Published in final edited form as:
Methods Mol Biol. 2010 ; 666: 395–417. doi:10.1007/978-1-60761-820-1_25.

Bioinformatics Techniques in Microarray Research: Applied Microarray Data Analysis Using R and SAS Software

Ryan T. Demmer[1], **Paul Pavlidis**[2], **Panos N. Papapanou**[3]

[1.]Department of Epidemiology, Mailman School of Public Health, Columbia University, New York, USA

[2.]Center for High-throughput Biology and Department of Psychiatry, University of British Columbia, Vancouver, Canada

[3.]Division of Periodontics, Section of Oral and Diagnostic Sciences, College of Dental Medicine, Columbia University, New York, USA

Abstract

Exploration of the underlying biological mechanisms of disease is useful for many purposes such as the development of novel treatment modalities in addition to informing on-going risk factor research. DNA-microarray technology is a relatively recent and novel approach to conducting genome-wide gene expression studies to identify previously unknown biological pathways associated with disease. The copious data arising from microarray experiments is not conducive to traditional analytical approaches. Beyond the analytical challenges, there are equally important issues related to the interpretation and presentation of results. This chapter outlines appropriate techniques for analyzing microarray data in a fashion that also yields a list of top genes with differential expression in a given experiment. Derivatives of the top genes list can be used as a starting point for the presentation of study results. The list also serves as the basis for additional techniques related to enhanced interpretation and presentation of results. All analyses described in this chapter can be performed using relatively limited computational resources such as a lap top PC with at least 2.0 GB of memory and 2.0 GHz of processing speed.

Keywords

Microarray; gene expression; mRNA; bioinformatics; epidemiology; statistics; R; SAS; computational genomics

1. Introduction

The advent of DNA-microarray technology has created new opportunities for the study of biological pathways relevant to human disease. However, analytical and bioinformatics challenges have developed alongside the advances in basic sciences accomplished by the use of genomic approaches. From an analytical standpoint, two of the greatest challenges are to (i) efficiently identify relevant biological patterns related to the disease process and (ii) handle the statistical issue of multiple comparisons. In this chapter, we provide an overview to help initiate researchers who are unfamiliar with microarray data analysis methods and concepts. Our aim is to give explicit, step-by-step examples of how one may conduct an analysis of their own data. It is expected that the reader will have some basic knowledge in statistics, causal inference, and data analysis using the relevant statistical analysis packages described herein. The analytical examples presented in this chapter are based on experiments using single channel Affymetrix GeneChips© but most of the analytical strategies and approaches discussed are applicable to other microarray designs as well.

1.1. Broad Overview of Microarray Data Analysis Concepts

Analytical approaches in microarray research generally fall into one of two broad categories defined as either supervised or unsupervised analysis. It is common to use both approaches when analyzing results from a single experiment but the necessary analytical approach will vary according to the design and aims of each experiment. A brief description of unsupervised and supervised analyses is provided below followed by a description of the experimental design on which we will focus our analytical methods in this chapter. For a more comprehensive overview and discussion of microarray data analysis, please *see* Quackenbush (1).

1.1.1. Unsupervised Analysis (Class Discovery)—Genomic research has contributed significantly to the identification of novel biological pathways underlying human disease. This has been accomplished partly through the use of unsupervised methods that can identify novel gene expression patterns. These techniques ignore any a priori information about the samples such as diagnosis, disease state, histologic characteristics but rather seek to identify potentially biologically meaningful patterns of expression. Various forms of cluster analysis, such as hierarchical cluster analysis or k-means clustering, have been utilized for these purposes. The premise of cluster analysis is to assign a measure of similarity to gene expression profiles. Expression similarity can be observed (i) between genes (across samples); or (ii) between samples (across genes); or (iii) as a combination of i and ii. A detailed description of cluster analysis theory and application is beyond the scope of this chapter and the specific analytical methods we describe will focus on a specific application of supervised analysis.

1.1.2. Supervised Analysis (Differential Expression)—In supervised analyses biological samples or participants are classified a priori based on relevant characteristics (i.e., disease diagnosis). These classifications then serve as independent variables in statistical models predicting variation in gene expression level (the dependent variable). The null hypothesis for each gene is that expression will not vary across the predefined classes. There are various approaches to supervised analysis and in this chapter, we will focus on appropriate techniques for conducting a supervised analysis for the identification of genes that are differentially expressed in states of periodontal health vs. disease. Therefore, our example will identify "top genes" with biological relevance to periodontal disease. We will use two examples stemming from the same study design but using two different statistical methods. A brief description of the study design and scientific aim follows. We have recently studied gene expression signatures in healthy and diseased gingival tissues (2) in 90 non-smokers, 63 with chronic and 27 with aggressive periodontitis, each contributing with ≥ 2 "diseased" interproximal papillae (with bleeding on probing, probing pocket depth ≥ 4 mm, and clinical attachment loss ≥ 3 mm) and a "healthy" papilla. This design allowed us to test whether expression levels of various genes are different when comparing diseased vs. healthy gingival tissue. The steps for identifying a list of differentially expressed genes using either mixed effects statistical models or two-sample *t* tests are outlined below.

All analyses described in this chapter were tested using both: i) a laptop PC running Windows Vista, with a 2.8 GHz processor and 3.0 GB of RAM; and ii) a laptop MacBook Pro, with a 2.53 GHz processor and 4.0 GB of RAM. We expect that the code will also work on other platforms such as UNIX, LINUX & XP but minor adjustments might be required for these systems.

2. Materials

2.1. Data

This chapter will assume that all original data files are located on the C drive in a folder named microarray: "C:\microarray"

1. Affymetrix CEL files for each experimental sample (*see* Note 1).
2. Gene annotation file (*see* Note 2).
3. Experimental design data file (*see* Note 3).

2.2. Statistical Analysis Software

1. R software and related analysis packages (*see* Note 4).
2. SAS software (optional, *see* Note 5).

3. Methods

In this section we provide explicit step-by-step instructions for manipulating and analyzing gene expression data. The analysis is based on our previously published work (2) as described above. It should be noted at the outset that there are no "absolutes" when it comes to developing an analytical strategy and corresponding computer code for analyzing gene expression data. The examples we provide are designed to be readable, transparent, and functional. However, in many situations they are not necessarily optimized for efficiency or programming eloquence. To accomplish efficiency of this nature, an understanding of computing, bioinformatics, and statistics beyond the scope of this chapter is required.

```
*Also, please note that downloadable text files containing the R and SAS
code presented in this chapter are available at:
http://www.chibi.ubc.ca/faculty/pavlidis/demmer-methods
```

3.1. Create List of Genes, p-Values and q-Values Using R

Open R on your PC and paste the following commands to the command line. The prompt symbols (">") should not be entered by the user; prompt symbols will appear by default in the R program and are presently included to reflect the beginning of a new command that should be typed directly into the R command line. Comments are denoted with a pound sign

(#) at the beginning of the comment. Comments should not be pasted to the R command line. Finally, be aware that R code is case sensitive.

3.1.1. Input Data Files and Normalize Expression Data

1.

```
#Set working directory in R (see Note 6).
> setwd("C:/microarray")
```

2.

```
#Load the affy library (see Note 4)
> library(affy)
```

3.

```
#Create normalized expression data based on all CEL files in the
working directory
>e<-justRMA(destructive=TRUE)
```

4.

```
#Write the expression data (see Note 7) in R object "e" to a text
file in the working directory
>write.exprs(e,file="expressionData.txt")
```

5.

```
#Read EDDF into an R object.
>design<-read.delim("EDDF.txt", row.names=1, header=T)
```

6.

```
#Remove .CEL file extensions and X "characters" from expression
data column names (see Note 8)
>d<-read.table("expressionData.txt", header=T, row.names=1)
>names(d)<-gsub(".CEL", "", gsub("X", "", names(d)))
>d<-round(d, 5)
```

7.

```
#Write a permanent expression data file to the working directory
>write.table(d, file="expressionDataFinal.txt", quote=F, sep='\t')
```

3.1.2. Modify Data

1.

```
#Read the expressionDataFinal into R as an object
>expression<-read.delim("expressionDataFinal.txt", row.names=1,
header=T)
```

2.

```
#Remove "X" characters from column names (corresponding to
Sample_ID numbers).
"X" characters are inserted automatically by the read.delim
function
```

```
>names(expression) <-sapply(strsplit(names (expression), “X”),
function(x) {x[2]})
```

3.

```
#Arrange expression data in ascending ID order (arranges columns
from left to right)
>expression<-expression [,as.character(sort
(as.numeric(names(expression))))]
```

4.

```
#Arrange experimental design data file in ascending ID order
(arranges rows from top to bottom)
>designOrder<-order(as.numeric(row.names(design)))
>design<-design[designOrder,]
```

5.

```
#Reduce the design and expression R objects to include only
biological samples that appear in both objects.
>exprsID<-data.frame(names(expression))
>designID<-data.frame(row.names(design))
>names(exprsID)<-c(“ID”)
>names(designID)<-c(“ID”)
>idoverlap<-merge(exprsID, designID, by = “ID”)
>row.names(idoverlap)<-idoverlap$ID
>design<-design[row.names(idoverlap),]
>expression<-expression[,row.names(idoverlap)]
```

3.1.3. Run Linear Mixed Effects Statistical Models—The following steps will run a linear mixed effects ANOVA (3) analysis exploring whether expression levels for each probe set (dependent variable) are differential by level of the biological sample status (independent variable – modeled as healthy vs. diseased gingival tissue in this example). We first set up a function in R and apply the function to the actual data and extract *p*-values from each regression. The end result (for this specific experiment) will be a list of 54,675 *p*-values – one *p*-value for each probe set on the particular Affymetrix GeneChip used in our experiments (Human Genome U133 Plus 2.0 Array).

1.

```
#Load the “nlme” library
>library(nlme)
```

2.

```
#Create a function that will run the mixed model regression for
all 54,675 probe sets on our GeneChip
>mm.funHD<-function(x)
{
d<-data.frame(x, design[,c(“Patient”, “Diseased_Tissue”)])
names(d)<-c(“E”, “Patient”, “Diseased”)
r<-NULL
try(silent=T, r<-anova(lme(E ~ Diseased, random = ~ 1 | Patient,
data=d))[,4][2])
r
}
```

3.

```
#Apply the function to the expression data and save the result to
an R object titled “hlthDis”
>hlthDis<-apply(expression, 1, mm.funHD)
```

4.

```
#Step #3 will take time (anywhere from seconds to many minutes
depending on the memory and processor speed of the computer being
used. Therefore it is advisable to save the R image so that the
analysis does not need to be rerun in the future
>save.image()
```

5.

```
#Replace “NULL” values with “NA” because R understands that NA =
missing data (missing p-value in this example) whereas R does not
know how to handle “NULL”
>hlthDis2<-gsub(“NULL”,”NA”,hlthDis)
>hlthDis2<-as.numeric(hlthDis2)
>names(hlthDis2)<- names(hlthDis)
```

6.

```
#Creates a text file in the working directory containing all probe
sets and their corresponding p-values in ascending p-value order
>write.table(sort(hlthDis2,na.last=TRUE),
file=“hlthDis.rem.txt”,col.names=“probe p-value”, quote=F, sep=
‘\t’)
```

7.

```
Create a reduced text file in the working directory that removes
any probe sets with p-values = “NA” because these values can
prevent the q-value function from working in subsequent steps. If
there are no “NA” values in “hlthDis.rem.txt” then this step is
unnecessary.
>hlthDis3<-hlthDis2[which(hlthDis2 != “NA”)]
>write.table(sort(hlthDis3,na.last=TRUE),file=“hlthDisNoNA.rem.txt”
,col.names=“probe p-value”, quote=F, sep=‘\t’)
```

3.1.4. Obtain q-Values and Merge q-Values with p-Values (see Note 9)

1.

```
#Load qvalue library
>library(qvalue)
```

2.

```
#Read the list of p-values created in Section 3.1.3 into an R
object
>pvalues<-read.table(“hlthDisNoNA.rem.txt”, skip=1,row.names=1)
>pvalues2<-pvalues[,1]
```

3.

```
#Apply the qvaule function to generate a table indicating the
number of significant calls for both the raw p-values and for the
calculated q-values using a set of cutoffs given by “cuts” (see
Notes 9,10).
>summary(qvalue(pvalues2), cuts = c(9.14e-07, 0.001, 0.01, 0.025,
0.05, 0.1, 1))
```

4.

```
#Create a vector of the q-values and merge with p-values
>q<-qvalue(pvalues2)$qvalues
>q<-data.frame(q)
>qp<-data.frame(row.names=row.names(pvalues), q, pvalues)
>names(qp)<-c(“qvalue”,”pvalue”)
>porder<-order(qp[,”pvalue”])
>qp<-qp[porder,]
>qpfinal<-data.frame(row.names(qp),qp$qvalue, qp$pvalue)
>names(qpfinal)<-c(“probe”,”qvalue”,”pvalue”)
>write.table(qpfinal,file=“hlthDis.qp.rem.txt”, row.names=F,
quote=F, sep=‘\t’)
```

3.1.5. Merge p-Values and q-Values with Gene Annotation File—At this point, we have generated a list of *p*-values and *q*-values corresponding to every probe set on the microarray chip (in addition to a summary corresponding to the number of significant calls using both *q*-value and *p*-value significance criteria). Now, in a final step, merge gene annotations to the file containing *p*-values and *q*-values (“hlthDis.qp.rem.txt” created in Section 3.1.4) and reduce the data to a file containing a list of “top genes.”

1.

```
#Read the gene annotation file into an R object (see Note 11)
>annotations<-read.delim(“C:/microarray/HG-
U133_Plus_2_bioproc.an.txt”, col.names=c(“probe”, “gene”,
“description”, “GOterms”), sep=“\t”, fill=T)
>annotations<-data.frame(annotations)
>probeorderannot<-order(annotations[,”Probe”])
```

2.

```
#Order the annotation file by probe ID
>annotations2<-annotations[probeorderannot,]
```

3.

```
#Order “qpfinal” by probe ID
>probeorder<-order(qpfinal[,”probe”])
>qpfinal<-qpfinal[probeorder,]
```

4.

```
#Combine the modified R objects containing the gene annotations
(“annotations2”) and the p-values and q-values (“qpfinal”)
>final<-merge(qpfinal,annotations2,by=“probe”) #(see Note 12)
>FinalPvalueOrder<-final[order(final[,”pvalue”]),]
>FinalPvalueOrder<-FinalPvalueOrder[,c(“probe”,
“gene”,”description”,”pvalue”,”qvalue”)]
>write.table(FinalPvalueOrder,file=“FinalPvalueOrder.txt”,
row.names=F, quote=F, sep=‘\t’)
```

5.

```
#Alternatively, if you only want a table including genes with a
qvalue < 0.05 then do:
>FinalPvalueOrder<-FinalPvalueOrder
[which(FinalPvalueOrder[,”qvalue”] < 0.05),]
>write.table(FinalPvalueOrder,file=“FinalPvalueOrder.txt”,
row.names=F, quote=F, sep=‘\t’)
```

3.2. Two Sample TTEST in R

The study design we have based our analytical approach on thus far is typical of clinical research in humans. Because of the repeated measures occurring within person, a sophisticated statistical analysis method was required (Mixed Model Regressions). However, other designs (and subsequent hypothesis testing scenarios) are also common in clinical research. For example, consider the following scientific question currently under investigation (4): “Does gene expression in diseased gingival tissue differ between patients diagnosed with chronic vs. aggressive periodontitis?” Now assume for the moment that the experimental design collects one diseased gingival tissue sample per patient, among $n=10$ patients ($n=5$ with chronic periodontitis and $n = 5$ with aggressive periodontitis). The gene expression data obtained = from this experiment are conducive to a classical two-sample t test which compares mean gingival tissue gene expression between the two groups of five patients. The following code demonstrates the appropriate steps to generate a list of top genes differentially expressed between these two hypothetical patient groups (*see* Note 13).

3.2.1. Modify Data

1.

```
#To calculate two-sample t-tests in the fashion described above,
we first need to modify the previously created R data objects,
“design” and “expression” by restricting the data to the first
diseased sample per patient.
>ttestobs<-which(design$Sample_Number = = 1 & design
$Diseased_Tissue = = 1)
>designT<-design[ttestobs,]
```

2.

```
#Restrict expression to include only samples in the modified
design, “designT”
expressionT<-expression[,ttestobs]
```

3.

```
#Create a function that will run two-sample t-tests for all 54,675
probe sets on the GeneChip.
>ttest.fundiag<-function(x)
{
designT2<-data.frame(x, designT[,”Diagnosis”])
names(designT2)<-c(“E”, “Diagnosis”)
r<-NULL
try(silent=T, r<-t.test(E~Diagnosis, data = designT2, var.equal=T)
[“p.value”])
r
}
```

4.

```
#Apply the function and write the results to the user defined R
object “DiagnosisT”.
>DiagnosisT<-apply(expressionT, 1, ttest.fundiag)
```

5.

```
#The next series of data manipulations allow for future sorting by
p-value and saving results to a text file
>DiagnosisT2<-unlist(DiagnosisT)
>DiagnosisT2<-gsub(“NULL”,”NA”,DiagnosisT2)
>DiagnosisT2<-as.numeric(DiagnosisT2)
>names(DiagnosisT2)<- names(DiagnosisT)
```

6.

```
#Creates a text file in the working directory containing all probe
sets and their corresponding two-sample t-test p-values in
ascending p-value order
>write.table(sort(DiagnosisT2,na.last=TRUE), file=“Final_t-
test_PvalueOrder.txt”, row.names=T, quote=F, col.names=“probe
pvalue”, sep=‘\t’)
```

7.

```
#At this point you can now use the exact same methods in Sections
3.1.4 and 3.1.5 to create a final text file containing genes,
descriptions, p-values and q-values. Simply substitute the old p-
value list “hlthDisNoNA.Rem.txt” with your new p-value list
“Final_t-test_PvalueOrder.txt”.
```

3.3. Create List of Genes, p-Values and q-Values Using SAS

The following sections generate results identical to those in Sections 3.1 and 3.2; only now we are using SAS software (*see* Note 5) to perform the analyses. The SAS examples also include the necessary code to generate gene expression fold changes (*see* Note 14). In SAS examples, a forward slash asterisk (/*) note the beginning of a comment and an asterisk forward slash (*/) note the end of a comment. We have also used ALL CAPS for SAS keywords and mixed case for user-supplied text in keeping with typographical conventions used in other SAS textbooks (5).

3.3.1. Import Data Files

1.

```
/*Import Gene Annotation File*/
DATA WORK.annotations;
INFILE “C:\microarry\GeneAnnotationFile.txt”
    DELIMITER=‘09’x
        MISSOVER DSD LRECL=32767 FIRSTOBS=2;
  INFORMAT probe $27.;
  INFORMAT Gene $10.;
  INFORMAT Description $100.;
  INFORMAT GOTerms $12.;
  FORMAT probe $27.;
  FORMAT Gene $10.;
  FORMAT Description $100.;
  FORMAT GOTerms $12.;
  INPUT probe $ Gene $ Description $ GOTerms $;
RUN;
```

2.

```
/*Sort Gene Annotation File by probe to allow for merging in later
data steps*/
PROC SORT DATA = annotations;
  BY probe;
RUN;
```

3.

```
/*Import Experimental Design Data File EDDF.txt*/
DATA WORK.design;
INFILE “C:\microarray\EDDF.txt” DELIMITER=‘09’x
       MISSOVER DSD LRECL=32767 FIRSTOBS=2;
  INFORMAT Sample_ID 8.1;
  INFORMAT Patient 8.;
  INFORMAT Sample_Number 1.;
  INFORMAT Diseased_Tissue 1.;
  INFORMAT Diagnosis 1.;
  FORMAT Sample_ID 8.1;
  FORMAT Patient 8.;
  FORMAT Sample_Number 1.;
  FORMAT Diseased_Tissue 1.;
  FORMAT Diagnosis 1.;

  INPUT Sample_ID Patient Sample_Number
    Diseased_Tissue Diagnosis;
RUN;
```

4.

/*Import Normalized Expression Data (*see* **Note 15**). When importing the normalize gene expression data, it will be necessary for the user to ensure that the order of INFORMAT, FORMAT and INPUT statements below, are identical to the column ordering (which correspond to each tissue sample) in the ““expressionData.txt” generated in **Section 3.1.1**. If the orders do not match, the wrong expression data will be matched to each tissue sample (*see* **Note 16**).*/

```
DATA WORK.EXPRIDORDER;
%LET _EFIERR_ = 0;
   INFILE ‘C:\microarray\expressionDataFinal.txt’ delimiter=‘09’x
MISSOVER DSD LRECL=32767 FIRSTOBS=2;
INFORMAT probe $27.;
INFORMAT ID_1_1 BEST32.;
INFORMAT ID_1_2 BEST32.;
INFORMAT ID_1_4 BEST32.;
INFORMAT ID_2_1 BEST32.;
INFORMAT ID_2_2 BEST32.;
INFORMAT ID_2_4 BEST32.;
INFORMAT ID_4_3 BEST32.;
FORMAT probe $27.;
FORMAT ID_1_1 BEST12.;
FORMAT ID_1_2 BEST12.;
FORMAT ID_1_4 BEST12.;
FORMAT ID_2_1 BEST12.;
FORMAT ID_2_2 BEST12.;
FORMAT ID_2_4 BEST12.;
FORMAT ID_4_3 BEST12.;
INPUT probe $ ID_1_1 ID_1_2 ID_1_4 ID_2_1 ID_2_2 ID_2_4 ID_4_3;
IF _ERROR_ THEN CALL SYMPUTX (‘_EFIERR_’,1);
RUN;
```

3.3.2. Manipulate Gene Expression Data Set in SAS

1.

```
/*Convert the data structure from “wide” to “long” form (see Note
17)*/
DATA expr2;
SET expridorder;
ARRAY origida [*] ID_1_1--ID_4_3;
ARRAY newida [7]$ _TEMPORARY_;
DO i = 1 TO DIM(origida);
newida[i] = VNAME(origida[i]);
END;
DO i = 1 to DIM(origida);
exprs=origida[i]; id=newida[i]; OUTPUT;
END;
KEEP probe exprs id;
RUN;
```

2.

```
/*Translate id number into same format as the format used the
design data set. This allows the two data sets to be merged in
future data steps*/
DATA expr2;
SET expr2;
pat=SCAN(id,2,’_’);
pat2=SCAN(id,3,’_’);
id=TRIM(pat)||”_”||TRIM(pat2);
id = TRANWRD(id,’_’,’.’);
id2 = INPUT(id,5.);
KEEP probe exprs id2;
RUN;
```

3.

```
/*Sort expression and design data sets by sample id number so they
can be merged*/
PROC SORT DATA=design; BY Sample_id; RUN;
PROC SORT DATA=expr2; BY id2; RUN;
```

4.

```
/*Merge expression data with design data by sample id number*/
DATA expr3;
MERGE expr2 (RENAME=(id2=id) IN=inexpr) design
(RENAME=(Sample_ID=id)IN=indesign);
BY id;
IF inexpr and indesign;
RUN;
```

3.3.3. Mixed Model Regressions

1.

```
/*Sort data set by probe to allow regressions to be run by probe*/
PROC SORT DATA=expr3;
BY probe;
RUN;
```

2.

```
/*see Note 18 for further information on SAS system options used
and the output delivery system in SAS*/
OPTIONS MSGLEVEL=N NONOTES STIMER;
```

3.

```
/*Mixed model regressions*/
ODS LISTING CLOSE;
ODS RESULTS OFF;
PROC MIXED DATA = expr3;
BY probe;
ODS OUTPUT TESTS3=perio (KEEP = probe Probf);
  *keeping p-values from F-statistic;
MODEL exprs = Diseased_Tissue/S;
RANDOM INTERCEPT/SUBJECT = patient;
RUN;QUIT;
ODS RESULTS ON;
ODS LISTING;
OPTIONS MSGLEVEL=I NOTES;
```

3.3.4. Merge SAS Intermediate Data Sets, Create Gene Expression Fold Changes and q-Values

1.

```
/*For each probe, create mean expression values across patient and
within Diseased_Tissue status (Diseased (1) vs. Healthy (0)). This
will allow for the creation of fold changes in a future data step*/
PROC MEANS DATA = expr3 NOPRINT NWAY;
CLASS probe Diseased_Tissue;
VAR exprs;
OUTPUT OUT = means (DROP = _TYPE_ _FREQ_)
  MEAN = exprs;
RUN;
```

2.

```
/*Get mean expression values for healthy and diseased tissue into
one observation*/
DATA means1;
SET means;
BY probe;
```

```
IF FIRST.probe THEN DO;
IF Diseased_Tissue = 0 THEN exprs0 = exprs;
RETAIN exprs0;
END;
IF Diseased_Tissue = 1 THEN exprs1 = exprs;
IF LAST.probe THEN OUTPUT;
RUN;

DATA final;
MERGE annotations means1 perio (RENAME=(probf=pvalue) IN=inperio);
BY probe; IF inperio;
FORMAT pvalue e16.;
RUN;
PROC SORT DATA = final;
BY pvalue;
RUN;
DATA final;
SET final;
obsnum+1;
/*(see Note 9)*/
qvalue = (pvalue*54675)/obsnum;
FC = 2**(exprs1-exprs0);
/*Calculate the absolute fold change so up- and down-regulated
genes can be compared on the same scale*/
absoluteFC = 2**(ABS(exprs1-exprs0));
KEEP Gene Description probe pvalue qvalue absoluteFC FC;
RUN;
```

3.

```
/*Sort the final data set by absolutFC - see Note 14 */
PROC SORT DATA = final; (see Note 19)
BY DESCENDING absoluteFC pvalue;
RUN;
```

3.3.5. Create Final Excel Spreadsheet

1.

```
/*Create a final Excel spreadsheet containing the results for all
genes sorted by absolute fold change*/
ODS LISTING CLOSE;
ODS HTML BODY = “C:\microarray\TopGenes.xls” style=minimal;
PROC PRINT DATA = final NOOBS;
RUN;
ODS HTML CLOSE;
ODS LISTING;
```

3.4. Two Sample TTEST in SAS

Refer to Section 3.2 and Note 13 for a brief introduction to the scientific question being addressed in the following SAS code.

1.

```
/*Restrict the data set “expr3” created in Section 3.3.2 step 4,
to include the appropriate samples (see Note 13)*/
DATA expr3;
SET expr3;
WHERE Diseased_Tissue = 1 and Sample_Number = 1;
KEEP id probe exprs Diagnosis;
RUN;
```

2.

```
/*The data set expr3 should already be sorted by probe but redo to
be sure*/
PROC SORT DATA = expr3;
BY probe;
RUN;
```

3.

```
/*Run t-tests for all 54,675 probe sets on the microarray chip*/
ODS LISTING CLOSE;
ODS RESULTS OFF;
PROC TTEST DATA = expr3;
BY probe;
CLASS Diagnosis;
VAR exprs;
ODS OUTPUT Statistics=stats (KEEP = probe class mean)
Ttests=ttests (KEEP = probe variances probt);
RUN;
ODS RESULTS ON;
ODS LISTING;
```

4.

```
/*Modify the data sets “ttests” and “stats” created in the ODS
OUTPUT statement from step 3*/
DATA ttests; SET ttests (RENAME=(probt=pvalue));
WHERE variances = “Equal”;
KEEP probe pvalue;
run;
DATA stats;
SET stats;
WHERE class = “Diff (1–2)”;
KEEP probe mean;
RUN;
```

5.

```
/*Sort SAS data sets for merging by probe*/
PROC SORT DATA = stats;
BY probe;
RUN;
PROC SORT DATA = ttests;
BY probe;
RUN;
PROC SORT DATA = annotations;
BY probe;
RUN;
```

6.

```
/*Merge necessary SAS data sets, create q-values and fold changes*/
DATA final;
MERGE annotations ttests stats;
BY probe;
FORMAT pvalue e16.;
RUN;
PROC SORT DATA = final;
BY pvalue;
RUN;
DATA final;
SET final;
obsnum+1;
qvalue = (pvalue*54675)/obsnum;
FC = 2**(mean);/*Chronic vs. Aggressive*/absoluteFC =
2**(ABS(mean));
KEEP probe Gene Description pvalue qvalue fc absoluteFC;
RUN;
```

7.

```
/*Sort the final data set by absolutFC - see Note 14*/
PROC SORT DATA = final;
BY DESCENDING absoluteFC;
RUN;
```

8.

```
/*Create final Excel spreadsheet*/
ODS LISTING CLOSE;*prevents printing to output screen;
ODS HTML BODY = “C:\microarray\TopGenesTTEST.xls” STYLE=minimal;
PROC PRINT DATA = final NOOBS; RUN;
ODS HTML CLOSE;
ODS LISTING;
```

3.5. Gene Ontology Analysis

After performing the appropriate statistical analysis to determine a level of statistical significance for each gene, it is often useful to identify groups of affected genes with similar biological function. Gene ontology analysis is an emerging method for this goal of grouping genes. Step-by-step instructions for a gene ontology analysis are beyond the scope of this chapter. However, two high quality and readily available tools are available for free download online. The user's manuals of these programs are sufficient for novice users to conduct a gene ontology analysis using the *p*-value list(s) generated above. We suggest the following two programs and provide their respective World Wide Web addresses, where more information can be found:

ErmineJ (6): http://www.bioinformatics.ubc.ca/ermineJ/index.html

Pathway Express (7, 8): http://vortex.cs.wayne.edu/Projects.html

4. Notes

1. Affymetrix CEL files are created by Affymetrix image analysis software. The CEL file stores the results of the intensity calculations for each probe on the GeneChip. The intensity is based on the pixel values of the DAT file. This information is used to generate an expression level for each probe and thereby each gene on the GeneChip. There is one CEL file for each biological sample collected.

2. Gene annotations files can be downloaded directly from the Affymetrix web site or alternatively, custom files developed by other research groups are also available for free download from the internet. In our studies, we have used the annotation file developed by Dr. Paul Pavlidis and colleagues (University of British Columbia, Canada). A detailed description of the annotation files can be found at the following WWW address: http://www.bioinformatics.ubc.ca/microannots/

   We recommend using the biological processes only version of the annotations corresponding to the microarray chip in your experiment. For the current example, the appropriate annotations file can be downloaded directly at the following WWW address: http://www.bioinformatics.ubc.ca/microannots/HG-U133_Plus_2_bioproc.an.zip

   After downloading this zipped file, you will need to unzip and save the file as a tab delimited text file in your working directory. If the file is not saved as a tab delimited text file, it will not import properly (this is true for both R and SAS imports). Table 25.1 provides a truncated example of a typical gene annotation file structure.

3. The Experimental Design Data File (EDDF) contains experimental design information that will be used to merge characteristics of each sample in the experiment (i.e., sample ID, which samples are healthy or diseased; treated or

untreated) and merge this information with the corresponding expression data. Table 25.2 provides a truncated example of the EDDF structure.

4. R software is freely available. Visit the following web site for information on the product and instructions regarding free download: http://cran.r-project.org/

   In addition to the base R software, download the following packages from the CRAN web site: "nlme", "qvalue".

   Also download and install Bioconductor, offered by the Bioconductor Project: http://www.bioconductor.org/packages/release/bioc/ In addition the "affy" package, will need to be downloaded and installed. Additional Bioconductor packages will likely be required (such as "Biobase") depending on the user's current R setup. Follow the prompts given by R when attempting to install the "affy" package.

5. SAS is a widely used data management and statistical analysis software package. SAS is not required to complete the analyses described in Sections 3.1 or 3.2. The SAS examples provided in Sections 3.3 and 3.4 generate (almost) identical results to those provided in R and we include sections based on SAS simply because this software is so widely used. Users without any prior SAS experience are advised to use the freely available R software only.

6. Working directory: R for PC recognizes forward slashes (/) in the file path. SAS recognizes back slashes (\).

7. The data structure of most microarray experiments is different than traditional experiments which have a limited number of study outcomes. Table 25.2 is an example of a traditional data structure in which study participants (or biological samples) are presented in rows and study outcomes or patient characteristics such as blood biomarkers or diagnosis are presented in columns. This type of table is commonly created by an investigator using readily available database programs such as Microsoft Access or Excel. However, in the context of microarray research, a data structure that can more efficiently handle large amounts of data is generally required. Table 25.3 presents a typical microarray data structure where participants (or biological samples) are presented in columns while gene expression levels for the various genes under study are presented in rows. The initial gene expression data files created in Section 3.1.1 will follow the format presented in Table 25.3.

8. This series of commands will remove "X" characters and .CEL file extensions from the variable names (column names) in the normalized expression file created in Section 3.1.1, step 4.

   Removing the "X" character is specific to the variable naming convention used in this chapter. As seen in Table 25.3, our variable names (which correspond to tissue samples) are numeric and not character. Because R does not handle numeric variable names, an "X" is automatically added to the variable name by R to avoid this conflict. Consequently, we need to remove the "X" so that the variable names in Table 25.3 match the Sample_IDs in Table 25.2. Accordingly, the CEL file extensions need to be removed for the same reason.

The "round" function is also introduced. The "round" function, rounds numerical values to a specified number of digits. This step is performed to reduce file size.

9. To paraphrase Storey & Tibshirani (9), the *q*-value provides a measure of each probe set's significance, automatically taking into account the fact that thousands of hypotheses are simultaneously being tested (i.e., in the current example, the expression of 54,675 probe sets is being compared between healthy and diseased gingival tissue). The *q*-value corresponds directly to the false discovery rate (FDR) and the FDR in turn refers to the percentage of all "significant" statistical tests that are truly null.

Results from the qvalue function will appear similar to those shown in Table 25.4. The interpretation based on Table 25.4 is that 39,690 probe sets were identified with a false discovery rate of <0.05 and 12,743 probe sets were identified with a *p*-value $<9.14\times10^{-7}$.

10. We use 9.14×10^{-7} as the lowest cutoff in this example because this corresponds to the very conservative Bonferroni adjustment (calculated as *p*-value/number of statistical tests, or 0.05/54,675 in the current example). A gene chip with a different number of probe sets would use a different Bonferroni adjustment.

11. The col.names option provides names for the variables in the raw annotation file. This allows a different name to be assigned to the variables in the original data file.

12. After this step you could print to the screen, a subset of the R object "final". The object "final" contains the probe variable twice and printing allows for visual confirmation that the correct annotations were merged with the correct *p*-values and *q*-values. The following code in R will print 15 rows and all columns of the R object: <final[1:15,]

13. In this example, we should clarify that (i) the gingival tissue samples arise from the same patient population as in the earlier "healthy vs. diseased" example, (ii) only the first diseased sample collected from each patient is used in this analysis. This is done because we wish to demonstrate an approach using two-sample *t*-test – a common statistical methods used in clinical research, and (iii) mixed model approaches could also be applied in this setting.

It should also be noted that although this example uses a study design applied to human subjects, the basic scientist could use the same techniques. For example, instead of patients the design might use mice as the experimental unit and rather than observing a diagnosis, the "exposure" of interest might be treatment with an experimental drug in a test group of mice vs. treatment with saline in a control group.

14. Fold change refers to the relative difference in gene expression between two groups of samples. For example, if we find that the mean expression level is

10,000 among diseased gingival tissues and 1,000 among healthy gingival tissues, the fold change for diseased vs. healthy tissue is 10.

15. The authors are currently unaware of any functions included in Base SAS that will convert raw data from .CEL files to a normalized SAS data set. Therefore, in this example, the user will still be required to use R to get normalized expression data (*see* Section 3.1.1). To conserve space we have only provided a truncated version of Expression Data import code. The INFORMAT, FORMAT, and INPUT statements should have a statement for each tissue sample in the experiment.

16. The import code we have supplied ignores the variable names supplied in the original "expressionData.txt" file. The FIRSTOBS = 2 option tells SAS to begin reading data at line 2 on the input file and the variable names are explicitly provided in the INFORMAT, FORMAT, and INPUT statements. There is no direct link between the variable names the user provides and the column names of the input file (expressionData.txt). Consequently, it is imperative that the variable names in the INFORMAT, FORMAT, and INPUT statements are ordered *identically* to the columns (which correspond to each tissue sample) in the "expressionData.txt" input file generated in Section 3.1.1. If the orders do not match, the wrong expression data will be matched to each tissue sample id.

The variable naming convention (names for each column) used for tissue samples in this example (ID_1_1, ID_1_2, etc.) combines the patient and sample ID such that ID_1_1 refers to patient 1, tissue sample 1. We begin the variable name with ID and use underscores as delimiters because SAS will not accept variable names that begin with numbers. Similarly, SAS will not accept variable names that contain '.' symbols. Therefore, the user is free to modify the naming convention in a manner that suits their data or preferences best but it will necessitate subsequent revisions throughout the remaining SAS code.

17. The file "expressionDataFinal.txt" is arranged as demonstrated in Table 25.3. The steps in this section convert the data structure to conform to a format outlined in Table 25.5, in which expression data are shown in the long form for one patient, two samples, three probes. This is necessary to accommodate the SAS statistical analysis procedures in subsequent steps.

18. Because the number of analyses (one for each probe set on the microarray) is generally very large, it is necessary to suppress normal printing to the SAS log and output windows. If the printing is not suppressed, most machines will run out of memory and the analysis will fail. By using SAS system options and ODS commands, this output can be easily suppressed. A brief overview of the options used presently is given below.

NOTES | NONOTES – controls whether notes (messages beginning with NOTE) are written to the SAS log. However, NONOTES does not suppress error or warning messages.

MSGLEVEL N | I – controls the level of detail in messages that are written to the SAS log. If the MSGLEVEL system option is set to N, the log displays notes, warnings, and error messages only. If MSGLEVEL is set to I, the log displays additional notes pertaining to index usage, merge processing, and sort utilities, along with standard notes, warnings, and error messages.

```
*More information on SAS options can be found here:
http://www.sfu.ca/sasdoc/sashtml/lrcon/z0998454.htm;
ODS LISTING CLOSE - Suppresses printing to the output window.
ODS RESULTS OFF - Suppresses printing to the results tab.
```

19. As a result of this sorting, the output file created in Section 3.3.5 will contain genes ordered by descending absolute fold changes using *p*-values to break ordering ties in absolute fold change. Other sort options are available. For example, the "final" data set could also be sorted by pvalue using the following code.

```
PROC SORT DATA = final;
BY pvalue DESCENDING absoluteFC;
RUN;
```

Using this alternate code would yield an output file in Section 3.3.5 containing genes ordered by ascending pvalue with absoluteFC used to break ordering ties in *p*-value.

Acknowledgments

The research described in this chapter has been supported by NIH grant R01 DE-015649. NIH grants K99 DE-018739, R01 DE-13094 have also provided support for the development of this chapter.

References

1. Quackenbush J (2006) Microarray analysis and tumor classification. N. Engl. J. Med 354, 2463–2472. [PubMed: 16760446]
2. Demmer RT, Behle JH, Wolf DL, Handfield M, Kebschull M, Celenti R, Pavlidis P, and Papapanou PN (2008) Transcriptomes in healthy and diseased gingival tissues. J. Periodontol 79, 2112–2124. [PubMed: 18980520]
3. Pavlidis P (2003) Using ANOVA for gene selection from microarray studies of the nervous system. Methods. 31, 282–289. [PubMed: 14597312]
4. Papapanou PN, Abron A, Verbitsky M, Picolos D, Yang J, Qin J, Fine JB, and Pavlidis P (2004) Gene expression signatures in chronic and aggressive periodontitis: a pilot study. Eur. J. Oral Sci 112, 216–223. [PubMed: 15154918]
5. Delwiche LS, and Slaughter SJ (2003) The little SAS book, 3rd ed. SAS Institute Inc., Cary.
6. Lee HK, Braynen W, Keshav K, and Pavlidis P (2005) ErmineJ: tool for functional analysis of gene expression data sets. BMC Bioinformat. 6, 269.
7. Draghici S, Khatri P, Bhavsar P, Shah A, Krawetz SA, and Tainsky MA (2003) Onto-Tools, the toolkit of the modern biologist: Onto-Express, Onto-Compare, Onto-Design and Onto-Translate. Nucleic Acids Res. 31, 3775–3781. [PubMed: 12824416]

8. Khatri P, Voichita C, Kattan K, Ansari N, Khatri A, Georgescu C, Tarcam AL, and Draghici S (2007) Onto-Tools: new additions and improvements in 2006. Nucleic Acids Res 35 (Web Server issue):W206–W211. [PubMed: 17584796]

9. Storey JD, and Tibshirani R (2003) Statistical significance for genomewide studies. Proc. Natl. Acad. Sci. USA 100, 9440–9445. [PubMed: 12883005]

Table 25.1

Truncated example of a gene annotation file

Probe ID	Gene	Description	GOTerms
91580_at	LRTM1	Leucine-rich repeats and transmembrane domains 1	
90610_at	LRCH4	Leucine-rich repeats and calponin homology (CH) domain containing 4	GO:0007399…
90265_at	CENTA1	Centaurin, alpha 1	GO:0050789…
89977_at	FLJ20581	Hypothetical protein FLJ20581	GO:0007582…
89948_at	C20orf67	Chromosome 20 open reading frame 67	
89476_r_at	NPEPL1	Aminopeptidase-like 1	GO:0044237…
87100_at	ABHD2	Abhydrolase domain containing 2	GO:0008150
823_at	CX3CL1	Chemokine (C-X3-C motif) ligand 1	GO:0009605

An example of eight probe sets (out of 54,675 total) and their descriptions included on the Affymetrix HG-U133 GeneChip.

Table 25.2

Truncated example of an experimental design data file

Sample_ID	Patient	Sample_Number	Diseased_Tissue	Diagnosis
1.1	1	1	1	1
1.2	1	2	1	1
1.3	1	3	0	1
2.1	2	1	1	2
2.2	2	2	1	2
2.3	2	3	0	2
3.1	3	1	1	2
3.2	3	2	1	2
3.3	3	3	0	2
4.1	4	1	1	1
4.2	4	2	1	1
4.3	4	3	0	1

Variable key: “Diseased_Tissue”, 1=diseased; 0 healthy. “Diagnosis”, 1=Chronic; 2=Aggressive.

Table 25.3

Truncated example of a gene expression data matrix

Probe	1.1	1.2	1.3	2.1
1007_s_at	10.14741	10.46277	10.43202	9.71754
1053_at	6.52359	6.77471	7.05892	6.6885
117_at	6.98609	6.92772	6.60945	8.07533
121_at	8.16319	8.02055	8.35759	8.44039
1255_g_at	3.27397	3.27663	3.32964	3.43425
1294_at	7.51381	7.28304	7.06534	7.32475
1316_at	5.14637	5.27665	5.14162	4.93963

Table 25.4

Example of “qvalue” function results

Cumulative number of significant calls							
	<9.14e-07	**<0.001**	**<0.01**	**<0.025**	**<0.05**	**<0.1**	**<1**
p-value	12,743	23,036	29,024	32,005	34,625	37,622	54,675
q-value	12,698	24,405	31,865	35,848	39,690	44,656	54,675

Table 25.5

Truncated example gene expression data matrix converted from wide to long form

Probe	Exprs	Sample_ID	Patient	Sample_Number
1007_s_at	10.14741	1.1	1	1
1053_at	6.52359	1.1	1	1
117_at	6.98609	1.1	1	1
1007_s_at	10.46277	1.2	1	2
1053_at	6.77471	1.2	1	2
117_at	6.92772	1.2	1	2