Obesity susceptibility genes in a Spanish population using sequencing data

Isaac De la Hoz

Introduction

Obesity

Obesity is defined as an increase in fat mass that is sufficient to adversely affect health.According World Health Organization, people with a body mass index (BMI; weight in kg/height in m^2) higher than 30 kg/m^2 are considered obese. Nowadays, obesity is considered as a worldwide epidemic associated with increased morbidity and mortality that imposes an enormous burden on individual and public health (Xu and Tong 2011). In the Europe population, for instance, 10%-20% of people are classified as obese (Klaauw and Farooqi 2015).

Obesity and genetics

Around 40-70% of inter-individual variability in BMI, commonly used to assess obesity, has been attributed to genetic factors(Xu and Tong 2011). The evidence for genetic contributions to body weight comes from family, twin, and adoption studies. Other studies cumulatively demonstrate that the heritability (fraction of the total phenotypic variance of a quantitative trait attributable to genes in a specified environment) of BMI is between 0.71 and 0.86 (Klaauw and Farooqi 2015).

Objectives

In order to expand the catalog of BMI susceptibility SNPs we perform an study on whole exome sequence data from 16 different obese individuals.

Methodology

From already aligned data, a pipeline which include variant calling, variant annotation and statistical analysis was performed.

Variant Calling

In order to find the best way to obtain variant from the alignment files, two different variant callers were proved and compared. One of them was selected for being used in this analysis.

R package: VariantTools

VariantTools is R package which allows to perform the variant calling using R. The following code was the used to perform the variant calling:

First of all, the libraries needed were loaded

```
library(GenomicAlignments)
library(VariantAnnotation)
library(Rsamtools)
library(VariantTools)
library(GenomicRanges)
library(BiocParallel)
```

```
library(BSgenome.Hsapiens.UCSC.hg38)
library(gmapR)
```

Once libraries were loaded, the Gmap genome of human Hg38 version (the human genome version used to perform the alignment) was created:

```
#Gmap genome object creation. The human genome is indexed
## IMPORTANT: the following lines have to be executed 1 time,
## the needed to create the dependency.
setwd("/scratch")
chrs <- standardChromosomes(Hsapiens)
hs <- getSeq(Hsapiens, chrs)
##genome seqlevels style correction so that it is equal to bam files' seqlevels
seqlevels(hs)[25]<-"chrMT"
names(hs)[25]<-"chrMT"
##Gmap genome creation
gmapGenomePath <- file.path(getwd(), "HSG")
gmapGenomeDirectory <- GmapGenomeDirectory(gmapGenomePath, create = TRUE)
gmapGenome <- GmapGenome(genome=hs, directory=gmapGenomeDirectory,
                         name="hg38", create=TRUE, k = 14L)
```

This last code have to be run only one time because once run, a gmap object is created and stored in the directory selected.

The following code take charge of variant calling and genotyping. It was executed once per Bam file we had, changing each time the object `FILE` by the correspondent bam file name.

```
#data loading
##filename
FILE <- "<file name>"
bamFile <- sprintf("~/data/WES_obesity/Data/%s.bam", FILE)

#Tallies creation (VRanges object with variant information)

chrs <- standardChromosomes(Hsapiens)
hs <- getSeq(Hsapiens, chrs)
seqlevels(hs)[25]<-"chrMT"
names(hs)[25]<-"chrMT"
gmapGenomePath <- file.path("/scratch/HSG")
gmapGenomeDirectory <- GmapGenomeDirectory(gmapGenomePath)
HGmapGenome <- GmapGenome(genome=hs, directory=gmapGenomeDirectory, name="hg38")
tiles <- tileGenome(seqinfo(HGmapGenome), ntile = 100)
param <- TallyVariantsParam(HGmapGenome, which = unlist(tiles), indels = TRUE)
bpparam <- MulticoreParam(workers = 5)
tallies <- tallyVariants(bamFile, param, BPPARAM = bpparam)
mcols(tallies) <- NULL
sampleNames(tallies) <- FILE

#Calling and filtering
##Call genotypes
cov <- coverage(bamFile)
params <- CallGenotypesParam(HGmapGenome, p.error = 1/1000, which = tiles)
genotypes <- callGenotypes(tallies, cov, params, BPPARAM = bpparam)

##The default variant calling filters:
```

```
##VariantCallingFilters(read.count = 2L, p.lower = 0.2, p.error = 1/1000)
calling.filters <- VariantCallingFilters()
post.filters <- VariantPostFilters()
variants <- callVariants(genotypes, calling.filters, post.filters)

### Saving as a R data
vcf <- asVCF(sort(variants))
save(vcf, file=sprintf("~/data/WES_obesity/genotypedVariants/%s.rda", FILE))
```

The resulting files were saved as R data to be easily read by R in the later analysis. This last procedure can be run iteratively through the following code:

```
Files <- c("<vector with the name of all bam files>")

chrs <- standardChromosomes(Hsapiens)
hs <- getSeq(Hsapiens, chrs)
seqlevels(hs)[25]<-"chrMT"
names(hs)[25]<-"chrMT"
gmapGenomePath <- file.path("/scratch/HSG")
gmapGenomeDirectory <- GmapGenomeDirectory(gmapGenomePath)
HGmapGenome <- GmapGenome(genome=hs, directory=gmapGenomeDirectory, name="hg38")
tiles <- tileGenome(seqinfo(HGmapGenome), ntile = 100)
param <- TallyVariantsParam(HGmapGenome, which = unlist(tiles), indels = TRUE)
bpparam <- MulticoreParam(workers = 4)
params <- CallGenotypesParam(HGmapGenome, p.error = 1/1000, which = tiles)
calling.filters <- VariantCallingFilters()
post.filters <- VariantPostFilters()

for (i in Files){
  bamFile <- sprintf("~/data/WES_obesity/Data/%s.bam", i)
  ##Tallies creation (VRanges object with variant information)
  tallies <- tallyVariants(bamFile, param, BPPARAM = bpparam)
  print("1")
  mcols(tallies) <- NULL
  sampleNames(tallies) <- i
  cov <- coverage(bamFile)
  ##Call genotypes
  genotypes <- callGenotypes(tallies, cov, params, BPPARAM = bpparam)
  ##Call variants and filtering
  variants <- callVariants(genotypes, calling.filters, post.filters)
  ##saving vcf file
  vcf <- asVCF(sort(variants))
  save(vcf, file=sprintf("~/data/WES_obesity/genotypedVariants/%s.rda", i))
}
```

Once we have all variant from all bam files, we need to obtain the minor allele frequency of the variants in order to perform the statistical analysis. For doing that, we need to merge all VCF files in only one multi-sample VCF file. But, there is a problem here. The VCFs do not have a line for every single locus. Samples that match consensus at a positions do not have an entry for that position, so if there is a SNP in a sample at a given position, other samples could have no entry for that position, and we do not know if the other samples really math consensus there, or if they have low coverage there. Therefore, the variants can not be safely called.

GATK haplotype caller

Considering the problem with the VariantTool R package, we proved the tool HaplotypeCaller from java-based tool named GATK. This tool allow us to call variants individually on each sample using it in -ERC GVCF mode, leveraging the previously introduced reference model to produce a comprehensive record of genotype likelihoods and annotations for each site in the exome, in the form of a gVCF file. By this way, we can safely call all variants.

The following bash code was used to obtain the gVCFs.

```
# Directories
DWD="$(pwd)"
DIRBAM=$DWD/Data
DIRVCF=$DWD/VCF_HapCaller
GREF=$DWD/hg38.fa

# 1. HaplotypeCaller

##Changing from chrM to chrMT
sed 's/chrM/chrMT/g' hg38.fa
##Reference dictionary creation
gatk CreateSequenceDictionary -R hg38.fa -O hg38.dict
##Creating index file
samtools faidx hg38.fa

##HaplotypeCaller
for BAM in `ls $DIRBAM| grep "bam$"|grep -v "2F759.bam"`
do
    gatk --java-options "-Xmx4g" HaplotypeCaller -R $GREF \
    -I $BAM -O $DIRVCF/${BAM%".bam"}.raw.snps.indels.g.vcf \
    -ERC GVCF &
done
wait
```

Once we got all gVCFs, we used the gatk's tool name `ValidateVarints` in order to validate the correctness of the formatting of VCF files. In addition to standard adherence to the VCF specification, this tool performs extra strict validations to ensure that the information contained within the file is correctly encoded. These include:

- REF. correctness of the reference base(s)
- CHR_COUNTS. accuracy of AC and AN values
- IDS. tests against rsIDs when a dbSNP file is provided
- ALLELES. that all alternate alleles are present in at least one sample

```
# 2. Variant validation
for FILE in `find $DIRVCF -name "*.raw.snps.indels.g.vcf"`
do
    gatk ValidateVariants -R $GREF -V $FILE &
done
wait
```

Once validated, we combined all gVCFs in only one VCF file through the GATK's tool named `CombineGVCFs`.

```
#3. Combine GVCFs
```

```
find $DIRVCF -name "*.raw.snps.indels.g.vcf" > $DWD/input.list
gatk CombineGVCFs -R $GREF -V $DWD/input.list -O $DIRVCF/RawVariants.vcf
```

Once we got the multi-sample VCF, we used the tool `GenotypeGVCFs` in order to perform joint genotyping.

```
# 4. GVCF Genotyping

mkdir $DIRVCF/finalVCF
gatk --java-options "-Xmx4g" GenotypeGVCFs -R $GREF \
    -V $DIRVCF/RawVariants.vcf -O $DIRVCF/finalVCF/variants.vcf
```

From the last code, a jointly genotyped VCF file was obtained. The next step consisted in filtering all low quality variants.

In order to filter in the best way, first of all, the SNPs and INDELs were separated because each type of variant has different filtering parameters. For selecting SNPs and INDELs the tool `SelectVariants` was used.

```
gatk SelectVariants -V $DIRVCF/finalVCF/variants.vcf \
    -select-type SNP -O $DIRVCF/finalVCF/variants.snps.vcf &
gatk SelectVariants -V $DIRVCF/finalVCF/variants.vcf \
    -select-type INDEL -O $DIRVCF/finalVCF/variants.indels.vcf
```

Once selected, the following filtering parameters were applied:

For SNPs

- QD< 2.0
- MQ < 40.0
- FS > 60.0
- SOR > 3.0
- MQRankSum < -12.5
- ReadPosRankSum < -8.0

For INDELs

- QD < 2.0
- ReadPosRankSum < -20.0
- InbreedingCoeff < -0.8
- FS > 200.0
- SOR > 10.0

```
# 5.Filtration
##SNPs filtration
gatk VariantFiltration -V $DIRVCF/finalVCF/variants.snps.vcf \
-filter "QD < 2.0" --filter-name "QD2" \
-filter "QUAL < 30.0" --filter-name "QUAL30" \
-filter "SOR > 3.0" --filter-name "SOR3" \
-filter "FS > 60.0" --filter-name "FS60" \
-filter "MQ < 40.0" --filter-name "MQ40" \
-filter "MQRankSum < -12.5" --filter-name "MQRankSum-12.5" \
-filter "ReadPosRankSum < -8.0" --filter-name "ReadPosRankSum-8" \
-O $DIRVCF/finalVCF/variants.snps_filtered.vcf
```

```
##Indels filtration

gatk VariantFiltration -V $DIRVCF/finalVCF/variants.indels.vcf -filter "QD < 2.0" \
--filter-name "QD2" \
-filter "QUAL < 30.0" --filter-name "QUAL30" \
-filter "FS > 200.0" --filter-name "FS200" \
-filter "ReadPosRankSum < -20.0" --filter-name "ReadPosRankSum-20" \
-O $DIRVCF/finalVCF/variants.indels_filtered.vcf
```

Once filtered, SNPs and INDELs were merged in a file and unfiltered variants were selected.

```
# 6. Merging and selection
##Merging SNPs and INDELs
gatk MergeVcfs \
          -I $DIRVCF/finalVCF/variants.snps_filtered.vcf \
          -I $DIRVCF/finalVCF/variants.indels_filtered.vcf \
          -O $DIRVCF/finalVCF/variants_filtered.vcf

##Selection of unfiltered variants
gatk SelectVariants \
              -V $DIRVCF/finalVCF/variants_filtered.vcf \
              -exclude-filtered true -O $DIRVCF/finalVCF/variant_filtered.vcf

bgzip -c $DIRVCF/finalVCF/variants_filtered.vcf > $DIRVCF/finalVCF/variants_filtered.vcf.gz
tabix -f -p vcf $DIRVCF/finalVCF/variants_filtered.vcf.gz
```

Finally, all variants were stored in a table, including genomic positions, alleles, type of variants and allele frequency.

```
# 7. Variants are stored in a table
##Information about the chromosome, postion, reference, alternative
##type and allele frequency are also included in the table.

gatk VariantsToTable \
    -V $DIRVCF/finalVCF/variants_filtered.vcf \
    -F CHROM -F POS -F REF -F ALT -F TYPE -F AF \
    -O $DIRVCF/finalVCF/Variants.table
##Variants.table looks like this:
```

```
var <- read.table(file="Data/Variants.table", sep="", dec= ".", header = TRUE )
head(var)
```

```
##   CHROM    POS REF ALT  TYPE    AF
## 1  chr1  19190  GC   G INDEL 0.100
## 2  chr1  66169  TA   T INDEL 0.250
## 3  chr1  98921  AG   A INDEL 0.250
## 4  chr1 102951   C   T   SNP 0.083
## 5  chr1 132991   G   A   SNP 0.167
## 6  chr1 133129   G   A   SNP 0.313
```

Statistical analysis

From the variants data, we calculated the minor allele frequency using the R package named `vcfR` through the following code.

```
library(vcfR)

#MAF calculation
vcfR<- read.vcfR("Data/variants_filtered.vcf.gz")
maf <- maf(vcfR)
##Number of individuals which contain the variant
N_ob <- as.vector(16-maf[,2])
##MAF and number of individuals are included in Variant.table
var$MAF <- as.vector(round(maf[,4],3))
var$N_ob <- N_ob
levels(var$CHROM)[23] <- "chrM"
##A GenomicRange object is created containing the data from Variant.table
gr <- makeGRangesFromDataFrame(var, seqnames.field="CHROM",
                               start.field ="POS", end.field="POS", ignore.strand = TRUE)
mcols(gr)$MAF <- var$MAF
mcols(gr)$N_ob <- var$N_ob
mcols(gr)$TYPE <- var$TYPE
##Maf calculation is a very time consuming step, so we save the file as a ckeckpoint
##just in case something wrong occurs
save(gr,file="maf_variants.rda")
##The GenomicRange object looks like this
```

```
head(gr)
```

```
## GRanges object with 6 ranges and 2 metadata columns:
##       seqnames    ranges strand |      N_ob       MAF
##          <Rle> <IRanges>  <Rle> | <numeric> <numeric>
##   [1]     chr1     19190      * |         5       0.1
##   [2]     chr1     66169      * |         2      0.25
##   [3]     chr1     98921      * |         4      0.25
##   [4]     chr1    102951      * |         6     0.083
##   [5]     chr1    132991      * |         6     0.167
##   [6]     chr1    133129      * |         8     0.312
##   -------
##   seqinfo: 25 sequences from an unspecified genome; no seqlengths
```

Once MAFs were calculated, we included the control allele frequencies for these variant from 1000 genomes phase 3 database for human genome version GRCh38 `MafDb.1Kgenomes.phase3.GRCh38`.

```
library(MafDb.1Kgenomes.phase3.GRCh38)
library(GenomicScores)

mafdb <- MafDb.1Kgenomes.phase3.GRCh38
populations(mafdb)
#The allele frequecies from european population are selected
obmaf <- gscores(mafdb, gr, pop = "EUR_AF")
#Another checkpoint is included
save(obmaf,file="obesity_maf.rda")
#This new object looks like this
head(obmaf)
```

```
## GRanges object with 6 ranges and 3 metadata columns:
##       seqnames    ranges strand |       MAF    EUR_AF      N_ob
##          <Rle> <IRanges>  <Rle> | <numeric> <numeric> <numeric>
##   [1]        1    494515      * |     0.083      0.02         6
```

```
##   [2]        1    591452      * |     0.071      0.02         7
##   [3]        1    591460      * |     0.071      0.03         7
##   [4]        1    598934      * |     0.167         0         6
##   [5]        1    633071      * |         0      0.16         1
##   [6]        1    727242      * |     0.062      0.11         8
##   -------
##   seqinfo: 25 sequences from GRCh38 genome; no seqlengths
```

Once the control frequencies were added, the following step was to calculate the p-value through fisher test using the following code.

```
#Pvalue calculation

library(parallel)

#All variants with EUR_AF=NA are excluded
obmaf <- obmaf[!is.na(obmaf$EUR_AF),]
#The number of individual where the control AF were calculated were added
obmaf$N_eur <- 669
#Function to calculate p-Value
testMAF <- function(i, dat) {
  x <- dat[i, ]
  ob <- round(x[1,3]*x[1,1])
  ob2 <- x[1,3] - ob
  eur <- round(x[1,4]*x[1,2])
  eur2 <- x[1,4] - eur
  tt <- matrix(c(ob2, ob, eur2, eur),
               byrow = TRUE,
               ncol=2)
  ans <- try(fisher.test(tt), TRUE)
  if (inherits(ans, "try-error"))
    out <- NA
  else
    out <- ans$p.value
  out
}
#Function execution, Very high time-consuming
#We parallelise in order to reduce the time-consuming
obmaf$pvalue <- mclapply(1:length(obmaf), testMAF,
                         dat=mcols(obmaf)[,c("MAF", "EUR_AF",
                                             "N_ob","N_eur")],
                         mc.cores=15)
#THe obmaf object looks like this once p-values are calculated
head(obmaf)
```

```
## GRanges object with 6 ranges and 4 metadata columns:
##       seqnames    ranges strand |       MAF    EUR_AF      N_ob
##          <Rle> <IRanges>  <Rle> | <numeric> <numeric> <numeric>
##   [1]        1    494515      * |     0.083      0.02         6
##   [2]        1    591452      * |     0.071      0.02         7
##   [3]        1    591460      * |     0.071      0.03         7
##   [4]        1    598934      * |     0.167         0         6
##   [5]        1    633071      * |         0      0.16         1
##   [6]        1    727242      * |     0.062      0.11         8
##                     pvalue
```

```
##                     <list>
##   [1]                    1
##   [2]                    1
##   [3]                    1
##   [4] 0.00888888888888889
##   [5]                    1
##   [6]                    1
##   -------
##   seqinfo: 25 sequences from GRCh38 genome; no seqlengths
```

Once we obtained the p-values, we calculated the adjusted p-value through the false discovery rate method (FDR). We used the following code.

```
#P-adjusted calcultion through fdr method
padj <- p.adjust(obmaf$pvalue, method = "fdr")
obmaf$Padj.fdr <- padj
```

The variants with a false discovery rate lower than 5% and 1% (p-adjusted lower than 0.05 and 0.01) were selected and saved in different objects (obmaf.05 and obmaf.01). We used the following code to do that.

```
obmaf$padj <- padj<=0.05
obmaf$padj2 <- padj<=0.01
obmaf.05<-obmaf[obmaf$padj==TRUE]
obmaf.01<-obmaf[obmaf$padj2==TRUE]
obmaf.05$padj<- NULL; obmaf.05$padj2 <- NULL
obmaf.01$padj2<- NULL; obmaf.01$padj <- NULL
```

Finally, we added the genes where this significant variant are located. We used the following code.

```
#Gene annotation

library(Homo.sapiens)
library(AnnotationDbi)
library(org.Hs.eg.db)

#Creation of Homo.sapiens object for hg38 genome

library(TxDb.Hsapiens.UCSC.hg38.knownGene)
library(OrganismDbi)
gd <- list(join1 = c(GO.db="GOID", org.Hs.eg.db="GO"),
           join2 = c(org.Hs.eg.db = "ENTREZID",
                     TxDb.Hsapiens.UCSC.hg38.knownGene = "GENEID"))
destination <- tempfile()
dir.create(destination)
makeOrganismPackage(pkgname = "Homo.sapiens.hg38", graphData = gd,
                    organism = "Homo sapiens", version = "1.0.0",
                    maintainer = "Maintainer<maintainer@email>",
                    author = "Author Name", destDir = destination,
                    license = "Artistic-2.0")
install.packages(sprintf("%s/Homo.sapiens.hg38",destination), repos = NULL, type="source")
library(Homo.sapiens.hg38)

#Extracting gene names and their genomic positions
geneRanges <-
  function(db, column="ENTREZID")
  {
```

```
    g <- genes(db, columns=column)
    col <- mcols(g)[[column]]
    genes <- granges(g)[rep(seq_along(g), elementNROWS(col))]
    mcols(genes)[[column]] <- as.character(unlist(col))
    genes
  }

splitColumnByOverlap <-
  function(query, subject, column="ENTREZID", ...)
  {
    olaps <- findOverlaps(query, subject, ...)
    f1 <- factor(subjectHits(olaps),
                 levels=seq_len(subjectLength(olaps)))
    splitAsList(mcols(query)[[column]][queryHits(olaps)], f1)
  }
gns <- geneRanges(Homo.sapiens.hg38, column="SYMBOL")

#Merging genes positions with SNPs' genomics positions
seqlevelsStyle(obmaf.01)<-seqlevelsStyle(gns);
seqlevelsStyle(obmaf.05)<-seqlevelsStyle(gns)
genome(obmaf.01)<-genome(gns); genome(obmaf.05)<-genome(gns)
###Obmaf.01
symInCnv = splitColumnByOverlap(gns, obmaf.01, "SYMBOL")
geneNames.01<-as.vector(unstrsplit(symInCnv, sep=", "))
obmaf.01$GENES <- geneNames.01
###Obmaf.05
symInCnv = splitColumnByOverlap(gns, obmaf.05, "SYMBOL")
geneNames.05<-as.vector(unstrsplit(symInCnv, sep=", "))
obmaf.05$GENES <- geneNames.05

#Saving results
save(obmaf.01, file="obmaf.01.rda")
save(obmaf.05, file="obmaf.05.rda")
```

The resulting data look like this:

```
#Variants with a fdr lower than 5%
head(obmaf.05,4)
```

```
## GRanges object with 4 ranges and 7 metadata columns:
##       seqnames    ranges strand |       MAF    EUR_AF      N_ob     N_eur
##          <Rle> <IRanges>  <Rle> | <numeric> <numeric> <numeric> <numeric>
##   [1]     chr1   3872630      * |       0.5         0        15       669
##   [2]     chr1  10451273      * |       0.4         0        10       669
##   [3]     chr1  13225068      * |       0.5      0.07        15       669
##   [4]     chr1  13230089      * |     0.375         0         4       669
##                     pvalue            Padj.fdr           GENES
##                     <list>           <numeric>     <character>
##   [1] 5.64239821139616e-15 9.81670316889798e-10            DFFB
##   [2] 2.39219018226442e-08 9.23921857621268e-05 CENPS-CORT, CORT
##   [3] 4.39269893391707e-06  0.00804614945765076        PRAMEF18
##   [4] 2.65336446614307e-05   0.0382201267727719
##   -------
##   seqinfo: 25 sequences from hg38 genome; no seqlengths
```

```
#Total number of variants fdr<0.05
length(obmaf.05)
```

```
## [1] 1177
```

```
#Variants with a fdr lower than 1%
head(obmaf.01,4)
```

```
## GRanges object with 4 ranges and 7 metadata columns:
##       seqnames    ranges strand |       MAF    EUR_AF      N_ob     N_eur
##          <Rle> <IRanges>  <Rle> | <numeric> <numeric> <numeric> <numeric>
##   [1]     chr1   3872630      * |       0.5         0        15       669
##   [2]     chr1  10451273      * |       0.4         0        10       669
##   [3]     chr1  13225068      * |       0.5      0.07        15       669
##   [4]     chr1  15192430      * |     0.455     0.002        11       669
##                     pvalue             Padj.fdr            GENES
##                     <list>            <numeric>      <character>
##   [1] 5.64239821139616e-15 9.81670316889798e-10             DFFB
##   [2] 2.39219018226442e-08 9.23921857621268e-05 CENPS-CORT, CORT
##   [3] 4.39269893391707e-06  0.00804614945765076         PRAMEF18
##   [4] 2.30463289454755e-09 1.49967125518657e-05           TMEM51
##   -------
##   seqinfo: 25 sequences from hg38 genome; no seqlengths
```

```
#Total number of variants fdr<0.01
length(obmaf.01)
```

```
## [1] 953
```

Enrichment analysis via hypergeometric test

In order to know if the results have biological sense, we performed an enrichment analysis. By this way, we knew what molecular function the genes, where the significant SNPs are located, have.

```
#Enrichment analysis

library(GO.db)
library(EnsDb.Hsapiens.v86)
library(org.Hs.egENSEMBL2EG)
library(GOstats)
```

Once the libraries needed were loaded, first of all, we annotated to genes all SNPs obtained from variant calling (obmaf) using the following code.

```
#Genes Obmaf (maf data used to perform the fdr analysis)
geneRanges <-
  function(db, column="ENTREZID")
  {
    g <- genes(db, columns=column)
    col <- mcols(g)[[column]]
    genes <- granges(g)[rep(seq_along(g), elementNROWS(col))]
    mcols(genes)[[column]] <- as.character(unlist(col))
    genes
  }

splitColumnByOverlap <-
  function(query, subject, column="ENTREZID", ...)
```

```
  {
    olaps <- findOverlaps(query, subject, ...)
    f1 <- factor(subjectHits(olaps),
                 levels=seq_len(subjectLength(olaps)))
    splitAsList(mcols(query)[[column]][queryHits(olaps)], f1)
  }
gns <- geneRanges(Homo.sapiens.hg38, column="SYMBOL")
seqlevelsStyle(obmaf)<-seqlevelsStyle(gns)
genome(obmaf)<-genome(gns)
symInCnv = splitColumnByOverlap(gns, obmaf, "SYMBOL")
geneNames<-as.vector(unstrsplit(symInCnv, sep=", "))
obmaf$GENES <- geneNames
obmaf<-obmaf[obmaf$GENES!=""]
```

Once we have genes annotated, the following step was add the ENSEMBL ids to all genes.

```
#Annotation of EMSEMBL genes IDs
edb <- EnsDb.Hsapiens.v86
length(obmaf.01)-length(obmaf.01[obmaf.01$GENES==""])
obmaf.01<-obmaf.01[obmaf.01$GENES!=""]
#Coincidents genes are splitted
##obmaf01
t <- strsplit(unlist(obmaf.01$GENES),"[,]")
t2<- grep("*",unlist(t), value = TRUE)
t2<-gsub(" ","", t2)
##obmaf
k <- strsplit(unlist(obmaf$GENES),"[,]")
k2<- grep("*",unlist(k), value = TRUE)
k2<-gsub(" ","", k2)
EnGenes <- genes(edb, filter= GeneNameFilter(as.vector(t2)))
EnGenesAll <- genes(edb, filter= GeneNameFilter(k2))
```

Finally, hypergeometric test was performed using Gene Ontology database.

```
#Enrichment procedure via hypergeometric test
Genes_01 <- unlist(mget(EnGenes$gene_id, envir=org.Hs.egENSEMBL2EG,ifnotfound = NA))
geneUniverse <- unlist(mget(EnGenesAll$gene_id, envir=org.Hs.egENSEMBL2EG, ifnotfound = NA))
##GO
params <- new("GOHyperGParams", geneIds=Genes_01,
              universeGeneIds=geneUniverse,
              annotation="org.Hs.eg.db", ontology="BP",
              pvalueCutoff=0.01 , conditional=FALSE,
              testDirection="over")
hgOver <- hyperGTest(params)
head(summary(hgOver))
save(hgOver, file="EnrichmentGO.rda")
```

References

Klaauw, Agatha A Van Der, and I Sadaf Farooqi. 2015. "Review The Hunger Genes : Pathways to Obesity." *Cell* 161 (1). Elsevier Inc.: 119–32. doi:10.1016/j.cell.2015.03.008.

Xu, Yuanzhong, and Qingchun Tong. 2011. "Expanding neurotransmitters in the hypothalamic neurocircuitry for energy balance regulation." *Protein & Cell* 2 (10): 800–813. doi:10.1007/s13238-011-1112-4.