Aumento de Visibilidade e Usabilidade do pacote MaigesPack

Trabalho Supervisionado de Formatura

Emerson Takeshi Hassegawa

Fábio Yoshio Sato

Ricardo Issao Shimanuki

Orientador: Eduardo Jordão Neves

Instituto de Matemática e Estatística – Universidade de São Paulo

Conteúdo

1. Introdução

O homem com a sua curiosidade vem se aprofundando nas pesquisas em diversas áreas desde tempos remotos. Hoje, estamos em um nível em que apenas uma área de conhecimento não é o suficiente para nos prover ferramentas para explicar os fenômenos que nos despertam curiosidade. A Bioinformática é uma nova área de pesquisa que une os conhecimentos dos estudos da biologia, como estudos genômicos e a medicina, com conhecimentos da área de exatas, como computação, estatística e física. Com a ajuda da Bioinformática, podemos estudar a influência dos genes na produção de proteínas em células, entre tantas outras possibilidades de estudos.

2. Objetivos

2.1. Proposta Anterior

Fazer uma refatoração do sistema de análise de dados de microarray R, redesenhando os processos de análise para que as dependências mostrem a realidade, melhorar a flexibilidade do sistema para que os usuários consigam ter maior domínio sobre o processo analítico realizado sobre os dados e aumentar a visibilidade para que o usuário consiga saber cada etapa do processo e entender a sua funcionalidade. Em relação à refatoração, pretendíamos não mudar nenhuma funcionalidade e não ter que escrever muito código, apenas quebrar métodos em métodos menores visando às melhoras já citadas.

2.2. Proposta Atual

Não se difere muito da proposta anterior nos objetivos e sim no método. Depois de estudar o código do sistema, percebemos que ele já estava bem modularizado e não havia muitos espaços para mudança sem modificar o código dos métodos. Como para os usuários do sistema objetivos do projeto seriam de grande utilidade, procuramos outra maneira de realizá-los. Trabalharemos com as funções envolvidas na construção de redes de relevância,criando novos métodos que quebrem o processo antes feito por apenas um método, para dar mais flexibilidade e transparência ao processo.

3. Microarray

3.1. Expressão gênica

Todo ser humano tem o ácido desoxirribonucléico (ADN, em português, ou mais, por convenção; DNA, do inglês, deoxyribose nucleic acid) em suas células. Ele carrega a informação genética, que é o que determina as características de cada pessoa, e conseqüentemente coordena o seu desenvolvimento e funcionamento, pois é usando essa informação que se produzem as proteínas, que são as biomoléculas funcionais indispensáveis para a manutenção do ciclo de vida celular.

O processo de produção de uma proteína a partir de um gene (DNA) é o que se chama de expressão gênica, um dos assuntos mais estudados na Biologia Molecular. Simplificando, a partir de um gene (seqüência de DNA) um mRNA (RNA Mensageiro) é formado através do processo de transcrição. Esse mRNA tem como função levar a informação sobre a cadeia protéica para o RNAt (RNA Transportador) que opera na formação da proteína.

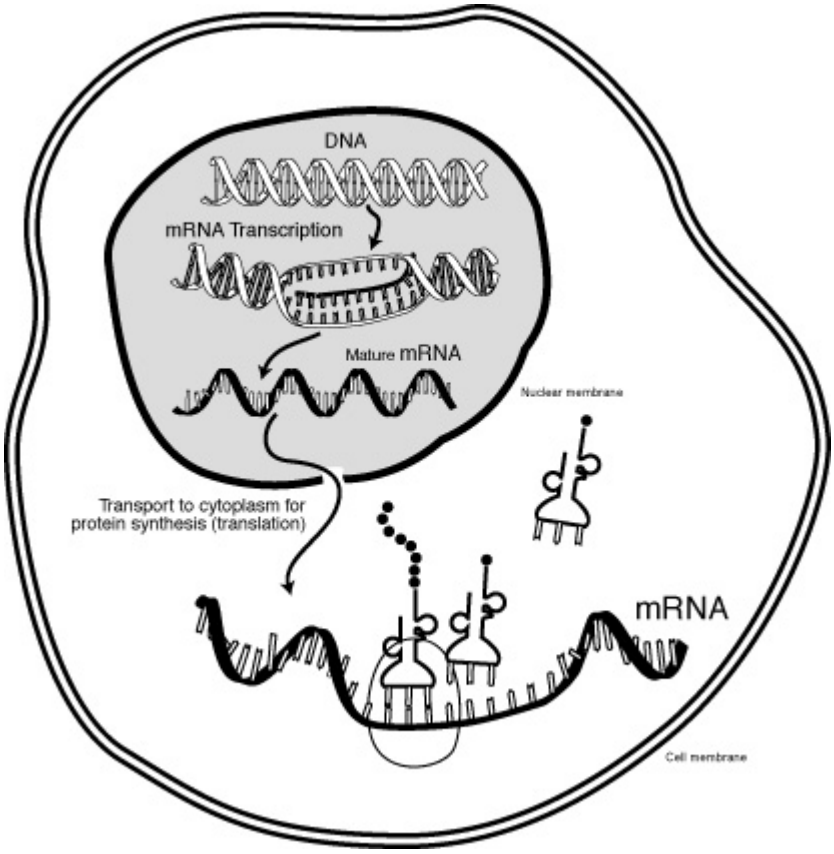

Figura 1: Transcrição, processo onde um mRNA leva a informação de um gene para ser expresso como uma proteína [8].

A Regulação gênica é a habilidade da célula de controlar quando e em que quantidade uma proteína irá ser produzida. Ela é a base da diferenciação celular, da versatilidade e adaptabilidade de qualquer organismo. Logo, entender em que momento um gene é expresso e o que tal expressão irá desencadear é um dos grandes desafios da biologia, medicina e bioinformática atualmente.

3.2. Análise Microarray

Em 2003, o “Projeto Genoma” foi considerado concluído, tendo seqüenciado 94% do DNA do genoma humano. Este foi um grande passo na evolução dos estudos sobre a genética [4], pois evidenciou o potencial do que a Biologia aliada à alta tecnologia pode fazer pela humanidade. Um genoma completo permite que se tenha uma visão global dos genes, porém não só evidencia a

seqüência destes, como também mostra suas co-localizações e relações com fatores regulatórios.

Com tantas descobertas e estudos genômicos, houve uma necessidade de métodos para analisar essas informações. Assim, a Bioinformática como área de pesquisa ganhou mais espaço e força, uma vez que tem como meta a utilização, integração e interpretação dos dados de genômica.

Nesse cenário, surgiu a plataforma de Microarray, em meados da década de 90 (DeRisi et al., 1996; Schena et al., 1995; Shalon et al., 1996), capaz de analisar a expressão gênica de milhares de genes simultaneamente (como uma "foto" da expressão de milhares de genes de um determinado instante), com a vantagem de permitir a comparação entre duas populações de mRNA (RNA mensageiro- ácido ribonucléico mensageiro), e com o objetivo de entender os mecanismos de transformação molecular dos tecidos analisados. Catalogando em banco de dados as diferenças entre as expressões gênicas das amostras estudadas, pode-se comparar a expressão dos genes em diversas condições biológicas, como por exemplo, em um tecido sadio versus em um patológico.

3.2.1. Processo de Análise de Microarray

Existem diversos métodos diferentes para cada etapa da análise de Microarray, porém para o nosso estudo não é interessante nos prender a detalhes de cada metodologia. Por isso, descreveremos agora o processo de análise de uma forma genérica, apenas citando algumas particularidades.

3.2.1.1. Obtenção do Microarray

A análise consiste em colocar amostras específicas de DNA que representam genes, que podem ser cDNAs ou oligonucleotídeos, em uma lâmina de

vidro ou em uma membrana de nylon, de dimensões mínimas. Para garantir a precisão ao se utilizar uma lâmina de vidro, é necessário o auxilio de uma estrutura robótica que, usando agulhas de impressão, deposita o material nas posições específicas da lâmina, conhecida como spots. As fitas de DNA são colocadas em pequenos poros da lâmina com a ajuda de adesivos de superfície como aminosilanos ou poli-l-lisina para maior adesão. Os mRNA são retirados dos tecidos a serem estudados, transformados em cDNA e marcados com fluocromos. Em um experimento típico de duas cores são usados o cy5, corante vermelho, e o cy3, corante verde. A lâmina é depositada em uma solução de hibridização. Neste processo, caso ocorra a complementaridade entre o cDNA fixado na lâmina e a amostra, teremos a proporção da expressão gênica de cada gene referenciado em um determinado spot.

3.2.1.2. Escaneamento da lâmina

A lâmina é escaneada usando dois comprimentos de ondas para diferenciar os fluocromos usados na marcação. Para ficar apta a ser estatisticamente analisada essa imagem é trabalhada em um programa de processamento de imagens. Diversos processos de análise de imagens são usados.

- O background (fundo da imagem) tem sua intensidade diminuída para não se confundir com os marcadores usando algumas transformações padrão. (Ishi et al., 2000; Schuchhardt et al., 2000; Kepler et al., 2002)
- Eliminam-se spots (cada ponto da imagem que representa a hibridização do RNAm com a fita de DNA) que tenham problemas de qualidade.
- É feita a identificação e o alinhamento dos spots.
- A expressão de cada spot é quantificada. Essa quantificação pode ser feita de diversas maneiras, podendo variar muito dependendo da escolha. O método a ser usado depende das suposições feitas sobre o modelo.
- Muitas vezes um mesmo gene está referenciado em mais de um spot para garantir que o valor estimado de expressão seja confiável.

- Invertem-se os canais atribuídos às amostras dos tecidos para compensar qualquer interferência indesejada.

Como resultado final se tem uma matriz de números reais que refletem o valor da expressão dos genes, cujas linhas são os genes e as colunas são as amostras.

3.2.2. Métodos estatísticos aplicados nos dados de Microarray

Com os dados da expressão gênica dos tecidos a serem estudados em mãos, existem diversas técnicas a serem aplicadas. Citaremos algumas explicando brevemente sua funcionalidade.

3.2.2.1. Normalização

Transformação usada para ajustar os valores medidos tirando interferências não relacionadas propriamente com as amostras como, quantidade de mRNA inicial, diferenças causadas pelos fluorescentes usados e possíveis erros de medição da expressão gênica.

O método de normalização usado é escolhido levando em consideração o modelo estatístico utilizado. Indiferentemente do método escolhido deve-se escolher um conjunto de genes que servirá de referência na normalização. Existem três principais critérios para essa escolha.

3.2.2.1.1. Utilizando todos os genes da lâmina

As análises geralmente são muito específicas. Logo, não é esperada diferença de expressão entre todos os genes, apenas entre um grupo pequeno. Tendo essa expressão constante na maioria dos genes, podemos usá-los como indicadores da intensidade relativa dos dois canais.

3.2.2.1.2. Genes expressados constantemente

Identifica-se um grupo de genes, chamados housekeeping, que pode ser considerado com expressão constante indiferente das condições. Existe uma grande dificuldade nessa identificação, mas pode ser feita para condições experimentais particulares.

3.2.2.1.3. Genes expressados controles

Controle Spiked-in consiste em adicionar genes previamente escolhidos na lâmina. Se eles forem colocados em quantidades iguais eles não apresentarão diferença na intensidade e poderão ser usados na normalização.

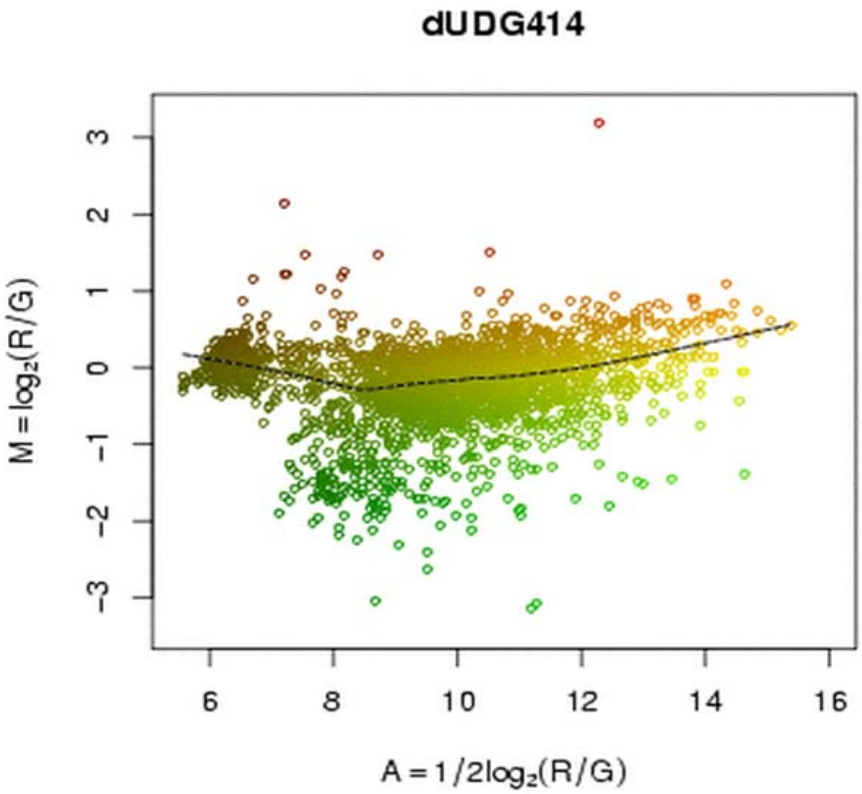

Figura 2: Dado sem Normalização.

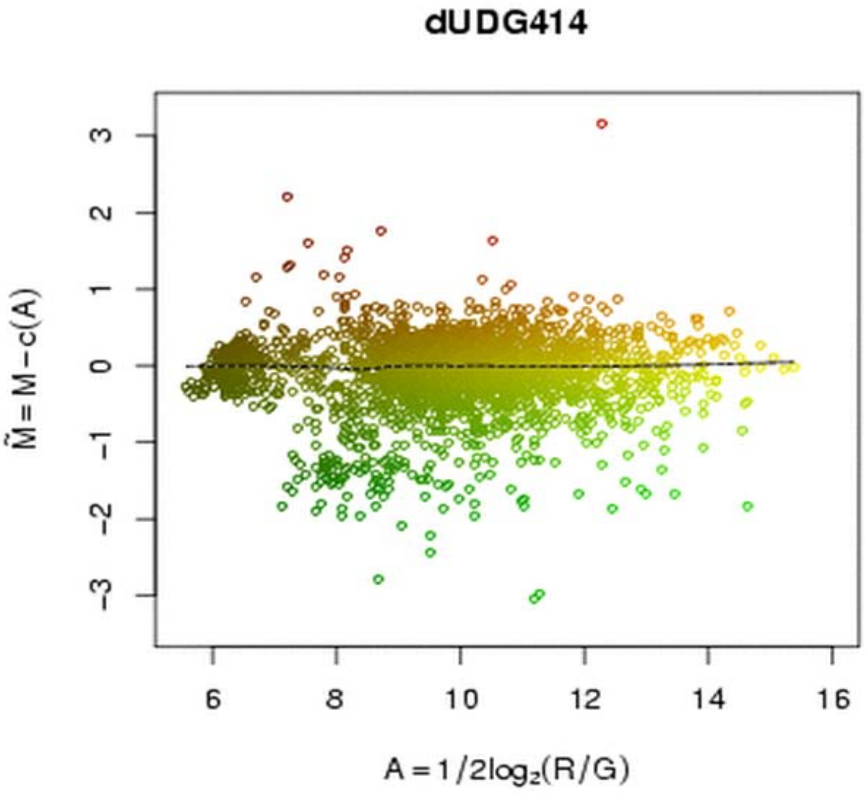

Figura 3: Dado com normalização global (lowess).

3.2.3. Agrupamento

Técnica usada para dividir os dados em grupos usando critérios tanto quantitativos quanto qualitativos. Primeiro, define-se quais serão os objetos (pontos) a serem considerados, podendo ser os genes ou amostras de pacientes. Após o objeto definido resta escolher a função que será usada para calcular a distância entre os pontos, quais pontos serão usados para medir a distância entre os grupos e quantos grupos serão formados.

Um dos métodos de agrupamento muito usado é o Agrupamento Hierárquico, esse método não pressupõe que cada objeto pertence somente a um grupo. Os objetos são ordenados de forma hierárquica onde a proximidade entre todos os objetos é mostrada. No primeiro nível, agrupam-se os objetos com similaridade máxima e depois, a cada nível, juntam-se os grupos segundo o critério de distância escolhido.

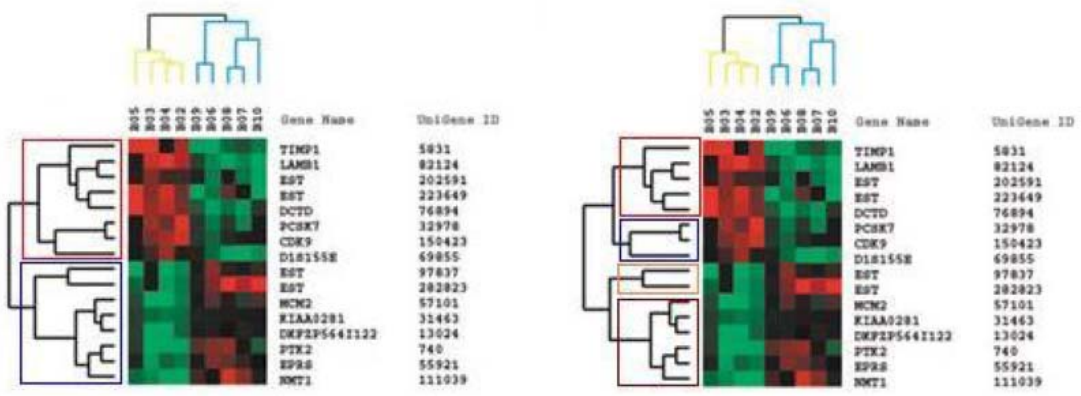

Figura 4: Duas árvores hierárquicas completas. A árvore pode ser cortada em diversos níveis para se obter diferentes números de grupos. A imagem da esquerda mostra dois grupos enquanto a da direta está divida em 4 grupos.

3.2.4. Classificação

É de interesse do estudo determinar genes ou grupos de genes que possam distinguir bem amostras de tipo diferentes. É necessário um grande número de amostras para se definir um critério de classificação eficiente.

4. MaigesPack

4.1. Introdução

No capítulo anterior, foi apresentado o processo de análise de microarray, o qual se mostrou complexo e constituído de diversas etapas experimentais de natureza diferente.

Os dados de expressão gênica obtidos desta análise podem sofrer modelagem estatística com o intuito de extrair informações genéticas mais precisas e detalhadas. Para isso, é necessário um ambiente computacional que integre métodos matemáticos e estatísticos de análise já disponíveis na literatura e que mantenha confiabilidade, reprodutibilidade e robustez às análises efetuadas.

É preciso também que este ambiente seja simples, modularizado e bem documentado, pois cada etapa precisa ser bem compreendida e controlada pelo usuário que irá utilizar este ambiente.

Neste contexto, surgiu o pacote denominado MaigesPack.

4.2. Definição

O MaigesPack é um pacote de arquivos com métodos matemáticos e estatísticos, escritos em linguagem de programação estatística R e em C, que desenvolve um ambiente computacional para análise de dados de microarray, integra estes com diversos algoritmos já desenvolvidos pelo projeto Bioconductor e implementa outros métodos de análise. Ele foi criado por Gustavo Henrique Esteves e está disponível no site do projeto Bioconductor ().

4.3. Estrutura

Neste novo ambiente, foram definidas classes de objetos para armazenar desde conjuntos de dados de microarray bruto até os resultados das análises específicas.

A figura abaixo contém um grafo representando a estrutura e as classes, que serão definidas na próxima seção.

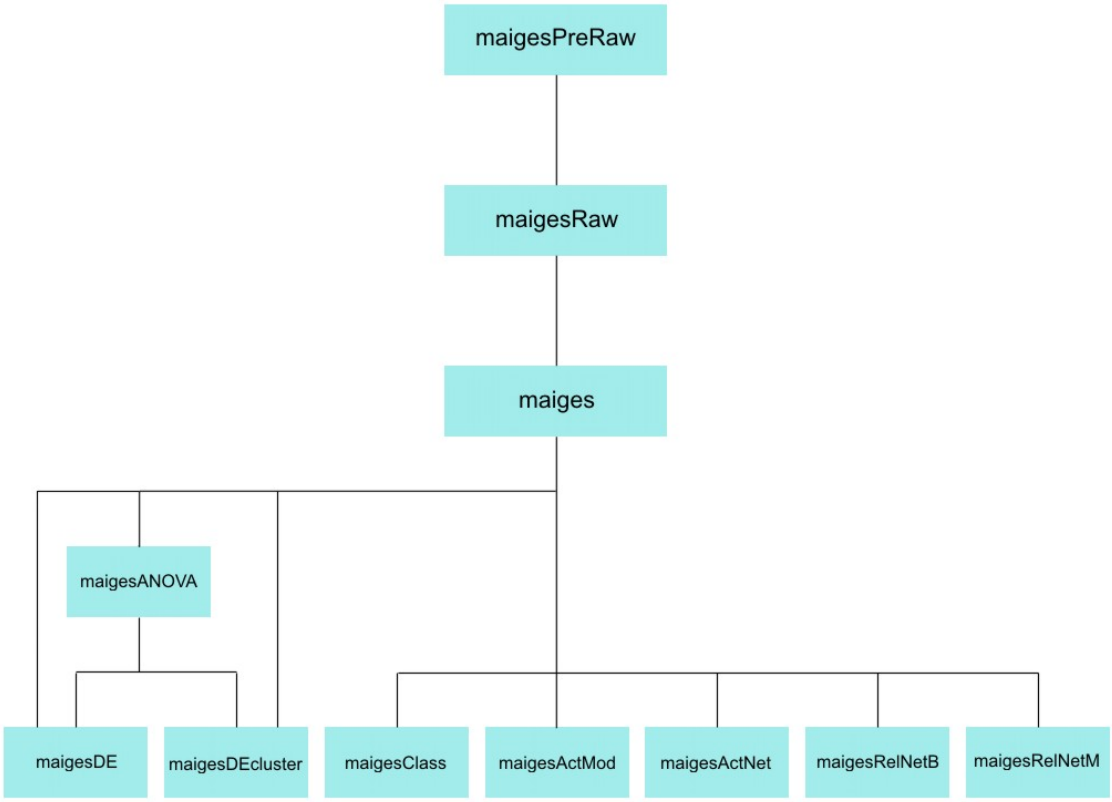

Figura 5: grafo da estrutura de classes do pacote MaigesPack

Todos os dados numéricos e informações de interesse para o conjunto a ser analisado encontram-se na primeira etapa do processo, na classe maigesPreRaw. Aqui, os dados sofrem análises exploratórias em busca de possíveis problemas com a carga dos dados e mesmo problemas experimentais. Quando verificado que algum spot não cumpre critérios de qualidade exigidos atualizam-se os campos referentes aos spots ruins.

Através de um método específico denominado create.maigesRaw, que recebe como parâmetro um objeto desta classe, variáveis de caracteres que especificam os campos numéricos a serem utilizados e dois argumentos de texto contendo a especificação dos rótulos de genes que serão usados para mapear os grupos e a rede gênica, o objeto da classe maigesPreRaw é transformado em um objeto da classe seguinte: maigesRaw.

Nesta classe, são aplicadas funções de normalização que utilizam os pacotes limma, marray e OLIN, já implementados no Bioconductor, seja para criar novas funções como o Mnorm.scaleLimma e o Mnorm.scaleMarray, ou apenas simplesmente para utilizar algum método já pronto como Mnorm.OLIN.

Após a aplicação das normalizações, são obtidos os dados da classe maiges, onde os objetos sofrem análises específicas para: genes diferencialmente expressos (maigesDE e maigesDEcluster), classificação (maigesClass, maigesActMod e maigesActNet) e redes de relevância (maigesRelNetB e maigesRelNetM).

A classe maigesANOVA é uma extensão da classe maiges.

4.4. Classes de objetos

Nesta seção, são apresentadas as características de cada classe do MaigesPack.

4.4.1. MaigesPreRaw

Como dito anteriormente, nesta classe estão todos os dados e informações para o conjunto a ser analisado. Ela possui listas que recebem campos numéricos a partir da análise de imagens, vetores de texto que especificam os grupos gênicos a serem avaliados, grafos que definem as redes de regulação de interesse, configuração das lâminas e rótulos para os genes e amostras de estudo.

Nesta primeira etapa, caso existam problemas, eles são informados em um campo lógico especificando spots ruins e campo de texto para informações adicionais.

4.4.2. MaigesRaw

Esta classe possui matrizes que: armazenam os valores de intensidade de sinal, indexam os spots a serem utilizados para a normalização e indexam genes pertencentes a diferentes grupos gênicos.

4.4.3. Maiges

Os objetos contidos nesta classe são gerados a partir da normalização aplicada sobre os objetos da classe maigesRaw. Aqui, são armazenados valores de razão de intensidade média dos spots em escala logarítmica e desvio padrão e intervalos de confiança para os valores normalizados.

- **MaigesANOVA**: é uma extensão da classe maiges que armazena matrizes de planejamento e de contrastes utilizadas para o ajuste de um modelo de análise de variância e para estimação de parâmetros.
- **MaigesDE**: contém resultados das análises de busca de genes diferencialmente expressos (genes DE)
- **MaigesDEcluster**: trata-se de uma extensão da classe anterior porém com matriz de valores de razão
- **MaigesClass**: possui resultados de análises de discriminação ou de classificação
- **MaigesActMod**: armazena as saídas das análises de classificação funcional de grupos gênicos
- **MaigesActNet**: contém resultados de análises de classificação funcional de redes de regulação gênica

- **MaigesRelNetB**: possui resultados de análises de redes de relevância
- **MaigesRelNetM**: armazena resultados de análises de redes de relevância usando o modelo aprimorado discutido

5. Redes de Relevância

5.1. Introdução

Muitos conjuntos de dados são planejados para comparar tipos de dados específicos. Com isso, a utilização de medidas de expressão gênica para a avaliação dinâmica da rede torna-se um método não muito próprio e adequado. Neste contexto, surge o conceito de redes de relevância.

Entre novembro de 1998 e fevereiro de 1999, Atul J. Butte e Isaac S. Kohane elaboraram uma lista de todas as análises laboratoriais clínicas feitas nos pacientes do Hospital Infantil de Boston, Massachusetts. Desta lista, foram excluídas as análises que não resultaram em um valor numérico após três ou mais vezes testes laboratoriais.

Para cada paciente, a lista de resultados do laboratório era colocada em uma tabela (figura 6), onde as colunas representavam os tipo de análises laboratoriais e as linhas, os intervalos de tempo. Os resultados do laboratório com sobreposição de intervalos válidos do tempo eram colocados na mesma linha. Assim, testes que não foram executados simultaneamente, mas que possuíam os mesmos intervalos de tempos válidos, poderiam ser comparados um a um na mesma linha. Isso permitiu que testes com intervalos válidos mais longos fossem comparados com os testes com intervalos válidos mais curtos.

Tais procedimentos reduziram a diferença dos testes rotineiros feitos similarmente que eram comparados um a um. As tabelas resultantes eram escassas, na qual nem todo teste laboratorial estava presente em toda interseção de intervalo válido. Os valores desconhecidos foram tratados diferentemente dos valores nulos.

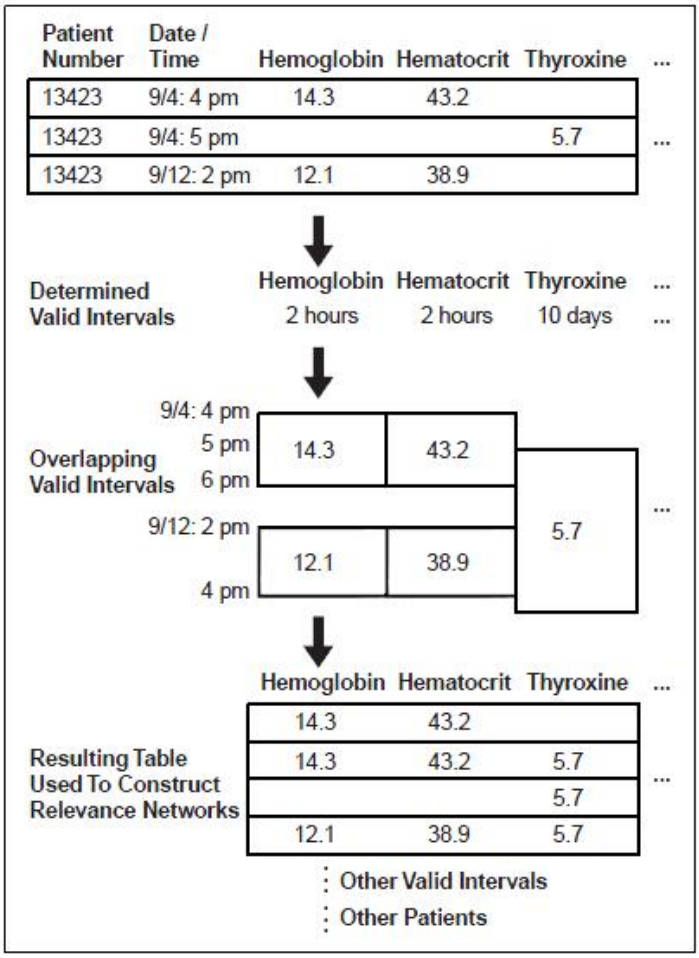

Figura 6: Motodologia para criação da tabela de sobreposição dos resultados laboratoriais[17].

5.2. Construção

Após gerar as tabelas descritas no item anterior utilizando a metodologia indicada pela figura 1, Butte e Kohane criaram um grafo com esses dados. Para isso, eles interpretaram cada coluna da tabela como uma variável separada e as analisaram, par a par.

A cada análise, era elaborado um modelo linear entre as variáveis analisadas e armazenava-se o coeficiente de correlação, r, que media a qualidade do ajuste do modelo linear, e o número de intervalos de cruzamento, n.

O resultado obtido mostrava que quase todas variáveis estavam conectadas às outras por um modelo linear de qualidade variável. Isto significa que o n e o r^2 de cada correlação variaram bastante.

Para dividir este grafo quase completamente conectado, foram escolhidos limiares iniciais n' e $r^{2'}$. Os modelos lineares com correlação menor $r^{2'}$ ou modelos com número de intervalos de interseção menores que o n' foram excluídos. Esse processo gerou sub-grafos denominados redes de relevância.

5.3. Utilização e Considerações

A aplicação da técnica de redes de relevância permite que sejam elaboradas hipóteses fisiológicas, fisiopatológicas e matemáticas sobre os testes e genes utilizados.

Esta técnica permite também encontrar resultados mais precisos e consistentes de acordo com o propósito da análise. Para isso, basta alterar os limiares iniciais n e r^2. Como conseqüência, o número de sub-grafos obtidos também será alterado.

Embora abrangente e útil, as redes de relevância possuem algumas limitações:

- Ligações representativas: cada ligação representa uma hipótese de relação linear entre duas variáveis.
- Variáveis categóricas que podem ou não estar presentes, mas que influenciam relações entre variáveis.
- Variáveis contínuas que estão em um modelo, mas agem como variáveis discretas.
- Variáveis contínuas que estão ocultas de um modelo.
- Variáveis que influenciaram direta ou indiretamente a tendência da seleção.

Embora esta técnica funcione bem em matrizes esparsas, seu desempenho pode ser melhorado para matrizes com menos dados faltantes.

O método proposto por Butte possui também uma outra importante restrição: só é aplicável com um único tipo de tecido por vez. Para comparar dois ou mais tipos biológicos, é necessário analisar visualmente os grafos gerados ou, ainda, procurar as diferenças nas redes de relevância utilizando resultados da teoria dos grafos. Porém, estes recursos não avaliam a significância dos resultados obtidos e seriam custosos para um conjunto de dados muito grandes.

A versão 1.4.0 do pacote MaigesPack, que está disponível no site do projeto Bioconductor[18], contém duas implementações para construção das redes de relevância: relNetworkB e relNetworkM. A primeira, é o método proposto por Butte. Já a segunda, trata-se deste mesmo método, porém com alterações que contornem a restrição descrita anteriormente.

No próximo capítulo, detalharemos a implementação dos algoritmos de redes de relevância.

6. Implementação

6.1. Introdução

A construção de redes de relevância é considerada adequada para a avaliação de redes de expressões gênica interagentes. O pacote maigesPack (versão 1.4) disponibilizada no site do Bioconductor [18] contém funções que possibilitam a construção dessas redes de relevância. Mostraremos como essas funções estão implementadas atualmente e como elas foram modificadas para se adequar as especificações dos usuários.

6.2. Processo

A construção das redes de relevância usando as funções disponíveis do pacote do maigesPack, na versão 1.4, pode ser feito através da função *relNetworkB* ou da função *relNetworkM.*

6.2.1. relNetworkB

A função relNetworkB usa o método proposto por Butte, que usa funções básicas do R para estimar os valores da correlação entre dois genes de interesse. Além das funções básicas do R, o relNetworkB usa outras funções existentes no pacote do maigesPack tais como: robustCorr, bootstrapCor, cor, MI e bootstrapMI. Essa função recebe os seguintes parâmetros:

1. **data:** é um objeto da classe maiges

2. **gLabelID:** Identificação do gene (nome, rótulo ou ID)
3. **sLabelID:** identificação da amostra a ser usada
4. **geneGrp:** caracter, string (ou índice numérico) especificando o grupo de genes onde será calculada a correlação entre eles. Se for nulo (junto com o *path*) todos os genes são usados.
5. **path:** caracter, string (ou índice numérico) especificando o percurso onde será calculada a correlação entre eles. Se for nulo (junto com o *geneGrp*), todos os genes são usados.
6. **samples:** um vetor de caracteres especificando o grupo a ser comparado.
7. **type:** tipo de estimativa a ser calculado. Pode ser 'Rpearson' (padrão), 'pearson', 'kendall', 'spearman' ou 'MI'.
8. **bRep:** número de "bootstraps" para correlação dos valores.
9. **... :** parâmetros adicionais para a função *robustCorr* ou *cor*

Os parâmetros que essa função recebe e o objeto que é devolvido pela função para usuário dar continuidade na análise são as únicas informações que o usuário tem conhecimento. Os parâmetros são os mencionados anteriormente e o retorno da função é um objeto do tipo *maigesRelNetB* da classe *maiges* que contém outros objetos como: tabela W, coeficientes de correlação, os p-valores, os boot máximos, data da construção desse objeto, o tipo de estimativa usado, os rótulos e a versão do pacote usado para realizar esses cálculos.

Para ilustrar o resultado da aplicação desta função, apresentaremos dois exemplos de figuras geradas após a execução do relNetworkB sobre a base de dados ‘gastro’, contida na versão 1.4.0 do pacote MaigesPack.

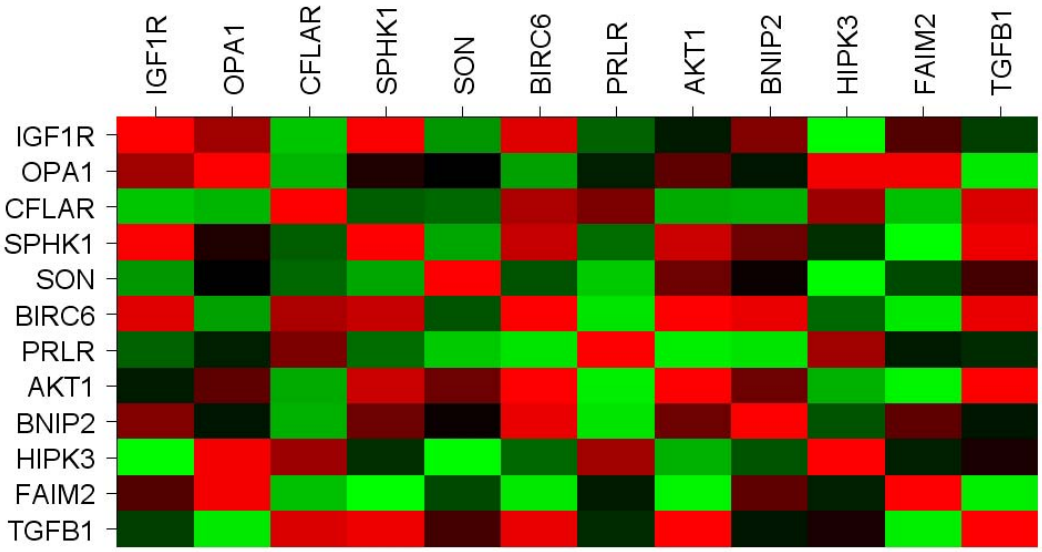

Figura 7: imagem dos valores de correlação para todos os pares de genes do banco de dados 'gastro', através da chamada de função image(gastro.net) onde gastro.net = relNetworkB(gastro.summ, sLabelID = "Tissue", samples = "Neso", geneGrp = 1)

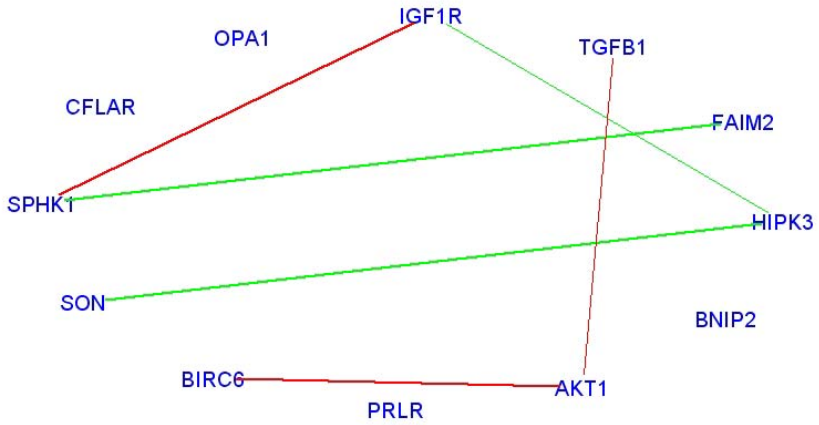

Figura 8: grafo das correlações de pares de genes com p-valores menores que 0,05 do banco de dados 'gastro', através da chamada da função plot(gastro.net, cutPval = 0.05) onde gastro.net = relNetworkB(gastro.summ, sLabelID = "Tissue", samples = "Neso", geneGrp = 1)

Todas as operações internas, como a remoção de genes e amostras que não estão nomeadas ou estão com nome ‘NA’, do objeto *data* (recebido como parâmetro da função) e cálculo dos coeficientes de correlação, por exemplo, não são transparentes para o usuário.

O objetivo do trabalho é realizar a modularização do código possibilitando ao usuário ter controle sobre as operações executadas dentro do método relNetworkB. Com isso, o usuário poderá utilizar os dados disponíveis da melhor maneira, adequando-o para o tipo de análise desejado. O usuário será capaz de modelar os dados da maneira que achar conveniente para a análise, podendo, por exemplo, remover ou substituir dados com valor ‘NA’ por algum outro valor que acharem conveniente, etc. Essas adaptações dos dados ficaram sob responsabilidade dos chamados *adaptadores*, sendo que cada tratamento específico dos ‘NA’, por exemplo, terá seu adaptador correspondente.

6.2.1.1. Implementação Atual

A implementação atual do relNetwork não permite intervenções do usuário no meio do processo de construção das redes de relevância, ou seja, o usuário passa os parâmetros na chamada da função e aguarda a saída. Abaixo segue os principais passos executados dentro dessa função

6.2.1.1.1. Obtendo genes de acordo com o grupo de genes

```
allGenes <- getLabels(data, gLabelID, FALSE)
allSamples <- getLabels(data, sLabelID)
```

6.2.1.1.2. Remoção de genes e amostras que não foram nomeados ou que são ‘NA’ que estão no objeto data.

```
idxGtmp <- which(allGenes != "" & !is.na(allGenes))
idxStmp <- which(allSamples != "" & !is.na(allSamples))
data <- data[idxGtmp, idxStmp]
```

6.2.1.1.3. Construção da tabela que será usada para armazenar os coeficientes de correlação

```
table <- calcW(data)
if(is.null(geneGrp) & is.null(path))
  genes <- allGenes
else if(!is.null(geneGrp)) {
  if(!is.numeric(geneGrp)) {
    geneGrp <- which(colnames(data@GeneGrps) == geneGrp)
    genes <- allGenes[data@GeneGrps[, geneGrp]]
  }
  else
    genes <- allGenes[data@GeneGrps[, geneGrp]]
}
else if(!is.null(path)) {
  genePath <- which(nodes(data@Paths[[path]]) %in% getLabels(data,
                    @Paths[[1]], FALSE))
  genes <- allGenes[genePath]
}
```

6.2.1.1.4. Construção da tabela que será usada e remoção de eventuais replicações de informação

```
tableTmp <- NULL
for(i in unique(genes)) {
  idx <- allGenes %in% i
  if(sum(idx) > 1)
    tableTmp <- rbind(tableTmp, apply(table[idx, ], 2, mean, na.rm=TRUE))
  else
    tableTmp <- rbind(tableTmp, table[idx, ])
}
```

6.2.1.1.5. Cálculo de coeficientes de correlação, p-valores e boot máximos de acordo com o tipo de estimativa

```
if(type == "Rpearson") {
  corObj <- robustCorr(tableTmp[, sample1], ...)[[1]]
  pValue <- bootstrapCor(tableTmp[, sample1], bRep=bRep, type=type,
                         "p-value")
  bootM <- bootstrapCor(tableTmp[, sample1], bRep=bRep, type=type, "max")
}
else if(type == "MI") {
  corObj <- MI(tableTmp[, sample1], ...)
  pValue <- bootstrapMI(tableTmp[, sample1], bRep=bRep, "p-value")
  bootM <- bootstrapMI(tableTmp[, sample1], bRep=bRep, "max")
}
else {
  corObj <- cor(t(tableTmp[, sample1]), method=type, ...)
  pValue <- bootstrapCor(tableTmp[, sample1], bRep=bRep, type=type,
                          "p-value")
  bootM <- bootstrapCor(tableTmp[, sample1], bRep=bRep, type=type, "max")
}
```

6.2.1.1.6. Obtendo informações das versões do R e do pacote que está sendo usada, nesse caso o maigesPack

```
tmp <- sessionInfo()
vInfo <- list()
vInfo$R.version <- tmp$R.version$version.string
vInfo$BasePacks <- tmp$basePkgs
tmp1 <- NULL
for (i in 1:length(tmp$otherPkgs))
  tmp1 <- c(tmp1, paste(tmp$otherPkgs[[i]]$Package, "version",
             $otherPkgs[[i]]$Version))

vInfo$AddPacks <- tmp1
```

6.2.1.1.7. Definindo e construção do objeto de retorno da função

```
result <- new("maigesRelNetB", W=tableTmp, Corr=corObj, Pval=pValue,
               maxB=bootM, Date=date(), type=paste(samples,
               collapse=", "), Slabel=sLabelID, V.info=vInfo)
```

6.2.1.2. Implementação Nova

A implementação nova do relNetwork permite ao usuário um maior controle sobre as operações que serão realizadas para análise de dados e criação das redes de relevância. O usuário pode inserir quaisquer novas operações entre as operações já existentes. A implementação abaixo demonstra como foi realizada a divisão das operações, em funções distintas, e como deve ser o uso dessas novas funcionalidades caso o usuário queira reproduzir a construção anterior.

6.2.1.2.1. Obtendo genes de acordo com o grupo de genes

A obtenção dos genes é realizada dentro da função *getGenes*. A função getGenes é responsável pela seleção de genes de acordo com um grupo de genes definido pelo usuário. Além de obter os genes a partir do objeto *data*, essa função realiza alguns testes para verificar se a solicitação dos genes procede, caso contrário, uma mensagem de erro é passada para o usuário. O parâmetro *geneGrp* da função getGenes é quem recebe o grupo de genes definido pelo usuário. Abaixo segue o trecho de código da função getGenes correspondente a esse passo.

```
getGenes <- function(data=NULL, gLabelID="GeneName", geneGrp=NULL,
                     path=NULL, ...) {

if(is.null(data))
  stop("The data MUST be specified!!!")

if(!is.null(geneGrp) & !is.null(path))
  stop("You must specify only one of geneGrp and path, or leave both
        NULL.")

## Getting gene
```

```
allGenes <- getLabels(data, gLabelID, FALSE)

allGenes <- getLabels(data, gLabelID, FALSE)
allGenes[data@BadSpots] <- paste(allGenes[data@BadSpots], "(*)")

if(is.null(geneGrp) & is.null(path))
  genes <- allGenes
else if(!is.null(geneGrp)) {
  if(!is.numeric(geneGrp)) {
    geneGrp <- which(colnames(data@GeneGrps) == geneGrp)
    genes <- allGenes[data@GeneGrps[, geneGrp]]
}
else
  genes <- allGenes[data@GeneGrps[, geneGrp]]
}
else if(!is.null(path)) {
  genePath <- which(nodes(data@Paths[[path]]) %in% getLabels(data,
                  data@Paths[[1]], FALSE))
  genes <- allGenes[genePath]
}

if (length(genes) < 2)
  stop("There is less than 2 elements in this group of genes.")
  return(genes)
}
```

6.2.1.2.2. Remoção de genes e amostras que não foram nomeados ou que são ‘NA’ que estão no objeto data.

A remoção dos genes e amostras que não foram nomeados ou que são ‘NA’ foi colocada em uma função específica, um adaptador, que irá tratar as informações do objeto data. Com as linhas da tabela ‘data’ representando genes e as colunas as amostras.

```
removeNaNames <- function(data =NULL, gLabelID="GeneName",
                          sLabelID="Classification"){

  ## Getting gene and sample labels
  allGenes <- getLabels(data, gLabelID, FALSE)
  allSamples <- getLabels(data, sLabelID)
```

```
  ## removing genes and samples not named or NA from the data object
  idxGtmp <- which(allGenes != "" & !is.na(allGenes))
  idxStmp <- which(allSamples != "" & !is.na(allSamples))
  data <- data[idxGtmp, idxStmp]

  return(data)
}
```

6.2.1.2.3. Construção da tabela que será usada para armazenar os coeficientes de correlação

A construção da tabela com os valores de W para os genes parametrizados, que será usada para se calcular os coeficientes de correlação, foi separada em uma função específica chamada *relNetworkBuildTable*. O processo de construção da tabela consiste em algumas validações para que a tabela resultante seja válida para dar continuidade ao processo. Caso haja alguma irregularidade nos parâmetros passados na chamada da função, uma mensagem será apresentada ao usuário. Os parâmetros que essa função recebe são: data, gLabelID, sLabelID, samples, type, processType, geneA, geneB e genes, sendo que a descrição dos parâmetros data, gLabelID, sLabelID e samples foram apresentados no item 5.1.2.1. O parâmetro processType é responsável por definir se o processo a ser executado será o baseado no método de Butte & Kohanne ou se será baseado no método de informações mútuas. Os parâmetros geneA e geneB são os parâmetros que definem os genes que serão analisados e caso ambos parâmetros sejam NULL, a análise será com todos os genes da lista de genes ‘genes’.

```
relNetworkBuildTable <- function(data=NULL, gLabelID="GeneName",
                                 sLabelID, samples=NULL, type="Rpearson",
                                 processType="B", geneA=NULL,
                                 geneB=NULL,genes=NULL, ...) {

  ## Doing a simple test...
  if(is.null(data))
    stop("The data MUST be specified!!!")

  if(!(type %in% c("Rpearson", "pearson", "kendall", "spearman", "MI")))
    stop("Parameter 'type' must be 'Rpearson', 'pearson', 'kendall',
         'spearman' or 'MI'.")
```

```
if(is.null(genes))
  stop("You must enter a valid list of genes")

allGenes <- getLabels(data, gLabelID, FALSE)
allGenes[data@BadSpots] <- paste(allGenes[data@BadSpots], "(*)")
allSamples <- getLabels(data, sLabelID)
if(processType == "B") {
  ## Getting the first 2 sample types to use in case of NULL samples
     parameter
  if(is.null(samples))
    samples <- unique(allSamples)[1]

  ## Finding indexes of the samples specified and respective lengths
  idxTmp <- allSamples %in% samples
}
else {
  if(is.null(samples)) {
    samples <- list(unique(allSamples)[1], unique(allSamples)[2])
      names(samples) <- unique(allSamples)[1:2]
  }
  ## Finding indexes of the samples specified and respective lengths
  idxTmp <- allSamples %in% c(samples[[1]], samples[[2]])
}

## Constructing the table to be used
table <- calcW(data)
table <- table[, idxTmp]
sTypes <- allSamples[idxTmp]
tableTmp = NULL

## Construct the table with 2 genes each time
if(is.null(geneA) & is.null(geneB)){
  ##Constructing the table to be used, averaging eventual replicates...
  for(i in unique(genes)) {
    idx <- allGenes %in% i

    if(sum(idx) > 1)
      tableTmp <- rbind(tableTmp, apply(table[idx, ], 2, mean,
                        na.rm=TRUE))
    else
```

```
        tableTmp <- rbind(tableTmp, table[idx, ])
    }
    ## Naming tableTmp
    rownames(tableTmp) <- unique(genes)
    colnames(tableTmp) <- sTypes
  }
  else if(!is.null(geneA) & !is.null(geneB)){
    idx <- allGenes %in% geneA
    if(sum(idx) > 1)
      tableTmp <- rbind(tableTmp, apply(table[idx,], 2, mean, na.rm=TRUE)
    else
      tableTmp <- rbind(tableTmp, table[idx, ])

    idx <- allGenes %in% geneB

    if(sum(idx) > 1)
      tableTmp <- rbind(tableTmp, apply(table[idx, ], 2, mean,
                        na.rm=TRUE))?index
    else
      tableTmp <- rbind(tableTmp, table[idx, ])

    ## Naming tableTmp
    id <- genes %in% c(geneA,geneB)
        rownames(tableTmp) <- unique(genes[id])
        colnames(tableTmp) <- sTypes

    if (processType == "M"){
      colnames(tableTmp)[sTypes %in% samples[[1]]] <- names(samples)[1]
      colnames(tableTmp)[sTypes %in% samples[[2]]] <- names(samples)[2]
    }
  }
  else
    stop("Enter two genes or leave both NULL!!")

  return(tableTmp)
}
```

6.2.1.2.4. Construção da tabela que será usada e remoção de eventuais replicações de informação

Como esse processo é sempre realizado para a construção das redes de relevância, então ele foi inserido dentro da execução do passo anterior, ou seja, dentro da função relNewtorkBuildTable.

6.2.1.2.5. Cálculo de coeficientes de correlação, p-valores e boot máximos de acordo com o tipo de estimativa

Foram criadas funções específicas para o cálculo de cada um dos tipos de valores. Foram criadas as seguintes funções: bootMCalc, corCalc e pValCalc que calculam, respectivamente, o boot máximo, os coeficientes de correlação e os p-valores. Além de essas funções serem usadas dentro da função *relNetworkBWrite* (explicação a seguir), elas também poderão ser chamadas fora do escopo da construção das redes de relevância.

6.2.1.2.5.1. Função bootMCalc

Essa função é responsável pelo cálculo dos coeficientes de correlação robusta, minimizando a influência do dado discrepante.

```
bootMCalc <- function(type ="Rpearson", table, sample, bRep) {


  ## Calculating the correlation coefficients
  if(type == "Rpearson") {
    bootM <- bootstrapCor(table[, sample], bRep=bRep, type=type, "max")
  }
  else if(type == "MI") {
    bootM <- bootstrapMI(table[, sample], bRep=bRep, "max")
  }
  else {
    bootM <- bootstrapCor(table[, sample], bRep=bRep, type=type, "max")
  }
  return(bootM)
}
```

6.2.1.2.5.2. Função corCalc

Essa função é responsável pelo cálculo dos coeficientes de correlação.

```
corCalc <- function(type ="Rpearson", table, sample){

  ## Calculating the correlation coefficients
  if(type == "Rpearson") {
    corObj <- robustCorr(table[, sample])[[1]]
  }
  else if(type == "MI") {
    corObj <- MI(table[, sample])
  }
  else {
    corObj <- cor(t(table[, sample]), method=type)
  }
  return(corObj)
}
```

6.2.1.2.5.3. Função pValCalc

Essa função é responsável pelo cálculo dos p-valores.

```
pValCalc <- function(type ="Rpearson", table, sample, bRep){

  ## Calculating the correlation coefficients
  if(type == "Rpearson") {
    pValue <- bootstrapCor(table[, sample], bRep=bRep, type=type,
    "p-value")
  }
  else if(type == "MI") {
    pValue <- bootstrapMI(table[, sample], bRep=bRep, "p-value")
  }
  else {
    pValue <- bootstrapCor(table[, sample], bRep=bRep, type=type,
    "p-value")
  }
  return(pValue)
}
```

6.2.1.2.6. Obtendo informações das versões do R e do pacote que está sendo usada, nesse caso o maigesPack

Como esse processo é sempre realizado na construção das redes de relevância, então ele foi inserido dentro da execução da função relNetworkBWrite. A função relNewtorkBWrite será explicada logo a seguir.

```
relNetworkBWrite <- function(data=NULL, gLabelID = "GeneName",
                             sLabelID="Classification", samples=NULL,
                             type="Rpearson", bRep=1000, geneA=NULL,
                             geneB=NULL, result=NULL, genes=NULL,
                             tableTmp=NULL, ...) {


  if(!(type %in% c("Rpearson", "pearson", "kendall", "spearman", "MI")))
    stop("Parameter 'type' must be 'Rpearson', 'pearson', 'kendall',
         'spearman' or 'MI'.")


  allSamples <- getLabels(data, sLabelID)

  ## Getting the first 2 sample types to use in case of NULL samples
     parameter
  if(is.null(samples))
    samples <- unique(allSamples)[1]


  if(is.null(result)){

    corObj <- matrix(,length(unique(genes)),length(unique(genes)),TRUE)
    rownames(corObj) <- unique(genes)
    colnames(corObj) <- unique(genes)

    pValue <- matrix(,length(unique(genes)),length(unique(genes)),TRUE)
    rownames(pValue) <- unique(genes)
    colnames(pValue) <- unique(genes)

    bootM <- matrix(,length(unique(genes)),length(unique(genes)),TRUE)
    rownames(bootM) <- unique(genes)
```

```
  colnames(bootM) <- unique(genes)

  ## Picking R and packages version information
  tmp <- sessionInfo()
  vInfo <- list()
  vInfo$R.version <- tmp$R.version$version.string
  vInfo$BasePacks <- tmp$basePkgs
  tmp1 <- NULL
  for (i in 1:length(tmp$otherPkgs))
    tmp1 <- c(tmp1, paste(tmp$otherPkgs[[i]]$Package, "version",
               tmp$otherPkgs[[i]]$Version))

  vInfo$AddPacks <- tmp1



  ## Defining the object to return
  result <- new("maigesRelNetB", Corr=corObj, Pval=pValue,
                maxB=bootM, Date=date(), type=paste(samples,
                collapse=", "), Slabel=sLabelID, V.info=vInfo)
}

## Finding indexes of the samples specified and respective lengths
sample1 <- which(colnames(tableTmp) %in% samples)

corObjTmp <- corCalc(type, tableTmp, sample1)
pValueTmp <- pValCalc(type , tableTmp, sample1, bRep)
bootMTmp <- bootMCalc(type, tableTmp, sample1, bRep)

if(is.null(geneA) & is.null(geneB)){
  result@Corr = corObjTmp
  result@Pval = pValueTmp
  result@maxB = bootMTmp
}
else{
  result@Corr[geneA,geneA] <-  corObjTmp[1,1]
  result@Corr[geneA,geneB] <-  corObjTmp[1,2]
  result@Corr[geneB,geneA] <-  corObjTmp[2,1]
  result@Corr[geneB,geneB] <-  corObjTmp[2,2]

  result@Pval[geneA,geneA] <-  pValueTmp[1,1]
```

```
    result@Pval[geneA,geneB] <-  pValueTmp[1,2]
    result@Pval[geneB,geneA] <-  pValueTmp[2,1]
    result@Pval[geneB,geneB] <-  pValueTmp[2,2]

    result@maxB[geneA,geneA] <-  bootMTmp[1,1]
    result@maxB[geneA,geneB] <-  bootMTmp[1,2]
    result@maxB[geneB,geneA] <-  bootMTmp[2,1]
    result@maxB[geneB,geneB] <-  bootMTmp[2,2]
  }
  if (dim(result@W)[1] == 0)
    result@W <- tableTmp
  else{
    while (dim(tableTmp)[2] != dim(result@W)[2])
      if(dim(tableTmp)[2] > dim(result@W)[2])
        result@W <- cbind(result@W,NA)
      else
        tableTmp <- cbind(tableTmp,NA)

    result@W <- rbind(result@W, tableTmp)
  }
  return(result)
}
```

6.2.1.2.7. Definindo e construção do objeto de retorno da função

Assim como a obtenção das informações das versões do R e do maigesPack, a definição e construção do objeto de retorno da função relNetworkB foi inserida dentro da função relNewtorkBWrite. Além da chamada dessas duas tarefas, o relNewtorkBWrite faz a chamada do cálculo dos coeficientes de correlação, p-valores e boot máximo. O trecho de código na função relNetworkBWrite correspondente à definição e construção do objeto de retorno da função segue abaixo.

```
relNetworkBWrite <- function(data=NULL, gLabelID = "GeneName",
                             sLabelID="Classification", samples=NULL,
                             type="Rpearson", bRep=1000, geneA=NULL,
                             geneB=NULL, result=NULL, genes=NULL,
                             tableTmp=NULL, ...) {
```

```
(...)
  if(is.null(result)){
     (...)
    ## Defining the object to return
     result <- new("maigesRelNetB", Corr=corObj, Pval=pValue,
                   maxB=bootM, Date=date(), type=paste(samples,
                   collapse=", "), Slabel=sLabelID, V.info=vInfo)
  }

  ## Finding indexes of the samples specified and respective lengths
  sample1 <- which(colnames(tableTmp) %in% samples)

  corObjTmp <- corCalc(type, tableTmp, sample1)
  pValueTmp <- pValCalc(type , tableTmp, sample1, bRep)
  bootMTmp <- bootMCalc(type, tableTmp, sample1, bRep)

  if(is.null(geneA) & is.null(geneB)){
    result@Corr = corObjTmp
    result@Pval = pValueTmp
    result@maxB = bootMTmp
  }
  else{
    result@Corr[geneA,geneA] <-  corObjTmp[1,1]
    result@Corr[geneA,geneB] <-  corObjTmp[1,2]
    result@Corr[geneB,geneA] <-  corObjTmp[2,1]
    result@Corr[geneB,geneB] <-  corObjTmp[2,2]

    result@Pval[geneA,geneA] <-  pValueTmp[1,1]
    result@Pval[geneA,geneB] <-  pValueTmp[1,2]
    result@Pval[geneB,geneA] <-  pValueTmp[2,1]
    result@Pval[geneB,geneB] <-  pValueTmp[2,2]

    result@maxB[geneA,geneA] <-  bootMTmp[1,1]
    result@maxB[geneA,geneB] <-  bootMTmp[1,2]
    result@maxB[geneB,geneA] <-  bootMTmp[2,1]
    result@maxB[geneB,geneB] <-  bootMTmp[2,2]
  }
  if (dim(result@W)[1] == 0)
    result@W <- tableTmp
  else{
```

```
    while (dim(tableTmp)[2] != dim(result@W)[2])
      if(dim(tableTmp)[2] > dim(result@W)[2])
        result@W <- cbind(result@W,NA)
      else
        tableTmp <- cbind(tableTmp,NA)

    result@W <- rbind(result@W, tableTmp)
  }
  (...)
}
```

6.2.2. relNetworkM

Este algoritmo possui alterações em relação ao método descrito por Butte & Kohanne. Além de uma nova medida de correlação robusta, ele busca alterações significativas em valores de correlação linear entre dois tipos biológicos distintos, o que pode trazer indícios biológicos de redes de expressão com comportamento antagônico entre os tecidos estudados.

Novamente, ilustraremos o resultado da aplicação desta função apresentando duas figuras geradas após a execução do relNetworkM sobre a base de dados ‘gastro’, contida na versão 1.4.0 do pacote MaigesPack.

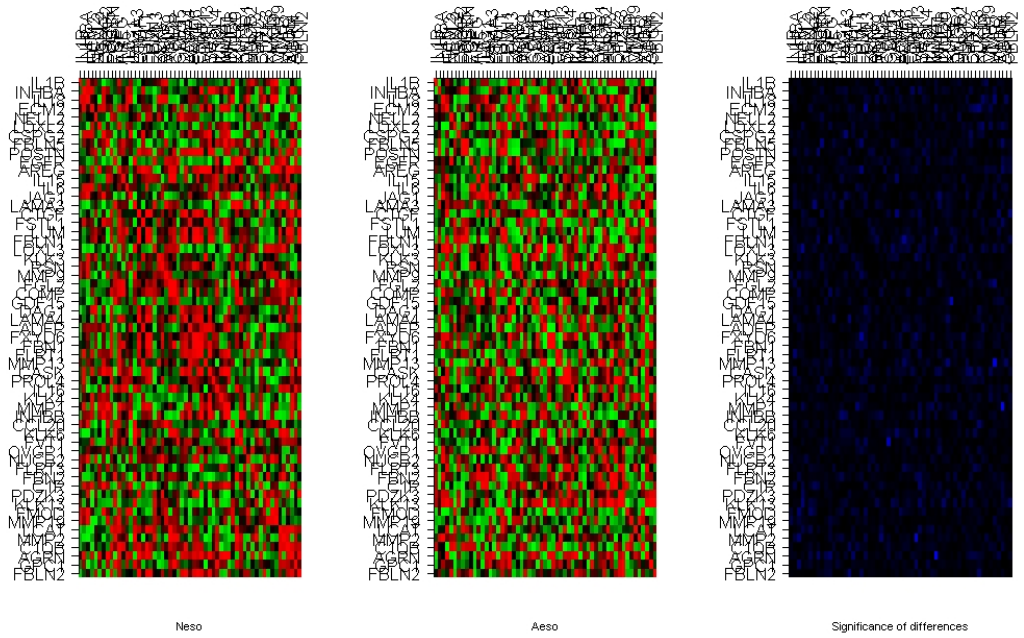

Figura 9: imagem dos valores de correlação para todos os pares de genes do banco de dados ‘gastro’, através da chamada de função image(gastro.net2) onde gastro.net2 = relNetworkM(gastro.summ, sLabelID = "Tissue, samples = list(Neso = "Neso", Aeso = "Aeso"), geneGrp = 7)

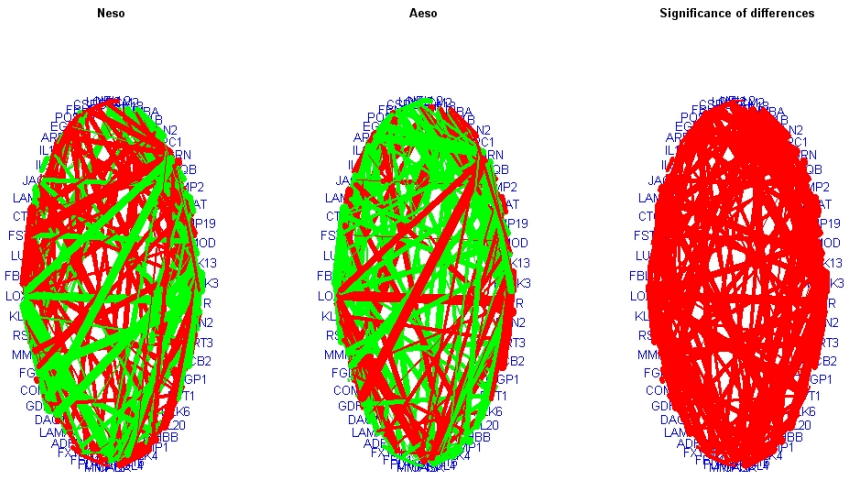

Figura 10: grafo das correlações de pares de genes com p-valores menores que 0,05 do banco de dados ‘gastro’, através da chamada da função plot(gastro.net2, cutPval = 0.05) onde gastro.net2 = relNetworkM(gastro.summ, sLabelID = "Tissue, samples = list(Neso = "Neso", Aeso = "Aeso"), geneGrp = 7)

6.2.2.1. Implementação Atual

Essa função é muito semelhante à função relNetworkB. Ela usa funções existentes do pacote maigesPack tais como: robustCorr, bootstrapCor, cor, MI e bootstrapMI. Os parâmetros que essa função recebe são os mesmos que o relNetworkB. São eles: data, gLabelID, sLabelID, geneGrp, path, samples, type e bRep.

A diferença entre as funções relNetworkB e relNetworkM está no processo de geração do objeto de retorno da função. Logo, a implementação da obtenção de genes por grupo de genes, remoção de genes e amostras que não foram nomeadas ou que são 'NA' que estão no objeto data, construção da tabela que será usada para armazenar os coeficientes de correlação, construção de tabela que será usada e remoção de eventuais replicações de informação, cálculo de coeficientes de correlação, p-valores e boot máximos de acordo com o tipo de estimativa, obtenção de informações das versões do R e maigesPack são processos que são executados usando as mesmas funções descritas no tópico anterior referente ao relNetworkB.

6.2.2.2. Implementação Nova

Na nova implementação, a geração do objeto de retorno usando o método de informações mútuas é responsabilidade da função *relNetworkMWrite.* Abaixo segue a codificação dessa função. Sendo o processo de análise, usando esta função, bem parecido com o processo utilizado quando se utiliza a função relNetworkBWrite.

```
relNetworkMWrite <- function(data=NULL, gLabelID = "GeneName",
                             sLabelID="Classification", samples=NULL,
                             type="Rpearson", bRep=1000, geneA=NULL,
                             geneB=NULL, result=NULL, genes=NULL,
                             tableTmp=NULL, ...) {
```

```
if(!(type %in% c("Rpearson", "pearson", "kendall", "spearman", "MI")))
  stop("Parameter 'type' must be 'Rpearson', 'pearson', 'kendall',
       'spearman' or 'MI'.")


allSamples <- getLabels(data, sLabelID)

## Getting the first 2 sample types to use in case of NULL samples
## parameter
if(is.null(samples)) {
  samples <- list(unique(allSamples)[1], unique(allSamples)[2])
  names(samples) <- unique(allSamples)[1:2]
}
if(is.null(result)){
  corObj1 <- matrix(,length(unique(genes)),length(unique(genes)),TRUE)
  rownames(corObj1) <- unique(genes)
  colnames(corObj1) <- unique(genes)

  corObj2 <- matrix(,length(unique(genes)),length(unique(genes)),TRUE)
  rownames(corObj2) <- unique(genes)
  colnames(corObj2) <- unique(genes)

 difPvalue <-matrix(,length(unique(genes)),length(unique(genes)),TRUE)
 rownames(difPvalue) <- unique(genes)
 colnames(difPvalue) <- unique(genes)

 ## Picking R and packages version information
 tmp <- sessionInfo()
 vInfo <- list()
 vInfo$R.version <- tmp$R.version$version.string
 vInfo$BasePacks <- tmp$basePkgs
 tmp1 <- NULL
 for (i in 1:length(tmp$otherPkgs))
   tmp1 <- c(tmp1, paste(tmp$otherPkgs[[i]]$Package, "version",
             tmp$otherPkgs[[i]]$Version))

 vInfo$AddPacks <- tmp1

## Defining the object to return
result <- new("maigesRelNetM", Corr1=corObj1, Corr2=corObj2,
DifP=difPvalue, Date=date(), types=names(samples), Slabel=sLabelID,
```

```
  V.info=vInfo)
}


## Finding indexes of the samples specified and respective lengths
sample1 <- which(colnames(tableTmp) %in% samples[[1]])
sample2 <- which(colnames(tableTmp) %in% samples[[2]])
n1 <- length(sample1)
n2 <- length(sample2)

## Calculating the correlation coefficients
if(type == "Rpearson") {
   corObj1 <- robustCorr(tableTmp[, sample1], ...)[[1]]
   corObj2 <- robustCorr(tableTmp[, sample2], ...)[[1]]
   ## Calculating the p-values of the differences
   difPvalue <- compCorr((n1-1), corObj1, (n2-1), corObj2)[[2]]
}
 else {
   corObj1 <- cor(t(tableTmp[, sample1]), method=type, ...)
   corObj2 <- cor(t(tableTmp[, sample2]), method=type, ...)
   ## Calculating the p-values of the differences
   difPvalue <- compCorr(n1, corObj1, n2, corObj2)[[2]]
 }

corObjTmp <- corCalc(type, tableTmp, sample1)
pValueTmp <- pValCalc(type , tableTmp, sample1, bRep)
bootMTmp <- bootMCalc(type, tableTmp, sample1, bRep)

if(is.null(geneA) & is.null(geneB)){
  result@Corr1 = corObj1
  result@Corr2 = corObj2
  result@DifP = difPvalue
}
else{
  result@Corr1[geneA,geneA] <-  corObj1[1,1]
  result@Corr1[geneA,geneB] <-  corObj1[1,2]
  result@Corr1[geneB,geneA] <-  corObj1[2,1]
  result@Corr1[geneB,geneB] <-  corObj1[2,2]

  result@Corr2[geneA,geneA] <-  corObj2[1,1]
  result@Corr2[geneA,geneB] <-  corObj2[1,2]
```

```
    result@Corr2[geneB,geneA] <-  corObj2[2,1]
    result@Corr2[geneB,geneB] <-  corObj2[2,2]

    result@DifP[geneA,geneA] <-  difPvalue[1,1]
    result@DifP[geneA,geneB] <-  difPvalue[1,2]
    result@DifP[geneB,geneA] <-  difPvalue[2,1]
    result@DifP[geneB,geneB] <-  difPvalue[2,2]
  }
  if (dim(result@W)[1] == 0)
    result@W <- tableTmp
  else{
    while (dim(tableTmp)[2] != dim(result@W)[2])
      if(dim(tableTmp)[2] > dim(result@W)[2])
        result@W <- cbind(result@W,NA)
      else
        tableTmp <- cbind(tableTmp,NA)

    result@W <- rbind(result@W, tableTmp)
  }
  return(result)
}
```

6.3. Utilização

A melhor forma de usar as novas funções é através da escrita de um script, pois o que antes era apenas a chamada de uma função se tornou a chamada de várias. Na realização dos testes foram criadas algumas funções que na realidade são os scripts que serão utilizados pelos usuários além da função simpleNaFilter que simula os filtros que serão usados nas análises. Ela tira os valores positivos sem nenhum propósito em particular, apenas para mostrar que uma função de filtragem da tabela pode ser utilizada facilmente.

O arquivo “scriptTeste.R” carrega os arquivos necessários e chama as funções que simulam os scripts do usuário, imprimindo os resultados para comparação.

6.3.1. Exemplo

Esse script faz a análise da correlação análoga à função relNetworkB, mas apenas entre o gene IGF1R e os genes OPA1 e CFLAR.

```
removeNaNames(data, gLabelID=”GeneName”,sLabelID=”Tissue)

genes <-getGenes(data, gLabelID=”GeneName”, geneGrp=1)

geneA= "IGF1R"
geneB= "OPA1"

table <- relNetworkBuildTable(data,gLabelID =“GeneName”
sLabelID=”Tissue”,type=”Rpearson”, geneA= geneA, geneB=geneB,NULL,
genes = genes, processType="B" )

table = simpleNaFilter(table)

result <- relNetworkBWrite(data, gLabelID ,sLabelID, samples = samples,
type, bRep, geneA= geneA, geneB=geneB, result=NULL, genes,tableTmp = table)

geneA= "IGF1R"
geneB= "CFLAR"

table <- relNetworkBuildTable(data,gLabelID =
gLabelID,sLabelID=sLabelID,samples = samples,type=type, geneA= geneA,
geneB=geneB,table, genes = genes, processType="B" )

table = simpleNaFilter(table)

result <- relNetworkBWrite(data, gLabelID ,sLabelID, samples = samples,
type, bRep, geneA= geneA, geneB=geneB, result, genes,tableTmp = table)
```

7. Conclusão

A Bioinformática tem como grande desafio a adequação dos sistemas computacionais para as necessidades reais dos pesquisadores. Seja quando o desenvolvedor do sistema é um bom programador, mas com baixo conhecimento da aplicação ou quando ele é um pesquisador sem muita experiência como programador. De qualquer forma, é consideravelmente complicada a conciliação de um sistema organizado e robusto com o processo de análise desejado pelos usuários.

Com esse cenário em mente, nós procuramos trabalhar no nível mais alto do pacote Maigespack. Nosso objetivo foi deixá-lo mais organizado, facilitando posteriores mudanças nele, como adição de novos métodos e modificação de outros já existentes. Após aprender mais sobre a linguagem R e a estrutura do Maigespack, percebemos que a nossa proposta era pouco aplicável, com baixos ganhos reais para os usuários, visto que o pacote já apresentava uma estrutura consistente.

Desta forma, optamos por mudar nossa proposta, tendo agora como objetivo uma melhoria real que fosse ajudar o usuário final. Conversando com os usuários, decidimos melhorar a flexibilidade e transparência da análise, em particular a de redes de relevância. Quebramos o processo em diferentes passos, como já foi explicado anteriormente. Com isso, o usuário terá um processo mais trabalhoso para adequar a análise de acordo com a sua necessidade, seja filtrando os valores de acordo com uma função escrita por ele ou aplicando a análise a apenas uma parte dos dados, do que, em contrapartida, teria na versão atual do código, onde chama apenas uma função.

8. Bibliografia

[1] CRISTO, Elier Broche - Métodos Estatísticos de Análise de Experimentos Microarray, 2003.

[2] BEISSBARTH, Tim; RUSCHHAUPT, Markus; JACKSON, David; LAWRENZ, Chris; Mansmann, Ulrich - Recommendations for normalization of microarray data [online]. Disponível na internet via WWW. URL: http://www.science.ngfn.de/dateien/Recommendations_for_normalization_of_microarray_data.pdf. Arquivo capturado em 12 de novembro de 2008**.**

[3] BILBAN, Martin; BUEHLER, Lukas K.; HEAD, Steven; DESIYE, Gernot; QUARANTA, Vito – Normalizing DNA Microarray Data

[4] WIKIPEDIA. Projeto Genoma [online]. Disponível na internet via WWW. URL: http://pt.wikipedia.org/wiki/Projeto_Genoma. Arquivo consultado em 4 de novembro de 2008.

[5] IRWIN, Richard D.; BOORMAN, Gary A.; CUNNINGHAM, Michael L.; HEINLOTH, Alexandra N.; MALARKEY, David E. - Application of Toxicogenomics to Toxicology: Basic Concepts in the Analysis of Microarray Data [online]. Disponível na internet via WWW. URL: http://tpx.sagepub.com/cgi/content/abstract/32/1_suppl/72. Arquivo consultado em 24 de setembro de 2008.

[6] GHANEM, Moustafa - Introduction to Bioinformatics - Microarrays2: Microarray Data Normalization

[7] TURNER, Heather. Clustering Microarray Data [online]. Disponível na internet via WWW. URL: http://www2.warwick.ac.uk/fac/sci/statistics/staff/research/turner/turnerchapter3.pdf. Arquivo capturado em 24 de setembro de 2008**.**

[8] Prof. Abraham B. Korol - Microarray cluster analysis and applications, 2003

[9] NCBI – National Center for Biotechnology Information. Molecular Biology Review [online]. Disponível na internet via WWW. URL: http://www.ncbi.nlm.nih.gov/Class/MLACourse/Modules/MolBioReview/images/mrna.gif. Arquivo capturado em 18 de novembro de 2008**.**

[10] FOWLER, Martin; BECK, Kent; BRANT, John; OPDYKE, William; ROBERTS, Don - Refactoring: Improving The Design of Existing Code. Addison-Wesley Object Technology Series, 1999.

[11] Sem Autor. Refactoring – Definição [online]. Disponível na internet via WWW. URL: http://www.javafree.org/wiki/Refactoring. Arquivo consultado em 20 de setembro de 2008.

[12] R PROJECT ORG. R - The R Projectfor Statistical Computing [online]. Disponível na internet via WWW. URL: http://www.r-project.org. Arquivo consultado em 3 de novembro de 2008.

[13] LANG, Jean-Phillipe. Refactoring R [online]. Disponível na internet via WWW. URL: http://www.r-developer.org/wiki/refactoring/RefactoringActivities. Arquivo consultado em 3 de novembro de 2008.

[14] EUGSTER, Manuel. Roxygen - Literate programming in R [online]. Disponível na internet via WWW. URL: http://roxygen.org. Arquivo consultado em 3 de novembro de 2008.

[15] BURGER, Matthias; JUENEMANN, Klaus; KOENIG, Thomas. RUnit Package [online]. Disponível na internet via WWW. URL: http://cran.r-project.org/web/packages/RUnit/RUnit.pdf. Arquivo capturado em 3 de novembro de 2008.

[16] ESTEVES, Gustavo H. - Métodos estatísticos para a análise de dados de cDNA microarray em um ambiente computacional integrado, Universidade de São Paulo, 2007

[17] BUTTE, Atul J; KOHANE, Isaac S. Unsupervised Knowledge Discovery in Medical Databases Using Relevance Networks [online]. Disponível na internet via WWW. URL: http://www2.amia.org/pubs/symposia/D005550.pdf. Arquivo capturado em 10 de janeiro de 2009.

[18] Bioconductor – open source software for bioinformatics [online] . Disponível na internet via WWW. URL: http://www.bioconductor.org/overview. Site consultado em 14 de setembro de 2008.