
fund_df_100_tickers <- fund_df_year_2013_processed[sample(nrow(fund_df_year_2013_processed),
                                                          100), ]
#View((fund_df_100_tickers))

# Subset 10 quantitative columns
col_names <- c('After.Tax.ROE', 'Cash.Ratio', 'Current.Ratio', 'Operating.Margin',
               'Pre.Tax.Margin', 'Pre.Tax.ROE', 'Profit.Margin', 'Quick.Ratio',
               'Total.Assets', 'Total.Liabilities')
fund_df_final_subset <- fund_df_100_tickers[col_names]
#View(fund_df_final_subset)

# Normalized subset
fund_df_final_subset_scaled <- scale(fund_df_final_subset)
#View(fund_df_final_subset_scaled)


### Lp-norm calculation ###

# Generalized Lp-norm function
lp_norm = function(x, y, p){
  return(sum((abs(x-y))^p)^(1/p))
}

## a) lp-norm: p=1
lp_norm_1_matrix <- matrix(, nrow = 100, ncol = 100)
for(i in 1:100){
  for(j in i:100){
    if(i!=j){
      lp_norm_1_matrix[i,j] <- lp_norm(fund_df_final_subset_scaled[i, ],
                                       fund_df_final_subset_scaled[j, ],
                                       1)
    }
  }
}

# Top 10 values for lp-norm where p=1
getTopOrBottom10(lp_norm_1_matrix)
```

```
## $values
##  [1] 36.70651 36.41936 35.70432 34.95868 34.01230 31.55456 30.92400
##  [8] 30.68233 30.26950 30.19575
```

```r
# Bottom 10 values for lp-norm where p=1
getTopOrBottom10(lp_norm_1_matrix, top = FALSE)
```

```
## $values
##  [1] 0.8606835 1.0198887 1.0269082 1.1065634 1.1163382 1.1450442 1.1554901
##  [8] 1.1660118 1.1886522 1.2117923
```

```
## b) lp-norm: p=2
lp_norm_2_matrix <- matrix(, nrow = 100, ncol = 100)
for(i in 1:100){
  for(j in i:100){
    if(i!=j){
      lp_norm_2_matrix[i,j] <- lp_norm(fund_df_final_subset_scaled[i, ],
                                       fund_df_final_subset_scaled[j, ],
                                       2)
    }
  }
}

# Top 10 values for lp-norm where p=2
getTopOrBottom10(lp_norm_2_matrix)
```

```
## $values
##  [1] 14.45117 13.81084 13.76067 12.98632 12.97345 12.81993 12.64782
##  [8] 12.64707 12.33425 12.31073
```

```
# Bottom 10 values for lp-norm where p=2
getTopOrBottom10(lp_norm_2_matrix, top = FALSE)
```

```
## $values
##  [1] 0.3948290 0.4213414 0.4289861 0.4406950 0.4466823 0.4491017 0.4533223
##  [8] 0.4653075 0.4723676 0.4763291
```

```
## c) lp-norm: p=3
lp_norm_3_matrix <- matrix(, nrow = 100, ncol = 100)
for(i in 1:100){
  for(j in i:100){
    if(i!=j){
      lp_norm_3_matrix[i,j] <- lp_norm(fund_df_final_subset_scaled[i, ],
                                       fund_df_final_subset_scaled[j, ],
                                       3)
    }
  }
}

# Top 10 values for lp-norm where p=3
getTopOrBottom10(lp_norm_3_matrix)
```

```
## $values
##  [1] 10.916919 10.445923 10.230647 10.111325 10.034654  9.899179  9.859255
##  [8]  9.742272  9.694196  9.693507
```

```
# Bottom 10 values for lp-norm where p=3
getTopOrBottom10(lp_norm_3_matrix, top = FALSE)
```

```
## $values
##  [1] 0.3137163 0.3137945 0.3330268 0.3382278 0.3410447 0.3440702 0.3449083
##  [8] 0.3474838 0.3641940 0.3783831
```

```r
## d) lp-norm: p=10
lp_norm_10_matrix <- matrix(, nrow = 100, ncol = 100)
for(i in 1:100){
  for(j in i:100){
    if(i!=j){
      lp_norm_10_matrix[i,j] <- lp_norm(fund_df_final_subset_scaled[i, ],
                                        fund_df_final_subset_scaled[j, ],
                                        10)
    }
  }
}

# Top 10 values for lp-norm where p=10
getTopOrBottom10(lp_norm_10_matrix)
```

```
## $values
##  [1] 7.745126 7.692788 7.692788 7.671205 7.668734 7.661659 7.653726
##  [8] 7.638804 7.635770 7.615981
```

```r
# Bottom 10 values for lp-norm where p=10
getTopOrBottom10(lp_norm_10_matrix, top = FALSE)
```

```
## $values
##  [1] 0.2275889 0.2389790 0.2414218 0.2542297 0.2585442 0.2591514 0.2717188
##  [8] 0.2780775 0.2863341 0.3078431
```

```r
## e) Minkovski function - taking p=2 (square root)

# Variable importance based on random forest
install.packages('party')
```

```
## Installing package into 'C:/Users/Dell/Documents/R/win-library/3.5'
## (as 'lib' is unspecified)
```

```
## Warning: package 'party' is in use and will not be installed
```

```r
library(party)

# Taking "profit margin" as target variable
cf1 <- cforest(Profit.Margin ~ . , data= data.frame(fund_df_final_subset_scaled),
               control=cforest_unbiased(mtry=2,ntree=50))
weights <- varimp(cf1)

# initialize default weight vec to all values 1
weights_vec <- c(1,1,1,1,1,1,1,1,1,1)

# add random forest weights to weight vec
for(i in 1:9){
  if(i>=7){
    weights_vec[i+1] <- weights[[i]]
  }else{
    weights_vec[i] <- weights[[i]]
  }
}

# Generalized minkovski function
```

```r
minkovski_dist = function(x, y, p){
  return(sum(weights_vec * (abs(x-y))^p)^(1/p))
}

minkovski_dist_matrix <- matrix(, nrow = 100, ncol = 100)
for(i in 1:100){
  for(j in i:100){
    if(i!=j){
      minkovski_dist_matrix[i,j] <- minkovski_dist(fund_df_final_subset_scaled[i, ],
                                                   fund_df_final_subset_scaled[j, ],
                                                   2)
    }
  }
}

#View(minkovski_dist_matrix)

# Top 10 values for minkovski where p=2
getTopOrBottom10(minkovski_dist_matrix)
```

```
## $values
##  [1] 7.133782 7.118060 7.076450 6.995679 6.841675 6.821379 6.756063
##  [8] 6.748253 6.726298 6.720327
```

```r
# Bottom 10 values for minkovki where p=2
getTopOrBottom10(minkovski_dist_matrix, top = FALSE)
```

```
## $values
##  [1] 0.01853580 0.03759598 0.06575722 0.06855356 0.07099290 0.07237877
##  [7] 0.07405943 0.08040957 0.08237891 0.08391694
```

```r
## f) Match based similarity
match_based_sim = function(x, y, p){
  final_sum = 0
  for(i in 1:10){
    final_sum = final_sum + ((1 - (abs(x[i]-y[i]))/2))^p
  }
  return((final_sum)^(1/p))
}

# taking p=2
match_based_sim_matrix <- matrix(, nrow = 100, ncol = 100)
for(i in 1:100){
  for(j in i:100){
    if(i!=j){
      match_based_sim_matrix[i,j] <- match_based_sim(fund_df_final_subset_scaled[i, ],
                                                   fund_df_final_subset_scaled[j, ],
                                                   2)
    }
  }
}

#View(match_based_sim_matrix)

# Top 10 values for match based similarity where p=2
```

```
getTopOrBottom10(match_based_sim_matrix)
```

```
## $values
##  [1] 5.050005 4.845273 4.767198 4.573800 4.513058 4.482688 4.475052
##  [8] 4.448747 4.436976 4.427854
```

```
# Bottom 10 values for match based similarity where p=2
getTopOrBottom10(match_based_sim_matrix, top = FALSE)
```

```
## $values
##  [1] 1.035152 1.062211 1.078117 1.108161 1.166112 1.203349 1.215196
##  [8] 1.219247 1.241119 1.261965
```

```
## g) Mahalanobis distance

install.packages('StatMatch')
```

```
## Installing package into 'C:/Users/Dell/Documents/R/win-library/3.5'
## (as 'lib' is unspecified)
```

```
## Warning: package 'StatMatch' is in use and will not be installed
```

```
library(StatMatch)
mahalanobisDist <- mahalanobis.dist(fund_df_final_subset_scaled)
#View(mahalanobisDist)

# Top 10 values for mahalanobis
getTopOrBottom10(mahalanobisDist)
```

```
## $values
##  [1] 12.70762 12.70762 12.08925 12.08925 11.81589 11.81589 11.79128
##  [8] 11.79128 11.73810 11.73810
```

```
# Bottom 10 values for mahalanobis
# removing bottom 100 values, because they are comparision
# of each record with itself
getTopOrBottom10_removing100(mahalanobisDist, top=FALSE)
```

```
## $values
##  [1] 0.5808778 0.5808778 0.6241160 0.6241160 0.6509340 0.6509340 0.6571258
##  [8] 0.6571258 0.6886095 0.6886095
```

```
# create subset with categorical data as well
combined_df_subset <- merge(x=fund_df_100_tickers, y=sec_df, by='Ticker.Symbol')
#View(combined_df_subset)

# subset only categorical columns
cat_col_names <- c('GICS.Sector', 'GICS.Sub.Industry')
combined_df_final_subset <- combined_df_subset[cat_col_names]
#View(combined_df_final_subset)

## h) Overlap measure
overlap_sims <- matrix(, nrow = 100, ncol = 100)
for(i in 1:100){
  for(j in i:100){
    if(i!=j){
      overlap_sims[i,j] <- sum(match(combined_df_final_subset[i, ],
                                     combined_df_final_subset[j, ], nomatch=0)>0)
```

```
    }
  }
}

#View(overlap_sims)

# Top 10 values for overlap measure
getTopOrBottom10(overlap_sims)
```

```
## $values
##  [1] 2 2 2 2 2 2 2 2 2 2
```

```
# Bottom 10 values for overlap measure
getTopOrBottom10(overlap_sims, top = FALSE)
```

```
## $values
##  [1] 0 0 0 0 0 0 0 0 0 0
```

```
## i) Inverse frequency
install.packages('nomclust')
```

```
## Installing package into 'C:/Users/Dell/Documents/R/win-library/3.5'
## (as 'lib' is unspecified)
```

```
## Warning: package 'nomclust' is in use and will not be installed
```

```
library('nomclust')
inverse_freq_measure <- iof(combined_df_final_subset)
#View(inverse_freq_measure)

# Top 10 values for Inverse frequency
getTopOrBottom10(inverse_freq_measure)
```

```
## $values
##  [1] 4.315924 4.315924 4.315924 4.315924 4.315924 4.315924 4.315924
##  [8] 4.315924 4.315924 4.315924
```

```
# Bottom 10 values for Inverse frequency
# removing bottom 100 values, because they are comparision
# of each record with itself
getTopOrBottom10_removing100(inverse_freq_measure, top=FALSE)
```

```
## $values
##  [1] 0 0 0 0 0 0 0 0 0 0
```

```
## j) Goodall measure
goodall_measure <- good1(combined_df_final_subset)
#View(goodall_measure)

# Top 10 values for Goodall measure
getTopOrBottom10(goodall_measure)
```

```
## $values
##  [1] 1 1 1 1 1 1 1 1 1 1
```

```
# Bottom 10 values for Goodall measure
# removing bottom 100 values, because they are comparision
# of each record with itself
```

```r
getTopOrBottom10_removing100(goodall_measure, top=FALSE)
```

```
## $values
##  [1] 0.01160 0.01160 0.01160 0.01160 0.01160 0.01160 0.01585 0.01585
##  [9] 0.01585 0.01585
```

```r
# Overall similarity on mixed type data
## k) Unnormalized
overall_sims_unnorm <- matrix(, nrow = 100, ncol = 100)
lambda <- 0.7
for(i in 1:100){
  for(j in i:100){
    if(i!=j){
      num_sim <- minkovski_dist_matrix[i,j]
      cat_sim <- inverse_freq_measure[i,j]
      overall_sims_unnorm[i,j] <- lambda * num_sim + (1-lambda) * cat_sim
    }
  }
}

#View(overall_sims_unnorm)

# Top 10 values for Overall similarity unnormalized
getTopOrBottom10(overall_sims_unnorm)
```

```
## $values
##  [1] 5.533057 5.423087 5.411899 5.366548 5.358174 5.347388 5.182054
##  [8] 5.092859 5.049631 5.048816
```

```r
# Bottom 10 values for Overall similarity unnormalized
getTopOrBottom10(overall_sims_unnorm, top = FALSE)
```

```
## $values
##  [1] 0.07778648 0.09671419 0.10164009 0.10498981 0.10586835 0.11738194
##  [7] 0.12574102 0.12671573 0.12979484 0.13208269
```

```r
## l) Normalized
overall_sims_norm <- matrix(, nrow = 100, ncol = 100)
lambda <- 0.7
sigma_num <- 10 #number of numeric features
sigma_cat <- 2 #number of categrical features
for(i in 1:100){
  for(j in i:100){
    if(i!=j){
      num_sim <- minkovski_dist_matrix[i,j]
      cat_sim <- inverse_freq_measure[i,j]
      overall_sims_norm[i,j] <- lambda * (num_sim/sigma_num)
                                + (1-lambda) * (cat_sim/sigma_cat)
    }
  }
}

#View(overall_sims_norm)

# Top 10 values for Overall similarity normalized
getTopOrBottom10(overall_sims_norm)
```

```
## $values
##  [1] 0.4993648 0.4982642 0.4953515 0.4896976 0.4789172 0.4774965 0.4729244
##  [8] 0.4723777 0.4708408 0.4704229
```

```
# Bottom 10 values for Overall similarity normalized
getTopOrBottom10(overall_sims_norm, top = FALSE)
```

```
## $values
##  [1] 0.001297506 0.002631718 0.004603006 0.004798749 0.004969503
##  [6] 0.005066514 0.005184160 0.005628670 0.005766524 0.005874186
```