MaLTE: Machine Learning of Transcript Expression
An R Package Implementing the MaLTE Framework

Paul Korir and Cathal Seoighe

Contents

Abstract

MaLTE is a novel approach to gene and transcript expression prediction that uses a supervised learning approach to achieve performance superior to conventional microarray summarisation. It learns from a gold-standard how best to use fluorescent probe intensity values. This leads to more accurate and absolute expression estimates as well as an simple facility to obtain expression estimates for individual transcript isoforms.

1   Introduction

The final step in microarray data preparation is quantification of raw intensities into gene or transcript expression estimates. This is referred to as *probe summarisation* [Irizarry 2003]. Many summarisation algorithms currently exist for this. Most of these algorithms suffer from two serious problems: the produce relative, as opposed to absolute, expression estimates [Irizarry 2005; Fu 2009] and they are unable to quantify the expression of individual transcript isoforms [cite].

We set out to supplement the summarisation step with a supervised learning approach. In supervised learning, the user is required to provide a true estimate by which to build learning models but in gene expression experiments the true estimate is seldom known. However, it is possible to use high quality estimates instead and here we chose to use RNA-Seq expression estimates.

MaLTE builds gene-specific models from a training and test dataset. The training dataset consists of microarray and RNA-Seq data while the test set consists of microarray data alone. The microarray fluorescence probe intensities are quantile-normalised and background corrected. We then train a model using the probe intensity values as predictors and the RNA-Seq values as the response. Finally, we predict RNA-Seq estimates for the test samples using only the probe intensities.

This approach overcomes the two main setbacks of conventional algorithms. It transforms expression estimates onto an absolute scale thus dramatically improving the within-sample correlations. MaLTE also 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, which increase statistical power for tasks such as differential expression analysis and eQTL[1] mapping. Finally, a tree-based learning introduces an easy way to filter out poorly-predicted genes through the use of out-of-bag estimates.

MaLTE is particularly suited to large-scale genetic studies involving hundreds to thousands of samples. A representative subset of samples would be subject to long-read, high-depth and wide-coverage RNA-Seq to ensure a high-quality gold-standard transcriptome upon which learning is based. All samples are subjected to the same

---

[1]Expression quantitative trait loci

type of microarray assay. We strongly recommend that the same RNA extractions are used for both RNA-Seq and microarray assays to eliminate batch effects, which can substantially reduce prediction performance. Such an approach will save costs (cheaper than an all-RNA-Seq experiment) while boosting statistical power (many samples with better expression estimates compared to microarray alone).

This documents describes how to used 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 roadmap for future development as well as a detailed description on how the MaLTE package is built. We invite bug reports, comments and suggestions through the following address: paul.korir@gmail.com.

2   Conventions Used

- We interchangeably use *high-throughput sequencing (HTS)* and *RNA-Seq*. RNA-Seq is one incarnation of HTS.

- *Array* and *microarray* are equivalent.

- A *map* is a tab-delimited text file with two columns with each column consisting of identifiers/names.

- Array data used here corresponds to that from *Affymetrix GeneChip$^{\circledR}$ Human Exon v.1.0 ST* arrays. So far we have only tested MaLTE using Affymetrix GeneChip$^{\circledR}$ Human Exon and Human Gene arrays.

- RNA-Seq data used is already quantified and is independent of the HTS platform used.

- 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)

3   System Requirements

- R (2.14.0 or greater) installed on GNU/Linux: MaLTE has been tested on Scientific Linux version 5

- Python 2.7 or later

- party R package

- multicore R package

3

4 Installation

MaLTE may be downloaded from <here>. There are two ways to install MaLTE: via
GNU/Linux shell and R shell.

1. Via GNU/Linux shell

   ```
   me@home ~$ R CMD INSTALL MaLTE_<release>.tar.gz
   ```

2. Via R shell

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

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

```
> library( MaLTE )
```

   or, quietly:

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

5 Gene Expression Prediction

5.1 Quick Start Guide

This section provides a quick introduction to using MaLTE. Detailed instructions in-
corporating descriptions of the various file and there respective formats is provided in
Section <>. We assume 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-normalised and back-
ground 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: Initialise training parameters

```
> tt.params <- TT.Params()
```

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 output to a text file for downstream analyses

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

5.2   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)

- 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 of between gene identifiers and probe set identifiers. Probe set identifiers are provided by the array manufacturer as part of the array description.

We now describe each file in detail under the following sub-headings: purpose, generic designation, header and structure.

5.2.1 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.

5.2.2 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.
Header	'gene_id<tab>Sample01<tab>...<tab>SampleN'
Structure	Tab-delimited
	All rows must be unique
	No comments are allowed

5.2.3 Microarray data file

Purpose	This file provides the microarray expression estimates.
Generic designation	'ma_data.txt' OR 'ma_data.txt.gz' if inessential columns have been removed (see section on Preparing Input Files)
	'raw_ma_data.txt' OR 'raw_ma_data.txt.gz' if inessential columns are still present (see section on Preparing Input Files)
	Any suitable name will do.
Header	'probe_id<tab>Sample01<tab>...<tab>SampleN'
Structure	Tab-delimited
	Comments must begin with a pound/hash ('#') symbol.

5.2.4   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.
Header	'gene_id<tab>probeset_id'
Structure	Tab-delimited
	Probe set identifiers may be missing for some genes.
	No comments are allowed.

5.3   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.

2. HTS data. This file may be constructed using customised scripts that collate the HTS expression estimates output from programmes such as Cufflinks and DESeq. The order of sample columns is unimportant. There are several online datasets 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-normalisation (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. You require to create an account to download library files and software from this site. 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:

   ```
   me@home ~$ 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
   ```

   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' like so:

```
me@home ~$ cut -f1,5,8- raw_ma_data.txt > ma_data.txt
```

4. Zip the file to save space.

```
me@home ~$ gzip raw_ma_data.txt
me@home ~$ gzip ma_data.txt
```

5. Gene-to-probeset map. This file may be downloaded directly from BioMart, particularly for popular arrays. Alternatively, the user will have prepare it themselves. 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 written in your favourite scripting language.

Please consult the BEDTools website on how to intersect two BED files.

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

6 Transcript Expression Prediction

6.1 Quick Start Guide

This describes how to perform transcript 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 V in the 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 )
> df.oob.txs <- get.predictions( tt.seq.oob.txs, train.names ) # again, just
  for the fun of it
```

6.2 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.

9

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.
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.

6.2.2 GEP training 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	'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
	Vectorised matrixfootnote of fluorescent probe intensities

6.2.3 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
	Vectorised matrixfootnote of fluorescent probe intensities

6.2.4 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.
Header	'tx_id<tab>Sample01<tab>...<tab>SampleN'
Structure	Tab-delimited
	All rows must be unique
	No comments are allowed

6.2.5 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.
Header	'gene_id<tab>tx_id'
Structure	Tab-delimited
	No comments are allowed.

6.3 Preparing Input Files

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

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

3. GEP test data. This file is automatically generated after running 'prepare.data()' Please see Section 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. Popular examples include Cufflinks (1 and 2), IsoEM and RSEM. As suggested in HTS data description

in 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 off BioMart [cite].

7 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.

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. This involves randomly selected a subset of the training samples and constructing a tree using a recursive partition approach. This is repeated for hundreds to thousands of trees. Each sample (observation) is only used to construct a subset of all the trees. We can then use the trees in which it is absent to predict its value. This also applies to all the other samples (observations).

The resulting predicted estimates are called the out-of-bag (OOB) estimates because they predict each observation using the subset of trees for which it was 'out of bag'. Our results suggest that the OOB estimates give a good indication of how reliably a particular gene will have its expression predicted. We measure this performance using the OOB Pearson correlation, which is the correlation between the training HTS and the OOB predictions. An OOB Pearson correlation threshold of zero ($r_{OOB} > 0$) is currently set as a default filter threshold. This is referred to as OOB filtering.

Steps IX and X of show GEP OOB filtering and step VII and VIII show TIEP OOB filtering.

Collation can only performed on filtered data. The sample names must first be obtained (GEP: Step XI; TIEP: Step X). Sample names are extracted from the Sample names file.

Finally, extracted data may be written to a tab-delimited (or otherwise) text file using the usual R functions.

8 License

```
or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
for more details.

You should have received a copy of the GNU General Public License along
with this program.  If not, see <http://www.gnu.org/licenses/>.
```

9   Future Work

We hope to make further improvements to MaLTE depending on whether it finds widespread use. The following list of features may be added at a future data:

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 GNU/Linux and 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* file.** Currently, the sample names file is underutilised. It should be 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.

10   Bug Reports

Please send all bug reports and feature requests to paul.korir@gmail.com with the subject 'MaLTE Bugs' of 'MaLTE Features', respectively.

A The MaLTE Framework

The MaLTE framework consists of applying a learning algorithm between gold-standard expression estimates and fluorescent probe intensities. We refer to is as a framework because the constituent components may be modified or entirely overhauled. It is not restricted to the learning algorithms used in the R MaLTE package (conditional random forest); any good learning algorithm may be used. It is also not restricted to one particular gold-standard. Currently, we use RNA-Seq as the gold-standard HTS quantification technique.

Figure NKK below shows a schematic of the MaLTE framework. Only one gene is presented. However, learning occurs for each gene independently. MaLTE incorporates simple feature selection by picking the best 15 probes that correlate with the RNA-Seq expression.

B 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, and OOB.

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

C   Function and Methods Table

Function/Methods	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 undlying 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, ntree, feature.select, min.probes, cor.thresh, OOB	tt.params	Constructor for objects of class TT.Params
run, oob.run	TT.Ready object, tt.params, OOB	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 'predict' The tuned parameters are obtained by running 'tune'.
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
tune	tt.ready	tt.params	Uses the training data to find the best parameters to use. Evaluation is based on OOB estimates only. tt.params is an object of class TT.Params
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