MaLTE: Machine Learning of Transcript Expression

An R Package Implementing the MaLTE Framework

Paul K Korir[1] and Cathal Seoighe[2]

[1]School of Biochemistry and Cell Biology,
University College Cork, Cork
[2]School of Mathematics, Statistics and Applied Mathematics,
National University of Ireland, Galway

Contents

1 Introduction

Quantification of oligonucleotide expression microarrays involves assembling probe fluorescence intensities from sets of probes into a single gene or transcript expression measure, a process referred to as *summarisation* [1]. There are many summarization algorithms [4] most of which exhibit from two main limitations: they produce *relative*, as opposed to *absolute*, estimates [2, 3] and they are oblivious to the abundance of individual transcript abundances [6].

The MaLTE framework supplements conventional algorithms with a supervised learning approach between a gold standard probe fluorescence intensities. As a framework, the various components may be modified or entirely overhauled; it is not restricted to the learning algorithms used in the R MaLTE package (conditional random forest (CRF) and quantile regression random forest (QRF)). Currently, we use RNA-Seq as the gold standard HTS quantification technique. Figure 1 below shows a schematic of the MaLTE framework. Learning occurs for each gene independently though only one gene is illustrated. For gene expression, MaLTE incorporates simple feature selection by picking the best 15 probes that correlate with the expression in the gold standard.

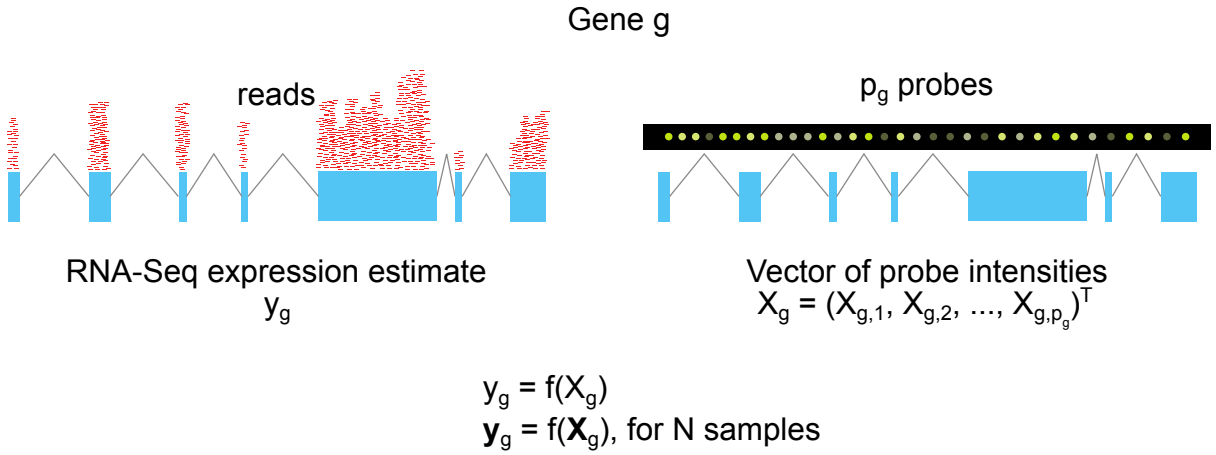

Figure 1: **Schematic of the MaLTE framework.** A supervised learning algorithm is used to learn the relationship between a gold-standard (RNA-Seq) and probe fluorescence intensities. The learned model may then be applied to a new set of probes.

This approach overcomes the two main setbacks of conventional algorithms: it transforms expression estimates onto an absolute scale thus improving the within-sample correlations and naturally extends to predicting expression of individual transcript isoforms by training on the multiple responses of individual transcript isoforms. Moreover, MaLTE leads to substantial improvements in cross-sample correlations when used on data from the same batch, which increases statistical power. Finally, a tree-based learning introduces an easy way to filter out poorly-predicted genes through the use of out-of-bag estimates.

This documents describes how to use the R MaLTE package. It begins with a description on how to get and install the package. It then outlines the two main ways in which the package may be used: *gene expression prediction (GEP)* and *transcript isoform expression prediction (TIEP)*. It concludes with a description on how to filter and collate expression predictions for downstream analyses. It also provides a tentative road map

for future development, a detailed description on how the MaLTE package is built, and includes a detailed use case of using data from the Genotype-Tissue Expression (GTEx) project [5] as training data together with an example of array transformation (e.g. from exon array to gene array). Bug reports, comments and suggestions are welcome through paul.korir@gmail.com.

2 Conventions Used

- Array data used here corresponds to that from *Affymetrix GeneChip® Human Exon 1.0 ST* arrays. At present only Affymetrix GeneChip® Human Exon and Human Gene arrays have been tested using MaLTE.
- All gene and transcript identifiers are from Ensembl (`http://www.ensembl.org`).
- All filenames are written in italics (*filename.txt*), variables in monospace (`my.var`), and functions in monospace terminated with parentheses (`my.function()`). Classes are in monospace beginning with a capital letter (`My.Class`) while corresponding constructors additionally terminate with parentheses (`My.Class()`). Names of software packages are in sans (R, MaLTE, Cufflinks)
- *High-throughput sequencing (HTS)* refers to *RNA-Seq*.
- A *map* is a tab-delimited text file with two columns with each column consisting of identifiers/names.
- The set of samples used for training are called *training samples*. *Test samples* refer to the samples that need to have their gene/transcript expression quantified.

3 System Requirements

- R (`2.14.0` or greater) installed on GNU/Linux: MaLTE has been tested on Scientific Linux version 5, Ubuntu 12.04, and Mac OS X.
- R packages: `party`, `multicore`, `quantregForest`, `limma`
- Python 2.7
- Affymetrix Power Tools (APT; `http://www.affymetrix.com`)
- Affymetrix GeneChip® library files (`http://www.affymetrix.com`)
- git (optional)

4 Installation

MaLTE may be downloaded from `https://github.com/polarise/MaLTE-package`. There are two ways to install MaLTE: via GNU/Linux shell and R shell.

1. Via GNU/Linux shell

```
R CMD INSTALL MaLTE_<release>.tar.gz
```

2. Via R shell

```
> install.packages( "/path/to/MaLTE_<release>.tar.gz" )
```

Once installed, MaLTE should be loaded in the R shell as follows:

```
> library( MaLTE )
```

or, quietly

```
> suppressMessages( library( MaLTE ))
```

5 Example Dataset

Example data files are provided with MaLTE, which are based on data from the HapMap project [7]. These are all contained in the `data` directory in the package. They will be used in Section 6 and Section 7 and are formatted as outlined in Section 6.2 and Section 7.2.

1. Files with sample names (several examples provided)
2. HTS data
3. Transcript HTS data
4. Raw microarray data (direct output from APT `apt-cel-extract`)
5. Truncated microarray data (non-essential rows and columns removed)
6. Gene-to-probeset maps for exon array
7. Gene-to-transcript maps

The complete set of files may be displayed on the R shell using the following line:

```
> dir( paste( system.file( package="MaLTE" ), "data", sep="/" ))
```

6 Gene Expression Prediction

6.1 Quick Start Guide

This section provides a quick introduction to using MaLTE. Detailed instructions incorporating descriptions of the various file and there respective formats is provided in Section 6.2. It is assumed that the user is in the R shell, the MaLTE library is loaded and the following data files are available: *samples.txt*, *hts_data.txt*, *ma_data.txt*, *gene_probesets.txt*.

Step I: Provide the location of the file containing a map between sample names on both platforms (RNA-Seq and microarray)

```
> samples.fn = paste( system.file( package="MaLTE" ), "data",
  "samples.txt.gz", sep="/" )
```

Step II: Provide the location of the file containing the high-throughput sequencing (RNA-Seq) data

```
> hts.fn = paste( system.file( package="MaLTE" ), "data",
  "hts_data.txt.gz", sep="/" )
```

Step III: Provide the location to the file containing quantile-normalized and background corrected fluorescence probe intensities

```
> ma.fn = paste( system.file( package="MaLTE" ), "data",
  "ma_data.txt.gz", sep="/" )
```

Step IV: Provide the location of a map showing the probe sets associated with each gene

```
> g2p.fn = paste( system.file( package="MaLTE" ), "data",
  "gene_probesets.txt.gz", sep="/" )
```

Step V: Prepare the data into training and test sets

```
> prepare.data( samples.fn=samples.fn, ma.fn=ma.fn, hts.fn=hts.fn,
  g2p.fn=g2p.fn )
```

Step VI: Read the data in preparation for the training and test phase

```
> tt.ready = read.data( train.fn="train_data.txt.gz",
  test.fn="test_data.txt.gz" )
```

Step VII: Initialize training parameters

```
# conditional random forest
> tt.params = TT.Params()

# quantile regression random forest
> tt.params = TT.Params( quantreg=TRUE )
```

Step VIII: Train and predict

```
> tt.seq = array2seq( tt.ready, tt.params )
```

Step IX: Perform out-of-bag (OOB) predictions

```
> tt.seq.oob = array2seq.oob( tt.ready, tt.params )
```

Step X: Filter based on OOB correlations

```
> tt.filtered = oob.filter( tt.seq, tt.seq.oob, thresh=0 )
```

Step XI: Get the names of test samples

```
> test.names = get.names( samples.fn, test=TRUE )
```

or

```
> test.names = get.test( samples.fn )   # get test sample names
```

Step XII: Aggregate predictions and write output to a text file for downstream analyses.

```
> df = get.predictions( tt.filtered, test.names )
> write.table( df, file="filt_preds.txt", col.names=TRUE, row.names=FALSE,
  quote=FALSE, sep="\t" )
```

6.2 Detailed Instructions

Gene expression prediction (GEP) depends on having four input files:

- **Sample names.** A map of sample names between both platforms (HTS and array; possibly zipped). An example file is provided with the package (Step I).
- **HTS data.** The HTS (RNA-Seq) data in text file (possibly zipped)
- **Microarray data.** The microarray probe data (possibly zipped)
- **Gene-to-probeset map.** A map between gene identifiers and probe set identifiers. Probe set identifiers are provided by the array manufacturer as part of the array description.

Each file is now described in detail under the following sub-headings: *purpose*, *generic designation*, *header* and *structure*.

6.2.1 Sample Names File

Purpose This file provides a one-to-one map between sample identifiers on both platforms.

Generic designation *samples.txt* OR *samples.txt.gz*; Any suitable name will do but the file name must end either with **.txt* or **.txt.gz*.

Header `hts<tab>ma`

Structure Two-columns separated by a single tab (tab-delimited)
All sample identifiers must be unique and must exactly correspond to sample names present in the headers of the HTS data and microarray data (to be described shortly). For example, microarray probe files will usually have headers with sample names terminated by **.CEL*; this must be retained in the sample names file.
Test samples are marked by having an asterisk (`'*'`) as the first character. Any other row is assumed to be a training sample.
Test samples may consist of having both HTS and array data. If only test array data is present then the first column must be `'*NA'`.
Comments must begin with a pound/hash (`'#'`) symbol.

6.2.2 HTS Data File

Purpose This file provides the HTS expression estimates.

Generic designation *hts_data.txt* OR *hts_data.txt.gz*
Any suitable name will do but the file name must end either with **.txt* or **.txt.gz*.

Header `gene_id<tab>Sample01<tab>...<tab>SampleN`

Structure Tab-delimited
All rows must be unique
No comments are allowed

6.2.3 Microarray Data File

Purpose This file provides the microarray probe intensities.

Generic designation *ma_data.txt* OR *ma_data.txt.gz* if inessential columns have been removed (see Section 6.3 on *Preparing Input Files*) *raw_ma_data.txt* OR *raw_ma_data.txt.gz* if inessential columns are still present (see Section 6.3 on *Preparing Input Files*)
Any suitable name will do but its name must terminate with **.txt* or **.txt.gz*.

Header `probe_id<tab>Sample01<tab>...<tab>SampleN`

Structure Tab-delimited
Comments must begin with a pound/hash (`'#'`) symbol.

6.2.4 Gene-to-Probe Set Map

Purpose	This file provides a one-to-many map of gene identifiers to probe set identifiers.
Generic designation	*gene_probesets.txt* OR *gene_probesets.txt.gz* Any suitable name will do but the file name must end either with **.txt* or **.txt.gz*..
Header	`gene_id<tab>probeset_id`
Structure	Tab-delimited Probe set identifiers may be missing for some genes. No comments are allowed.

6.3 Preparing Input Files

1. **Sample names.** This file can be prepared using a text editor or preferably using a spreadsheet application such as Microsoft Excel or LibreOffice/OpenOffice Calc. The file must be saved as a tab-delimited text file or a comma-separated values (CSV) file with the field-delimiter set to TABS and the quote-character set to NONE. The file extension must be as described above.

2. **HTS data.** This file may be constructed using customized scripts that collate the HTS expression estimates output from programs such as Cufflinks or DESeq. The order of sample columns is unimportant. There are several online datasets[1] that are provided in this format making it easy to proceed with using MaLTE.

3. **Microarray data.** Raw microarray data is provided as CEL files. The contents of CEL files need to be extracted and additional pre-processing steps may be applied to the raw data. Two pre-processing steps we recommend are quantile-normalization (QN) and background correction (BC). The Affymetrix Power Tools (APT) suite is recommended for this and other analytical steps though several R packages have been developed to supplement APT. Here we describe how to extract fluorescence probe intensities and how to remove unnecessary columns.

   (i) **Extracting QN and BC microarray probe data.** We assume that all CEL files are contained in a single folder. APT requires a set of library files that are available from the Affymetrix website (`http://www.affymetrix.com`). An account will have to be created in order to download library files. The library files consist of array description files used by APT to carry out analyses. More information on these can be found in the manuals provided for each array type.

   To extract probes with QN and BC:

```
apt-cel-extract -o raw_ma_data.txt
  -c /path/to/HuEx-1_0-st-v2.2/HuEx-1_0-st-v2.r2.clf
  -p /path/to/HuEx-1_0-st-v2.2/HuEx-1_0-st-v2.r2.pgf
  -b /path/to/HuEx-1_0-st-v2.2/HuEx-1_0-st-v2.r2.antigenomic.bgp
  -a quant-norm,pm-gcbg *.CEL
```

[1] `http://bowtie-bio.sourceforge.net/recount/`

This provides 'raw' data that can directly be used with MaLTE. To do so, the `raw` argument in `prepare.data()` must be set to `TRUE` (as it is `FALSE` by default). However, unnecessary columns can be excluded as shown below.

(ii) **Excluding unnecessary columns.** Unnecessary columns can be easily excluded using the bash utility `cut` as follows:

```
cut -f1,5,8- raw_ma_data.txt > ma_data.txt
```

4. **Zip the file to save space.**

```
gzip raw_ma_data.txt
gzip ma_data.txt
```

5. **Gene-to-probeset map.** This file may be downloaded directly from BioMart, particularly for popular arrays. Alternatively, the user may prepare it for him/herself. This can be done using BEDTools. BEDTools takes as input two BED files having coordinates of gene and probe sets, respectively. The `intersect` BEDTools utility then finds all overlaps between both files and writes them to an extended BED file. The appropriate columns can then be combined to provide the required file.

   To use the BED approach, the gene annotation must be provided as a BED file. Similarly, the array's annotation files (available from the Affymetrix website) must be converted to BED format. Both tasks may be performed using custom scripts.

   Please consult the BEDTools website (`http://bedtools.readthedocs.org/en/latest/`) on how to intersect two BED files.

   The gene and probe set columns can then be isolated using `cut` in a manner similar to sub-step (2) above.

7 Transcript Isoform Expression Prediction

7.1 Quick Start Guide

This section describes how to perform transcript isoform expression prediction in quick steps. It assumes that the user has logged into an R shell, the MaLTE package is loaded and that the following files are available: *samples.txt*, *train_data.txt.gz*, *test_data.txt.gz*, *hts_txs_data.txt*, and *gene_transcripts.txt*.

The files *train_data.txt.gz* and *test_data.txt.gz* are produced by running Step I-V in Section 6.1 *Gene Expression Prediction: Quick Start Guide*.

Step I: Provide the location of the file containing a map between sample names on both platforms (RNA-Seq and microarray)

```
> samples.fn = paste( system.file( package="MaLTE" ), "data",
  "samples.txt.gz", sep="/" )
```

Step II: Provide the location of the file containing the high-throughput sequencing (RNA-Seq) transcript isoform expression estimates

```
> hts.txs.fn = paste( system.file( package="MaLTE" ), "data",
  "hts_txs_data.txt.gz", sep="/" )
```

Step III: Provide the location of map of gene-to-transcript identifiers

```
> g2tx.fn = paste( system.file( package="MaLTE" ), "data",
  "gene_transcripts.txt.gz", sep="/" )
```

Step IV: Prepare the data into training and test sets

```
> prepare.txs.data( samples.fn=samples.fn, train.fn="train_data.txt.gz",
  test.fn="test_data.txt.gz", hts.txs.fn=hts.txs.fn, g2tx.fn=g2tx.fn )
```

Step V: Read in the data in preparation for training and preparation

```
> tt.ready.txs = read.txs.data( train.fn="train_txs_data.txt.gz",
  test.fn="test_txs_data.txt.gz" )
```

Step VI: Train and predict

```
> tt.seq.txs = array2seq( tt.ready.txs, tt.params )
```

Step VII: Train and predict for OOB estimates

```
> tt.seq.oob.txs = array2seq.oob( tt.ready.txs, tt.params )
```

Step VIII: Filter based on OOB correlations

```
> tt.filtered.txs = oob.filter( tt.seq.txs, tt.seq.oob.txs, thresh=0 )
```

Step IX: Get test sample names

```
> test.names = get.names( samples.fn, test=TRUE )
```

or

```
> test.names = get.test( samples.fn )   # get test sample names
```

Step X: Collate predicted transcript isoform predictions and write them to a text file for downstream analyses

```
> df.txs = get.predictions( tt.filtered.txs, test.names )
```

7.2 Detailed Instructions

Transcript isoform expression prediciton (TIEP) depends on having five input files:

- **Sample names.** A map of sample names between both platforms (HTS and array; possibly zipped). This is the exact same file used in GEP above.
- **GEP Training data.** The name of this file is *train_data.txt.gz*. It is the first zipped output produced by running `prepare.data()` prior to carrying out GEP.
- **GEP Test data.** The name of this file is *test_data.txt.gz*. It is the second zipped output produced by running `prepare.data()` prior to carrying out GEP.
- **Transcript HTS data.** The HTS (RNA-Seq) transcript isoform data in text file (possibly zipped)
- **Gene-to-transcript map.** A map of between gene identifiers and transcript identifiers.

7.2.1 Sample Names File

Purpose	This file provides a one-to-one map between sample identifiers on both platforms.
Generic designation	*samples.txt* OR *samples.txt.gz* Any suitable name will do but the file name must end either with **.txt* or **.txt.gz*..
Header	`hts<tab>ma`
Structure	Two-columns separated by a single tab (tab-delimited) All sample identifiers must be unique and must exactly correspond to sample names present in the headers of the HTS data and microarray data. For example, microarray probe files will usually have headers with sample names terminated by **.CEL*; this must be retained in the sample names file. Test samples are marked by having an asterisk (`'*'`) as the first character. Any other row is assumed to be a training sample. Test samples may consist of having both HTS and array data. If only test array data is present then the first column must be `'*NA'`. Comments must begin with a pound/hash (`'#'`) symbol.

7.2.2 GEP Training Data File

Purpose This file contains gene-to-probe training data that will be used to create new transcripts-to-probes training data.

Generic designation *train_data.txt.gz*
This file is produced after running `prepare.data()`.

Header None

Structure This file has six columns. This data is automatically generate by the `prepare.data()` function.
Gene identifier
Number of training samples
Number of probes associated with this gene
Probe (not probe set) identifiers associated with this gene
HTS expression estimates
Vectorized matrix[1] of fluorescence probe intensities

[1] A *vectorized matrix* is a stack of the columns into a single column vector.

7.2.3 GEP Test Data File

Purpose This file contains that gene-to-probe training data that will be used to create new transcripts-to-probes training data.

Generic designation *test_data.txt.gz*

Header None

Structure This file has six columns. This data is automatically generate by the `prepare.data()` function.
Gene identifier
Number of test samples
Number of probes associated with this gene
Probe (not probe set) identifiers associated with this gene
HTS expression estimates
Vectorized matrix of fluorescence probe intensities

7.2.4 Transcript HTS Data File

Purpose This file provides the HTS transcript isoform expression estimates.

Generic designation *hts_txs_data.txt* OR *hts_txs_data.txt.gz*
Any suitable name will do but the file name must end either with **.txt* or **.txt.gz*..

Header `tx_id<tab>Sample01<tab>...<tab>SampleN`

Structure Tab-delimited
All rows must be unique
No comments are allowed

7.2.5 Gene-to-Transcript Map

Purpose	This file provides a one-to-many map of gene identifiers to transcript identifiers.
Generic designation	*gene_transcripts.txt* OR *gene_transcripts.txt.gz* Any suitable name will do but the file name must end either with **.txt* or **.txt.gz*..
Header	`gene_id<tab>tx_id`
Structure	Tab-delimited No comments are allowed.

7.3 Preparing Input Files

1. **Sample names.** Please see Section 6 *GEP: Preparing Input Files*.

2. **GEP training data.** This file is automatically generated after running `prepare.data()` Please see Section 6 Steps I-V of *GEP: Quick Start Guide*.

3. **GEP test data.** This file is automatically generated after running `prepare.data()` Please see Section 6 Steps I-V of *GEP: Quick Start Guide*.

4. **Transcript HTS data file.** Several programs are available that perform transcript isoform expression quantification from HTS data. Widely used examples include Cufflinks, IsoEM and RSEM. As suggested in HTS data description in Section 6.3 *GEP: Preparing Input Files*, custom scripts should be used to combine expression estimates. Sample names in the header must exactly correspond to those in the Sample names file. The order of samples is not important.

5. **Gene-to-transcript map.** A man of gene to transcript identifiers is most easily obtained from BioMart.

8 Filtering and Collating Predictions

The training and prediction step produces data in an internal format that is not suitable for further bioinformatic analyses. The prediction estimates need to be filtered to exclude poor predictions then finally converted into a data frame that can then be saved as a tab-delimited text file. Steps IX and X show GEP OOB filtering and step VII and VIII show TIEP OOB filtering. In order for expression predictions to be collated, the sample names must first be obtained (GEP: Step XI; TIEP: Step X) from the sample names file (*samples.txt*). Finally, extracted data may be written to a tab-delimited text file using base R functions.

Filtering takes advantage of the tree-based learning algorithm. Conditional random forests are used in the current implementation of MaLTE. A forest is an ensemble of trees, with each tree constructed by bootstrapping on the training data then using a recursive partition approach to define the branches. The forest consists of hundreds to thousands of such trees and predictions are aggregated either by a simple voting scheme (classification) or averaging (regression). Each training observation is used to train a subset of all the trees therefore the trees from which it is absent (for which it is *out-of-bag*) can be used to predict its value. The resulting predicted estimates are

therefore called the *out-of-bag (OOB) estimates* from which we can estimate a correlation with the training response. It is this correlation that acts as a heuristic for prediction performance and results in *OOB filtering*. An OOB Pearson correlation threshold of zero ($r_{OOB} > 0$) is currently set as a default filter threshold.

Alternatively, filtering can be based on the training RPKM/FPKM values. For example, we could include all genes with at least ten samples with an FPKM above one.

```
# tt.seq     - predictions from training data
# tt.seq.oob - OOB predictions (contains training RNA-Seq)
# only include genes with at least 10 samples having
#  RNA-Seq expression of at least 1
> filt.indexes = .filter.trues( tt.seq.oob, filter.fpkm=1, filter.count=10 )
> tt.filt = tt.seq[ filt.indexes ]

# extract a data frame of filtered predictions
> test.names = get.test( samples.fn ) # get names of test samples
> df.malte = get.predictions( tt.filt, test.names )
```

9 Training Parameters

There are eight main parameters that can be specified when running MaLTE:

```
> tt.params = TT.Params()
> tt.params
mtry             = 2
ntree            = 1000
feature.select   = TRUE
min.probes       = 15
cor.thresh       = 0
OOB              = FALSE
QuantReg         = FALSE
Tune (OOB cor.P) = NA
```

1. `mtry`. Specifies the number of predictors (probes) to be used in creating splits in the tree. The default and optimized parameter value is `mtry=2`.

2. `ntree`. The number of trees in the forest. The default and optimized value is `ntree=1000`.

3. `feature.select`. A boolean that sets whether feature selection is carried out. The default value is `feature.select=TRUE`.

4. `min.probes`. If `feature.select=TRUE` then what is the minimum number of probes below which no feature selection is carried out? Defaults to `min.probes=15` (i.e. feature selection is only carried out for 16 or more probes).

5. `cor.thresh`. Feature selection is implemented using a simple method: the top `min.probes` probes that correlate with the response (RNA-Seq) are used if they have a Pearson correlation of at least `cor.thresh`.

6. `OOB`. A boolean that specifies whether or not OOB predictions are performed. Defaults to `OOB=FALSE`.

7. `quantreg`. A boolean that specifies whether or not to use quantile regression forest as the learner. Defaults to `quantreg=FALSE`.

8. `tune`. This is an experimental feature in which parameter values are tune for each gene individually. Parameters are tuned in a cascade in the following order: `feature.select`, `mtry`, then `ntree`. Other parameters are retained as specified above. Defaults to `tune=FALSE`.

10 Future Work

The following list of features may be added at a future date:

1. **Replace the underlying Python scripts with C/C++ programs.** We used Python because it was relatively simple to put together and it has mature data structures. Python scripts are slower than C/C++ programs but we have applied multiprocessing to shorten the data preparation step. However, this arrangement restricts MaLTE to UNIX-like systems.

2. **Configure training and test data using a standardised structured file format.** Currently, training and test data is held in custom tab-delimited files. We would like to transition either to XML or HDF.

3. **Expand the use of the sample names (*samples.txt*) file.** Currently, the sample names file is underutilised. It is possible to include additional columns that could be incorporated into the learning process. For example, a column on batch information could be passed to ComBat to minimise batch effects. Other variables such as tissue type could be important for training.

4. **Automatic parameter tuning.** We would like to incorporate a tuning utility that uses a random sample of the training data to optimise the training parameters.

11 Bug Reports

Please send bug reports and feature requests to paul.korir@gmail.com with the subject 'MaLTE Bugs' of 'MaLTE Features', respectively. Alternatively, you may visit `https://github.com/polarise/MaLTE/issues` and click the 'Issues' button to file a report.

12 Citing MaLTE

MaLTE has been submitted for peer-review. This section will be updated shortly.

References

[1] Irizarry, Rafael A., et al. "Summaries of Affymetrix GeneChip probe level data." Nucleic acids research 31.4 (2003): e15-e15.

[2] Irizarry, Rafael A., et al. "Multiple-laboratory comparison of microarray platforms." Nature Methods 2.5 (2005): 345-350.

[3] Fu, Xing, et al. "Estimating accuracy of RNA-Seq and microarrays with proteomics." BMC Genomics 10.1 (2009): 161.

[4] Irizarry, Rafael A., Zhijin Wu, and Harris A. Jaffee. "Comparison of Affymetrix GeneChip expression measures." Bioinformatics 22.7 (2006): 789-794.

[5] John Lonsdale, Jeffrey Thomas, Mike Salvatore, Rebecca Phillips, Edmund Lo, Saboor Shad, Richard Hasz, Gary Walters, Fernando Garcia, Nancy Young, et al. The Genotype-Tissue Expression (GTEx) project. *Nature Genetics*, 45(6):580–585, 2013.

[6] Malone, John H., and Brian Oliver. "Microarrays, deep sequencing and the true measure of the transcriptome." BMC biology 9.1 (2011): 34.

[7] Gibbs, Richard A., et al. "The international HapMap project." Nature 426.6968 (2003): 789-796.

A Classes

The R MaLTE package uses three main classes.

1. `TT.Ready`. This class handles data *ready* for training and test. It is an abstract base class from which two other classes are derived:

   (i) `TT.Ready.Gene`. This class handles gene expression training and prediction data.

   (ii) `TT.Ready.Txs`. This is for transcript isoform expression data.

2. `TT.Seq`. This class handles the results of training and prediction. Just like `TT.Ready`, this is an abstract base class with two derived classes:

   (i) `TT.Seq.Gene` for gene predictions. Both test and OOB predictions are of this class.

   (ii) `TT.Seq.Txs` for transcript isoform predictions similar to `TT.Seq.Gene`.

3. `TT.Params`. This class handles training and prediction parameters passed to the `train.and.predict()` (alias `run()`) methods of `TT.Ready` objects. It has the following slots: `mtry`, `ntree`, `feature.select`, `min.probes`, `cor.thresh`, `OOB`, `quantreg`, `tune`.

More information on these classes can be found in the MaLTE manual that accompanies the package.

B Using Alternative Learning Algorithms (Experimental)

MaLTE is designed as a framework and this allows the user to specify alternative learning algorithms. To do this requires two steps: importing the required library and incorporating the appropriate call to the model-building function.

For example, to use multivariate adaptive regression splines (MARS) as the learner, we would do the following:

1. Add the name of the library into line 8 of the `DESCRIPTION` file in the package root. In this example, we import the `mda` package (assuming it is installed).

```
8: Depends: R (>= 2.14.0), methods, party, multicore, \
   quantregForest, limma, mda
```

2. We then modify the `model` by implementing the appropriate signature. Changes have to be made to either line 77 or line 80 of `train_and_predict.R`:

```
74: # the model
75: if ( quantreg )
76: {
77:  model <- quantregForest( y=D[,1], x=D[,2:ncol( D )],\
    mtry=mtry, ntree=ntree )
78: } else
79: {
80:  model <- mars( x=D[,2:ncol(D), y=D[,1] )
81: }
```

Modifying the learning algorithm may disable certain features. For example, out-of-bag (OOB) estimates are only available using forest models. Additionally, prediction of expression of transcript isoforms depends on the configuration of the learning algorithm to provide multiple response regression.

Please use the contact information provided in Section 11 to report any undesirable behaviour.

C Function and Methods Table

Function/Method	Input	Output	Comments
`prepare.data()`	`samples.fn`, `ma.fn` (OR `raw_ma.fn` with `raw=TRUE`), `hts.fn`, `g2p.fn`	*train_data.txt.gz* and *test_data.txt.gz* (together with log files indicating which genes are missing)	This function calls the underlying Python script *prepare_data.py*, which can be called directly by the user. All output is written to the current directory.
`read.data()`	`train.fn='train_data.txt.gz'`, `test.fn='test_data.txt.gz'`	`tt.ready`	`tt.ready` is a list of objects of class `TT.Ready.Gene` that has embedded within it the training and testing data.
`prepare.txs.data()`	`samples.fn`, `train.fn`, `test.fn`, `hts.txs.fn`, `g2tx.fn`	*train_txs_data.txt.gz*, *test_txs_data.txt.gz* (together with log files indicating which genes are missing)	This function works like `prepare.data()` by calling the underlying Python script *prepare_txs_data.py*, which can be called directly. All output is written to the current directory.
`read.txs.data()`	`train.fn='train_txs_data.txt.gz'`, `test.fn='test_txs_data.txt.gz'`	`tt.ready.txs`	`tt.ready.txs` is a list of objects of class `TT.Ready.Txs`
`TT.Params()`	`mtry=2`, `ntree=1000`, `feature.select=TRUE`, `min.probes=15`, `cor.thresh=0`, `OOB=FALSE`, `quantreg=FALSE`, `tune=FALSE`	`tt.params`	Constructor for objects of class `TT.Params`
`run()`, `oob.run()`	`TT.Ready` object, `tt.params`, `OOB=FALSE`	`TT.Seq` object	Performs prediction on a single `TT.Ready.Gene` or `TT.Ready.Txs` object.
`array2seq()`, `array2seq.oob()`	`tt.ready`/`tt.ready.txs`, `tt.params`	`tt.seq` OR `tt.seq.txs`	`tt.seq` is a list of `TT.Seq.Gene` or `TT.Seq.Tx` objects. The parallelised-list version of `run()`/`oob.run()`
`oob.filter()`	`tt.seq`/`tt.seq.txs`, `tt.seq.oob`/`tt.seq.oob.txs` (resp.), `thresh`	`tt.filtered`	`tt.filtered` is a list of objects of class `TT.Seq.Gene` `tt.filtered.txs` is a list of objects of class `TT.Seq.Tx`
`predictions()`	`TT.Seq.Gene` OR `TT.Seq.Txs`	`tt.predicted` OR `tt.predicted.txs`	Method to extract predictions only
`get.predictions()`	`tt.filtered` OR `tt.filtered.txs`	`df`	`df` is a data frame of predictions
`cor.P()`	`tt.seq.oob` OR `tt.seq.oob.txs`		Method to extract Pearson correlations only when HTS is available for test data
`cor.S()`	`tt.seq.oob` OR `tt.seq.oob.txs`		Method to extract Spearman correlations only when HTS is available for test data
`get.names()`, `get.train()`, `get.test()`	`samples.fn='samples.txt'`	`sample.names`	Returns a list of names of train/test samples

D Use Case: MaLTE Trained with GTEx Data Applied on Exon Array

Using MaLTE with the GTEx training dataset consists of three main steps illustrated in Figure 2.

- Obtaining the package and training data
- Preparing the data for training-and-testing
- Predicting, filtering and collating expression estimates

In the first step, the user needs to download a set of auxiliary data and scripts before installing the MaLTE R package. The training (and test) data may then be downloaded after which several preprocessing steps may be carried out depending on the type of array involved. For example, we will need to transform the exon array to a gene array. The transformed array data is then quantile normalized to reduce batch effects.

The second and third steps are relatively straightforward and employ only MaLTE functions as outlined in Sections 6.

D.1 Obtaining Auxiliary Data and Scripts

The use-case presented here requires several auxiliary data and scripts to convert the exon array to a gene array. These are all contained in the MaLTE-aux Github repository which may be directly downloaded from `https://github.com/polarise/MaLTE-aux`. Click the 'Download' button to get the whole directory as a zip file.

```
unzip MaLTE-aux-master.zip
cd MaLTE-aux
```

Alternatively, it is much faster to use git as follows:

```
git clone git@github.com:polarise/MaLTE-aux.git
cd MaLTE-aux
```

This directory contains pre-compiled resource files described in Table 1. **We strongly recommend that all analysis be carried out in this directory.**

The script *create_samples_template.py* compiles the resource file *samples.txt* by choosing a random subset of samples (the user specifies the number) to be used for training specified in *training_samples.txt*. More information on how it works can be obtained by running

```
./create_samples_template.py --help
```

The script *transform_microarrays.py* converts one array to another using several resource files provided in MaLTE-aux. A detailed specification of how to use this script is available in the file *transform_microarrays.help*.

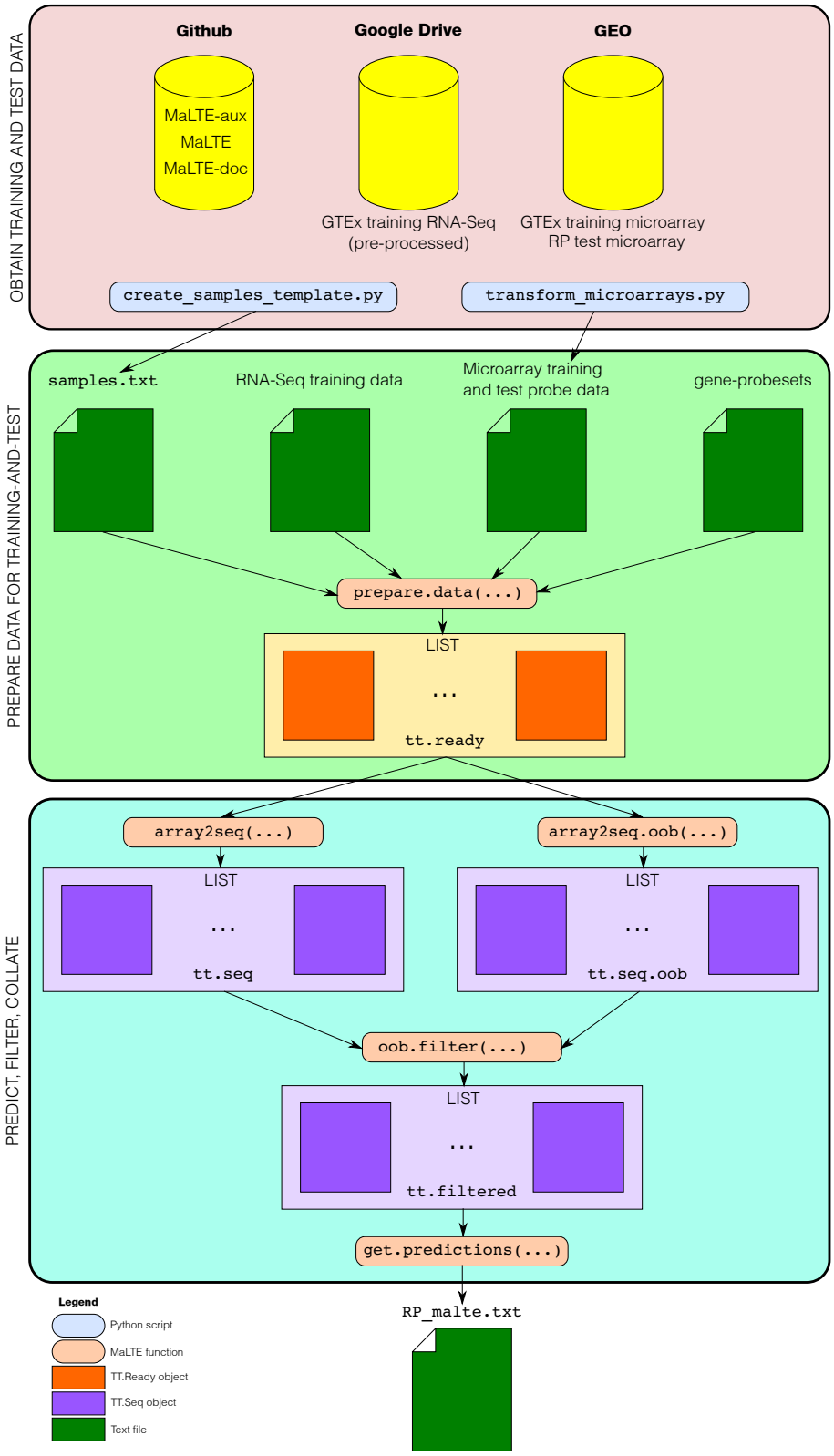

Figure 2: **Schematic representing a MaLTE use-case.** The four text files (dark green) are the main input to MaLTE. These files are obtained by processing the training data using the auxiliary scripts.

File	Purpose
create_samples_template.py *training_samples.txt*	These script and input data files are used to generate *samples.txt*.
transform_microarrays.py *transform_microarrays.help*	The Python script is used to transform the exon array to a gene array. Some probes are omitted in the process. The `.help` file gives the explicit use of the data files below.
HuEx-1_0-st-v2.r2.dt1.hg18.full.mps.gz *HuEx-1_0-st-v2.r2.pgf.txt.gz* *HuExVsHuGene_BestMatch.txt.gz* *HuGene-1_1-st-v1.r4.mps.gz* *HuGene-1_1-st-v1.r4.pgf.txt.gz*	Data files required by *transform_microarrays.py*
gene_probesets_HuGene_Ens72.txt.gz *transcript_probesets_HuGene_Ens72.txt.gz*	Input data files required to run the MaLTE function `prepare.data()`

Table 1: Contents of MaLTE-aux and the functions they provide

D.2 Working Directory Structure

As all analysis will be carried out in MaLTE-aux directory, we need to create two directories for the training and test array CEL files.

```
mkdir GTEx_CEL
mkdir RP_CEL
```

D.3 Obtaining MaLTE

MaLTE may be downloaded as a pre-built or source package. Pre-built versions can be found at `https://github.com/polarise/MaLTE-packages`. The latest version should be downloaded (files are not sorted by version), ensuring that all prerequisites (see Section 3) are installed before installing MaLTE.

```
R CMD INSTALL MaLTE_<version>.tar.gz
```

The source package requires an installation of git and is obtained as followed:

```
git clone git@github.com:polarise/MaLTE.git
```

then built with

```
R CMD build MaLTE
```

and installed using `R CMD INSTALL` as shown above.

D.3.1 Creating the file *samples.txt*

The *samples.txt* file is created by running

```
./create_samples_template.py
```

The names of the test samples should be appended at the bottom of this file using a text editor. For example, to add a test sample called *Test01.CEL* add the following line to *samples.txt*:

```
*NA<tab>Test01.CEL
```

The asterisk (`'*'`) marks this sample as a test sample and `'NA'` indicates that RNA-Seq data is not available. Repeat this for as many samples as are present in the test data.

D.3.2 Obtaining the training data

GTEx RNA-Seq data. Gene expression levels estimated from RNA-Seq data may be downloaded from `http://www.broadinstitute.org/gtex`. The downloaded data needs to be processed to arrive at the training state by excluding extraneous information and converting gene annotation to Ensembl (from GENCODE) then quantile-normalizing only those genes with probe sets. Pre-processed data may be downloaded from `https://drive.google.com/file/d/0BxLgaMV5aZahVDNjM29WeGh2NHM/edit?usp=sharing`.

GTEx gene array data. CEL files for the GTEx microarray data are available as a single zipped archive which may be downloaded from GEO archive GSE45878 into the directory `GTEx_CEL`. Background-corrected probes should be extracted as follows using the Affymetrix Power Tools function `apt-cel-extract`, which depends on the Affymetrix GeneChip® Human Gene 1.1 ST Array library files available at `http://www.affymetrix.com/estore/catalog/prod350003/AFFY/Human-Gene-ST-Array-Strips#1_3`. Select the 'Technical Documentation' tab to get a link to the library files.

```
apt-cel-extract -o GTEx_probe_intensities.txt \
-c /path/to/HuGene-1_1-st-v1.r4.clf \
-p /path/to/HuGene-1_1-st-v1.r4.pgf \
-b /path/to/HuGene-1_1-st-v1.r4.bgp \
-a pm-gcbg GTEx_CEL/*.CEL
```

It may be necessary to extract background probes file using `apt-dump-pgf`.

```
apt-dump-pgf \
-p /path/to/HuGene-1_1-st-v1.r4.pgf \
-c /path/to/HuGene-1_1-st-v1.r4.clf \
--probeset-type antigenomic \
-o HuGene-1_1-st-v1.r4.bgp
```

Exon array data. We outline this procedure using a previously uploaded dataset available under GEO accession GSE43134 into the directory `RP_CEL`. The Affymetrix GeneChip® Human Exon 1.0 ST Array library files are available from `http://www.affymetrix.com/estore/catalog/131452/AFFY/Human-Exon-ST-Array`. Similarly, background-corrected probe fluorescence intensities should be extracted as follows:

```
apt-cel-extract \
-o RP_probe_intensities.txt \
-c /path/to/HuEx-1_0-st-v2.r2.clf \
-p /path/to/HuEx-1_0-st-v2.r2.pgf \
-b /path/to/HuEx-1_0-st-v2.r2.antigenomic.bgp \
-a pm-gcbg RP_CEL/*.CEL
```

D.4 Transforming exon array probes to gene array probes

We now run *transform_microarrays.py* using the following template

```
./transform_microarrays.py \
-e RP_probe_intensities.txt \
-g GTEx_probe_intensities.txt \
-c HuExVsHuGene_BestMatch.txt.gz \
-i HuGene-1_1-st-v1.r4.pgf.txt.gz \
-f HuEx-1_0-st-v2.r2.pgf.txt.gz \
-p HuEx-1_0-st-v2.r2.dt1.hg18.full.mps.gz \
-q HuGene-1_1-st-v1.r4.mps.gz \
--huex-out BestMatch_HuEx_probe_intensities.txt.gz \
--huge-out BestMatch_HuGe_probe_intensities.txt.gz
```

Once the modified probe intensity files have been created they need to be quantile-normalized together with the training data as follows:

```
> library( limma )

> # BESTMATCH DATA
> huex.best <- read.table( "BestMatch_HuEx_probe_intensities.txt.gz",
  header=TRUE,  stringsAsFactors=FALSE,  check.names=FALSE)

> huge.best <- read.table( "BestMatch_HuGe_probe_intensities.txt.gz",
  header=TRUE,  stringsAsFactors=FALSE,  check.names=FALSE)

> # all raw data
> data <- cbind( huge.best[,8:ncol( huge.best )],
  huex.best[,8:ncol( huex.best )] )

> # quantile normalisation only
> qnorm.data <- normalizeQuantiles( data, ties=FALSE )
> # save the quantile-normalised data for later
> # (extracting principal components)
```

```
> save( qnorm.data, file="qnorm.data.Rdata" )

> # re-insert probe metadata
> hugeex.qnorm.data <- cbind( huge.best[,1:7], qnorm.data )

> # save to file
> write.table( hugeex.qnorm.data,
  file="BestMatch_GTEx_RP_probe_intensities_QN.txt", col.names=TRUE,
  row.names=FALSE,  quote=FALSE,  sep="\t" )
```

All training data needed is now available to use MaLTE.

D.5 Preparing the data for training-and-testing

We can now use MaLTE to prepare the data for training and testing. First, we load the package after launching R.

```
> library( MaLTE )
```

We use the `prepare.data()` function which takes as arguments the *samples.txt, GTEx_subset_gene_rpkm_QN.txt, BestMatch_GTEx_RP_probe_intensities_QN.txt.gz*, and *gene_probesets_HuGene_Ens72.txt.gz* (available in MaLTE-aux directory).

```
> prepare.data( samples.fn='samples.txt',
  hts.fn='GTEx_subset_gene_rpkm_QN.txt.gz',
  ma.fn='BestMatch_GTEx_RP_probe_intensities_QN.txt',
  g2p.fn='gene_probesets_HuGene_Ens72.txt.gz', raw=TRUE )
```

We then read in the training and test data.

```
> tt.ready = read.data( train.fn='train_data.tar.gz',
  test.fn='test_data.tar.gz' )
```

D.6 Prediction

Prediction can either be performed on the test data training data (out-of-bag, OOB). We outline OOB shortly. First, we need to define training-and-test parameters (additional parameters are specified in Section 9).

```
> tt.params = TT.Params() # default parameters
```

This sets the following values:

```
> tt.params
mtry             = 2
ntree            = 1000
feature.select   = TRUE
min.probes       = 15
cor.thresh       = 0
OOB              = FALSE
QuantReg         = FALSE
Tune (OOB cor.P) = NA
```

Predictions are performed using the `array2seq()` function.

```
> tt.seq = array2seq( tt.ready, tt.params )
```

OOB predictions are performed using the `array2seq.oob()` function.

```
> tt.seq.oob = array2seq.oob( tt.ready, tt.params )
```

D.7 Filtering by OOB

OOB filtering is performed using the `oob.filter()` function, which takes a `tt.seq` pair (`tt.seq` and `tt.seq.oob`) and a filtering threshold together with the correlation to use for filtering (Pearson (default) or Spearman),

```
> tt.filtered = oob.filter( tt.seq, tt.seq.oob, thresh=0.5, method='spearman' )
```

D.8 Collating predicted gene expression values

Predicted values should be collated as a data frame using the `get.predictions()` function. First, we get the names of test samples using the `get.names()` or `get.test()` functions.

```
> test.names = get.names( samples.fn='samples.txt', test=TRUE )
> test.names = get.test( samples.fn='samples.txt' )
```

The predictions can be quantile-normalized but this may be omitted.

```
> df.malte.qn = get.predictions( tt.filtered, sample.names=test.names,
  qnorm=TRUE )
```

then saved to file

```
> write.table( df.malte, file='RP_malte_OOB_0p5_QN.txt',
  col.name=TRUE, row.names=FALSE, quote=FALSE, sep='\t' )
```

E Experimental features

E.1 Per-gene/transcript tuning

Set `tune.cor.P=TRUE` in `TT.Params()`

```
> tt.params = TT.Params( tune.cor.P=TRUE )
```

E.2 Incorporating principal components

Principal components of probe intensities may be incorporated into the prediction. The same principal component will be used for all genes/transcript therefore they must be incorporated into the *samples.txt* file to be dispatched to all genes/transcripts. We have experimented with the first 10 principal components.

```
> load( "dnorm.data.Rdata" ) # load 'qnorm.data' from before
> pc.data = princomp( qnorm.data )

# save the first 10 principal components
> pcs = pc.data$loadings[,1:10]
> samples = read.table( 'samples.txt', header=TRUE )

# create new samples data frame augmented with principal components
> samples.aug = cbind( samples, pcs )
> colnames( samples.aug ) = paste( 'P', 1:10, sep='' )

# create a new samples file
> write.table( samples.aug, file='samples_aug.txt',
  col.names=TRUE, row.names=FALSE, quote=FALSE, sep='\t' )
```

We then run `prepare.data()` using *samples_aug.txt* (instead of *samples.txt* as before) setting the variable `PCs=TRUE`. This creates two files: *train_PCs.txt* having the training principal components and a corresponding *test_PCs.txt*.

```
> prepare.data( samples.fn=samples.fn, hts.fn=hts.fn, ma.fn=ma.fn,
  g2p.fn=g2p.fn, PCs=TRUE )
```

Also, we need to run `read.data()` setting `PCs.present=TRUE`, `train.PCs.fn='train_PCs.txt'` and `test.PCs.fn='test_PCs.txt'`.

```
> tt.ready.pc <- read.data( train.fn="train_data.txt.gz",
  test.fn="test_data.txt.gz", PCs.present=TRUE,
  train.PCs.fn="train_PCs.txt", test.PCs.fn="test_PCs.txt"  )
```

The principal components are now embedded in the data for each gene.