1 Introduction

CyberT works by calculating a bayesian regularized t-statistic for each gene. This is a lot of language to basically say that we calculate a t-statistic for each gene, but we tweak the variance of each group a little using genes with similar expression values.

The Mann-Whitney (or Wilcoxon Rank-Sum, or Mann-Whitney-Wilcoxon (MWW)) test is the non-parametric version of the t-test. Non-parametric in this case means that the statistic works on the ranks of the data, rather than assuming some parameterized underlying distribution. Proper analysis of the t-statistic is dependent on assuming that each group is from a normal distribution, i.e., is from a distribution parameterized by a mean and variance. The MWW statistic is very closely related to the area under the receiver operating characteristic curve, i.e., the area under the ROC curve, or AUC. In fact, if x is a vector of predictions on a set of positive examples, and y is a vector of predictions on a set of negative examples, then:

$$\mathrm{AUC}(x, y) = \mathrm{MWW}(x, y)/(|x| * |y|)$$

In this post, I'm going to demonstrate how to calculate the AUC for each gene in a microarray dataset. This basically says, if we used a single gene's expression value as a predictor for two different phenotypes, what would the AUC of this predictor be? Or another way to think of it is as a MWW statistic that is normalized to be between 0 and 1, or really between 0.5 and 1. One can also calculate p-values for these AUC's based on the p-values of the MWW stats.

2 The Function

Here is a function to do this calculation. This actually calculates a one-vs-rest for all groups.

```
> ##############################################################################
> #   calcAUCArray
> #
> #   Function to calculate the AUC by row for a matrix of predictions
> #
> #   Input:
> #       x           -   the data.frame or array
> #       repPerCond  -   a vector of the number per condition (assumes
> #                       the data.frame is in order).
> ##############################################################################
> calcAUCArray <- function(x, repPerCond, calcPVals = T){
+     classFactor <- factor(rep(1:length(repPerCond), repPerCond))
+
+     #The Auc is equiv to the Mann-Whitney or Wilcox test.
```

```
+     #For generalizations beyond 2 class, do a one-vs-rest scheme.
+     classIndices <- tapply(1:ncol(x), classFactor, list)
+
+     theRes <- sapply(classIndices, function(inds){
+         sapply(as.data.frame(t(x)), function(y){
+             maxW <- length(inds)*length(y[-inds]);
+             theTest <-  wilcox.test(y[inds], y[-inds]);
+             res <- theTest$statistic/maxW;
+
+         })
+     })
+
+     ## This line flips the prediction (AUC < 0.5 is worse than random)
+     theRes <- apply(theRes, 2, function(x){ifelse(x > 0.5, x, 1-x)})
+     colnames(theRes) <- paste("AUC_Cond_", 1:length(repPerCond), sep='');
+     if (calcPVals){
+         thePVals <- sapply(classIndices, function(inds){
+             sapply(as.data.frame(t(x)), function(y){
+                 theTest <-  wilcox.test(y[inds], y[-inds]);
+                 theTest$p.value;
+             })
+         })
+         colnames(thePVals) <- paste("AUC_PVal_Cond_", 1:length(repPerCond), sep='');
+         theRes <- cbind(theRes, thePVals)
+     }
+     theRes;
+ }
```

3 An Example

Here, let's fake up some data and show how the MWW could be useful in finding some genes of interest. First, make some fake completely normal data:

```
> data <- data.frame(matrix(rnorm(100), nrow=10))
> colnames(data) <- c(sprintf('Ctrl%d', 1:5),
+                     sprintf('Expt%d', 1:5))
> head(data)
```

```
        Ctrl1       Ctrl2       Ctrl3      Ctrl4      Ctrl5      Expt1
1  1.05075025 -1.30088148  2.04442575  0.2396374 -0.1640639 0.46255951
2 -0.04274524  0.14674350  0.19292496 -3.0934266  1.3382232 1.44391446
3  0.32498054 -0.38788724 -1.18156093  0.4588947  0.4427220 1.74483569
4  1.61588825 -0.78319953 -0.07895495 -0.3512329  1.7633533 0.08020991
5 -0.15815873 -0.71218677 -0.54358175  1.1996652 -0.9436978 0.64061365
6 -1.68811312 -0.01021523  0.81717540  1.0148944 -1.6237105 1.39887543
       Expt2      Expt3       Expt4        Expt5
```

```
1  0.1977868  1.2951139 -0.94278682 -0.458105214
2  1.1318220 -0.3654534  0.45353342 -0.514351083
3 -0.9689614 -0.7129198  0.07177276 -1.175522844
4 -1.7997450 -0.4597961  0.56408353  1.044657360
5 -0.4004646  0.5248907  0.49722154  0.009011202
6  0.5671171  0.9390662 -0.12485235  0.362181267
```

Then, let's randomly pick a gene and just right shift the expression of just the Expt cols here by 2.

```
> fakeGeneId <- sample(1:10, 1)
> fakeGeneId

[1] 10

> data[fakeGeneId, 6:10] <- data[fakeGeneId, 6:10] + 2;
```

Now, let's calculate the AUCs and p-values here.

```
> repPerCond <- c(5,5) # 5 in each group
> aucs <- calcAUCArray(data, repPerCond)
> aucs

      AUC_Cond_1 AUC_Cond_2 AUC_PVal_Cond_1 AUC_PVal_Cond_2
V1.W        0.56       0.56     0.841269841     0.841269841
V2.W        0.60       0.60     0.690476190     0.690476190
V3.W        0.60       0.60     0.690476190     0.690476190
V4.W        0.60       0.60     0.690476190     0.690476190
V5.W        0.76       0.76     0.222222222     0.222222222
V6.W        0.68       0.68     0.420634921     0.420634921
V7.W        0.52       0.52     1.000000000     1.000000000
V8.W        0.80       0.80     0.150793651     0.150793651
V9.W        0.56       0.56     0.841269841     0.841269841
V10.W       1.00       1.00     0.007936508     0.007936508

> aucs[fakeGeneId,]

     AUC_Cond_1      AUC_Cond_2 AUC_PVal_Cond_1 AUC_PVal_Cond_2
    1.000000000     1.000000000     0.007936508     0.007936508
```

There you go! AUCs and you should see a much lower p-value for the fakeGeneId!