# Tumor Tumor adj. Sex covariate

Annika Jorgensen

2023-01-17

# R Markdown

Title: “DEA Viral Etiology with Sex as Covariate”
Author: Annika Jorgensen
Date: 01/17/2023 Purpose: The Purpose of this file is to do a DEA with the ICGC liver cancer data on all tumor tumor adjacent data with sex added as a covariate. The purpose is to make sure that sex is accounted for sex as a potential influence of the expressed genes.

## Libraries

The first chunk of code is dedicated to installing the libraries. These libraries are to help execute the differential analysis and helps visualize the data. The code was not included for concision.

# Environment parameters

This next section of code is dedicated to the environmental parameters. Environmental parameters are a series of variables and other code that will help make the rest of the script be easier to make and run later on.

## Working Directory

A working directory is a code that iterates a file path on your computer th.t sets where the default location of any files that you read into R. Working directories work different in R files than R Markdowns. R Markdown files require directories to be defined at the end of each code chunk. Meaning from here on out you will see working directories being defined at the end of each code chunk.

```
setwd('~/Desktop/Research Projects/Liver Cancer Project')
```

## Defining Colors

This chunk defines color palette variables that are going to be used in plots later on the script. These variables are defined by conversiting BrewerCode palettes into palettes that can be used in R.

```
viralPalette <- brewer.pal(8, "Set1")
hbvColor <- viralPalette[1]
hcvColor <- viralPalette[2]
bothColor <- viralPalette[3]
neitherColor <- viralPalette[4]

sexTissuePalette <- brewer.pal(12, "Paired")
maleTumorColor <- sexTissuePalette[4]
maleAdjacentColor <- sexTissuePalette[3]
femaleTumorColor <- sexTissuePalette[6]
femaleAdjacentColor <- sexTissuePalette[5]
```

## Read in data

This code is where you read in all the data files that are going to be used in the script. The data is also converted into a variety of variables that makes the data easier to handle. The data is also cleaned up to make sure the analysis done later is accurate and precise.

```
metadata <- read.table("~/Desktop/Research Projects/Liver Cancer Project/Metadata/metada
ta_for_de.csv", row.names=1,header=TRUE, sep=",") #changing the name of the file
tumorAdjacentExp <- read.table("~/Desktop/Research Projects/Liver Cancer Project/Metadat
a/japan_all_samples_salmon_expression_counts.txt", row.names = 1, header=TRUE) #changing
the name of the file
colnames(tumorAdjacentExp) <- gsub("\\.", "-", colnames(tumorAdjacentExp)) #changing the
column names
```

#### Gene Length

This next code chunk is very similar However, it does calculate **gene length** which is the done by first defining a variable named “gene”” and then changing the data type to a data frame. You then redefine “tumorAdjacentExp” (defined above) to have the rows of the previous “tumorAdjacentExp” and then the columns of “GENEID”” that lies within “gene”.
Gene length is then defined to have “with” of genes in the rows and ‘end-start’ as a column

#### Running Identical Function

We ran the identical function to see if the inputs of the match function are of length one. The function outputted a true value therefore they are identical.

```
genes <- read.table("~/Desktop/Research Projects/Liver Cancer Project/Metadata/gencodeTr
anscripts.txt", header=TRUE, sep="\t")
genes <- data.frame(genes)
tumorAdjacentExp <- tumorAdjacentExp[rownames(tumorAdjacentExp) %in% genes$GENEID ,]
genes <- genes[match(rownames(tumorAdjacentExp), genes$GENEID),]

identical(rownames(tumorAdjacentExp),genes$GENEID)
```

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

```
# Calculating gene length, this is needed for calculating the FPKM values
genes$length <- with(genes, end - start)
```

### Low Quality

The next line shows a sample being removed due to low quality.

```
metadata<-metadata[!(metadata$ID == "RK023"),]
```

### Subsetting data

This next chunk of data is dedicated to sub-setting and organizing the data to make it easier to use going forward. Sub-setting means that the data is being organized to match a count matrix. In this specific case the count matrix is the sample ID attached to the tumors.

```
tumorAdjacentExpSubset <- tumorAdjacentExp[,colnames(tumorAdjacentExp) %in% metadata$sam
pleid]
metadataSubset <- metadata[metadata$sampleid %in% colnames(tumorAdjacentExpSubset),]
metadataSubset <- metadataSubset[match(colnames(tumorAdjacentExpSubset), metadataSubset
$sampleid),]
identical(colnames(tumorAdjacentExpSubset),metadataSubset$sampleid)
```

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

```
rownames(metadataSubset) <- metadataSubset$sampleid
```

### Tissue object

This next chunk of data is taking the meta data and subsetting it in such a way that converts a series of categorical variables into factors. This data also adds a tissue type.

```
metadataSubset$tumor <- as.numeric(grepl('tumor', metadataSubset$sampleid, ignore.case=
T))
metadataSubset$gender_tissue <- paste(metadataSubset$Gender, metadataSubset$tumor, sep
="_")
metadataSubset$gender_tissue_viral <- paste(metadataSubset$gender_tissue, metadataSubset
$Virus_infection, sep="_")
metadataSubset$library_type <- metadataSubset$strandedness
metadataSubset$library_type <- factor(metadataSubset$library_type)
metadataSubset$tumor <- factor(metadataSubset$tumor)
metadataSubset$Ta <- factor(metadataSubset$Ta)
metadataSubset$Portal_vein_invasion <- factor(metadataSubset$Portal_vein_invasion)
metadataSubset$Hepatic_vein_invasion <- factor(metadataSubset$Hepatic_vein_invasion)
metadataSubset$Bile_duct_invasion <- factor(metadataSubset$Bile_duct_invasion)
metadataSubset$Liver_fibrosisc <- factor(metadataSubset$Liver_fibrosisc)
metadataSubset$Prognosis <- factor(metadataSubset$Prognosis)
```

### Creating DGE Object

This next chunk of code creates something called a DGEList Object. This object contains a dataset that is to be analyzed later in the script. Specifically the object contains:
1. Counts– numeric matrix containing read counts
2. group– vector giving the experimental conditiona for each sample
3. genes– data frame information for the genes for which we have count data
4. remove.zeros– whether to remove rows that have 0 total count

The last line of code takes the amount of samples and places them into a table for easy read and inspection.

```
dge <- DGEList(counts=tumorAdjacentExpSubset, genes=genes)
colnames(dge) <- colnames(tumorAdjacentExpSubset)
dge$samples$sex <- metadataSubset$Gender
dge$samples$viral <- paste(metadataSubset$Virus_infection, metadataSubset$tumor, sep
="_")
dge$samples$ID <- metadataSubset$ID
dge$samples$tumor <- metadataSubset$tumor
dge$samples$gender_tissue <- metadataSubset$gender_tissue
dge$samples$gender_tissue_viral <- metadataSubset$gender_tissue_viral
dge$samples$library_type <- metadataSubset$library_type
dge$samples$edmonson_grade <- metadataSubset$Edmondson_grade
dge$samples$Ta <- metadataSubset$Ta
dge$samples$survival <- metadataSubset$Overall_survival_month
##=====================

#Inspecting the N of samples in each group

table(dge$samples$gender_tissue_viral)
```

```
## 
##  F_0_HBV  F_0_HCV F_0_NBNC  F_1_HBV  F_1_HCV F_1_NBNC M_0_both  M_0_HBV 
##        8       34        3        9       36        3        4       33 
##  M_0_HCV M_0_NBNC M_1_both  M_1_HBV  M_1_HCV M_1_NBNC 
##       59       25        4       40       71       25
```

**Calculating fpkm values**

This chunk of code takes the fpkm of all the genes in the dataset and calculates the mean. They also filter out genes that have a fpkm of 0.5 or lower.

Fpkm stands for fragments per kilo base of exon per million this term is interchangable with Rpkm (reads per kilobase of exon per million. This measure is a normalization method which allows us to compare gene expression levels by rescaling both library size and gene length.

Fpkm is calculated by multiplying the number of reads mapped to a gene by 1,000, 1,000,000 and then dividing that number by the total number of mapped reads to gene length in base pairs.

Please note that that calculation is done for RPKM which is analogous to Fpkm.

```
# =======================================
# Filtering expression data
# =======================================

# Keeping genes that have a mean FPKM of at least 0.5 in at least one of the
# groups under investigation and at least 6 reads in at least 10 samples
fpkm <- rpkm(dge, gene.length=dge$genes$length)
```

Here the fpkm is calculated from all the various tissue samples, are filtered for greater than 0.5 and put into a variable named “keep” which a cutoff point that is going to be used in limma/voom analysis

```
M_1_HBV_mean_fpkm <- apply(as.data.frame(fpkm)[(dge$samples$gender_tissue_viral=="M_1_HB
V")],1,mean,na.rm=TRUE)
M_0_HBV_mean_fpkm <- apply(as.data.frame(fpkm)[(dge$samples$gender_tissue_viral=="M_0_HB
V")],1,mean,na.rm=TRUE)
M_1_HCV_mean_fpkm <- apply(as.data.frame(fpkm)[(dge$samples$gender_tissue_viral=="M_1_HC
V")],1,mean,na.rm=TRUE)
M_0_HCV_mean_fpkm <- apply(as.data.frame(fpkm)[(dge$samples$gender_tissue_viral=="M_0_HC
V")],1,mean,na.rm=TRUE)
M_1_HBVHCV_mean_fpkm <- apply(as.data.frame(fpkm)[(dge$samples$gender_tissue_viral=="M_1
_both")],1,mean,na.rm=TRUE)
M_0_HBVHCV_mean_fpkm <- apply(as.data.frame(fpkm)[(dge$samples$gender_tissue_viral=="M_0
_both")],1,mean,na.rm=TRUE)
M_1_NBNC_mean_fpkm <- apply(as.data.frame(fpkm)[(dge$samples$gender_tissue_viral=="M_1_N
BNC")],1,mean,na.rm=TRUE)
M_0_NBNC_mean_fpkm <- apply(as.data.frame(fpkm)[(dge$samples$gender_tissue_viral=="M_0_N
BNC")],1,mean,na.rm=TRUE)

F_1_HBV_mean_fpkm <- apply(as.data.frame(fpkm)[(dge$samples$gender_tissue_viral=="F_1_HB
V")],1,mean,na.rm=TRUE)
F_0_HBV_mean_fpkm <- apply(as.data.frame(fpkm)[(dge$samples$gender_tissue_viral=="F_0_HB
V")],1,mean,na.rm=TRUE)
F_1_HCV_mean_fpkm <- apply(as.data.frame(fpkm)[(dge$samples$gender_tissue_viral=="F_1_HC
V")],1,mean,na.rm=TRUE)
F_0_HCV_mean_fpkm <- apply(as.data.frame(fpkm)[(dge$samples$gender_tissue_viral=="F_0_HC
V")],1,mean,na.rm=TRUE)
F_1_NBNC_mean_fpkm <- apply(as.data.frame(fpkm)[(dge$samples$gender_tissue_viral=="F_1_N
BNC")],1,mean,na.rm=TRUE)
F_0_NBNC_mean_fpkm <- apply(as.data.frame(fpkm)[(dge$samples$gender_tissue_viral=="F_0_N
BNC")],1,mean,na.rm=TRUE)

keep <- (M_1_HBV_mean_fpkm > 0.5 | M_0_HBV_mean_fpkm > 0.5 |
           M_1_HCV_mean_fpkm > 0.5 | M_0_HCV_mean_fpkm > 0.5 |
           M_1_HBVHCV_mean_fpkm > 0.5 | M_0_HBVHCV_mean_fpkm > 0.5 |
           M_1_NBNC_mean_fpkm > 0.5 | M_0_NBNC_mean_fpkm > 0.5 |
           F_1_HBV_mean_fpkm > 0.5 | F_0_HBV_mean_fpkm > 0.5 |
           F_1_HCV_mean_fpkm > 0.5 | F_0_HCV_mean_fpkm > 0.5 |
           F_1_NBNC_mean_fpkm > 0.5 | F_0_NBNC_mean_fpkm > 0.5)
```

**DGE object organization**

This chunk is further organizes and counts the libraries to be more tangible for later on as well as calculates the normalization factors (not normalizing the data) to use later on in the limma/voom DEG.

The normalization factors are calculated using Trimmed Mean of M-values (TMM). TMM is a between sample normalization that assumes that most genes are not differentially expressed. TMM normalizes the total RNA output among the samples, not considering gene length nor library size. TMM also considers the RNA population which makes it effective with samples that have diverse RNA repertoires.

TMM takes the library size normalized read count for each gene in each sample and calculates the log2 fold change between two samples (M-value). From there you calculate the absolute expression count (A values) which is the sum of the log2 fold change of treated sample count plus the log2 fold change of the control sample count divided by two.

M-values and A-values are double trimmed by 30% and 5% respectively. You then get the weight mean M after trimming and calculate the normalization factor.

```
dge <- dge[keep,,keep.lib.sizes=FALSE]
dge <- calcNormFactors(dge, method="TMM")
keep <- rowSums(dge$counts > 6) >= 10
dge <- dge[keep,,keep.lib.size=FALSE]
dge <- calcNormFactors(dge, method="TMM")
```

**Counting number of FPKM genes**

This code counts the number of genes that make it past the cutoff point.

```
# N of genes retained after filtering
dim(dge$genes)
```

```
## [1] 13384     7
```

# ** DEA tumor vs. tumor- adjacent

This section is doing voom/limma DEA on all tumor vs. tumor adjacent samples

```
# ============================================================
# ============================================================
# Analysis of all tumor vs. tumor-adjacent regardless of sex
# ============================================================
# ============================================================

# Creating a new design model matrix with the variable of interest and the
# library type
design <- model.matrix(~0+dge$samples$tumor+dge$samples$library_type+dge$samples$Ta+dge
$samples$sex)

#I just put design matrix in twice because I didn't remember what inputs to put in.
identical(design,design,num.eq=TRUE)
```

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

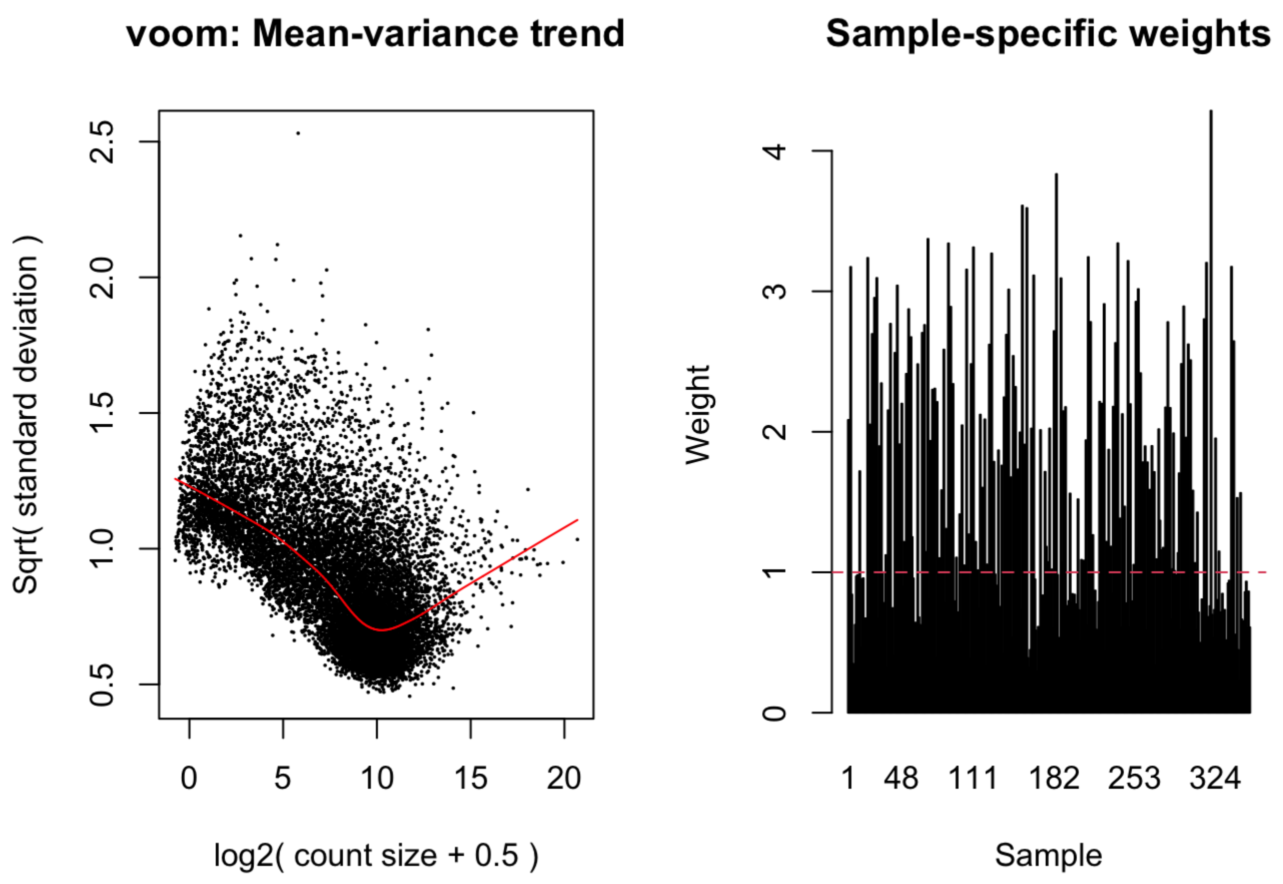
```
colnames(design) <- gsub("dge\\$samples\\$tumor", "tumor", colnames(design))
colnames(design) <- gsub("dge\\$samples\\$library_typeunstranded", "library_type", colna
mes(design))
colnames(design) <- gsub("dge\\$samples\\$Ta2", "Ta2", colnames(design))
colnames(design) <- gsub("dge\\$samples\\$Ta3", "Ta3", colnames(design))
colnames(design) <- gsub("dge\\$samples\\$Ta4", "Ta4", colnames(design))
colnames(design) <- gsub("dge\\$samples\\$sex", "Sex", colnames(design))
head(design)
```

```
##   tumor0 tumor1 library_type Ta2 Ta3 Ta4 SexM
## 1      1      0            1   1   0   0    1
## 2      0      1            1   1   0   0    1
## 3      1      0            1   0   0   1    1
## 4      0      1            1   0   0   1    1
## 5      1      0            1   1   0   0    1
## 6      0      1            1   1   0   0    1
```

### voom

voom is a function that lies within a package called limma. limma/voom is used in DGE analysis. voom is a function that takes the counts in a metadata set and transforms them into log2 of TMM values calculated above in the normalization factors. A linear model is then fitted to the TMM for each gene and residuals are calculated. A smoothed curve is then fitted to the square root of the residual standard deviation by the average expression (this is the red line). This smooth curve is then used to obtain weights for each gene and sample that are passed into limma along with the TMM values.

```
# Running voom again with the new design matrix.
v <- voomWithQualityWeights(dge, design, plot=TRUE)
```



### limma

This sections marks the beginning of running limma. limma creates a linear fit to the data, makes comparisons of the fitted data, and applies Bayes smoothing. limma startes by creating a variable that has all of the duplicate correlation values on v and design. These correlation values will be used later in a linear fit.

```
# ======================================================================================
# Differential expression analysis with limma - all male tumor adjacent vs. non-tumor-adjacent
# =====================================================================================

# Block design for individual. This is used in tumor-normal comparisons with
# paired samples.
corfit <- duplicateCorrelation(v, design, block = v$targets$ID)
# This should give a positive correlation value. It represents the
# correlation between measurements made on the same person.
corfit$consensus
```

```
## [1] 0.1579944
```

### limma graph

This is the linear model with limma, notice that the correlation values were the duplicate correlations used earlier

```
# Fitting the linear model with limma.
# If using paired samples, the within-patient correlation and a block design
# for patient is used to account for pairwise samples
fit <- lmFit(v, design, block = v$targets$ID, correlation = corfit$consensus)
```

### Coefficient vector

This code chunk involves extracting coefficients from the linear fit model and storing them in a vector for later use.

```
# Contrast design for differential expression
# Defining pairwise comparisons
contrasts <- makeContrasts(Adjacent_vs_Tumor = tumor1 - tumor0,
                           levels=colnames(design))
head(contrasts)
```

```
##               Contrasts
## Levels         Adjacent_vs_Tumor
##   tumor0                      -1
##   tumor1                       1
##   library_type                 0
##   Ta2                          0
##   Ta3                          0
##   Ta4                          0
```

```
# Assigning all comparisons to a vector for later
allComparisons <- colnames(contrasts)
```

### Contrast Analysis

This next code chunk reorients the linear model obtained earlier and obtains the coefficients and standard errors from the model. This step also sets us up to apply Empirical Bayes smoothing.

```
# Running contrast analysis
vfit <- contrasts.fit(fit, contrasts = contrasts)
# Look at N of DEGs with adj. p <0.01 and log2FC>2
summary(decideTests(vfit, adjust.method = "BH", p.value = 0.05, lfc = 2))
```

```
##        Adjacent_vs_Tumor
## Down                 512
## NotSig             12714
## Up                   158
```

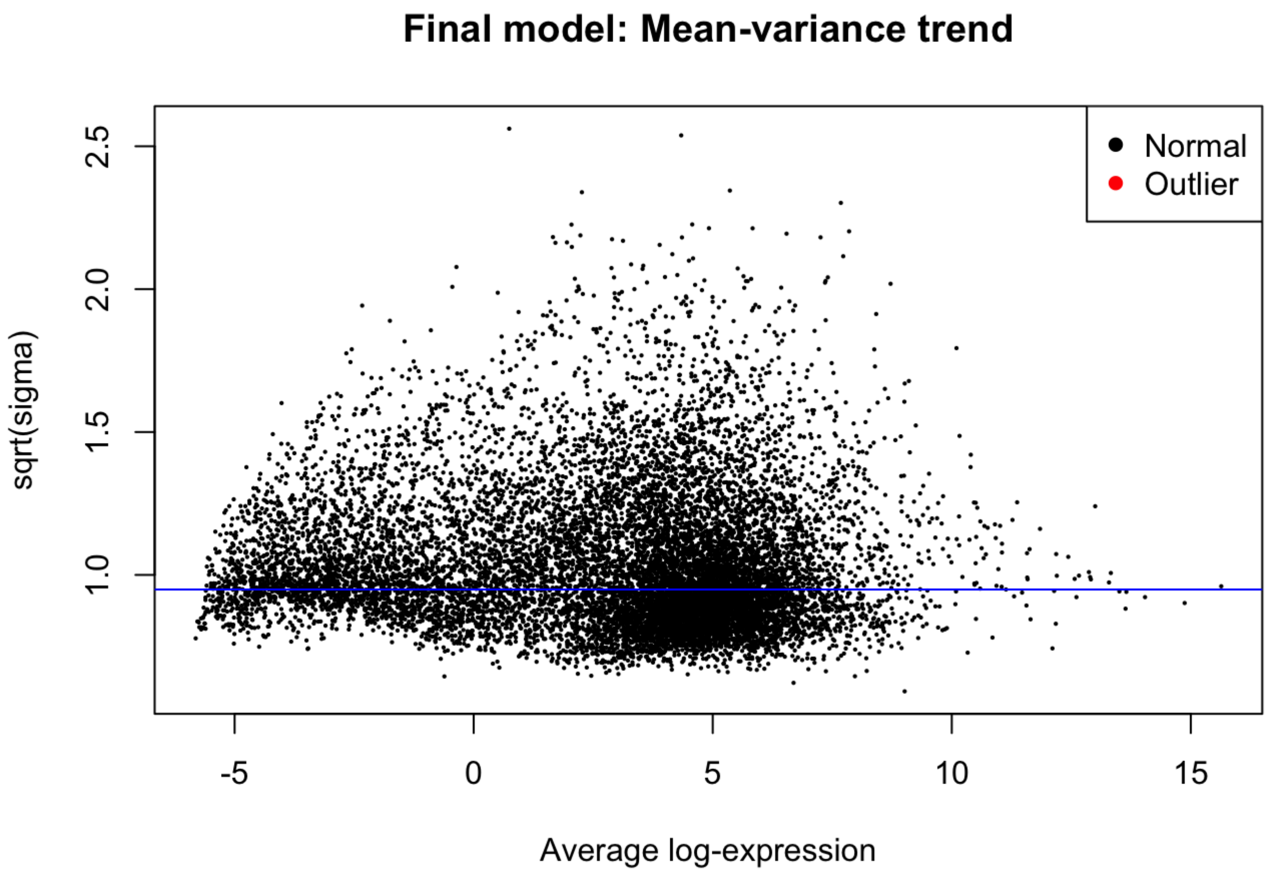
### Bayes smoothing

This code chunk uses Empirical Bayes smoothing to plot the final model after doing the limma and voom analysis.

Empirical Bayes smoothing is a way to account for uncertainty. The technique uses the population in a region as a measure of confidence. Meaning that areas with low margin of error are left untouched while estimates with higher margin of error are moved closer to the global average.

The write.csv is commented out because it is not needed for this file

```
# Computing differential expression based on the empirical Bayes moderation of
# the standard errors towards a common value. Robust = should the estimation of
# the empirical Bayes prior parameters be robustified against outlier sample
# variances?
veBayesFit <- eBayes(vfit, robust=TRUE)
plotSA(veBayesFit, main = "Final model: Mean-variance trend")
```



```
vTopTable <- topTable(veBayesFit, n=Inf, p.value=1, lfc=0)
DEGs <- topTable(veBayesFit, n=Inf, p.value=0.05, lfc=2)
#DEGs_print <- data.frame(DEGs$GENEID, DEGs$gene_name, DEGs$adj.P.Val, DEGs$logFC)
#write.csv(DEGs_print, "~/R/gene_list_tumor_vs_tumor_adjacent.csv")
```