**package** ATACanalysis

The ATACanalysis package provides turnkey data analysis for the microarray readout of Surajit Dhara’s cancer prognostic assay. It implements operations first explored in the Matlab file ATACarrayWorkflow.m.

Copyright © 2019 by Bo Curry. All rights reserved.

# Class ATACanalysis

The class ATACanalysis implements the sequential steps of the ATAC array analysis. It is called with two arguments, the full path to the ATAC region definitions (the design directory), and the full path to the directory containing the Agilent Feature Extraction (FE) output files (the experimental directory). It then sequentially calls public methods to proceed stepwise through the analysis workflow. Although these methods are currently not invoked from outside the class, they are defined as **public** in case future hosts might choose to do so. The ATACanalysis class has no publicly accessible variables.

**public** ATACanalysis**(String** exptdir**)**

The ATACanalysis constructor does nothing but store the path to the experimental directory .

## readFEdata ()

**public int** readFEdata**()**

Creates a temporary instance of the FEdata class to read all the valid FE output files of a single design that it can find in the exptdir. It reads the name fields “FeatureNum", "ControlType", "ProbeName", and "SystematicName" for the design, and the value fields "gBGSubSignal", "rBGSubSignal", "LogRatio", "gIsFeatNonUnifOL", and "rIsFeatNonUnifOL" for each array in the experimental directory. It then invokes the FEdata methods filterValues() and backgroundNoise() to filter out (i.e. replace with NaN) the values of features flagged as nonuniformity outliers, and to compute the background noise levels for each channel of each array from the gBGSubSignal and rBGSubSignal values. The ControlType, ProbeName, and SystematicName name fields and the LogRatio value field are stored locally, and the FEdata instance is discarded.

## readHybSummary ()

**public int** readHybSummary**()**

Reads an association between FE output filenames and user defined sample names. It looks for a .txt file in the experimental directory containing “HybSum” in its name. If such a file exists, it should be a tab delimited txt file containing columns headed “Array” and “Sample”. It may also contain other columns, which are ignored. This method looks at each valid FE output file that was read by readFEdata() and reduces its file name to a canonical arrayName. This name consists of an “S”, followed by the last five digits of the slide barcode, followed by the subarray designation, if any. For example, Feature Extraction produces an output file named SG19018632\_258579010004\_4\_17\_19\_S001\_CGH\_1201\_Sep17\_2\_1.txt from subarray 2\_1 of slide 258579010004. The arrayName of this sample will be S10004\_2\_1. For each such FE file, readHybSummary() searches the Array field of the HybSummary file for an exact match to this arrayName. If it finds one, it assigns that sample a sampleName from the corresponding Sample column. If it doesn’t find one, the sample is named with the arrayName.

## combineFeaturesToProbes ()

**public void** combineFeaturesToProbes**()**

The array design contains negative and positive control features, CGH backbone probes, ATAC control probes, and ATAC differential probes. See the Array Design section below for details. Many of the CGH probes and all of the ATAC probes have multiple replicate features on the array. This method computes the median log ratio of all replicated features that pass the nonoutlier filter, to generate a single log ratio value for each of the 7405 CGH backbone probes, the 1544 ATAC control probes, and the 5210 ATAC differential probes.

## readProbeRegions ()

**public int[]** readProbeRegions**(String** designpath**)**

Dhara et al identified 336 ATAC control regions, which are expected to be accessible in all patients, and 1092 differential regions, whose varying accessibility from patient to patient constitutes the diagnostic signal. These regions are defined by their genomic coordinates in the files ATACctrlRegions.txt and ATACdiffRegions.txt. These files must exist in the directory pointed to by designpath.

The ATAC differential regions are further characterized as “red” or “blue”, depending on whether their average accessibility tends to increase or decrease in the clinically relevant subset of samples.

This method reads these two files, and assigns each non-control probe to a category based on its genomic coordinates. Probes which overlap ATAC regions are assigned to those regions. Probes that don’t overlap any regions are classified as CGH backbone probes.

Agilent’s SureCall probe design software was unable to find any probes with sufficiently high quality scores in some of these regions, whereas many probes were found in other regions. A total of 312/336 ATAC control regions, 244/370 red differential regions, and 688/722 blue differential regions had at least one probe.

## centerLogratios ()

**public void** centerLogratios**()**

The log ratios are normalized by setting the median log ratio of the CGH backbone probes to zero. This requires a previous call to readProbeRegions() to identify which probes those are.

## combineProbesInRegions ()

**public void** combineProbesInRegions**()**

Each of the ATAC regions is targeted by a number of probes, sometimes zero. This method computes the log ratio of each region as the median of probes targeting the region. It assumes a previous call to centerLogratios ().

## writeRegions ()

**public void** writeRegions**(String** outfile**)**

Writes the log ratios of each region in each sample to a file in the experimental directory called "ATACregionLRs.txt". The differential regions are written first, then the ATAC control regions, and finally the log rations of the CGH backbone probes, for comparison.

## isSampleRed ()

**public boolean[]** isSampleRed**()**

The goal of this analysis is to classify the samples as either “red” or “blue”, depending on which differential ATAC regions predominate. This method computes the median log ratio of red regions and of blue regions, and returns true for a sample if the red medians are higher. In red samples, the log ratios in red regions tend to be significantly higher, but in blue samples the red and blue regions are often similar. Thus it can happen that a blue sample has a slightly higher log ratio in the red regions, and is incorrectly called red by this method. These cases can be distinguished from true red samples, because diffMlogp**()** will report very low significance. Requires a prior call to combineProbesInRegions**().**

## diffMlogp ()

**public float[]** diffMlogp**()**

This method performs a t-test on the log ratios of red and blue regions. It returns the negative log base ten of the probability that the log ratios of the two types of regions have equal means. Values higher than 2.0 correspond to p < 0.01, signifying that the red and blue regions are significantly different. Requires a prior call to combineProbesInRegions**().**

# Class FEdata

The FEdata class implements methods to read annotations and data from Agilent Feature Extraction output files, and selectively filter and access these data.

**public** FEdata**(String** exptdir**, String[]** nameFields**, String[]** valueFields**)**

**public int** amadid; // The design ID

**public String[]** sampleNames; // Simplified filenames

An instance of the FEdata class can store the data from FE output files from a single experiment using a single array design. The FEdata constructor saves the pathname to the directory containing the FE output files, and copies the requested nameFields and valueFields. The FE output files include everything you might want to know about a microarray assay and much, much more. Only a handful of the dozens of data fields are interesting for a particular analysis, but different experiments will be interested in different fields. Name fields are associated with an array design and will be the same for all the arrays in an experiment. Examples are ProbeName and ControlType. Value fields, such as gBGSubSignal and gIsFeatNonUnifOL, will typically have different values for different arrays in the experiment. The constructor doesn’t actually read the data, but it determines where the data files are to be found and which fields to read.

## ReadFEdata ()

**public int** ReadFEdata**()**

This method loads data into the requested fields from FE output files in the experimental directory. It considers all \*.txt files in the exptdir to potentially be FE output files. For each such file, it looks for the header field “FeatureExtractor\_Barcode” in line two of the file. If it finds this field name, it then parses the field value from the next line, and extracts the integer amadid, which is the Agilent code for the array design (AgilentMicroArrayDesignID). If this is the first valid FE file encountered, then this amadid determines the array design that this instance of FEdata will consider. If this isn’t the first valid FE file encountered, then the amadid of this file must match the amadid of earlier files, or the file is ignored.

Once ReadFEdata has decided which files to include in the experiment, it sorts them alphabetically, and makes simplified versions of the file names available to the host in the sampleNames array. It then opens the first file, and parses line 10, looking for exact matches to the requested nameFields and valueFields, ignoring any it can’t find. It then reads the contents of the name fields for each feature of the array. These are all stored as Strings, even though some of them could be parsed as booleans or ints. ReadFEdata() then reads the contents of the value fields for each feature of each array, parsing them all as floats. It returns the number of valid FE files it found.

## FEnameFieldValue ()

**public String[]** FEnameFieldValue**(String** fieldname**)**

**public String[]** FEnameFieldValue**(int** ifld**)**

This method gives the host access to the name fields. It returns a vector of strings as long as the number of features on the array.

## FEvalueFieldValue ()

**public float[][]** FEvalueFieldValue**(String** fieldname**)**

**public float[][]** FEvalueFieldValue**(int** ifld**)**

This method gives the host access to the value fields. It returns a matrix of floats[narrays][nfeatures].

## backgroundLevels ()

**public float[]** backgroundLevels**(String** gsigfield**, String** rsigfield**)**

The background signal level is estimated as the median signal of the features designated as negative controls. The results are returned in a single float[2\*narrays], the green channel backgrounds first for each array, if any, then the red channel backgrounds, if any. In a one-color assay the backgrounds reported for the unscanned (usually red) channel will be zero.

## backgroundNoise ()

**public float[]** backgroundNoise**(String** gsigfield**, String** rsigfield**)**

The background noise level is estimated as the robust standard deviation of the signals from negative control features. This is one standard deviation above zero of negative features, and is used to surrogate signals and to estimate detection limits. The results are returned in a single float[2\*narrays], the green channel background noise first for each array, if any, then the red channel backgrounds, if any. In a one-color assay the background noise reported for the unscanned (usually red) channel will be zero.

## filterValues ()

**public void** filterValues**(String** valueField**, String** filterField**)**

FE provides several boolean fields to flag features as aberrant or questionable. The most useful of these are the nonuniformity flags gIsFeatNonUnifOL and rIsFeatNonUnifOL. The filterValues() method uses any field to filter any other field. Each feature on each array that reports a nonzero value of the filter field will have its valueField replaced with Float.NaN. Downstream methods then ignore these features.

# Running standalone

A standalone version of the java code exists in [\\shasta.scs.agilent.com\kryptonite\share4\EpiGenomics\DharaATACchip\javaCode\ATACanalysis](file:///\\shasta.scs.agilent.com\kryptonite\share4\EpiGenomics\DharaATACchip\javaCode\ATACanalysis%20) .

In the bin subdirectory is the Windows batch file ATACanalysis.bat, which takes as its input an experimental directory containing FE output files, and generates the output files ATACregionLRs.txt and SampleClassification.txt in the same directory.

The jar file is executed from the command line as

java -jar %cd%\..\bin\ATACanalysis.jar %designdir% %exptdir%

where designdir is the full pathname to the directory containing the “ATACctrlRegions.txt” and “ATACdiffRegions.txt” files defining the ATAC regions, and exptdir is the full pathname to the directory containing the FE output files. ATACanalysis.bat defaults the designdir to ..\config\, where .. is the current directory.

Running this batch file will generate two files in the experimental directory, “ATACregionLRs.txt” containing the average log ratios of probes in each region, and “SampleClassification.txt”, containing the classification and p-value for each of the samples.

# The assay

The original Dhara assay used ATAC-seq to explore chromatic accessibility (<https://en.wikipedia.org/wiki/ATAC-seq>). Dhara postulated that a microarray readout of the ATAC samples would be faster and cheaper than a sequencing readout. The experiment can be summarized as:

1. Use ATAC to attach PCR primers to accessible genomic regions
2. Amplify those regions.
3. Label the sample with cy5 using the Agilent ULS labeling kit
4. Hybridize 200 ng of DNA to a microarray, along with 200 ng of cy3 labeled genomic DNA.
5. Read out the log ratios of probes in selected genomic regions to the reference signal.

# Array design

The 20181011\_eArray\_Design1, amadid 85790, was the original design tested. This design contains 724 negative and 3162 positive control features, 7405 CGH backbone probes of which 1002 are replicated five or more times (11450 features total), 1544 ATAC control and 5210 ATAC differential probes replicated mostly seven times.

The control probes are used by FE to estimate backgrounds, correct for spatial nonuniformities, and compute various array QC metrics. They are not used by ATACanalysis. The CGH backbone probes are high quality probes from a standard sparse tiling of the whole genome. They are used to estimate intraarray reproducibility and to center log ratios. The ATAC probes are probes in Agilent’s genomic probe database that overlap regions designated by Surajit Dhara et al as interesting regions. All probes in the Agilent database scoring 0.4 or above in the specified regions were chosen. Many of these probes have never before been empirically tested, and are expected to include many probes of significantly lower quality than the CGH backbone probes (low quality probes are generally those that cross hybridize to nontargeted regions of the genome, thus diluting the signal from targeted regions and compressing log ratios, The Agilent database scores probes by their predicted quality, but the scores of untested probes are not always reliable). Probes to the ATAC control regions are expected to be accessible in all patient samples, whereas probes to the ATAC differential regions are expected to be differentially accessible in different patient samples.

Now that data from this array design Is available, we can cull the more poorly performing ATAC probes in a followup v2 design. The control grid and CGH backbone probes are unlikely to change significantly, while there will be fewer ATAC probes replicated more frequently.