Bioinformatics and Biological Data Science Project

Malek Sabri and Ryan Canfield

Professor Geist

Gene Expression Analysis in Bumblebees

**Background and Question**

Our team was very excited to start working on this project. We knew that it would entail some complex thinking, but the practicality of it was very intriguing to us. Investigating the genetic responses of organisms to climate change is crucial for understanding and mitigating its effects on ecosystems. Bumblebees are vital for agriculture and food security. Currently, the climate is changing at a rapid rate and in a way that is not very favorable to the bees. When they get exposed to very cold temperatures this causes a reaction in their bodies which leads them to go into a coma. This is a problem because as temperatures get colder in areas, the bees in those areas will go into a coma, potentially for too long causing them to die. This can cause lots of problems down the line in their ecosystems since they are vital to the pollination of many plants.

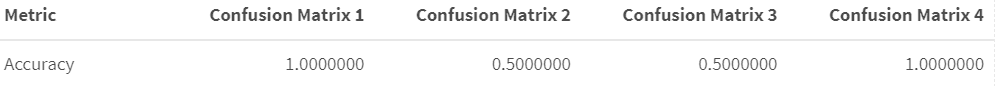
Our team sought to study the gene expression that changed during these chill comas, which would shed light on why the bee’s molecular structure responds this way and potentially find a way of making them more resilient to climate-induced stress.   
 The primary goal of this project is to analyze the chill coma gene expression data using machine learning techniques, and then reanalyze a smaller sample and compare the results with traditional methods. Identifying the key genes involved in cold tolerance will enhance our understanding of how the bumblebee copes with temperature stress. This knowledge is crucial for conservation efforts and maintaining the stability of pollination services, which are essential for agricultural productivity and biodiversity.

**Data Source**

The data for this analysis comes from a variety of researchers and universities, which examined gene expression in bumblebees following cold-induced chill coma. However, no one has really looked at the data in the way that we will. Bumblebees were kept inside and exposed to cold temperatures for different time intervals. RNA samples were collected at different intervals post-coma to measure gene expression changes. These samples were flash-frozen to preserve RNA integrity, allowing for accurate analysis of the bee’s physical responses to the cold. The original study provides a foundational dataset for re-evaluation using advanced machine learning methods.

**Analysis**

The model team alpha chose to classify the bumblebee RNA data, was a random forest model. In this section we will analyze the results of the model as well as talk through the code and what the team did to get the results. After loading in all the packages, functions, and data, we start off by doing an 80/20 training test split and creating a 10-fold cross validation for our random forest. Next, we trained our four different random forests all using different searches to find the best hyperparameters. The first one was filtered to remove variables with low variance, the second one was made with the best hyperparameters from a 10-fold CV. The third one was created with hyperparameters taken from a grid search, and the final fourth one was created with hyperparameters taken from an adjusted grid search. The CV hyperparameters were 130 for mtry, a *gini* split rule, and a node size of 1. The first grid search’s hyperparameters were 2138 for mtry, an *extratrees* split rule, and a node size of 1. The tuned grid searches hyperparameters were a 427 mtry, 50 trees used to vote on a decision, a *gini* split rule, and a minimum node size of 10. The four confusion matrices gave us accuracies of either 100% or 50%.



*Figure 1: shows accuracies of each confusion matrix.*

*CM1 = first RF, CM2 = 10 fold CV RF, CM3 = grid search RF, CM4 = Tuned grid search RF*

After looking at all the confusion matrices we can see that our model is either 100% accurate and can classify the data set perfectly, the model is overfitted and is just pushing every data point to one side or the other. The reason why we think we have two models with 50% accuracy is due to the model only having 1 node. This effectively turns the RF into a binary question where the data can either go into one group or the other.

When looking at other accuracy metrics we produced two visuals showing the inner and outer folds ROC. The AUC we get is around ~75% for both showing that the model does a good job distinguishing between false positives and false negatives but there is still room for improvement. Now after looking at all metrics for each model, we can choose the best one and extract any meaningful data. Of the ten extracted RNA sequences two matched the DESeq2 database. LOC10077885, and LOC10076272 were the two that matched.

**Conclusions**

In conclusion, the model that was built for the research question did a good job at analyzing the smaller data set of bumblebee data. A lot of challenges arose during this project due to unforeseen events. Our team also had limited knowledge on the subject, as well as building advanced classification models. With all that being said, we were able to successfully overcome these obstacles, and utilize machine learning to our advantage. With the focus of our research being on the re-evaluation of the data, we were still able to identify multiple significant gene expression changes when bumblebees were exposed to the cold. After reviewing the results and looking at the performance metrics of the model, it is still questionable if the model is fully ready to be used at a commercial level, or have our findings be published in a paper. Even though we still found significant genes we only have a good base and are on the right track, but we still need to do a lot more work before everything gets finalized. Overfitting was the biggest challenge we encountered, and we will talk about it further in the section below.

**Recommendations and Next Steps**

Our team decided to take option one out of the two which focused on building a random forest model instead of a support vector machine model. Instead of the full set of DESeq data we rebuilt the random forest model but for the smaller sample called the DESeq 30-minute data. This was more straightforward and more on the level of knowledge our team was comfortable with. To start, we would recommend refitting the data again but building a support vector machine and analyzing those results since the SVM model can better deal with the overfitting problem and does a good job at handling high dimensional data. Another recommendation our team would give is to force the tree of our random forest to be bigger than a tree size of one node. For some of the hyperparameter tuning, the code recommended using one node for a tree, but having one node turns the model into a binary question: does the data belong here to there which could explain why some of the model’s accuracies were 50%. We would also recommend ensemble learning methods to try to increase the accuracy of the models. Bagging and boosting would be two examples of this. To try other classifying models, our team recommends trying a deep learning model to see if it does a better job classifying and identifying significant data. Neural networks, or recurrent neural networks, would probably give a more comprehensive analysis and decrease overfitting since they are one of the most advanced models that we have. To conclude, working with a larger test sample would increase our confidence in the findings, ensuring that the results are more generalized and reliable. If there is more of a focus on exploring these alternative methods and expanding the dataset, then we are confident that there would be an improvement in the model and the classifications obtained from them would be excellent and meaningful.

\*\*\*Note\*\*\* The 15 questions are answered down below in the code appendix.

**Code Appendix**

Project 1: Bioinformatics of Gene Expression

Merrimack College DSE6630: Healthcare & Life Sciences Analytics

Ryan Canfield & Malek Sabri

2025-5-20

#### Questions were completed by look at the html document on github.

##### **Question 1**: Look up these rules and try to explain what they are doing for yourself.

The gini rule is a measure of randomness in a dataset’s target variable. The gini rule is calculated for a given node by considering the proportion of each class in the node. The extratrees rule creates many decision trees, but the sampling for each tree is random, without replacement. This creates a dataset for each tree with unique samples.

##### **Question 2**: Take a moment to go look at the function and the documentation I have written. Also try playing with the parameters a bit - how many more or fewer genes do you get when you lower or raise the filterCutoff, respectively?

The doSplits() function is one that is very useful and does a good job at hiding a lot of things behind the scenes. doSplits() can aide in doing a number of preprocessing steps and saving lots of time later on. Based on what I’m seeing from the code it appears that when the I lower the ‘filterCutoff’ there are more genes and when I raise it there are less genes included. This all makes sense and is pretty much intuitive once you have an understanding of what the functions is actually doing. There is then a larger or lower set of genes depending on what I do in the code. = Malek Sabri

##### **Question 3**: Do some research on **data leakage** and try to explain what that means in this context. Why do we care about data leakage?

In this context, data leakage means where information from outside the training dataset is inadvertently used to create a machine learning model. We care about data leakage because it can mislead the model’s performance, invalidate conclusions, and generalize issues.

##### **Question 4**: What combination of hyperparameters gives the highest accuracy in the **training model**? (Use plot above to answer.)

Initially the gini splitting rule has the highest accuracy until the randomly selected predictors hit around 6800 then extratrees has the highest accuracy.

##### **Question 5**: Can you see an impact of the pre-processing (to remove lowly expressed genes)?

I can not see an impact of the pre-processing.

##### **Question 6**: Can you see an impact of the 10-fold cross-validation?

I’d say that you can see an impact fromt the 10-fold CV. You can see this in figure 1. It shows the increased accuracy for extratees and decreased accuracy for gini. Then at around 6800 randomly selected predictors there is an accuracy overlap. The good thing about the 10-fold CV is that we are reducing overfitting along with it doing automated hyperparameter tuning and thus must be increasing accuracy = Malek Sabri

##### **Question 7**: How many different combinations are we going to search with this grid?

This grid will utilize the 5 values for mtry, the 2 values for splitrule, and the 4 values for min.node.size leading me to believe there could be 40 combinations = Malek Sabri

##### **Question 8**: What combination of hyperparameters gives the highest accuracy in the **training model**? (Use plot above to answer.)

When the min.node.size is 5, splitrule is extratees, and mtry is 23 I see the highest accuracy = Malek Sabri

##### **Question 9**: How well did hyperparameter tuning with the grid search perform?

Based on the confusion matrices there is not an improvement and thus the tuning was not useful when being utilized with the grid search = Malek Sabri

##### **Question 10**: Use the modelLookup function to help you figure out what hyperparameters are available for the rf vs. ranger methods for performing random forest algorithm with caret. What are the key differences you observce in the hypers you can tune?

The key differences in the tuning between rf and ranger is that ranger allows for more tuning capabilites. It appears that rf can tune mtry, but ranger can also tune min.node.size and splitrule = Malek Sabri

##### **Question 11**: What combination of hyperparameters gives the highest accuracy in the **training model**? (Use plot above to answer.) How does it compare to previous results?

When using 50 trees you get the highest accuracy when you use 10, or 20 nodes, and 1919 mtry or number features randomly sampled at each split. The accuracy is around 98~99% on when training the model. If you are using 200 trees you want to use either 5 or 20 nodes, and still the 1919 mtry. This will also give you an accuracy of 98~99%. This is similar to the results of the Grid Search on Random Forest since it also found that 5 nodes and 1919mtry were the best hyperparameters. –Ryan Canfield

##### **Question 12**: How well did hyperparameter tuning with the grid search perform?

The hyperparameter tuning with the grid search preformed exactly the same as with the other grid search both got eactly 57%. I think this is because the new tuned model has a node size of 1 which turns the descions tree into a binary answer which is why we would be getting the low accuracy. – Ryan Canfield

##### **Question 13**: Notice that the y-axis is now **log-Loss** and not **accuracy**. You may need to do some digging, but can you figure out why? Sources can be helpful if you’re not sure you’re on the right track!

While researching I found that when evaluating the performance of a model using log-loss instead of accuracy, it is because the model is now trained on probabilistic predictions. An example of models with probability outputs are logistic regression or random forests. Finally, the website says its just more a comprehensive evaluation of model performance when the outputs are probabilities not numbers. – Ryan Canfield

[https://datascience.stackexchange.com/questions/39825/log-loss-vs-accuracy-for-deciding-between-different-learning-rates](about:blank) [https://medium.com/@fzammito/whats-considered-a-good-log-loss-in-machine-learning-a529d400632d#](about:blank):~:text=Log%20Loss%20is%20similar%20to,reference%20values%20to%20compare%20it.

##### **Question 14**: The ROC and precision-recall curves are suggesting very high accuracy. Woohoo!!… Right?! What is the confusion matrix telling us? How do you know?

The confusion matrix is on the testing data set and contains 14 data points 7 of each points. 6 are correctly labeled a control point and 7 are correctly labeled cold point. the last point was miscorrectly labeled a cold point when it was actually labeled a control point. This type of error is a Type II error or False Negative. I know this the accuracy is accurate because it was taken from the output based off of the confusion matrix. 13/14 ~ 92%.

##### **Question 15**: Should we have any confidence in these results? Why or why not?

I think we should have confidence because the repeated appearance of LOC100747964 suggests it could have potential for being biologically significant. This is just and inference and not a fact, I think it is important to build other models and test their validation their answers so we can have a more reliable way to confirm it’s the gene’s relevance.

### Enter Phase 2 of this Project:

As we move forward with our benchmarking study, you have two choices. Choose your own adventure!

#### **Path 1** - Potentially easier and more satisfying.

Perform -fold & nested CV Random Forest, as well as hyperparameter tuning, on a different VST from the 30-minute experiment only.

If you decide to choose this adventure, you will use the VST data for the 30-min time point expression data only. You will also be given the DESeq results so you can use the findOverlappingGenes() function again to look at performance. Your task will be to take this vst, and work through all the same steps as in this first-half of the project. **Make sure to look at the important features at the end and comment on whether you have any confidence in your results!**

#### Team Alpha chose Path 1!

## Functions to make your life easier:

#### 1. doSplits()

Prepare the datasets, depending on algorithm, quickly and easily. This could also facilitate automated testing of filtering cutoffs or split-ratios by algorithm.

#### 2. findOverlappingGenes()

Function that compares the DESeq results at a given LFC with the list of important genes from an ML classifier. This could again be iterated over to automate collation of results for easier comparison.

#### 3. compareConfusion()

Function that compares the results of confusion matrices by printing them in a table side-by-side. You could also use this to graph the results, if desired.

# RANDOM FOREST

#### Now, reset filterCutoff = 5 and partition data for random forest tuning.

splits <- doSplits(vst = vsData, algorithm = "rf", splitRatio = 0.8, filterCutoff = 5)

## [1] "After filtering, the number of genes remaining in the dataset are: 8553"

train <- splits[[1]]  
test <- splits[[2]]

## Apply -fold Cross-Validation (CV) to Random Forest

#### 1. Leverage the trainControl() and train() functions from the caret package.

First, fit the RF **except now applying the filtering for low variance / expression!**

rfOOB <- randomForest::randomForest(  
 Treatment ~ .,   
 data = train)  
  
pred.test.rf <- predict(rfOOB, test, type = "response")  
confMat <- confusionMatrix(pred.test.rf, test$Treatment)

#### 2. Employ a -fold CV where .

(n\_repeats\*nresampling)+1 Set the control for and then run the RF:

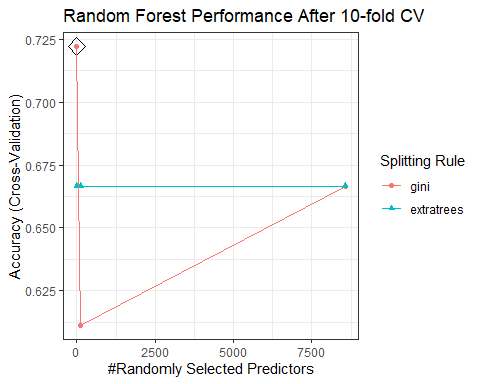
# Set the CV arguments  
kFoldCtrl <- trainControl(method = "cv", # for k-fold CV  
 number = 10) # k

#### Fitting the Random Forest model:

rfCV <- train(Treatment ~.,   
 data = train,  
 method = "ranger",  
 trControl = kFoldCtrl) ## added in the 10-fold CV

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,  
## : There were missing values in resampled performance measures.

##### Figure 1. Visualize the 10-fold CV Performance



#### 3. Extract best model.

#### A close-up of a white background Description automatically generated

#### 4. Fit the test data to the best tuned model from the 10-fold CV.

rfCV <- train(Treatment ~.,   
 data = train,  
 method = "ranger",  
 trControl = kFoldCtrl,   
 tuneGrid = rfCV$bestTune) # Add in the results of the CV and auto tuning

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,  
## : There were missing values in resampled performance measures.

pred.test.rf <- predict(rfCV, test, type = "raw") ## type is now 'raw'   
# Store the confusion matrix  
confMatCV <- confusionMatrix(pred.test.rf, test$Treatment)

#### 5. Compare the two confusion matrices.

### A screenshot of a computer Description automatically generated

### Grid Search

#### 1. Set the search grid.

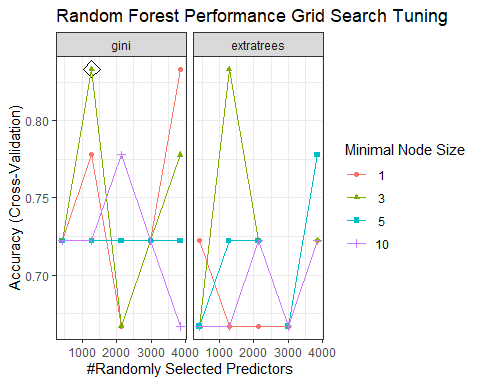
searchGrid <- expand.grid(  
 mtry = floor(ncol(train) \* c(.05, .15, .25, .35, .45)),  
 splitrule = c("gini", "extratrees"),  
 min.node.size = c(1, 3, 5, 10)   
)

#### 2. Run the RF, this time adding in the search grid.

rfTuned <- train(Treatment ~.,   
 data = train,  
 method = "ranger",  
 trControl = kFoldCtrl,   
 tuneGrid = searchGrid # Add in the search grid  
)

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,  
## : There were missing values in resampled performance measures.

##### Figure 2. Visualize the Grid Search Performance:



#### 3. Take a look at the hyperparameters of the best model:

rfTuned$bestTune %>%   
kable(  
 format = "html",  
 caption = "Table 3. Results of the Grid Search on Random Forest") %>%  
 kable\_styling(bootstrap\_options = c("hover", full\_width = F))

#### A white background with black lines Description automatically generated

#### 4. Refit the best model’s hyperparameters to the test data set:

rfTuned <- train(Treatment ~.,   
 data = train,  
 method = "ranger",  
 trControl = kFoldCtrl,   
 tuneGrid = rfTuned$bestTune) # Add in the results of the CV and auto tuning

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,  
## : There were missing values in resampled performance measures.

#### 5. Make the new predictions to assess accuracy:

pred.test.rf <- predict(rfTuned, test, type = "raw") ## type is now 'raw'   
# Store the confusion matrix  
confMatTuned <- confusionMatrix(pred.test.rf, test$Treatment)

#### 6. Compare the three confusion matrices:

compareConfusion(confusionList = list(confMat, confMatCV, confMatTuned)) %>%   
 kable(  
 format = "html",  
 caption = "Table 4. Comparing Accuracy - Random Forest") %>%  
 kable\_styling(bootstrap\_options = c("hover", full\_width = F))

A screenshot of a computer

Description automatically generated

### Grid Search Using a Customized caret Method

#### 1. Write a customized caret method for random forest using the ranger method:

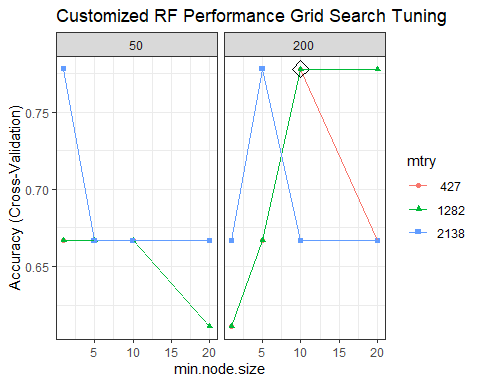
#### 2. Run the customized method:

customRF <- makeCustomMethod(paramList = c("mtry",   
 "num.trees",   
 "min.node.size",  
 "splitrule"),  
 methodName = "ranger")  
  
searchGrid <- expand.grid(.mtry = floor(ncol(train) \* c(.05, .15, .25)),  
 .num.trees = c(50, 200),   
 .min.node.size = c(1, 5, 10, 20),  
 .splitrule = "gini"  
 )  
  
customTuned <- train(y = train$Treatment, x = train,   
 method = customRF,   
 metric = "Accuracy",   
 tuneGrid = searchGrid,   
 trControl = kFoldCtrl)

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,  
## : There were missing values in resampled performance measures.

##### Figure 3. Visualize the Customized Grid Search Performance.

ggplot(customTuned, highlight = TRUE) +  
 ggtitle("Customized RF Performance Grid Search Tuning") +   
 theme\_bw()



#### 3. Take a look at the hyperparameters of the best model.

customTuned$bestTune %>%   
kable(  
 format = "html",  
 caption = "Table 4. Results of the Customized Grid Search on Random Forest") %>%  
 kable\_styling(bootstrap\_options = c("hover", full\_width = F))

#### A screenshot of a computer Description automatically generated4. Refit the best model’s hyperparameters to the test data set.

newTuned <- train(Treatment ~.,   
 data = train,  
 method = customRF,  
 trControl = kFoldCtrl,   
 tuneGrid = customTuned$bestTune)

#### 5. Make the new predictions to assess accuracy.

pred.test.rf <- predict(newTuned, test, type = "raw")   
# Store the confusion matrix  
confMatTuned2 <- confusionMatrix(pred.test.rf, test$Treatment)

#### 6. Compare the three confusion matrices.

Table 4. Comparing Accuracy - Random Forest

A screenshot of a computer

Description automatically generated

### Nested CV

#### 1. Set a search grid and train the model using the nestedcv.train() function.

searchGrid <- expand.grid(.mtry = floor(ncol(train) \* c(0.01, 0.05, 0.10)),  
 .min.node.size = c(1, 5, 10),  
 .splitrule = "gini"   
 )  
  
ncv <- nestcv.train(y = train$Treatment, x = train,  
 method = 'ranger',  
 tuneGrid = searchGrid,   
 savePredictions = "final")

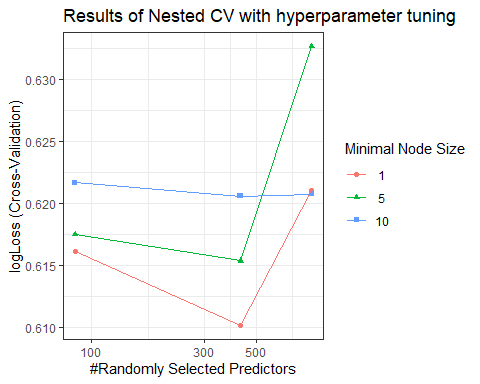
## 1 predictor(s) have var=0

## Fitting final model using CV on whole data

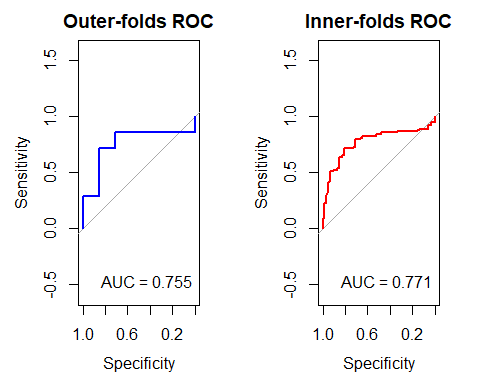
## Performing 10-fold outer CV, using 1 core

## Duration: 16.8797 mins

##### Figure 4. Visualize the Nested CV Performace on RF while also using a Grid Search of hyperparameters.



#### 2. Plot the Receiver-operator curves (ROC) and precision-recall curves for both the inner- and outer- loops.



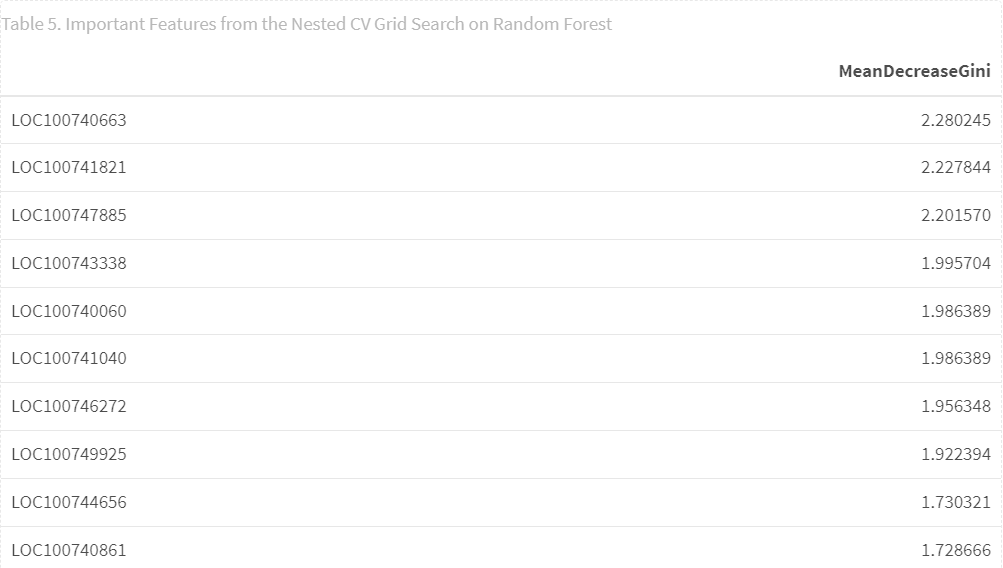
#### 5. Get that final accuracy by fitting to the test data.

preds <- predict(ncv,   
 newdata = test)  
confusionMatrix(preds, test$Treatment)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Control Cold  
## Control 1 0  
## Cold 0 1  
##   
## Accuracy : 1   
## 95% CI : (0.1581, 1)  
## No Information Rate : 0.5   
## P-Value [Acc > NIR] : 0.25   
##   
## Kappa : 1   
##   
## Mcnemar's Test P-Value : NA   
##   
## Sensitivity : 1.0   
## Specificity : 1.0   
## Pos Pred Value : 1.0   
## Neg Pred Value : 1.0   
## Prevalence : 0.5   
## Detection Rate : 0.5   
## Detection Prevalence : 0.5   
## Balanced Accuracy : 1.0   
##   
## 'Positive' Class : Control   
##

#### 4. Grab the important features.

## Selecting by MeanDecreaseGini



#### 5. Compare our gene results to the ones out of DESeq2.

Let’s filter the nested CV randfom forest importance:A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated