5-2017

Statistical Methods for Assessing Individual Oocyte Viability Through Gene Expression Profiles

Michael O. Bishop
Utah State University

Statistical Methods for Assessing Individual Oocyte Viability Through Gene Expression Profiles

By

Michael O. Bishop

A report submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Statistics

Approved:

[illegible]
Dr. John R. Stevens
Major Professor

[illegible]
Dr. Adele Cutler
Committee Member

[illegible]
Dr. Guifang Fu
Committee Member

Utah State University
Logan, Utah

2017

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. John R. Stevens for his resources, guidance, mentoring and example throughout my undergraduate degree but even more so during my graduate degree at Utah State University. Dr. Stevens has been patient with me and made this work possible. He has guided me to the roles and responsibilities of an active researcher, while being considerate of the additional responsibilities a father has in raising a young family.

Above all others, at this time I am forever grateful for the constant support and love given by my wife, Alessandra. None of this would have been possible without her constant efforts to keep our home life in order, including raising our son Owen. Despite the many hours spent on this project, Alessandra has always been understanding, encouraging and supportive.

ABSTRACT

STATISTICAL METHODS FOR ASSESSING INDIVIDUAL OOCYTE VIABILITY THROUGH GENE EXPRESSION PROFILES

By

Michael O. Bishop

Utah State University, 2017

Major Professor: Dr. John R. Stevens
Department: Mathematics and Statistics

Oocytes are the precursor cells to the female gamete, or egg. While reproduction may vary from species to species, within humans and most domesticated animals, the oocyte maturation process is fairly similar. As an oocyte matures, there are various processes that take place, all of which have an effect on the viability of the individual oocyte. Barring outside damage that may come to the oocyte, one of the primary reasons for non-viability is that of abnormal gene expression. Within this project, we focus on two oocyte maturation techniques: *in vivo* (IVV) derived oocytes (our gold-standard) and *in vitro* matured (IVM) oocytes. A great disparity exists between the viability rates of the two origination techniques, and this disparity has led to low yields and inefficiency in the fields of cloning, fertility treatments, as well as personalized medicine.

Within our project we use existing swine oocyte gene expression profile data as a proxy measure of viability, based on the similarity to IVV oocytes. Four statistical techniques for assessing the individual oocyte viability are proposed and compared, including: a weighted root mean squared deviation (wRMSD) approach, a distance kernel p-value approach, a distance tolerance interval approach, and a classification tree method. The relative performance of these four measures is discussed.

1. Introduction

Oocytes are the precursor cells to what we often think of as the female egg cell. They operate and grow in essentially the same way within swine as they do in humans (Thomas, 2016). One main reason that one would be interested in studying oocytes is that they are the key to studying life, reproduction, and all other pertinent matters. Without access to oocytes, researchers in the fields of cloning, fertility treatments, and others would not be able to do their work. Currently, there are numerous methods of deriving (growing) oocytes, each with its benefits. *In vivo* (IVV) derived oocytes are held as the gold standard for viability, and other known origination methods are sub-par by comparison. However, alternatively derived oocytes do have many traits that are desirable. The *in vitro* maturation (IVM) method is one such method of oocyte derivation, as it allows the researchers to have access to the oocyte from early on. There are however, many problems associated with this method. From past studies, it is shown that the viability rate associated with IVV derived oocytes is upwards of 95%, however for IVM oocytes (and other similar methods) the viability rate is roughly only 25% (Dr. S. Clay Isom, personal communication). Similar low viability concerns arise with the SCNT (somatic cell nuclear transfer, or traditional cloning) origination method. With such a low viability rate it is only natural to try to alleviate the frustration and waste that comes from focusing time, resources and energy on oocytes that are simply not viable. But how can one know if an oocyte is viable prior to investing in it? In general, one cannot. However, methods can be developed that test individual oocytes for viability, establishing a viability-optimized general gene expression profile. As laboratory originated oocytes can be treated with a wash that promotes growth and expression patterns in chosen directions (Dr. S. Clay Isom, personal communication), it is apparent that establishing methods to test individual oocyte viability and thereby derive a general viable profile for oocytes would, in theory, increase the viability of non-IVV oocytes.

Kwon et al. (2015) took embryos that were obtained from three different origination methods (IVV, IVM, and SCNT) and examined the gene expression profiles for 15 different genes of interest. In their project, a Weighted Root Mean Squared Deviation (wRMSD) was calculated based on expression level deviation from the mean expression level of the origination group. The average wRMSD for each origination group was calculated and compared. They found that the difference between the IVV group and the SCNT group was greater than the difference between the IVV group and the IVM group. These conclusions were relevant for comparing methods of origination, however we desire to compare individual oocyte viability.

Within our project, the goal has been to establish origin-independent methods of evaluating individual oocyte viability, based simply on the observed gene expression profiles. As the process of collecting the observed gene expression profiles of oocytes is destructive, making the oocytes non-viable (and thereby making it impossible to tell if the oocyte would have been viable or not, simply by observing), there are a number of assumptions that one must make. First, assume that IVV-derived oocytes are the "gold standard" for oocyte viability. This assumption does not seem to be erroneous, as historically over 95% of all IVV-derived oocytes have been observed to be viable (Dr. S. Clay Isom, personal communication). The second assumption that is made is that the closer an oocyte's gene expression profile gets to the mean of the IVV gene expression profiles, the more likely the oocyte is to be viable. Third, assume that the IVV oocyte gene expression profiles are a representative sample of all IVV oocyte gene expression profiles, and likewise for the IVM oocytes within the data.

Dr. S. Clay Isom provided the project dataset in the form of an Excel spreadsheet, which contained the gene expression profiles for 29 IVV derived oocytes as well as 29 IVM derived oocytes. Each of the gene expression profiles was expressed in the form of a log2 fold-change on 67 genes of interest compared to a common "housekeeping" gene. These specific 67 genes were selected, as they are affiliated with early cell growth, regulation and viability (Dr. S. Clay Isom, personal communication). Overall there were few values that were missing (about 4.4%), but computationally if the case arose that a value was missing, that value was omitted from gene level calculations in the following methods. Once again, actual viability of these individual sample oocytes is unknown, as they yielded the "ultimate sacrifice" for science, though from the assumptions, greater similarity to IVV is treated as more likely to be viable.

In this MS project, an adaption (on the single oocyte level) of the Kwon et al. (2015) wRMSD method is summarized in Section 2, along with three other novel methods – a Distance Kernel P-value method in Section 3, a Tolerance Interval method in Section 4, and a Decision Tree method in Section 5. Each of these methods is then compared for accuracy via simulation in Sections 6 and 7, and the methods' performance is discussed in Section 8.

2. Weighted Root Mean Squared Deviation Kernel Density P-value

We use an application of the weighted root mean squared deviation (wRMSD) approach proposed by Kwon et al (2015). Here, we compute the wRMSD from the center of the oocyte viability class. In the more practical case that viability status is unknown, we would then compute the wRMSD from the center of the oocyte maturation class (center of IVV if using an IVV oocyte, or IVM if using an IVM oocyte). Again, we operate under the assumption that IVV is considered viable. Kwon et al. (2015) said "We considered each gene expression to be an independent event; therefore, we combined all of the expression measurements of each (gene) sample in the calculation of the wRMSD. To minimize the bias from a measurement error of a gene expression profile with a low coefficient of variation (CV), the deviation of each gene expression level from the mean was weighted with the CV of the gene in the group." Within the wRMSD calculation there are three main parts: the reference expression level (mean expression level for the gene within group), the expression level of the specified gene within the oocyte of interest, and the weighting coefficient. The weighting coefficient was further made up of "the proportion of the CV for the expression level of the ith gene to the sum of CV for those of all genes in the group" (Kwon, et al., 2015):

$$wRMSD = \sqrt{\sum_i w_i \cdot (Em_i - E_i)^2}. \tag{1}$$

In Equation 1, wRMSD is defined for a given oocyte. Here, E_i refers to the expression level of the ith gene in the oocyte, Em_i refers to the reference expression level for the oocyte (i.e. mean of the IVV if the oocyte is IVV or IVM if the oocyte is in the IVM group) of the ith gene, and w_i refers to the weight of the mean squared deviation of the gene expression, defined as follows:

$$w_i = \frac{CV_i}{\sum_j CV_j}.$$

That is, the weight (w_i) in Equation 1 is defined as the proportion of the Coefficient of Variation (CV) for the expression level of the ith gene (across all oocytes in the group – IVV or IVM) to the sum of the CV for those of all genes in the group.

We set up the null hypothesis that an observed oocyte (with an accompanying wRMSD) came from the viable (or IVV) class, with the respective alternative hypothesis that it did not. P-values for likelihood of belonging to the "viable" class were then computed for the individual oocytes by comparing the oocyte's observed wRMSD to the kernel density of the viable (or IVV) wRMSD distribution, and computing an upper tail area (see Figure 1). These p-values were then compared to an alpha 0.05 level for determining if there was enough evidence to reject our null hypothesis (that the observed oocyte was viable). This Kernel Density p-value portion was not utilized in the Kwon et al. paper, however it is a reasonable and necessary application of their published method that allows for individual oocyte classification and comparison between results from this and other methods.

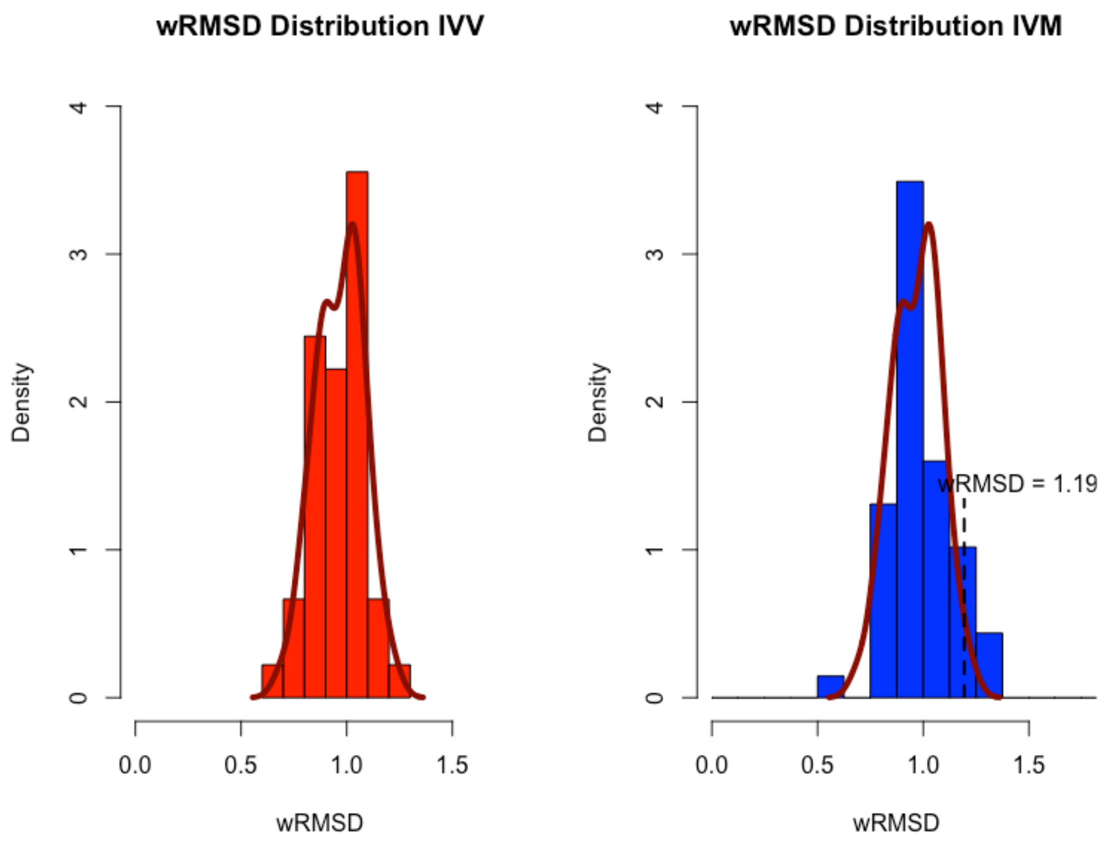

Figure 1. The wRMSD distribution of IVV and IVM oocytes with calculated kernel density for the IVV wRMSD distribution overlaid on both histograms. An example IVM oocyte has been selected and its wRMSD has been found to be 1.19. The wRMSD of the oocyte is then compared to the kernel density of the wRMSD for the IVV group; we can then set up a hypothesis test with a null hypothesis that the observed oocyte comes from the IVV group. By computing an upper tail area from the kernel density, we can observe a p-value for our example oocyte equal to 0.052. This p-value is larger than our 0.05 cutoff, so we would classify the observed oocyte as viable (by our assumption that the more closely related to the IVV group that an oocyte is, the more likely it is to be viable).

3. Distance Kernel Density P-value Method

We considered also a distance measurement method as a modification to the wRMSD approach presented by Kwon et al (2015). It utilizes only a subset of the available genes; those that a limma eBayes approach has determined are differentially expressed (between IVV and IVM groups) at an alpha 0.05 level (Ritchie, 2015). Briefly, the limma eBayes approach performs a modified t-test on the expression level of each gene, testing for differential expression between two conditions (IVV and IVM here). The gene expression profile of all IVV (or viable) oocytes is computed, and the mean expression is taken for each gene in order to form an IVV group mean gene expression profile. Using only the subset of differentially expressed genes, the distance measure for each oocyte from the mean of the IVV group gene expression profile is then computed. Here, distance is measured as an adaption to the Kwon et al. wRMSD approach:

$$Distance = \sqrt{\sum_i w_i \cdot (Em_i - E_i)^2}. \tag{2}$$

In Equation 2, we only use information from differentially expressed genes. W_i is the weight for the specified gene i, and is defined as before, based on CVs. Em_i represents the mean of the IVV (or viable) group for gene i. E_i represents the expression level of the i^{th} differentially expressed gene for the specified oocyte.

The distance kernel density is then calculated for the IVV (or viable) oocytes, based upon the calculated IVV distances. We then can use this kernel density distribution to set up a hypothesis test for each of the individual oocytes, using the following null and alternative hypotheses: H_0 – The oocyte distance comes from the IVV (viable) distribution of distances, i.e. the observed distance for the oocyte is within an expected range if it was of the IVV (viable) class, vs. H_a – The oocyte distance does not come from the IVV (viable) distribution of distances, i.e. the observed distance for the oocyte is outside the expected range if it is of the IVV (viable) class. In order to make a decision for the hypothesis test, we compute the upper tail probability above the distance observed in the oocyte of interest (see Figure 2). This upper tail area is our observed p-value.

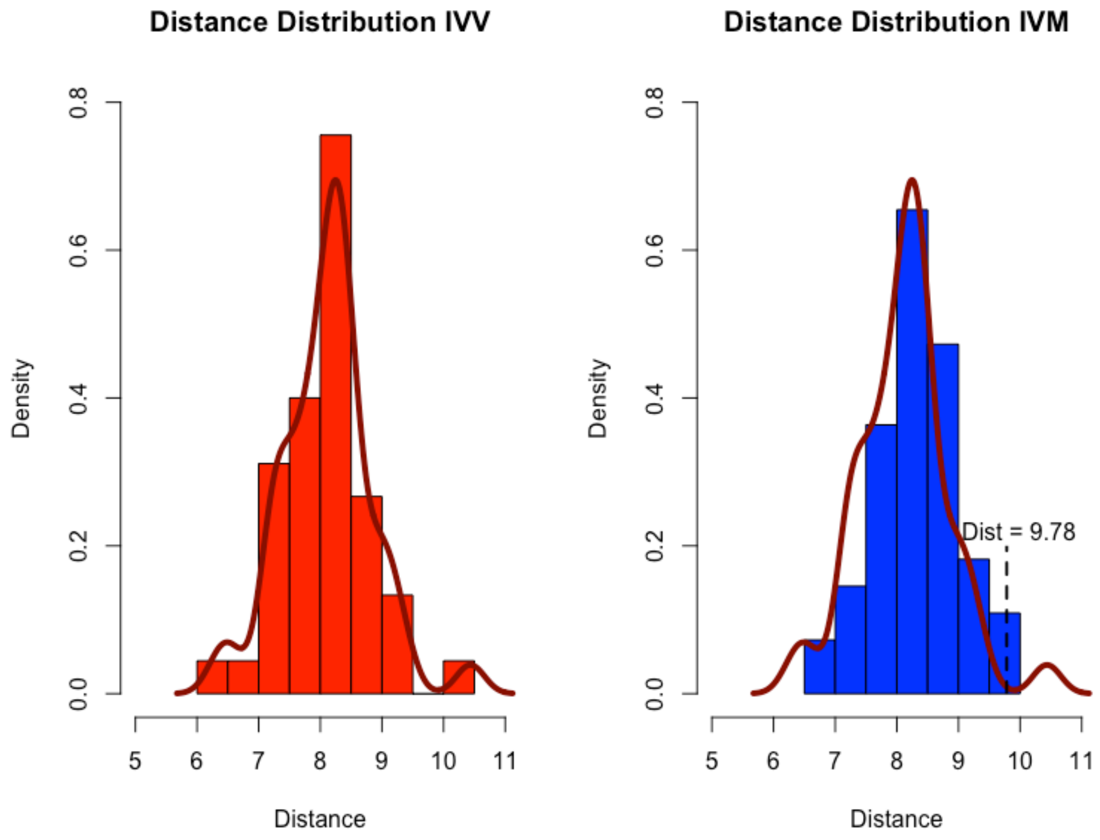

Figure 2. The distance distribution of IVV and IVM oocytes with calculated kernel density for the IVV distance distribution overlaid on both histograms. The same example IVM oocyte has been selected as in Figure 1, and its calculated distance has been found to be 9.78. The distance of the oocyte is then compared to the kernel density of the distances for the IVV group; we can then set up a hypothesis test with a null hypothesis that the observed oocyte comes from the IVV group. By computing an upper tail area from the kernel density, we can observe a p-value for our example oocyte equal to 0.019. This p-value is smaller than our 0.05 cutoff, so we would classify the observed oocyte as non-viable (by our assumption that the more closely related to the IVV group that an oocyte is, the more likely it is to be viable).

4. Tolerance Interval Method

A tolerance interval is a numerical interval that is calculated to provide limits wherein at least a specified proportion of a sampled population falls with an indicated level of confidence. Oftentimes, tolerance intervals are constructed and applied in areas such as quality control or manufacturing to establish that certain product standards are being met by the overall bulk of the products. "More specifically, a $100 \times p\%/100 \times (1-\alpha)$ tolerance interval provides limits within which at least a certain proportion (p) of the population falls with a given level of confidence $(1-\alpha)$" (Young, 2010). Tolerance intervals are based on the sampled data; however they allow us to say something about the population distribution. "A tolerance interval differs from a confidence interval in that the former encloses a proportion of the entire population distribution, while the latter is constructed to contain the value of a population parameter" (Millsap, 1988). Frequently, tolerance intervals are based on a specified distribution of the data, and more often than not, that distribution is assumed to be normal. However, we can also make the tolerance interval more general by taking a non-parametric approach (Wilks, 1941). In Wilks' paper, he proves that there is a systematic way of calculating a confidence interval for an unknown data distribution; a tolerance interval can be calculated for a given population coverage proportion, level of confidence and minimum sample size, that guarantees at least the given population coverage proportion. This approach is outlined as follows:

- Let a be the average value which p is to have, where p is the proportion of the population to be included in the interval (the mean coverage).
- Draw a sample of size n from the population subject to the constraint that $[(1-a)(n+1)]/2 = r$, a positive integer.

- Order the sampled data according to increasing magnitude from x_1 to x_n
- Let $L_1 = x_{n-r+1}$, the upper tolerance limit of our 1-sided non-parametric tolerance interval

As noted above, we need a minimum sample size value to ensure proper coverage of the population with the specified level of confidence. Within the framework of this project, we utilized the principles of this method to create a 95% non-parametric one-sided tolerance interval for 95% coverage of the IVV distance distribution. The distribution of distances for the Isom data IVV oocytes is right skewed (see Figure 3), so we used an application of the non-parametric tolerance interval method.

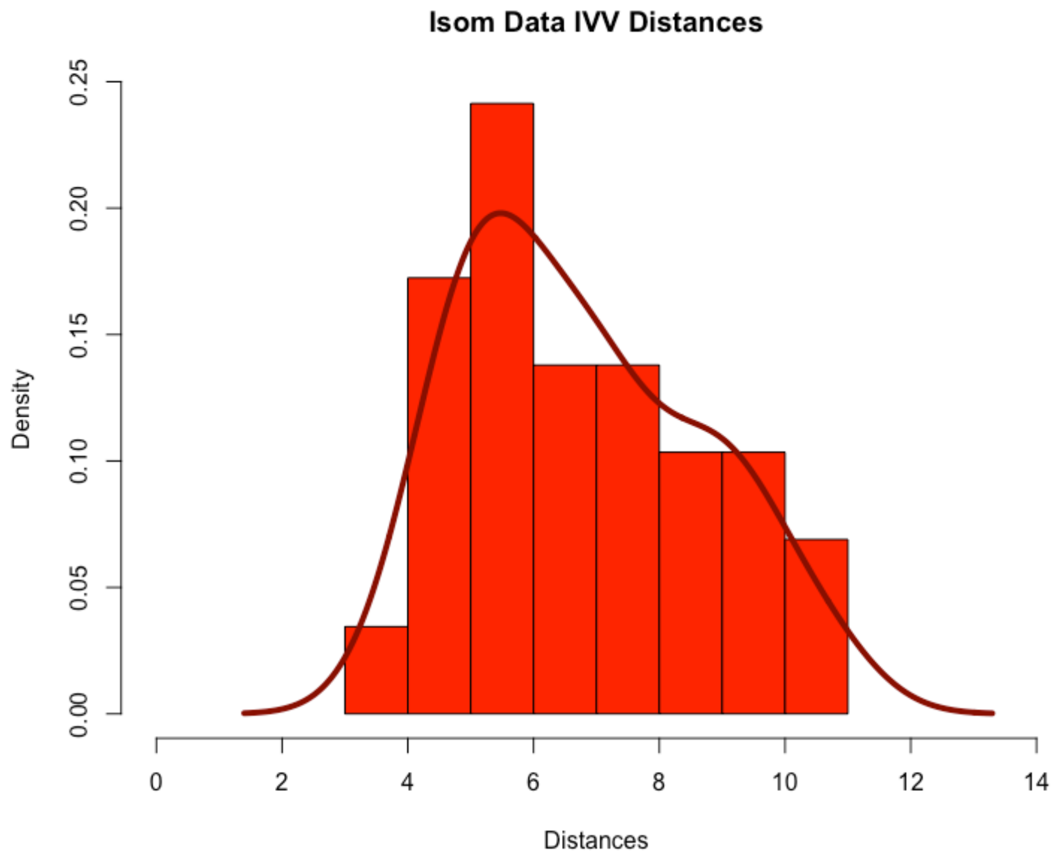

Figure 3. The distribution of IVV distances for the Isom data. The kernel density was overlaid to help depict the skewedness of the distribution.

For a non-parametric approach, the calculation of the tolerance interval can be different than when we specify a distribution for the data. As we do not restrict our interval to a specific distribution, we require a larger sample size in order to maintain the same level of confidence and coverage for our tolerance interval than that of a specified distribution (i.e. Gaussian, Weibull, etc). The calculations for the minimum required sample size of a non-parametric tolerance interval are shown below (NIST, 2012):

$$n \approx \frac{1}{4}\frac{(1+p)}{(1-p)}\chi^2_{1-\gamma,4} + \frac{1}{2}.$$

The above equation is an approximation for the minimum sample size n, needed for a non-parametric tolerance interval with confidence level γ , and population coverage proportion p. We also note that we are calling a specified value from a chi-squared distribution with 4 degrees of freedom.

For our project, the minimum required sample size for a 95% confidence and at least 95% coverage was n=94, so we ended up needing a sample size of 100. In the simulations of Section 6 below, we use n=100 oocytes, and we proceed with the construction of the tolerance interval here solely for demonstration purposes (though the actual sample size in the project dataset is n=58, so the actual coverage in this demonstration example is likely less than 95%). For each oocyte in the project, the IVV distance distribution was computed using the same distance function as previously discussed. That distance function again only looks at genes that are differentially expressed in the Isom dataset, as indicated by a limma eBayes approach, using a significance level of 0.05 as a cutoff for the FDR-adjusted p-values. Those oocytes that individually have a distance statistic outside of the constructed tolerance interval for the IVV oocytes (i.e. their distance was greater than that of the upper tolerance bound) were selected as being nonviable (see Figure 4).

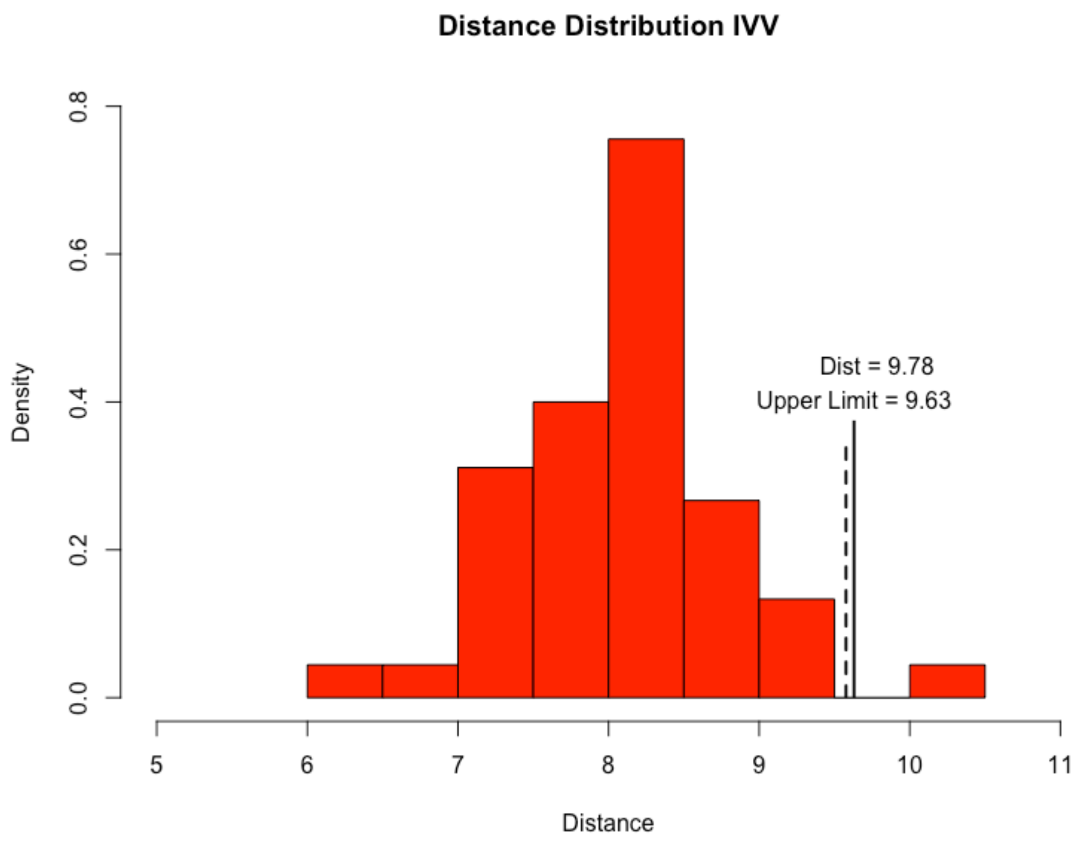

Figure 4. The distance distribution of IVV. The upper limit of the 95% confidence, 95% coverage non-parametric tolerance interval for the IVV group was calculated to be 9.63. The same example IVM oocyte as in Figures 1 and 2 has been selected and its distance was calculated to be 9.78. The distance of the oocyte is then compared to the upper limit of the tolerance interval for the IVV group. As the observed distance of 9.78 is larger than the upper tolerance limit of 9.63, we would classify this oocyte as non-viable, as it was classified as being less closely related to the IVV group (by our assumption that the more closely related to the IVV group that an oocyte is, the more likely it is to be viable).

5. Classification Tree Method

Classification trees or decision trees are a graphical representation of a set of rules used to classify data into categories. They are appropriate to use when one has a 2 or more level categorical variable as an outcome, with one or more variables as predictors. In general, we would use a classification tree to predict the class or outcome level of a number of observations within a dataset, based on the observed values of the predictor variables. In R, the default index for choosing the best split in the data (for classification) is the Gini index. The Gini index is a measure of impurity of a node (or a whole tree). The Gini impurity measure can be calculated in the following way (Kingsford, 2008):

$$Gini = 1 - \sum_{i=1}^{m} p_i^2.$$

Within the Gini calculation, we are trying to classify items into m classes using a set of training items E. Let p_i (i= 1,…,m) be the fraction of the items of E that belong to class i. Thus, the Gini index reaches an optimal value of zero when the set E contains items from only one class. Construction of a decision tree using this index can be broken down into a number of simple steps:

1. The tree algorithm uses a training data set to build the tree, in other words, one needs to know the true outcomes in order to build a classification tree that can be used on other datasets drawn from the same population.
2. The algorithm then uses the predictor variables to make the most optimal splits in the data. An optimal split is defined as using a variable to split the data from one group into two that provides the greatest differentiation between the different classes, separating them from one another as well as possible.

3. The algorithm uses the most important variables to make splits in the data. If the observations in a specified branch of the tree contain a diverse group of classes, then the algorithm finds “the best” rule based on a single variable/feature to split that branch into two smaller branches. The quality of the split is again measured based upon a reduction in the Gini impurity measure.
4. Only if every observation in the “branch” of the tree is from the same class, does the tree form a terminal node or leaf.
5. One can invoke the *minsplit* argument in the *rpart* function of the package *rpart* (Therneau, 2015) for R (R Core Team, 2016) to indicate the minimum number of observations needed to make a split in the data (to mitigate overfitting and nodes with single observations).

This process of actually using a tree to classify objects was aptly summarized in the following way: “In order to classify an object, we start at the root of the tree, evaluate the test, and take the branch appropriate to the outcome. The process continues until a leaf is encountered, at which time the object is asserted to belong to the class named by the leaf” (Quinlan, 1986).

Within the framework of this project, the outcome or class variable is oocyte viability. A key advantage to a decision tree is that it provides a visual rule for distinguishing between viable and non-viable classifications, based again on the most differentially expressed genes (as was the case with our previous methods using the limma eBayes method). Again, the genes are the variables on which the tree is split, and while all genes have a chance of being chosen initially, the tree will chose the genes to split on that give the best split as calculated by the Gini Index (see Figure 5).

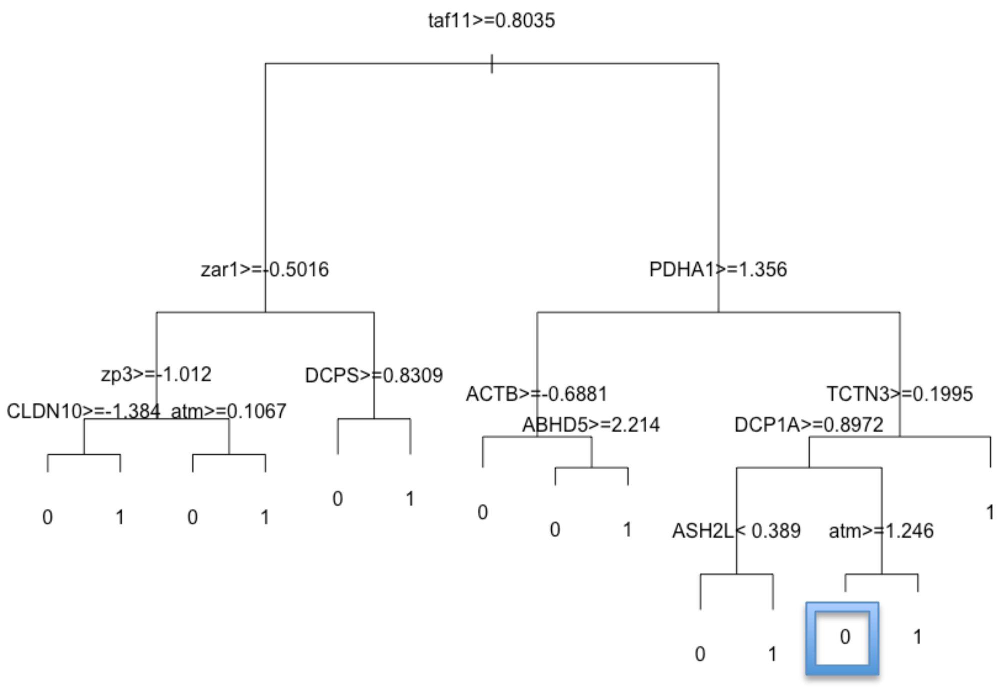

Figure 5. A classification tree that was grown for the dataset; the splits are conducted based upon the gene expression values of the oocytes in the dataset. The resulting classification values of 0 and 1 refer to non-viable and viable respectably. Using the same example oocyte as in Figures 1, 2 and 4, we arrive at a classification of non-viable (shown in box above).

6. SIMULATIONS

In order to adequately compare the four previously summarized methods from Sections 2 through 5, simulations were conducted, in which datasets were generated and results were gathered for the four methods. As the goal was to determine how the methods compared to one another, the simulation was constructed as a function in R (see Appendix II) with the following five variables that could be altered to yield different situations for the simulated data:

- n, the number of oocytes to be simulated
- Pi, the mixing proportion for the mixture distribution of the IVM oocytes (i.e. the degree of similarity between the viable IVM group and the IVV group)
- δ, the magnitude of differential expression for genes that are differentially expressed between IVM and IVV groups
- WhichGenes, a given list of genes to be differentially expressed in the simulation
- ProbViableIVM, the prior probability that any given IVM oocyte is viable

At the start of a simulation, the function starts with the first of the n oocytes (for our project, n=100), and decides whether it will be generated as an IVV or IVM oocyte. This process is done randomly using a binomial generator with probability of IVV equal to 0.5. After maturation type is decided, we determine if the oocyte will be simulated as a viable oocyte or not. For the sake of the simulation, all IVV oocytes were given the viable class, while the probability that an IVM oocyte is deemed viable depends on the simulation variable "ProbViableIVM" (binomial distribution with probability of viable being equal to "ProbViableIVM"). After each of the oocytes has been assigned a maturation and viability type, the individual gene expression profiles for each oocyte are generated.

Within the process of gene expression profile generation, the simulation function takes into account specific genes that the user wants to make sure are differentially expressed between viable and non-viable classes. The list of these gene names is to be supplied to the function by the user in the simulation argument "WhichGenes" (for our project, these were derived from the Isom data). For the genes identified as being differentially expressed, we then determine the degree of differential expression. Each named gene has its individual degree of differential expression, δ_i, which is generated randomly from a uniform distribution, with minimum equal to zero, and maximum equal to the absolute value of the user supplied variable δ (in our case, this is calculated as the median of the log fold-change for the differentially expressed genes from the Isom data). If the gene of interest is not included in the list of genes to be differentially expressed, δ_i is simply set to zero. Next, for each gene within the oocyte, we generate an expression value in the following manner:

$$Degree\,of\;Differential\;Expression\;for\;Gene\,i\;=\delta_i\sim U(0,\delta),$$

$$IVV\;Expression\;for\;Gene\,i=X\sim N(0,1),$$

$$Viable\,IVM\;Expression\;for\;Gene\,i\;=\beta*X+(1-\beta)*Y,$$

$$NonViable\;IVM\;Expression\;for\;Gene\,i=Y\sim N(\delta_i,1),$$

$$\beta\;\sim\;Binom(1,Pi).$$

The expression value for gene i of the oocyte is simply a random value from a standard normal distribution if the oocyte is an IVV oocyte. If the oocyte is a non-viable IVM oocyte, then the expression value for gene i is a random value from a normal distribution with mean equal to δ_i and standard deviation equal to one. If the oocyte is a viable IVM oocyte, then the expression level for gene i is a mixture of the two aforementioned distributions, with the mixing proportion, Pi, determining the degree of the mixture (i.e. the probability it will be more similar to the IVV group, see Figure 6).

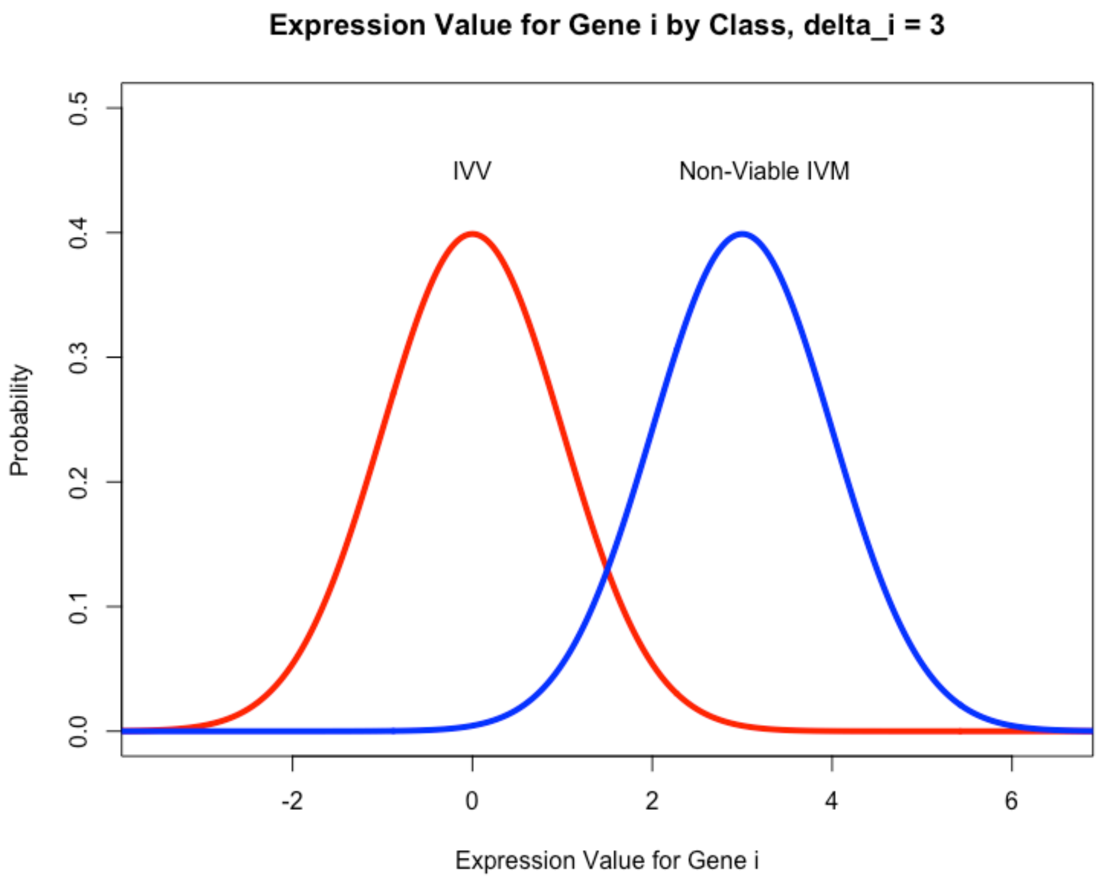

Figure 6. Example expression value distributions for IVV and Non-Viable IVM oocytes for gene i, such that δ_i is equal to three (for example purposes). The distribution of the Viable IVM oocyte expression value is represented as a mixture of the two above distributions. The proportion of the mixture is determined by the simulation argument Pi. In other words, the probability that an expression value for gene i in a Viable IVM oocyte will come from the IVV distribution is equal to Pi.

For the purpose of this project, we have chosen only to vary the variables Pi and ProbViableIVM using the values 0.1, 0.3, 0.5, 0.7 and 0.9; all other variables remained constant throughout the simulations. Constant values and the list of differentially expressed genes came from the provided Isom data. We simply wished to see the effects of our mixing proportion as well as the probability of viable IVM oocytes, and how they affected our methods. Note, as we increase the mixing proportion, we are simply increasing the similarity between viable IVV and viable IVM oocytes within the simulation. When we increase the probability of viable IVMs, what we are doing is increasing the probability that a viable IVM oocyte is generated within our simulation set, as our oocyte number stayed constant across simulations. We note that this creates an imbalance in the group sizes, which we can then use to test how our methods handle such conditions. The support and resources from the Center for High Performance Computing at the University of Utah are gratefully acknowledged in providing the means of running the simulations (for a helpful tutorial, see Barton 2016). As the University of Utah system is set up for cluster computing, we were able to run 500 iterations at each simulation level in under 14 hours whereas it would have taken multiple days to compute the same amount on a traditional PC.

In addition to gene expression data generation, the functions created for the simulations also compile results from our primary measures of interest for model evaluation, as averaged over 500 simulations: proportion correctly classified (PCC), sensitivity and specificity for each of the models. In this context, we define proportion correctly classified as the count of those oocytes that were classified as either being viable or non-viable when it was truly their respective viability status, divided by the total number of oocytes that were classified within the simulation (in this case 100). This was repeated 500 times and then averaged across each simulation level combination. A similar process was done for sensitivity and specificity. Sensitivity was defined as the count of correctly classified viable oocytes divided by the total number of viable oocytes, and specificity was defined as the count of correctly classified non-viable oocytes divided by the total number of non-viable oocytes.

7. RESULTS

On examining Figures 7 and 8 (as well as the full results summarized in Appendix I), we notice that in general, the tolerance method approach tends to outclass the other methods when looking at the proportion correctly classified (PCC). A similar result is seen in Figures 9 and 10 when sensitivity is used as our measured outcome. In both outcomes, the tolerance interval approach scores roughly two to three percentage points higher on average (after averaging across 500 simulations at each mixing proportion by probability viable IVM combination) than the relatable wRMSD and distance kernel methods. In the aforementioned areas, the disparity between tolerance intervals and that of a decision tree are even more pronounced, as the decision tree looks to be simply lack-luster.

Also, in reviewing Figure 7 and Figure 8, we note that it appears that the variable Pi (mixing proportion) seems to have very little effect on the method outcomes. We do see some slight changes in the slopes of the lines within the classification tree method in Figure 8 and Figure 9; however, nothing of major consequence that could accurately be attributed to the variable Pi.

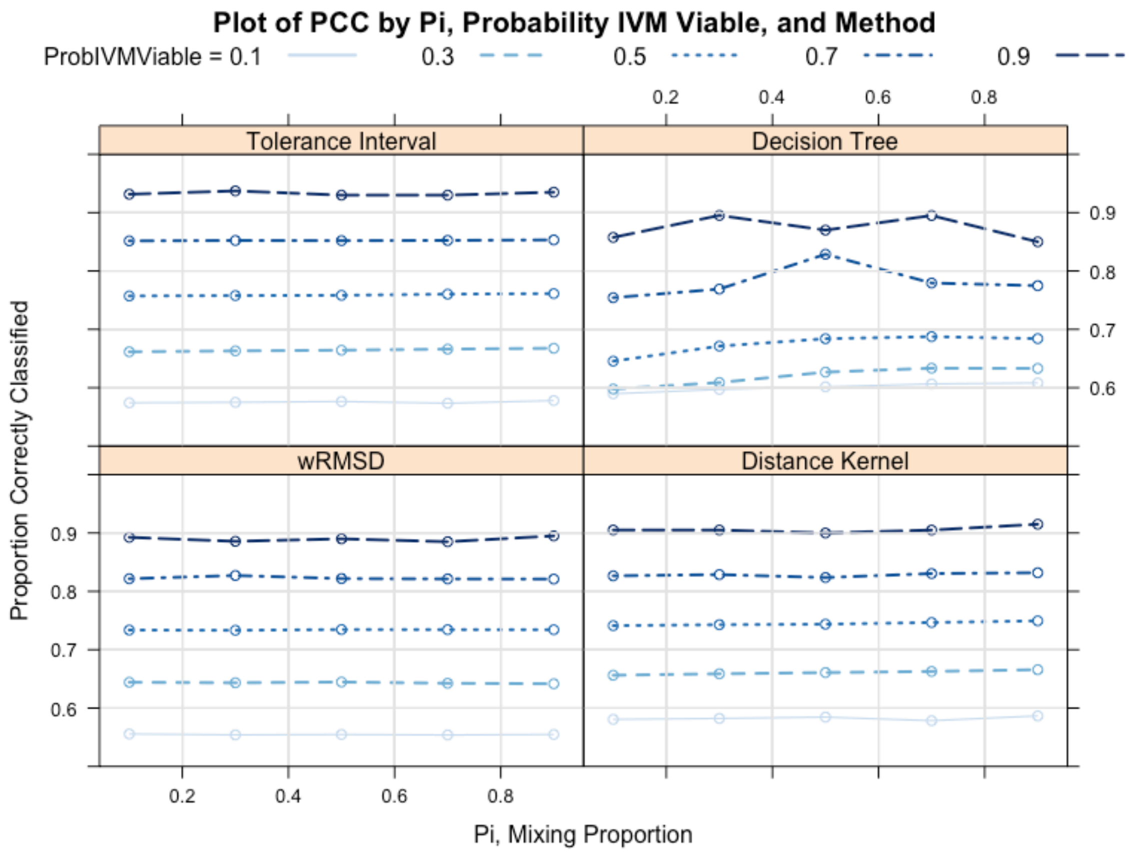

Figure 7. *The Proportion Correctly Classified (PCC) of the different simulation combinations, as it specifically relates to the Mixing Proportion (Pi). The different line types, as well as the increasing depth of the blue color signify probability of a given IVM being viable across the four different methods.*

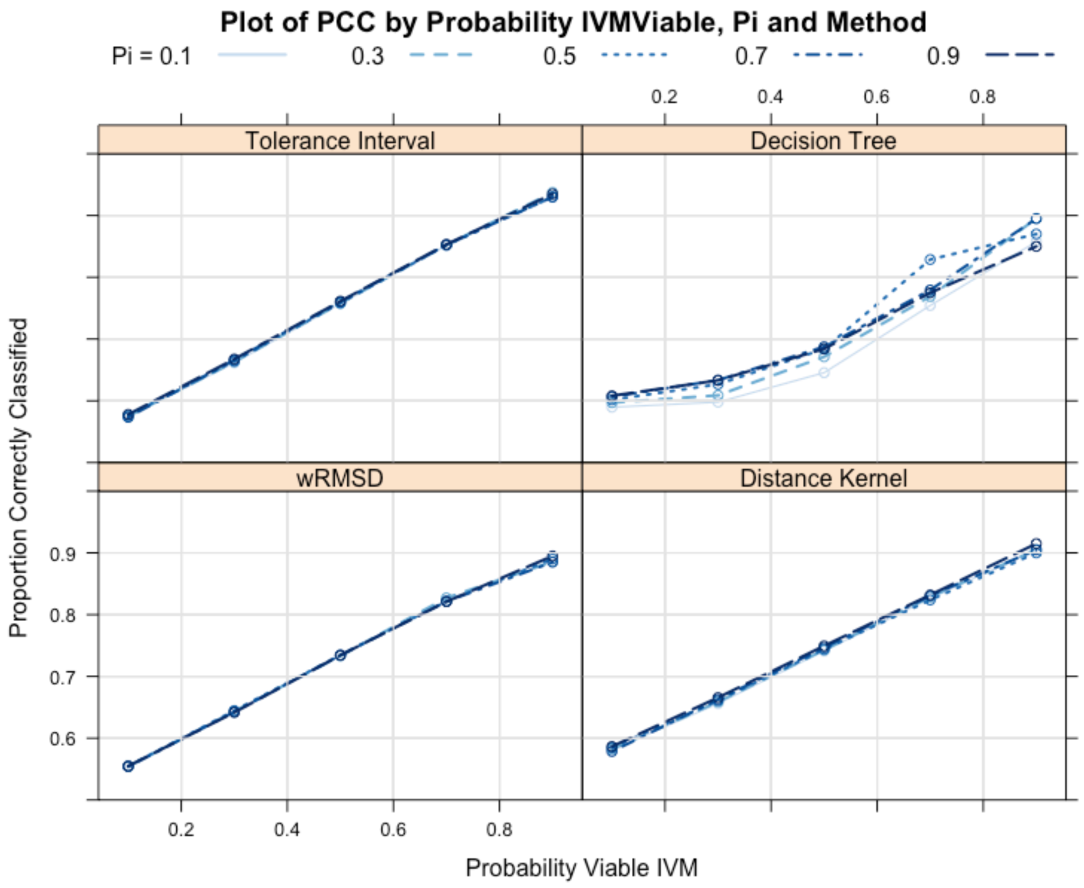

Figure 8. The Proportion Correctly Classified (PCC) of the different simulation combinations, as it specifically relates to the Probability of a given IVM being viable. The different line types, as well as the increasing depth of the blue color signify the mixing proportion Pi across the four different methods.

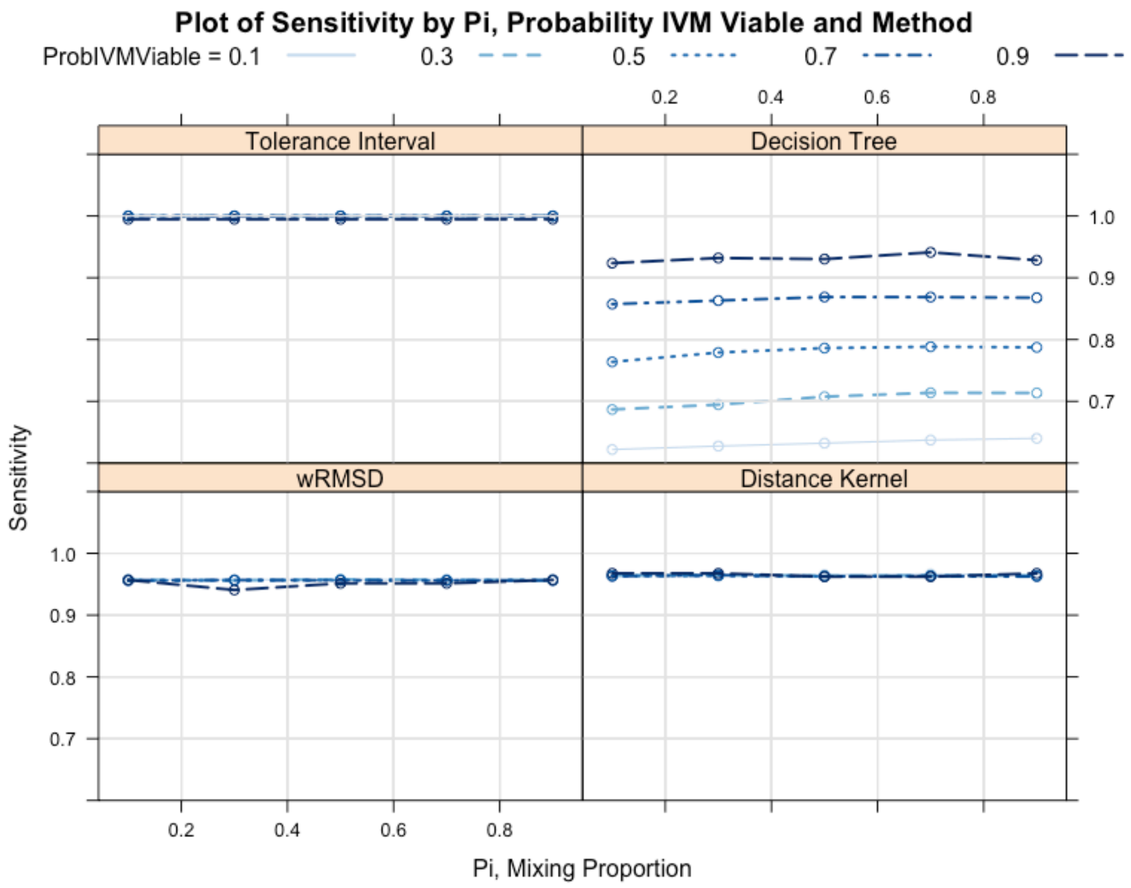

Figure 9. The Sensitivity, or ability to correctly classify the viable oocytes of the different simulation combinations, as it specifically relates to the mixing proportion Pi. The different line types, as well as the increasing depth of the blue color signify the probability of a given IVM oocyte being viable across the four different methods.

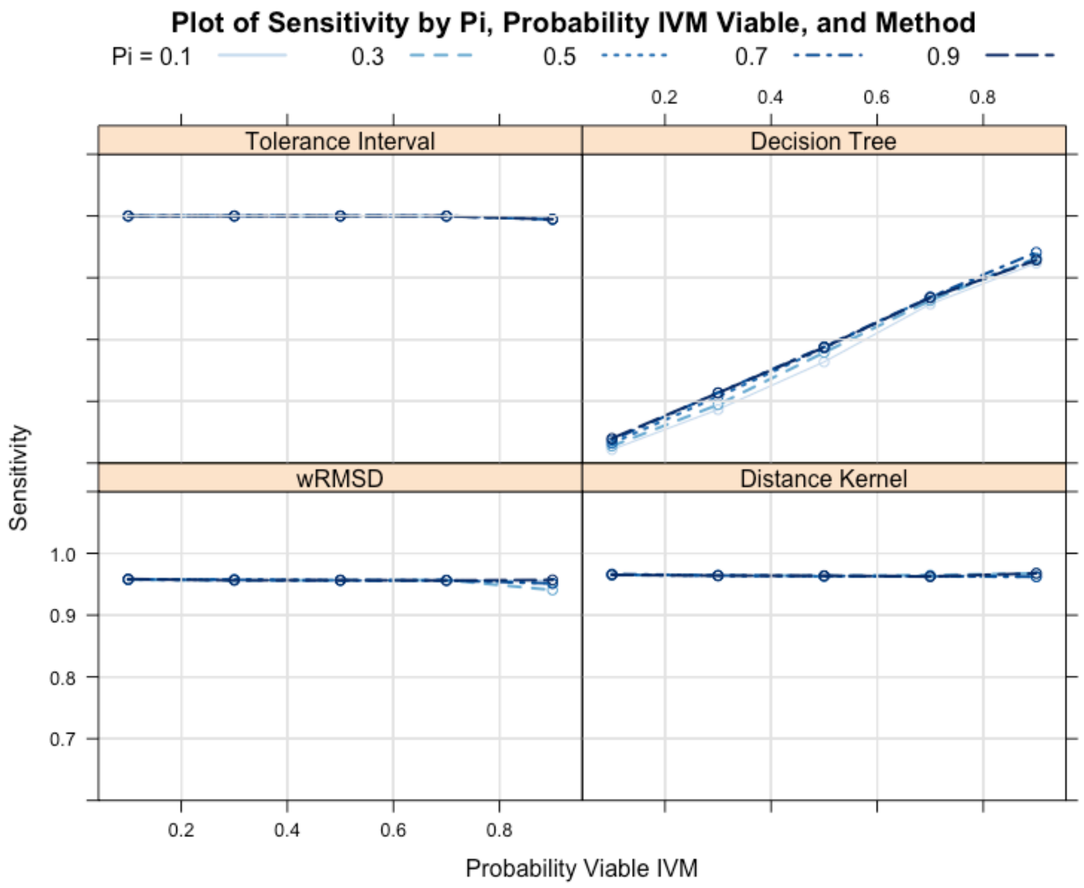

Figure 10. The Sensitivity, or ability to correctly classify the viable oocytes of the different simulation combinations, as it specifically relates to the Probability of a given IVM being viable. The different line types, as well as the increasing depth of the blue color signify the mixing proportion Pi across the four different methods.

Something to note is that, while it appears that the classification tree is greatly outclassed by these other three methods (specifically tolerance intervals), it does have situations where it outperforms all others, and in some ways, these situations may actually be quite valuable. In looking at the specificity of the methods (i.e. the ability of the methods to correctly classify non-viable oocytes) in Figure 11, classification trees actually outperform the other methods. This is an important feature, as part of our main goal was to establish a method that could identify both viable and non-viable oocytes.

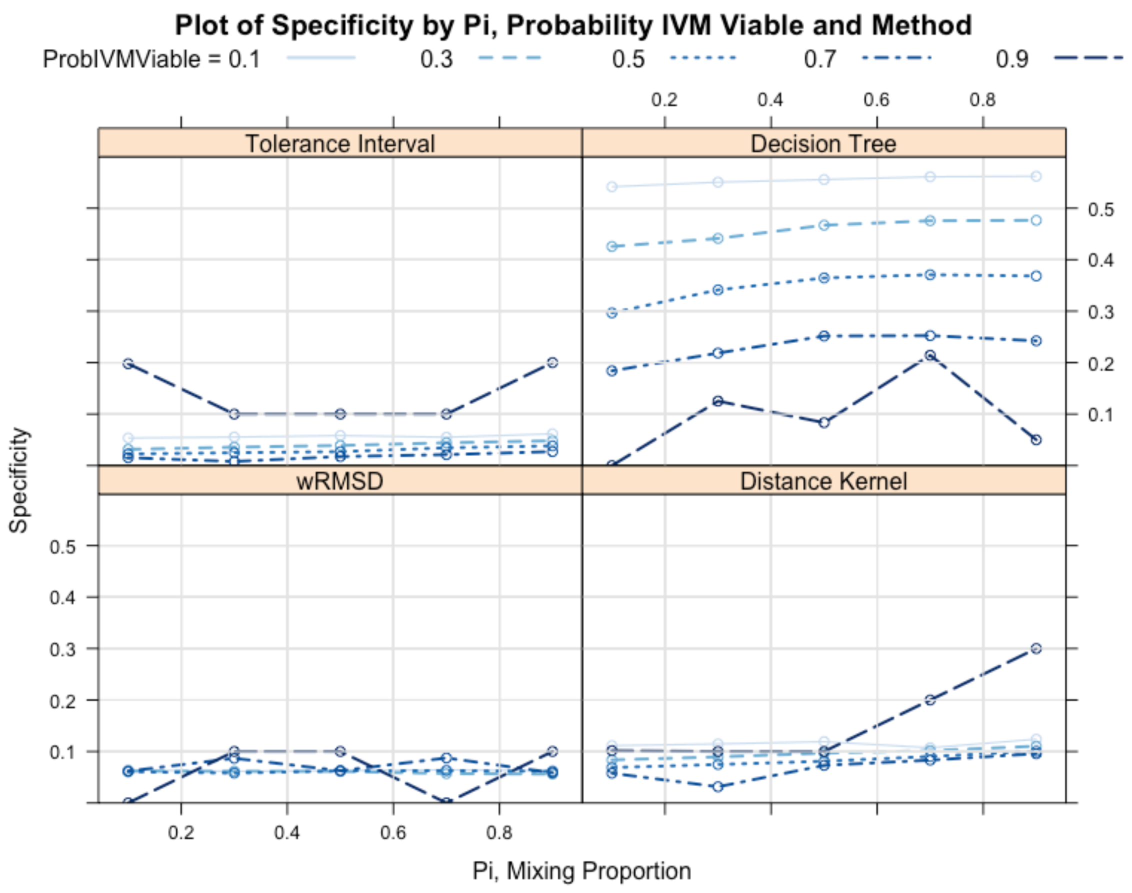

Figure 11. The Specificity, or ability to correctly classify the non-viable oocytes of the different simulation combinations, as it specifically relates to the mixing proportion Pi. The different line types, as well as the increasing depth of the blue color signify the probability of a given IVM oocyte being viable across the four different methods.

In Figure 11, it appears that as the mixing proportion (Pi) increases in the decision tree method, the harder time it has correctly classifying the non-viable samples. This would make sense, as we bring the two distributions closer together (see Figure 6), such that the IVV oocytes are appearing more similar to the IVM oocytes, and so the harder it would be to accurately separate them.

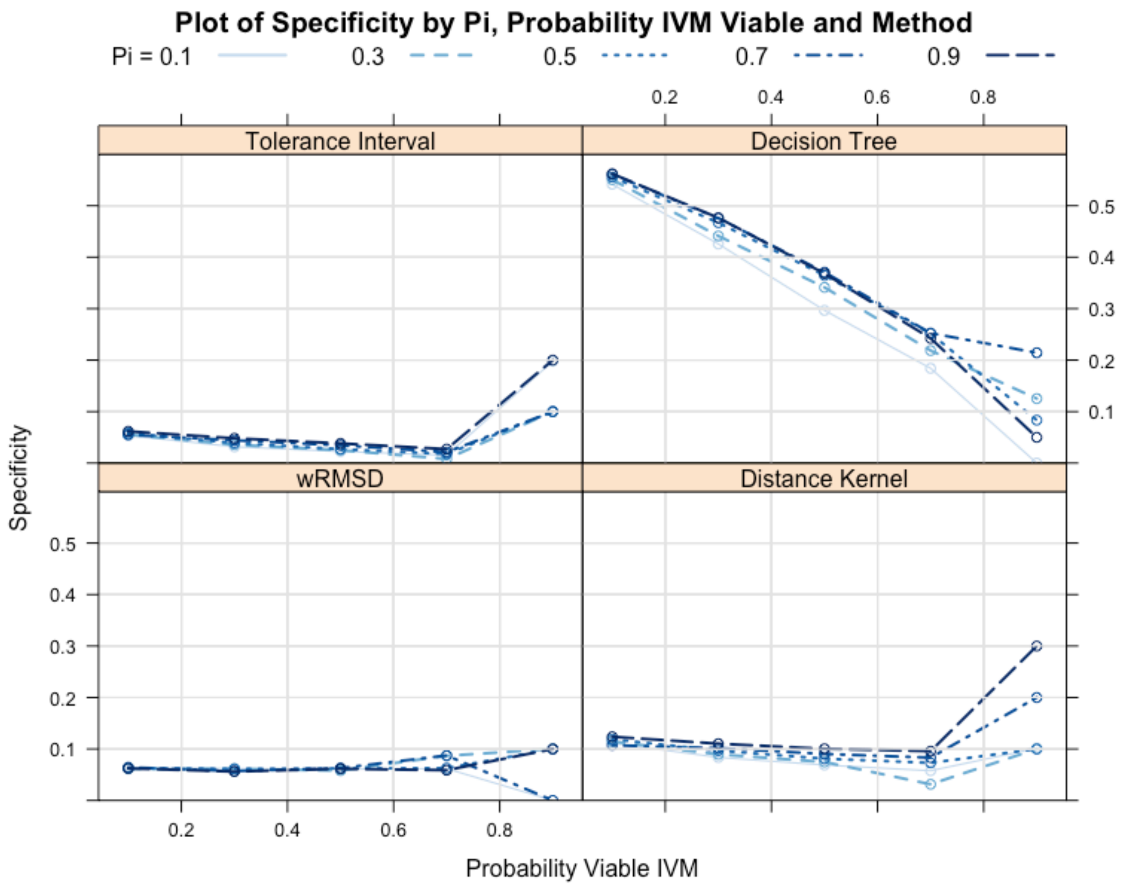

Figure 12. *The Specificity, or ability to correctly classify the non-viable oocytes of the different simulation combinations, as it specifically relates to the Probability of a given IVM being viable. The different line types, as well as the increasing depth of the blue color signify the mixing proportion Pi across the four different methods.*

In Figure 12, we observe yet another interesting phenomenon. As the probability increases that any given IVM oocyte will be viable, it makes it increasingly difficult to correctly classify the non-viable samples, as it is simply more likely that any given oocyte will be viable than non-viable at that point. As the probability of any given IVM oocyte being viable increases, it creates more of dispersion between the group sample sizes, and one would expect a drop in the care that is given to oocytes of the lesser class. However, as we see in Figure 12, it appears as though the distance kernel and the tolerance interval methods actually get slightly better at classifying the non-viable IVM oocytes for the specificity measure, as the probability of viable IVM increases to its highest value.

8. Discussion

Currently in the fields of cloning and embryology, researchers constantly run into complications and unplanned procedural failures due to the effects of low viability amongst oocytes derived in a laboratory setting. Non-viability of samples is a costly and unwanted outcome that is worth evaluating and attempting to mitigate. Past focus has been in evaluating origination methods and determining their 'closeness' to one another; specifically, the method's proximity to IVV derived oocytes, the gold standard. While this may be a decent approach at evaluating origination methods, we claim that we can use alternate approaches to evaluate, and thereby later improve the quality of individual oocytes within a given method. With recent advances in gene expression technology, nutrient washes can be created in order to encourage the oocytes to express specific genes within a given origination method, thereby improving the viability and quality of the samples, while still retaining the flexibility of using the origination method of choice.

While the goal was to identify methods that were best to use overall, we notice that we actually have a few different situations represented in the data, and as a result we get different "top methods" depending on the situation at hand. As noted above in Figures 7 and 9, when looking at PCC and sensitivity, tolerance intervals seemed to always be the method that performed the best, however the kernel density p-value approach wasn't far behind, with our modification of the published Kwon wRMSD method following shortly behind that. However, one reason that these methods could be performing so well in these result measurements (PCC and sensitivity), is that there is an imbalance of sample sizes. The viable oocyte group is consistently larger than the non-viable group, simply due to the nature of the probabilities, as all IVV oocytes are simulated as being viable, while an increasing number of IVM oocytes are counted as viable as the variable ProbIVMViable starts to increase. So, as a reiteration, as the variable ProbIVMViable increases, the disparity between the viable and non-viable group sizes starts to increase (note, when ProbIVMViable is set to 0.90, nearly all oocytes are in the viable group). It appears as though the classification tree method is more balanced and more immune to the effects of this imbalance in the data, as it appears to perform roughly equally across all groups, though it does still get swayed some as ProbIVMViable reaches higher values (see Figures 8, 10, and 12). So, if one is interested in a method that is more robust to the effects of imbalanced samples, or has a vested interest in the accuracy of identifying the non-viable samples (where they severely outperformed the other methods, see Figures 11 and 12), classification trees may be the method to choose. If not, we would recommend the non-parametric tolerance interval approach based upon our calculated distance measure.

At this point, we acknowledge the possible limitations of these simulations. First, we note that we have made a number of assumptions that we cannot fully prove with the resources at hand. First, we assume that oocytes that have gene expression profiles that are more like those of IVV oocytes are going to be more viable. Second, we assume the center of the observed IVV distribution to be the optimal gene expression profile. Third, we assume that the data provided to us from Dr. Isom is a representative sample of all IVV and IVM oocytes (viable and non-viable). Fourth, assumptions were made about the relative rates of viability in the different origination method groups based upon conversations with Dr. Isom, as he was our expert in the field. While these are a fair amount of assumptions, we did take precautions as to try to minimize their effects. The methods used were built or conceived in a way that they operate independent of oocyte origination method, they require no distributional assumptions of the oocyte data, and as demonstrated, do not require a complete balance of group sizes (though it may have yielded slightly different results had we forced this on the generated data).

While overall the methods seemed to perform well in specific areas, there are some procedural aspects of these simulations that can be better handled. One such example is the techniques for handling missing values in the data. With oocyte data, when obtaining the gene expression profiles of the oocyte, we destroy the oocyte, thereby making it possible only to attempt to retrieve the profile once; if we miss a few values, we cannot go back and "re-observe" them. Hence, having a set method for handling missing values is advised. One recommendation could be to impute or borrow information from other similar oocytes in the same origination group. For the purposes of these simulations, if there was a missing value in the original Isom data, that value was simply omitted from the gene level calculations. Despite not being able to go back and "re-observe" the missing values, they were fairly infrequent (about 4.4% of the specific gene-level data) in the original dataset.

Further method application and development is necessary in this field, as we showed how under the simulated circumstances that our three proposed methods consistently outperformed the previously published wRMSD method; however, our proposed methods were quite rudimentary or basic. More advanced classifiers such as Random Forest, neural networks, and the like, may end up doing a better job on these types of classification problems. But, also bear in mind, if the dataset has a large number of variables (such as genes here), then these other methods, while more complex and possibly better classifiers, may require a large increase in processing power and computing time.

References

1. Barton, S.W. (2016). "Tutorial for Using the Center for High Performance Computing at The University of Utah and an example using Random Forest". Unpublished M.S. Report, Utah State University. http://digitalcommons.usu.edu/gradreports/873

2. Kingsford, C., & Salzberg, S. L. (2008). “What are decision trees?” Nature Biotechnology, 26(9), 1011–1013. http://doi.org/10.1038/nbt0908-1011.

3. Kwon, S., et al. (2015). “Assessment of Difference in Gene Expression Profile Between Embryos of Different Derivations”, Cellular Reprogramming, VOL. 17, NO. 1. DOI: 10.1089/cel.2014.0057.

4. Millsap, R. E. (1988). “Tolerance Intervals: Alternatives to Credibility Intervals in Validity Generalization Research”, Applied Psychological Measurement, March 1988, VOL. 12, NO. 1, PP.27-32.

5. NIST/SEMATECH (2012). “e-Handbook of Statistical Methods”, http://www.itl.nist.gov/div898/handbook/prc/section2/prc265.htm, 3-3-17.

6. Quinlan, J. R. (1986). "Induction of decision trees." Machine learning 1.1: 81-106. doi:10.1023/A:1022643204877. http://link.springer.com/article/10.1023/A:1022643204877.

7. R Core Team (2016). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/.

8. Ritchie, M. E., Phipson, B., Wu, D., Hu, Y., Law, C.W., Shi, W., and Smyth, G.K. (2015). “limma powers differential expression analyses for RNA-sequencing and microarray studies”, Nucleic Acids Research 43(7), e47.

9. Therneau, T., Atkinson B., and Ripley B. (2015). rpart: Recursive Partitioning and Regression Trees. R package version 4.1-10. https://CRAN.R-project.org/package=rpart

10. Thomas, F. H., Vanderhyden, B. C. (2006). “Oocyte-granulosa cell interactions during mouse follicular development: regulation of kit ligand expression and its role in oocyte growth”, Reproductive Biology and Endocrinology 2006 4:19. doi: 10.1186/1477-7827-4-19. https://rbej.biomedcentral.com/articles/10.1186/1477-7827-4-19.

11. Wilks, S. S. (1941). “Determination of Sample Sizes for Setting Tolerance Limits”. Ann. Math. Statist. 12, no. 1, 91--96. doi:10.1214/aoms/1177731788. http://projecteuclid.org/euclid.aoms/1177731788.

12. Young, D. S. (2010). Book Reviews: "Statistical Tolerance Regions: Theory, Applications, and Computation", Technometrics, February 2010, VOL. 52, NO. 1, pp.143-144.

Appendix I: Full Results

Full Simulation Results: 500 iterations for each result number; the mean of the outcomes was recorded; see visualization in Figures 7-12.

Result Number	Pi	Probability Viable IVM	PCC	Sensitivity	Specificity	Method
1	0.1	0.1	0.55562	0.957686	0.06418214	wRMSD
2	0.1	0.3	0.6442	0.9572701	0.06176947	wRMSD
3	0.1	0.5	0.73368	0.9559364	0.06065979	wRMSD
4	0.1	0.7	0.8216	0.9562596	0.06179166	wRMSD
5	0.1	0.9	0.89254	0.9569693	0	wRMSD
6	0.3	0.1	0.55408	0.9575066	0.06136217	wRMSD
7	0.3	0.3	0.64334	0.9563099	0.06149518	wRMSD
8	0.3	0.5	0.73326	0.9565633	0.05799724	wRMSD
9	0.3	0.7	0.82716	0.9567615	0.08672111	wRMSD
10	0.3	0.9	0.88568	0.9407172	0.1	wRMSD
11	0.5	0.1	0.5547	0.9576374	0.06277166	wRMSD
12	0.5	0.3	0.64462	0.9579131	0.06129233	wRMSD
13	0.5	0.5	0.73454	0.9565436	0.06231212	wRMSD
14	0.5	0.7	0.82184	0.9564298	0.06268898	wRMSD
15	0.5	0.9	0.89	0.9514748	0.1	wRMSD
16	0.7	0.1	0.5539	0.9577265	0.06433873	wRMSD
17	0.7	0.3	0.64252	0.957133	0.05701877	wRMSD
18	0.7	0.5	0.73436	0.9560838	0.0628393	wRMSD
19	0.7	0.7	0.8213	0.9561947	0.08737186	wRMSD
20	0.7	0.9	0.885	0.9514748	0	wRMSD
21	0.9	0.1	0.5548	0.9583906	0.06224052	wRMSD
22	0.9	0.3	0.64164	0.956466	0.05610936	wRMSD
23	0.9	0.5	0.73422	0.9567238	0.0614104	wRMSD
24	0.9	0.7	0.82104	0.9561652	0.05874454	wRMSD
25	0.9	0.9	0.895	0.9569693	0.1	wRMSD
26	0.1	0.1	0.58062	0.9648194	0.1113315	Distance Kernel
27	0.1	0.3	0.65626	0.9640163	0.08360072	Distance Kernel
28	0.1	0.5	0.74122	0.9634611	0.06880955	Distance Kernel
29	0.1	0.7	0.8266	0.9629717	0.05768953	Distance Kernel
30	0.1	0.9	0.90508	0.9677274	0.10152754	Distance Kernel
31	0.3	0.1	0.5822	0.9649558	0.1146015	Distance Kernel
32	0.3	0.3	0.65872	0.964505	0.08951383	Distance Kernel
33	0.3	0.5	0.74284	0.9637019	0.07468482	Distance Kernel
34	0.3	0.7	0.8287	0.964736	0.03139803	Distance Kernel
35	0.3	0.9	0.90504	0.967727	0.1	Distance Kernel
36	0.5	0.1	0.5845	0.9654879	0.1189987	Distance Kernel

37	0.5	0.3	0.66078	0.9638159	0.09678404	Distance Kernel
38	0.5	0.5	0.74372	0.9627725	0.08110845	Distance Kernel
39	0.5	0.7	0.82366	0.9627791	0.07302354	Distance Kernel
40	0.5	0.9	0.9	0.9622325	0.1	Distance Kernel
41	0.7	0.1	0.57838	0.9665355	0.1072164	Distance Kernel
42	0.7	0.3	0.66278	0.9641232	0.101658	Distance Kernel
43	0.7	0.5	0.74662	0.9634287	0.09028951	Distance Kernel
44	0.7	0.7	0.83046	0.9631017	0.08317705	Distance Kernel
45	0.7	0.9	0.905	0.9622325	0.2	Distance Kernel
46	0.9	0.1	0.58656	0.9651825	0.1237739	Distance Kernel
47	0.9	0.3	0.66576	0.9640038	0.1102951	Distance Kernel
48	0.9	0.5	0.74944	0.9638118	0.1003812	Distance Kernel
49	0.9	0.7	0.83188	0.9624808	0.09573614	Distance Kernel
50	0.9	0.9	0.915	0.967727	0.3	Distance Kernel
51	0.1	0.1	0.57412	1	0.05354377	Tolerance Interval
52	0.1	0.3	0.66158	1	0.03151418	Tolerance Interval
53	0.1	0.5	0.7572	1	0.02256178	Tolerance Interval
54	0.1	0.7	0.8516	0.9998506	0.01533592	Tolerance Interval
55	0.1	0.9	0.93154	0.9947368	0.19783924	Tolerance Interval
56	0.3	0.1	0.57496	1	0.05533945	Tolerance Interval
57	0.3	0.3	0.6631	1	0.03570271	Tolerance Interval
58	0.3	0.5	0.75788	1	0.02497714	Tolerance Interval
59	0.3	0.7	0.8523	0.9998932	0.007981687	Tolerance Interval
60	0.3	0.9	0.93722	0.9947368	0.1	Tolerance Interval
61	0.5	0.1	0.57636	1	0.0582746	Tolerance Interval
62	0.5	0.3	0.66428	1	0.03899848	Tolerance Interval
63	0.5	0.5	0.75832	1	0.02703368	Tolerance Interval
64	0.5	0.7	0.852	0.9998506	0.01754385	Tolerance Interval
65	0.5	0.9	0.93	0.9947368	0.1	Tolerance Interval
66	0.7	0.1	0.57334	1	0.05503084	Tolerance Interval
67	0.7	0.3	0.66612	1	0.04427516	Tolerance Interval
68	0.7	0.5	0.76036	1	0.03448108	Tolerance Interval
69	0.7	0.7	0.8524	0.9998506	0.02123251	Tolerance Interval
70	0.7	0.9	0.93	0.9947368	0.1	Tolerance Interval
71	0.9	0.1	0.57794	1	0.06161823	Tolerance Interval
72	0.9	0.3	0.66758	1	0.0481722	Tolerance Interval
73	0.9	0.5	0.76126	1	0.03819451	Tolerance Interval
74	0.9	0.7	0.85324	0.9998506	0.02712096	Tolerance Interval
75	0.9	0.9	0.935	0.9947368	0.2	Tolerance Interval
76	0.1	0.1	0.58964	0.6219182	0.5416533	Decision Tree
77	0.1	0.3	0.59776	0.6867474	0.4256442	Decision Tree

78	0.1	0.5	0.6455	0.7638124	0.2965806	Decision Tree
79	0.1	0.7	0.75428	0.8573771	0.1840564	Decision Tree
80	0.1	0.9	0.85754	0.9236677	0	Decision Tree
81	0.3	0.1	0.59704	0.6274832	0.5505429	Decision Tree
82	0.3	0.3	0.60888	0.6945525	0.4412472	Decision Tree
83	0.3	0.5	0.67126	0.7789462	0.3412931	Decision Tree
84	0.3	0.7	0.769	0.8631029	0.2185776	Decision Tree
85	0.3	0.9	0.89519	0.9320175	0.1252257	Decision Tree
86	0.5	0.1	0.60158	0.6321843	0.5556561	Decision Tree
87	0.5	0.3	0.6267	0.7074931	0.4669443	Decision Tree
88	0.5	0.5	0.68396	0.7861372	0.3645009	Decision Tree
89	0.5	0.7	0.8287	0.8688106	0.2516531	Decision Tree
90	0.5	0.9	0.87	0.9302729	0.08333333	Decision Tree
91	0.7	0.1	0.60648	0.6372164	0.5610909	Decision Tree
92	0.7	0.3	0.63342	0.7137988	0.4756032	Decision Tree
93	0.7	0.5	0.68754	0.7883776	0.370693	Decision Tree
94	0.7	0.7	0.77938	0.8686988	0.2524446	Decision Tree
95	0.7	0.9	0.895	0.9411462	0.2142857	Decision Tree
96	0.9	0.1	0.60816	0.6399419	0.5620293	Decision Tree
97	0.9	0.3	0.63308	0.7134873	0.4764793	Decision Tree
98	0.9	0.5	0.68412	0.7873414	0.3682815	Decision Tree
99	0.9	0.7	0.77484	0.8677161	0.2423266	Decision Tree
100	0.9	0.9	0.85	0.9283816	0.05	Decision Tree

Appendix II: R Code

```r
library(tidyr)
library(dplyr)
library(tolerance)
library(KernSmooth)
library(ggplot2)
library(mclust)
library(rpart)
library(limma)
library(graphics)
library(grDevices)
library(methods)
library(stats)
library(utils)
library(lattice)
library(latticeExtra)
library(RColorBrewer)

#Read in the data#
FunDat <- read.csv("dataDropped.csv")

initiate <- function(x = FunDat){
  x$Log2.FC <- as.numeric(levels(x$Log2.FC))[x$Log2.FC]
  #Create a single column key#
  x <- x %>% arrange(Maturation, Mother, Oocyte)
  x$OociteID <- x %>% unite(OocyteID, Maturation, Mother,
Oocyte)
  x$OocyteID <- x$OociteID$OocyteID
  x <- x[,c(1,2,3,4,5,7)]
  tableDat <- table(x$OocyteID)
  numGenes <- max(tableDat)
  goodIDs <- tableDat[tableDat == numGenes]
  numOocytes <- length(goodIDs)
  test <- as.vector(names(goodIDs))
  #Only oocytes with full profiles#
  x <- x[ which(x$OocyteID %in% test), ]
  x$GeneID <- rep(1:numGenes, numOocytes)
  x$OocyteID <- rep(1:numOocytes, each = numGenes)
  return(x)
}

FunDat <- initiate(FunDat)

InitialGlean = function(dataset = FunDat){
  Obs <- as.data.frame(matrix(seq(max(dataset$GeneID,
na.rm = T)), nrow=max(dataset$GeneID, na.rm = T), ncol =
10))
  colnames(Obs) <- c("Mui NonViable", "SDi NonViable",
"CVi NonViable", "absCVi NonViable", "wi NonViable",
"Mui Viable", "SDi Viable", "CVi Viable", "absCVi Viable",
"wi Viable")
  subsetNonViable <- 0
  subsetViable <- 0
  ViabletestVals = matrix(NA, nrow = nrow(Obs), ncol = 1)
  NonViabletestVals = matrix(NA, nrow = nrow(Obs),
  ncol = 1)
  for(j in (1:nrow(Obs))){
    subsetNonViable <- dataset$Log2.FC[
which(dataset$Viable == 0 & dataset$GeneID == j)]
    Obs[j,1] <- mean(subsetNonViable, na.rm = TRUE)
    Obs[j,2] <- sd(subsetNonViable, na.rm = TRUE)
    Obs[j,3] <- (Obs[j,2]/Obs[j,1])
    Obs[j,4] <- abs(Obs[j,3])

    subsetViable <- dataset$Log2.FC[ which(
      dataset$Viable == 1 & dataset$GeneID == j)]
    Obs[j,6] <- mean(subsetViable, na.rm = TRUE)
    Obs[j,7] <- sd(subsetViable, na.rm = TRUE)
    Obs[j,8] <- (Obs[j,7]/Obs[j,6])
    Obs[j,9] <- abs(Obs[j,8])
  }
  SumCV_NonViable <- sum(Obs$`absCVi NonViable`, na.rm =
TRUE)
  SumCV_Viable <- sum(Obs$`absCVi Viable`, na.rm = TRUE)
  #Get the gene by gene weights for the group
  Obs[,5] <- Obs[,4]/SumCV_NonViable
  Obs[,10] <- Obs[,9]/SumCV_Viable
  return(Obs)
}

wRMSD <- function(x = FunDat){
  Obs = InitialGlean(x)
  dat <- as.data.frame(matrix(seq(max(x$OocyteID)),
nrow=max(x$OocyteID), ncol = 4))
  colnames(dat) <- c("OocyteID", "Maturation", "Viable",
"wRMSD")
  #get viability for every Oocyte (58)
  for(k in (1:nrow(dat))){
    for(l in (1:nrow(x)))
      if(x$OocyteID[l] == k){
        dat[k,"Viable"] <- x$Viable[l]
        if(dat[k,"Viable"] == 1){
          dat[k,"Viable"] = 0
        }
        else{
          dat[k,"Viable"] = 1
        }
      }
  }
  #get maturation type for every Oocyte (58)
  for(k in (1:nrow(dat))){
    for(l in (1:nrow(x)))
      if(x$OocyteID[l] == k){
        dat[k,"Maturation"] <- x$Maturation[l]
      }
  }
  #for every Oocyte (58)
  for(i in (1:nrow(dat))){
    SumValue <- 0
    #for every Gene within Oocyte (69)
    for(j in (1:nrow(Obs))){
      if(dat$Viable[i] == 0){
        Emi <- Obs[j,1]
        Wi <- Obs[j,5]
      }
      else{
        Emi <- Obs[j,6]
        Wi <- Obs[j,10]
      }
#if the log2FC is missing for the gene, don't do anything
      if(is.na(x$Log2.FC[[ which(x$OocyteID == i &
x$GeneID == j)]]) == 1){
      }
      else{
        Ei <- x$Log2.FC[[ which(x$OocyteID == i &
x$GeneID == j)]]
        value <- Wi*(Emi - Ei)^2
        SumValue <- SumValue + value
      }
    }
    dat[i,"wRMSD"] <- sqrt(SumValue)
  }
  return(dat)
}

#The Function#
wRMSDPvals <- function(dat = wRMSDdat){
  # Sort the data by wRMSD #
  wRMSDdat <- dat[order(dat$wRMSD),]
  wRMSDdat$Order <- c(1:nrow(wRMSDdat))
  e <- density(na.omit(wRMSDdat$wRMSD[wRMSDdat$Viable ==
1]), from = 0, to = 30, n=3000, bw = "SJ")
  est2 <- approxfun(e)
  NonViabledist <- wRMSDdat[wRMSDdat$Viable == 0, ]
  Viabledist <- wRMSDdat[wRMSDdat$Viable == 1, ]
  for(i in (1:nrow(wRMSDdat))){
    if(wRMSDdat$wRMSD[i] > max(Viabledist$wRMSD)){
      wRMSDdat$Pval[i] <- 0
    }
    else{
      temp <- (integrate(est2, 0, wRMSDdat$wRMSD[i],
stop.on.error = FALSE, subdivisions = 200))
      wRMSDdat$Pval[i] <- 1 - temp$value
    }
    if(wRMSDdat$Pval[i] < 0){
      wRMSDdat$Pval[i] <- 0
    }
    if(wRMSDdat$Pval[i] > 1){
      wRMSDdat$Pval[i] <- 1
    }
  }
  return(wRMSDdat)
}

distances <- function(x = FunDat){
  Obs = InitialGlean(x)
  #Change the data format for the eBayes method
  #Note Oocytes 1-29 are Maturation 1 (i.e. IVM), 30-58 are
Maturation 2, (IVV) for the Isom data
  #Use Viability for the simulated data, use maturation for
the Isom data
  y = arrange(x, Viable)
  NumNonViable = nrow(y[y$Viable == 0,])/max(y$GeneID)
  NumViable = nrow(y[y$Viable == 1,])/max(y$GeneID)
  eBayesDat = spread(y[,c("Log2.FC","OocyteID","GeneID")],
OocyteID, Log2.FC)
  eBayesDat2 = eBayesDat[,-1]
  groupLabels = as.factor(c(rep("NonViable",
NumNonViable),rep("Viable", NumViable)))
  design = model.matrix(~0 + groupLabels)
  colnames(design) = levels(groupLabels)
  xTemp = lmFit(eBayesDat2, design)
  cont.matrix<-makeContrasts(NonViable-Viable,
levels=design)
  fit2<-contrasts.fit(xTemp, cont.matrix)
  ebfit<-eBayes(fit2)
  xTemp = topTable(ebfit, coef=1, number = nrow(xTemp))
  pvalTemp = cbind(row.names(xTemp),xTemp[,6])
  colnames(pvalTemp) = c("GeneID", "P-value")
  #Select the differentially expressed Genes
  sigPvalTemp = pvalTemp[pvalTemp[,2] < 0.05,]
  for(i in 1:nrow(y)){
    if(is.element(y$GeneID[i], sigPvalTemp[,1])){
      y[i,"Log2.FC"] = y[i,"Log2.FC"]
    }
    else{
      y[i,"Log2.FC"] = NA
    }
  }
  dat <- as.data.frame(matrix(seq(max(y$OocyteID)),
nrow=max(y$OocyteID), ncol = 4))
  colnames(dat) <- c("OocyteID", "Maturation", "Viable",
"Distances")
  #get maturation type for every Oocyte (58)
  for(k in (1:nrow(dat))){
    for(l in (1:nrow(y)))
      if(y$OocyteID[l] == k){
        dat[k,"Maturation"] <- y$Maturation[l]
      }
  }
  #get viability for every Oocyte (58)
  for(k in (1:nrow(dat))){
    for(l in (1:nrow(y)))
      if(y$OocyteID[l] == k){
        dat[k,"Viable"] <- y$Viable[l]
        if(dat[k,"Viable"] == 1){
          dat[k,"Viable"] = 0
        }
        else{
          dat[k,"Viable"] = 1
        }
      }
  }
  #for every Oocyte (58)
  for(i in (1:nrow(dat))){
    SumValue <- 0
    #for every Gene within Oocyte (67)
    for(j in (1:nrow(Obs))){
      # Compared to the IVV Gene Mean Expressions #
      ViablegeneMeanExpr <- mean(y[which(y$Viable == 1 &
y$GeneID == j), "Log2.FC"], na.rm = TRUE)
      if(dat$Viable[i] == 0){
        Emi <- ViablegeneMeanExpr
        Wi <- 1/Obs[j,"SDi NonViable"]
      }
      else{
        Emi <- ViablegeneMeanExpr
        Wi <- 1/Obs[j,"SDi Viable"]
      }
      #if the log2FC is missing for the gene, don't do
anything
      if(is.na(y$Log2.FC[[ which(y$OocyteID == i &
y$GeneID == j)]]) == 1){ }
      else{
        Ei <- y$Log2.FC[[ which(y$OocyteID == i &
          y$GeneID == j)]]
        value <- Wi*(Emi - Ei)^2
        SumValue <- SumValue + value
      }
    }
    dat[i,"Distances"] <- sqrt(SumValue)
  }
  return(dat)
}

DistancePvals <- function(dat = distData){
  # Sort the data by distances #
  dat <- dat[order(dat$Distances),]
  dat$Order <- c(1:nrow(dat))

  e <- density(na.omit(dat$Distances[dat$Viable == 1]),
from = 0, to = 30, n=3000, bw = "SJ")
  est2 <- approxfun(e)
  NonViabledist <- dat[dat$Viable == 0, ]
  Viabledist <- dat[dat$Viable == 1, ]
  for(i in (1:nrow(dat))){
    if(dat$Distances[i] > max(Viabledist$Distances)){
      dat$Pval[i] <- 0
    }
    else{
      temp <- (integrate(est2, 0, dat$Distances[i]))
      dat$Pval[i] <- 1 - temp$value
    }
    if(dat$Pval[i] < 0){
      dat$Pval[i] <- 0
    }
    if(dat$Pval[i] > 1){
      dat$Pval[i] <- 1
    }
  }
```

```
    return(dat)
}

tolInt <- function(x = distFunDat){
  distFunDat <- distances(x)
  Viabledist <- x[x$Viable == 1, ]
  tolInterval <- nptol.int(Viabledist$Distances, alpha =
.05, P = 0.95, side = 1)

  return(tolInterval)
}

TreeFunc <- function(x = FunDat){
  #Reformat the data#
  TreeDat <- x[,c("Gene","Viable","Log2.FC","OocyteID")]
  TreeDat <- spread(TreeDat, Gene, Log2.FC)
  ifelse(TreeDat$Viable == 1, TreeDat$Maturation <-
"Viable", TreeDat$Maturation <- "Non-Viable")
  cols <- as.formula(paste(colnames(TreeDat)[1],
"~",paste(colnames(TreeDat)[c(3:ncol(TreeDat))], collapse =
"+"), sep = ""))
  #Grow Trees#
  csv.cp=csv.rpartfull<-
rpart(cols,control=rpart.control(cp=0.0,minsplit=2),data=Tr
eeDat)
  csv.cp.xval=rep(0,nrow(TreeDat))
  xvs=rep(1:10,length=nrow(TreeDat))
  xvs=sample(xvs)
  data.cp.xval=rep(0,nrow(TreeDat))
  xvs=rep(1:10,length=nrow(TreeDat))
  xvs=sample(xvs)
  for(i in 1:10){
    test=TreeDat[xvs==i,]
    train=TreeDat[xvs!=i,]
    glub=rpart(cols, control=rpart.control(cp=0.0,
minsplit=2),data=train)
    data.cp.xval[xvs==i]=predict(glub,test,type="class")}
  #Use this for real data without vialbe/non-viable markers

#data.cpconfuse.xval=table(TreeDat$Maturation,data.cp.xval)
  #Use this for simulated data
  data.cpconfuse.xval=table(TreeDat$Viable,data.cp.xval)
  colnames(data.cpconfuse.xval) <- c("True Non-Viable",
"True Viable")
  rownames(data.cpconfuse.xval) <- c("Pred Non-Viable",
"Pred Viable")
  #Overall Error Rate#
  #100-100*sum(diag(data.cpconfuse.xval))/nrow(TreeDat)
  data.cpconfuse.xval
  #plot(csv.cp, margin = 0.1, main = "4. Trimmed Decision
Tree, Isom Data");text(csv.cp)

  return(data.cpconfuse.xval)
}

DataSimulation = function(numOocytes, pi, delta, GeneNames,
whichDEgenes, probabilityViableIVM, dat = eBayesDat2){
  #Error Checking#
  if(pi > 1 | pi < 0) stop("pi, the mixing
proportion/degree of similarity is invalid, must be between
0 and 1 inclusive")
  if(probabilityViableIVM > 1 | probabilityViableIVM < 0)
stop("probabilityViableIVM is invalid, must be between 0
and 1 inclusive")

  #Oocyte Level#
  # Note: Maturation = 0 means IVM, Maturation = 1 means
IVV
  Oocytes = as.data.frame(matrix(seq(1:numOocytes),
     nrow = numOocytes, ncol = 1))
  colnames(Oocytes) = "OocyteID"
  Oocytes$Maturation = rbinom(numOocytes, size = 1,
     prob = 0.5)
  for(i in 1:numOocytes){
    if(Oocytes$Maturation[i] == 1){
      Oocytes$Viable[i] = 1
    }
    else{
      Oocytes$Viable[i] = rbinom(1,1,probabilityViableIVM)
    }}
  while(nrow(Oocytes[Oocytes$Viable == 0,]) < 2){
    for(i in 1:numOocytes){
      if(Oocytes$Maturation[i] == 1){
        Oocytes$Viable[i] = 1
      }
      else{
        Oocytes$Viable[i] =
rbinom(1,1,probabilityViableIVM)
      }}
    while(nrow(Oocytes[Oocytes$Viable == 1,]) < 2){
      for(i in 1:numOocytes){
        if(Oocytes$Maturation[i] == 1){
          Oocytes$Viable[i] = 1
        }
        else{
          Oocytes$Viable[i] =
rbinom(1,1,probabilityViableIVM)
        }}
  }
  }
  #Gene Level#
  #Setup#
  GeneI = as.vector(matrix(NA, nrow = numOocytes,
     ncol = 1))
  for(j in 1:nrow(dat)){
    #j goes from 1 to number of genes
    #This delta_i
    if(is.element(j, whichDEgenes)){deltai =
     runif(1, min = 0, max = abs(delta))}
    else{deltai = runif(1, min = 0, max = 0)}

    #Oocyte Level Within Gene Level#
    for(i in 1:numOocytes){
      #i goes from 1 to number of oocytes
      #Gene Expression Distributions#
      IVVdist = rnorm(1, mean = 0, sd = 1)
      IVMdist = rnorm(1, mean = deltai, sd = 1)

      IVVSample = IVVdist
      comp = rbinom(1,1,pi)
      IVMViableSample = comp*IVVdist + (1-comp)*IVMdist
      IVMNonViableSample = IVMdist

      if(Oocytes$Maturation[i] == 1){
        GeneI[i] = IVVSample
      }
      else{
        if(Oocytes$Viable[i] == 0){
          GeneI[i] = IVMNonViableSample
        }
        else{
          GeneI[i] = IVMViableSample
        }
      }
    }
    Oocytes = cbind(Oocytes, GeneI)
    GeneNum = j
    colnames(Oocytes) =
c(colnames(Oocytes[,c(1:(ncol(Oocytes)-1))]), GeneNum)
  }
  Oocytes = gather(Oocytes, "GeneID", "Log2.FC", 4:70)
  Oocytes = arrange(Oocytes, OocyteID)
  Gene = as.vector(levels(FunDat$Gene))
  Oocytes = cbind(Oocytes, Gene)
  Oocytes$GeneID = as.numeric(Oocytes$GeneID)
  Oocytes$Viable = as.factor(Oocytes$Viable)
  return(Oocytes)
}

wRMSDres = function(dat){
  temp = wRMSD(dat)
  temp = wRMSDPvals(temp)
  Results = as.data.frame(matrix(NA, nrow = 1, ncol = 4))
  colnames(Results) =
c("M1negneg","M1negpos","M1posneg","M1pospos")
  Results$M1negneg = nrow(temp[temp$Viable == 0 &
     temp$Pval < 0.05,])
  Results$M1negpos = nrow(temp[temp$Viable == 0 &
     temp$Pval >= 0.05,])
  Results$M1posneg = nrow(temp[temp$Viable == 1 &
     temp$Pval < 0.05,])
  Results$M1pospos = nrow(temp[temp$Viable == 1 &
     temp$Pval >= 0.05,])
  Results$M1PCC = (Results$M1negneg +
Results$M1pospos)/(Results$M1negneg + Results$M1negpos +
Results$M1posneg + Results$M1pospos)
  Results$M1Sens = (Results$M1pospos)/(Results$M1posneg +
Results$M1pospos)
  Results$M1Spec = (Results$M1negneg)/(Results$M1negneg +
Results$M1negpos)
  return(Results)
}

distancesRes = function(dat){
  temp = distances(dat)
  temp = DistancePvals(temp)
  Results = as.data.frame(matrix(NA, nrow = 1, ncol = 4))
  colnames(Results) =
c("M2negneg","M2negpos","M2posneg","M2pospos")
  Results$M2negneg = nrow(temp[temp$Viable == 0 &
     temp$Pval < 0.05,])
  Results$M2negpos = nrow(temp[temp$Viable == 0 &
     temp$Pval >= 0.05,])
  Results$M2posneg = nrow(temp[temp$Viable == 1 &
     temp$Pval < 0.05,])
  Results$M2pospos = nrow(temp[temp$Viable == 1 &
     temp$Pval >= 0.05,])
  Results$M2PCC = (Results$M2negneg + 
Results$M2pospos)/(Results$M2negneg + Results$M2negpos + 
Results$M2posneg + Results$M2pospos)
  Results$M2Sens = (Results$M2pospos)/(Results$M2posneg + 
Results$M2pospos)
  Results$M2Spec = (Results$M2negneg)/(Results$M2negneg + 
Results$M2negpos)
  return(Results)
}

tolIntRes = function(dat){
  tempDist = distances(dat)
  temp = as.data.frame(tolInt(tempDist))
  Results = as.data.frame(matrix(NA, nrow = 1, ncol = 4))
  colnames(Results) = 
c("M3negneg","M3negpos","M3posneg","M3pospos")
  Results$M3negneg = nrow(tempDist[tempDist$Viable == 0 & 
tempDist$Distances > temp[1,4],])
  Results$M3negpos = nrow(tempDist[tempDist$Viable == 0 & 
tempDist$Distances <= temp[1,4],])
  Results$M3posneg = nrow(tempDist[tempDist$Viable == 1 & 
tempDist$Distances > temp[1,4],])
  Results$M3pospos = nrow(tempDist[tempDist$Viable == 1 & 
tempDist$Distances <= temp[1,4],])
  Results$M3PCC = (Results$M3negneg + 
Results$M3pospos)/(Results$M3negneg + Results$M3negpos + 
Results$M3posneg + Results$M3pospos)
  Results$M3Sens = (Results$M3pospos)/(Results$M3posneg + 
Results$M3pospos)
  Results$M3Spec = (Results$M3negneg)/(Results$M3negneg + 
Results$M3negpos)
  return(Results)
}

treeFuncRes = function(dat){
  temp = TreeFunc(dat)
  temp = as.data.frame(temp)
  Results = as.data.frame(matrix(NA, nrow = 1, ncol = 4))
  colnames(Results) = c("M4negneg","M4posneg","M4negpos", 
"M4pospos")
  Results$M4negneg = temp$Freq[1]
  Results$M4negpos = temp$Freq[2]
  Results$M4posneg = temp$Freq[3]
  Results$M4pospos = temp$Freq[4]
  Results$M4PCC = (Results$M4negneg + 
Results$M4pospos)/(Results$M4negneg + Results$M4negpos + 
Results$M4posneg + Results$M4pospos)
```

```
  Results$M4Sens = (Results$M4pospos)/(Results$M4posneg +
Results$M4pospos)
  Results$M4Spec = (Results$M4negneg)/(Results$M4negneg +
Results$M4negpos)
  return(Results)
}

SimulationRes = function(NumItterations = 100, Dat =
FunDat){
  #Simulations#
  #We will keep delta and proportionDEgenes constant for
this project#
  #delta should be found by looking at the log fold change
of the eBayes (observed) data and take the median
  #proportionDEgenes should be found by looking at the
multiple-testing adjusted p-values of the eBayes (observed)
  #data and noting the total number of genes that have a p-
value of <0.05, divided by total number of genes in
dataset.

  # The Check for delta #
  #Note Oocytes 1-29 are Maturation 1 (i.e. IVM), 30-58 are
Maturation 2, (IVV)
  #Change the data format for the eBayes method
  eBayesDat = spread(Dat[,c(5,6,7,1)], OocyteID, Log2.FC)
#"Log2.FC","OocyteID","GeneID","Gene"
  Genes = as.character(eBayesDat[,"Gene"])
  eBayesDat2 = eBayesDat[,-c(1,2)]
  groupLabels = as.factor(c(rep("IVV",
ncol(eBayesDat2)/2),rep("IVM", ncol(eBayesDat2)/2)))
  design = model.matrix(~0 + groupLabels)
  colnames(design) = levels(groupLabels)
  fit = lmFit(eBayesDat2, design)
  cont.matrix<-makeContrasts(IVM-IVV, levels=design)
  fit2<-contrasts.fit(fit, cont.matrix)
  ebfit<-eBayes(fit2)
  #Get the Log-fold change of the eBayes data (ebfit) and
take median of differentially expressed genes to find delta
  tempTab = topTable(ebfit, coef=1, number=30)

  Delta = median(tempTab[tempTab$adj.P.Val < 0.05,"logFC"])

  # The Check for proportionDEgenes #
  ProportionDEgenes = nrow(tempTab[tempTab$adj.P.Val <
0.05,])/nrow(eBayesDat)
  #Identify Which Genes
  DEgenes = row.names(tempTab[tempTab$adj.P.Val < 0.05,])
  #Set up pi and proportion Viable IVM to loop over various
values
  Pi = as.vector(c(0.1, 0.3, 0.5, 0.7, 0.9))
  ProbabilityViableIVM = as.vector(c(0.1, 0.3, 0.5, 0.7,
0.9))

  #Result matricies
  pcc1 = matrix(NA, nrow = length(Pi), ncol =
length(ProbabilityViableIVM))
  pcc2 = matrix(NA, nrow = length(Pi), ncol =
length(ProbabilityViableIVM))
  pcc3 = matrix(NA, nrow = length(Pi), ncol =
length(ProbabilityViableIVM))
  pcc4 = matrix(NA, nrow = length(Pi), ncol =
length(ProbabilityViableIVM))

  sens1 = matrix(NA, nrow = length(Pi), ncol =
length(ProbabilityViableIVM))
  sens2 = matrix(NA, nrow = length(Pi), ncol =
length(ProbabilityViableIVM))
  sens3 = matrix(NA, nrow = length(Pi), ncol =
length(ProbabilityViableIVM))
  sens4 = matrix(NA, nrow = length(Pi), ncol =
length(ProbabilityViableIVM))

  spec1 = matrix(NA, nrow = length(Pi), ncol =
length(ProbabilityViableIVM))
  spec2 = matrix(NA, nrow = length(Pi), ncol =
length(ProbabilityViableIVM))
  spec3 = matrix(NA, nrow = length(Pi), ncol =
length(ProbabilityViableIVM))
  spec4 = matrix(NA, nrow = length(Pi), ncol =
length(ProbabilityViableIVM))

  #Data Generation and Results
  for(i in 1:length(Pi)){
    for(j in 1:length(ProbabilityViableIVM)){
      r1 = r2 = r3 = r4 = as.data.frame(matrix(NA, nrow =
NumItterations, ncol = 7)) #should have the the confusion
matrix results (4) in addition to pcc, sens, spec, etc.
      for(k in 1:NumItterations){
        data <- DataSimulation(numOocytes = 100, pi =
Pi[i], delta = Delta, GeneNames = Genes, whichDEgenes =
DEgenes, probabilityViableIVM = ProbabilityViableIVM[j],
dat = eBayesDat2)
```

```
        while(nrow(data[data$Viable == 0,]) <
2*length(Genes)){
          data <- DataSimulation(numOocytes = 100, pi =
Pi[i], delta = Delta, GeneNames = Genes, whichDEgenes =
DEgenes, probabilityViableIVM = ProbabilityViableIVM[j],
dat = eBayesDat2)
        }
        r1[k,] = wRMSDres(data)
        r2[k,] = distancesRes(data)
        r3[k,] = tolIntRes(data)
        r4[k,] = treeFuncRes(data)
      }
      #Rows are pi values, columns are probabilityViableIVM
      temp = apply(r1[,c(5:7)], 2, mean)
      pcc1[i,j] = temp[1]
      sens1[i,j] = temp[2]
      spec1[i,j] = temp[3]

      temp = apply(r2[,c(5:7)], 2, mean)
      pcc2[i,j] = temp[1]
      sens2[i,j] = temp[2]
      spec2[i,j] = temp[3]


      temp = apply(r3[,c(5:7)], 2, mean)
      pcc3[i,j] = temp[1]
      sens3[i,j] = temp[2]
      spec3[i,j] = temp[3]


      temp = apply(r4[,c(5:7)], 2, mean)
      pcc4[i,j] = temp[1]
      sens4[i,j] = temp[2]
      spec4[i,j] = temp[3]
    }
  }
  SimulationResults = list(pcc1, pcc2, pcc3, pcc4, sens1,
sens2, sens3, sens4, spec1, spec2, spec3, spec4)
  return(SimulationResults)
}

set.seed(122816)
source("~/Documents/Current
Classes/Research/FunctionSource.R")
start = proc.time()
example = SimulationRes(NumItterations = 500, Dat = FunDat)
end = proc.time()
```

```
totalTime = end-start

temp = as.data.frame(matrix(NA, nrow = 25, ncol = 1))
temp$Pi = c(rep(0.1, 5),rep(0.3, 5),rep(0.5, 5),rep(0.7,
5),rep(0.9, 5) )
temp$ProbViableIVM = c(rep(c(0.1, 0.3, 0.5, 0.7, 0.9), 5))
temp = temp[,-1]
count = 1
Method = list(NA)
for(l in 1:4){
for(i in 1:nrow(Results[[l]])){
  for(j in 1:ncol(Results[[l]])){
    temp$PCC[count] = Results[[l]][i,j]
    count = count + 1
  }
}
count = 1
for(i in 1:nrow(Results[[l+4]])){
  for(j in 1:ncol(Results[[l+4]])){
    temp$Sens[count] = Results[[l+4]][i,j]
    count = count + 1
  }
}
count = 1
for(i in 1:nrow(Results[[l+8]])){
  for(j in 1:ncol(Results[[l+8]])){
    temp$Spec[count] = Results[[l+8]][i,j]
    count = count + 1
  }
}

Method[[l]] = temp
temp = as.data.frame(matrix(NA, nrow = 25, ncol = 1))
temp$Pi = c(rep(0.1, 5),rep(0.3, 5),rep(0.5, 5),rep(0.7,
5),rep(0.9, 5) )
temp$ProbViableIVM = c(rep(c(0.1, 0.3, 0.5, 0.7, 0.9), 5))
temp = temp[,-1]
count = 1
}
Method[[1]]$Method = as.factor(rep("wRMSD",25))
Method[[2]]$Method = as.factor(rep("Distance Kernel",25))
Method[[3]]$Method = as.factor(rep("Tolerance
Interval",25))
Method[[4]]$Method = as.factor(rep("Decision Tree",25))

temp = rbind(Method[[1]], Method[[2]], Method[[3]],
Method[[4]])
```

```
#PCC by Pi
Prob1 = temp[seq(1,100,5),]
mypanel <- function(x, y) {
       panel.superpose(x, y, groups =
as.factor(Prob1$ProbViableIVM), type = "o", subscripts =
as.factor(Prob1$ProbViableIVM),lty = 1, col =
brewer.pal(9,"Blues")[3])
       panel.grid(h=-1, v=-1)
   }
plot1 = xyplot(Prob1$PCC~Prob1$Pi|Prob1$Method, xlab = "Pi,
Mixing Proportion", ylab = "Proportion Correctly
Classified", panel = mypanel, ylim = c(0.5,1.0), main=
"Plot of PCC by Pi, Probability IVM Viable, and Method",
               key=list(columns=5,
                        text=list(lab=c("ProbIVMViable =
0.1","0.3","0.5","0.7","0.9")),
                        lines=list(lty = c(1,2,3,4,5), col
= c(brewer.pal(9,"Blues")[3], brewer.pal(9,"Blues")[5],
brewer.pal(9,"Blues")[7], brewer.pal(9,"Blues")[8],
brewer.pal(9,"Blues")[9]), lwd = 2)))

Prob2 = temp[seq(2,100,5),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups =
as.factor(Prob2$ProbViableIVM), type = "o", subscripts =
as.factor(Prob2$ProbViableIVM),lty = 2, col =
brewer.pal(9,"Blues")[5], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot2 = xyplot(Prob2$PCC~Prob2$Pi|Prob1$Method, panel =
mypanel)

Prob3 = temp[seq(3,100,5),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups =
as.factor(Prob3$ProbViableIVM), type = "o", subscripts =
as.factor(Prob3$ProbViableIVM),lty = 3, col =
brewer.pal(9,"Blues")[7], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot3 = xyplot(Prob3$PCC~Prob3$Pi|Prob1$Method, panel =
mypanel)

Prob4 = temp[seq(4,100,5),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups =
as.factor(Prob4$ProbViableIVM), type = "o", subscripts =
as.factor(Prob4$ProbViableIVM),lty = 4, col =
brewer.pal(9,"Blues")[8], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot4 = xyplot(Prob4$PCC~Prob4$Pi|Prob1$Method, panel =
mypanel)

Prob5 = temp[seq(5,100,5),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups =
as.factor(Prob5$ProbViableIVM), type = "o", subscripts =
as.factor(Prob5$ProbViableIVM),lty = 5, col =
brewer.pal(9,"Blues")[9], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot5 = xyplot(Prob5$PCC~Prob5$Pi|Prob1$Method, panel =
mypanel)

FinalPlot1 =
plot1+as.layer(plot2)+as.layer(plot3)+as.layer(plot4)+as.la
yer(plot5)


#PCC by ProbIVMViable
Prob1 = temp[c(1:5,26:30,51:55,76:80),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups = as.factor(Prob1$Pi), type
= "o", subscripts = as.factor(Prob1$Pi),lty = 1, col =
brewer.pal(9,"Blues")[3])
  panel.grid(h=-1, v=-1)
}
plot1 = xyplot(Prob1$PCC~Prob1$ProbViableIVM|Prob1$Method,
xlab = "Probability Viable IVM", ylab = "Proportion
Correctly Classified", panel = mypanel, ylim = c(0.5,1.0),
main= "Plot of PCC by Probability IVMViable, Pi and
Method",
               key=list(columns=5,
                        text=list(lab=c("Pi =
0.1","0.3","0.5","0.7","0.9")),
                        lines=list(lty = c(1,2,3,4,5), col
= c(brewer.pal(9,"Blues")[3], brewer.pal(9,"Blues")[5],
brewer.pal(9,"Blues")[7], brewer.pal(9,"Blues")[8],
brewer.pal(9,"Blues")[9]), lwd = 2)))
```

```
Prob2 = temp[c(6:10,31:35,56:60,81:85),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups = as.factor(Prob2$Pi), type
= "o", subscripts = as.factor(Prob2$Pi),lty = 2, col =
brewer.pal(9,"Blues")[5], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot2 = xyplot(Prob2$PCC~Prob2$ProbViableIVM|Prob1$Method,
panel = mypanel)

Prob3 = temp[c(11:15,36:40,61:65,86:90),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups = as.factor(Prob3$Pi), type
= "o", subscripts = as.factor(Prob3$Pi),lty = 3, col =
brewer.pal(9,"Blues")[7], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot3 = xyplot(Prob3$PCC~Prob3$ProbViableIVM|Prob1$Method,
panel = mypanel)

Prob4 = temp[c(16:20,41:45,66:70,91:95),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups = as.factor(Prob4$Pi), type
= "o", subscripts = as.factor(Prob4$Pi),lty = 4, col =
brewer.pal(9,"Blues")[8], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot4 = xyplot(Prob4$PCC~Prob4$ProbViableIVM|Prob1$Method,
panel = mypanel)

Prob5 = temp[c(21:25,46:50,71:75,96:100),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups = as.factor(Prob5$Pi), type
= "o", subscripts = as.factor(Prob5$Pi),lty = 5, col =
brewer.pal(9,"Blues")[9], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot5 = xyplot(Prob5$PCC~Prob5$ProbViableIVM|Prob1$Method,
panel = mypanel)

FinalPlot2 =
plot1+as.layer(plot2)+as.layer(plot3)+as.layer(plot4)+as.la
yer(plot5)

#Sens by Pi
Prob1 = temp[seq(1,100,5),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups =
as.factor(Prob1$ProbViableIVM), type = "o", subscripts =
as.factor(Prob1$ProbViableIVM),lty = 1, col =
brewer.pal(9,"Blues")[3])
  panel.grid(h=-1, v=-1)
}
plot1 = xyplot(Prob1$Sens~Prob1$Pi|Prob1$Method, xlab =
"Pi, Mixing Proportion", ylab = "Sensitivity", panel =
mypanel, ylim = c(0.6,1.1), main= "Plot of Sensitivity by
Pi, Probability IVM Viable and Method",
               key=list(columns=5,
                        text=list(lab=c("ProbIVMViable =
0.1","0.3","0.5","0.7","0.9")),
                        lines=list(lty = c(1,2,3,4,5), col
= c(brewer.pal(9,"Blues")[3], brewer.pal(9,"Blues")[5],
brewer.pal(9,"Blues")[7], brewer.pal(9,"Blues")[8],
brewer.pal(9,"Blues")[9]), lwd = 2)))

Prob2 = temp[seq(2,100,5),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups =
as.factor(Prob2$ProbViableIVM), type = "o", subscripts =
as.factor(Prob2$ProbViableIVM),lty = 2, col =
brewer.pal(9,"Blues")[5], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot2 = xyplot(Prob2$Sens~Prob2$Pi|Prob1$Method, panel =
mypanel)

Prob3 = temp[seq(3,100,5),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups =
as.factor(Prob3$ProbViableIVM), type = "o", subscripts =
as.factor(Prob3$ProbViableIVM),lty = 3, col =
brewer.pal(9,"Blues")[7], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot3 = xyplot(Prob3$Sens~Prob3$Pi|Prob1$Method, panel =
mypanel)

Prob4 = temp[seq(4,100,5),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups =
as.factor(Prob4$ProbViableIVM), type = "o", subscripts =
as.factor(Prob4$ProbViableIVM),lty = 4, col =
brewer.pal(9,"Blues")[8], lwd = 2)
  panel.grid(h=-1, v=-1)
```

```
}
plot4 = xyplot(Prob4$Sens~Prob4$Pi|Prob1$Method, panel =
mypanel)

Prob5 = temp[seq(5,100,5),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups =
as.factor(Prob5$ProbViableIVM), type = "o", subscripts =
as.factor(Prob5$ProbViableIVM),lty = 5, col =
brewer.pal(9,"Blues")[9], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot5 = xyplot(Prob5$Sens~Prob5$Pi|Prob1$Method, panel =
mypanel)

FinalPlot3 =
plot1+as.layer(plot2)+as.layer(plot3)+as.layer(plot4)+as.la
yer(plot5)


#Sens by ProbIVMViable
Prob1 = temp[c(1:5,26:30,51:55,76:80),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups = as.factor(Prob1$Pi), type
= "o", subscripts = as.factor(Prob1$Pi),lty = 1, col =
brewer.pal(9,"Blues")[3])
  panel.grid(h=-1, v=-1)
}
plot1 = xyplot(Prob1$Sens~Prob1$ProbViableIVM|Prob1$Method,
xlab = "Probability Viable IVM", ylab = "Sensitivity",
panel = mypanel, ylim = c(0.6,1.1), main= "Plot of
Sensitivity by Pi, Probability IVM Viable, and Method",
               key=list(columns=5,
                        text=list(lab=c("Pi =
0.1","0.3","0.5","0.7","0.9")),
                        lines=list(lty = c(1,2,3,4,5), col
= c(brewer.pal(9,"Blues")[3], brewer.pal(9,"Blues")[5],
brewer.pal(9,"Blues")[7], brewer.pal(9,"Blues")[8],
brewer.pal(9,"Blues")[9]), lwd = 2)))


Prob2 = temp[c(6:10,31:35,56:60,81:85),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups = as.factor(Prob2$Pi), type
= "o", subscripts = as.factor(Prob2$Pi),lty = 2, col =
brewer.pal(9,"Blues")[5], lwd = 2)
  panel.grid(h=-1, v=-1)
```

```
}
plot2 = xyplot(Prob2$Sens~Prob2$ProbViableIVM|Prob1$Method,
panel = mypanel)

Prob3 = temp[c(11:15,36:40,61:65,86:90),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups = as.factor(Prob3$Pi), type
= "o", subscripts = as.factor(Prob3$Pi),lty = 3, col =
brewer.pal(9,"Blues")[7], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot3 = xyplot(Prob3$Sens~Prob3$ProbViableIVM|Prob1$Method,
panel = mypanel)

Prob4 = temp[c(16:20,41:45,66:70,91:95),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups = as.factor(Prob4$Pi), type
= "o", subscripts = as.factor(Prob4$Pi),lty = 4, col =
brewer.pal(9,"Blues")[8], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot4 = xyplot(Prob4$Sens~Prob4$ProbViableIVM|Prob1$Method,
panel = mypanel)

Prob5 = temp[c(21:25,46:50,71:75,96:100),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups = as.factor(Prob5$Pi), type
= "o", subscripts = as.factor(Prob5$Pi),lty = 5, col =
brewer.pal(9,"Blues")[9], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot5 = xyplot(Prob5$Sens~Prob5$ProbViableIVM|Prob1$Method,
panel = mypanel)

FinalPlot4 =
plot1+as.layer(plot2)+as.layer(plot3)+as.layer(plot4)
+as.layer(plot5)

#Spec by Pi
Prob1 = temp[seq(1,100,5),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups =
as.factor(Prob1$ProbViableIVM), type = "o", subscripts =
as.factor(Prob1$ProbViableIVM),lty = 1, col =
brewer.pal(9,"Blues")[3])
  panel.grid(h=-1, v=-1)
}
```

```
plot1 = xyplot(Prob1$Spec~Prob1$Pi|Prob1$Method, xlab =
"Pi, Mixing Proportion", ylab = "Specificity", panel =
mypanel, ylim = c(0,0.6), main= "Plot of Specificity by Pi,
Probability IVM Viable and Method",
               key=list(columns=5,
                        text=list(lab=c("ProbIVMViable =
0.1","0.3","0.5","0.7","0.9")),
                        lines=list(lty = c(1,2,3,4,5), col
= c(brewer.pal(9,"Blues")[3], brewer.pal(9,"Blues")[5],
brewer.pal(9,"Blues")[7], brewer.pal(9,"Blues")[8],
brewer.pal(9,"Blues")[9]), lwd = 2)))

Prob2 = temp[seq(2,100,5),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups =
as.factor(Prob2$ProbViableIVM), type = "o", subscripts =
as.factor(Prob2$ProbViableIVM),lty = 2, col =
brewer.pal(9,"Blues")[5], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot2 = xyplot(Prob2$Spec~Prob2$Pi|Prob1$Method, panel =
mypanel)

Prob3 = temp[seq(3,100,5),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups =
as.factor(Prob3$ProbViableIVM), type = "o", subscripts =
as.factor(Prob3$ProbViableIVM),lty = 3, col =
brewer.pal(9,"Blues")[7], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot3 = xyplot(Prob3$Spec~Prob3$Pi|Prob1$Method, panel =
mypanel)

Prob4 = temp[seq(4,100,5),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups =
as.factor(Prob4$ProbViableIVM), type = "o", subscripts =
as.factor(Prob4$ProbViableIVM),lty = 4, col =
brewer.pal(9,"Blues")[8], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot4 = xyplot(Prob4$Spec~Prob4$Pi|Prob1$Method, panel =
mypanel)

Prob5 = temp[seq(5,100,5),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups =
as.factor(Prob5$ProbViableIVM), type = "o", subscripts =
as.factor(Prob5$ProbViableIVM),lty = 5, col =
brewer.pal(9,"Blues")[9], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot5 = xyplot(Prob5$Spec~Prob5$Pi|Prob1$Method, panel =
mypanel)

FinalPlot5 =
plot1+as.layer(plot2)+as.layer(plot3)+as.layer(plot4)+as.la
yer(plot5)

#Spec by ProbIVMViable
Prob1 = temp[c(1:5,26:30,51:55,76:80),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups = as.factor(Prob1$Pi), type
= "o", subscripts = as.factor(Prob1$Pi),lty = 1, col =
brewer.pal(9,"Blues")[3])
  panel.grid(h=-1, v=-1)
}
plot1 = xyplot(Prob1$Spec~Prob1$ProbViableIVM|Prob1$Method,
xlab = "Probability Viable IVM", ylab = "Specificity",
panel = mypanel, ylim = c(0,0.6), main= "Plot of
Specificity by Pi, Probability IVM Viable and Method",
               key=list(columns=5,
                        text=list(lab=c("Pi =
0.1","0.3","0.5","0.7","0.9")),
                        lines=list(lty = c(1,2,3,4,5), col
= c(brewer.pal(9,"Blues")[3], brewer.pal(9,"Blues")[5],
brewer.pal(9,"Blues")[7], brewer.pal(9,"Blues")[8],
brewer.pal(9,"Blues")[9]), lwd = 2)))

Prob2 = temp[c(6:10,31:35,56:60,81:85),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups = as.factor(Prob2$Pi), type
= "o", subscripts = as.factor(Prob2$Pi),lty = 2, col =
brewer.pal(9,"Blues")[5], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot2 = xyplot(Prob2$Spec~Prob2$ProbViableIVM|Prob1$Method,
panel = mypanel)

Prob3 = temp[c(11:15,36:40,61:65,86:90),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups = as.factor(Prob3$Pi), type
= "o", subscripts = as.factor(Prob3$Pi),lty = 3, col =
brewer.pal(9,"Blues")[7], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot3 = xyplot(Prob3$Spec~Prob3$ProbViableIVM|Prob1$Method,
panel = mypanel)

Prob4 = temp[c(16:20,41:45,66:70,91:95),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups = as.factor(Prob4$Pi), type
= "o", subscripts = as.factor(Prob4$Pi),lty = 4, col =
brewer.pal(9,"Blues")[8], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot4 = xyplot(Prob4$Spec~Prob4$ProbViableIVM|Prob1$Method,
panel = mypanel)

Prob5 = temp[c(21:25,46:50,71:75,96:100),]
mypanel <- function(x, y) {
  panel.superpose(x, y, groups = as.factor(Prob5$Pi), type
= "o", subscripts = as.factor(Prob5$Pi),lty = 5, col =
brewer.pal(9,"Blues")[9], lwd = 2)
  panel.grid(h=-1, v=-1)
}
plot5 = xyplot(Prob5$Spec~Prob5$ProbViableIVM|Prob1$Method,
panel = mypanel)

FinalPlot6 =
plot1+as.layer(plot2)+as.layer(plot3)+as.layer(plot4)+
as.layer(plot5)
```