Week 6 Discussion: Logistic, Multinomial and Polynomial Regression

Ken Hydock

2022-08-07

# Load the Required Libraries

I have a few extra libraries that were not in the demo. Additionally, I didn’t encounter anything that required dplyr. I commented with what function is utilized by each library.

# Import, Convert, and Inspect Data

Let’s get to work! As always, don’t forget to update your working directory. We can see we’re working with a .data file. I wonder how that’s going to look? I’ll also go ahead and use the set.seed function for repeatable results with randomization functions and I created a variable for our threshold cutoff value in case we want to easily adjust it throughout the document.

setwd("~/MSDS/MSDS660/wk6")  
dt <- read.csv("breast-cancer-wisconsin.data")  
dt <- as.data.table(dt)  
  
# Set the seed to the meaning of life and everything.  
set.seed(42)  
  
cutoff = 0.5   
  
str(dt)

## Classes 'data.table' and 'data.frame': 698 obs. of 11 variables:  
## $ X1000025: int 1002945 1015425 1016277 1017023 1017122 1018099 1018561 1033078 1033078 1035283 ...  
## $ X5 : int 5 3 6 4 8 1 2 2 4 1 ...  
## $ X1 : int 4 1 8 1 10 1 1 1 2 1 ...  
## $ X1.1 : int 4 1 8 1 10 1 2 1 1 1 ...  
## $ X1.2 : int 5 1 1 3 8 1 1 1 1 1 ...  
## $ X2 : int 7 2 3 2 7 2 2 2 2 1 ...  
## $ X1.3 : chr "10" "2" "4" "1" ...  
## $ X3 : int 3 3 3 3 9 3 3 1 2 3 ...  
## $ X1.4 : int 2 1 7 1 7 1 1 1 1 1 ...  
## $ X1.5 : int 1 1 1 1 1 1 1 5 1 1 ...  
## $ X2.1 : int 2 2 2 2 4 2 2 2 2 2 ...  
## - attr(\*, ".internal.selfref")=<externalptr>

So that .data file didn’t do us any favors - it doesn’t have a header! Thankfully, it generated something akin to headers so it didn’t mess with the first row. Time to get this data ready for use.

# Data Preparation

First up, we have a conveniently prescribed set of column names from our assignment. For the curious one, I did find an original reference to the data set which corroborates these names as well as explains a few other things…

<https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Original)>

# Name all the variables  
colnames(dt) <- c("id", "clump\_thickness", "cell\_size", "cell\_shape" ,"marginal\_adhesion" , "se\_cell\_size",   
 "bare\_nucleoli" ,"bland\_chromatin", "normal\_nucleoli", "mitoses", "Class")  
str(dt)

## Classes 'data.table' and 'data.frame': 698 obs. of 11 variables:  
## $ id : int 1002945 1015425 1016277 1017023 1017122 1018099 1018561 1033078 1033078 1035283 ...  
## $ clump\_thickness : int 5 3 6 4 8 1 2 2 4 1 ...  
## $ cell\_size : int 4 1 8 1 10 1 1 1 2 1 ...  
## $ cell\_shape : int 4 1 8 1 10 1 2 1 1 1 ...  
## $ marginal\_adhesion: int 5 1 1 3 8 1 1 1 1 1 ...  
## $ se\_cell\_size : int 7 2 3 2 7 2 2 2 2 1 ...  
## $ bare\_nucleoli : chr "10" "2" "4" "1" ...  
## $ bland\_chromatin : int 3 3 3 3 9 3 3 1 2 3 ...  
## $ normal\_nucleoli : int 2 1 7 1 7 1 1 1 1 1 ...  
## $ mitoses : int 1 1 1 1 1 1 1 5 1 1 ...  
## $ Class : int 2 2 2 2 4 2 2 2 2 2 ...  
## - attr(\*, ".internal.selfref")=<externalptr>

Looking better… We can see a bunch of interger columns and one column of character encoded numbers. That’s something we need to fix, as well as dropping the extraneous index column id and factorizing our Class as our dependent variable. From the reference linked above, we can see that 2 is Benign and 4 is Malignant. That’s a useful bit of information we needed. Lastly, if running these adjustment individually, we’d note that doing the coercion as.integer function, we create a few NA records as whatever was in the bare\_nucleoli column may not have been **all** character encoded integers.

# Remove id column  
dt$id <- NULL  
  
# Factor Class column  
dt$Class <- factor(dt$Class, labels = c('B', 'M'))  
  
# Change bare\_nucleoli column to integer  
dt$bare\_nucleoli <- as.integer(dt$bare\_nucleoli)

## Warning: NAs introduced by coercion

dt <- dt[complete.cases(dt),]  
  
str(dt)

## Classes 'data.table' and 'data.frame': 682 obs. of 10 variables:  
## $ clump\_thickness : int 5 3 6 4 8 1 2 2 4 1 ...  
## $ cell\_size : int 4 1 8 1 10 1 1 1 2 1 ...  
## $ cell\_shape : int 4 1 8 1 10 1 2 1 1 1 ...  
## $ marginal\_adhesion: int 5 1 1 3 8 1 1 1 1 1 ...  
## $ se\_cell\_size : int 7 2 3 2 7 2 2 2 2 1 ...  
## $ bare\_nucleoli : int 10 2 4 1 10 10 1 1 1 1 ...  
## $ bland\_chromatin : int 3 3 3 3 9 3 3 1 2 3 ...  
## $ normal\_nucleoli : int 2 1 7 1 7 1 1 1 1 1 ...  
## $ mitoses : int 1 1 1 1 1 1 1 5 1 1 ...  
## $ Class : Factor w/ 2 levels "B","M": 1 1 1 1 2 1 1 1 1 1 ...  
## - attr(\*, ".internal.selfref")=<externalptr>

Looking ready to go. I do want to mention a misgiving I had with the as.factor function this go round. While we did only have two encodings (2 and 4), the function does not specify how it works. It works in the logical sense simplifying 2 to 1 as B and 4 to 2 as M, but there would have been some piece of mind if I could explicitly state that to the function so I *know* what to expect of it.

# Train & Test Sets

Now that our data is ready, we can split it up into training and test sets. Why would we do this? Well in the Machine Learning (ML) sense, we’re building a model to test against data. So realistically, we want it to learn from one data set and perform on another - train and test!

I saw a bunch of ways to achieve this, the most common other way involved dplyr, but I found the demo way to be a bit more elegant and saved on including an additionally library for our purposes. The 0.8 below signifies that we’re creating a sample of 80% of our data set. Then we make our training set out of that 80% and a test set of the remaining 20%. We want our training data to be more robust than the testing data to generate a reliable model. This also ensures that we’re not using the same data for both sets.

dt.sample <- sample.split(dt$Class, SplitRatio = 0.8)  
dt.train <- subset(dt, dt.sample == TRUE)  
dt.test <- subset(dt, dt.sample == FALSE)

# Create a Binomial Logistic Regression Model

I got super-duper confused between the two demos on this part. I went a proverbial rabbit hole. Let me save you some time - the way the lm and glm are done in the different demos are the same. Doing the lm way allows you to specify attributes using the + and \* operators as we’ve come to know. Additionally, you can use the poly function with lm to specify your polynomial. This is useful with a simple plot that you can estimate the right polynomial to your goal. Using glm with the family = "binomial"(link="logit") flags does exactly what our course content was talking about with much less thinking on our end.

dt.model <- glm(Class ~ ., data = dt.train, family = "binomial"(link="logit"))  
summary(dt.model)

##   
## Call:  
## glm(formula = Class ~ ., family = binomial(link = "logit"), data = dt.train)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -3.8396 -0.1037 -0.0445 0.0086 2.1898   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -11.28064 1.58982 -7.096 1.29e-12 \*\*\*  
## clump\_thickness 0.67179 0.18490 3.633 0.000280 \*\*\*  
## cell\_size -0.06026 0.23629 -0.255 0.798724   
## cell\_shape 0.18627 0.25993 0.717 0.473607   
## marginal\_adhesion 0.58459 0.17561 3.329 0.000872 \*\*\*  
## se\_cell\_size 0.14349 0.18739 0.766 0.443827   
## bare\_nucleoli 0.49968 0.12472 4.007 6.16e-05 \*\*\*  
## bland\_chromatin 0.42480 0.18599 2.284 0.022370 \*   
## normal\_nucleoli 0.16080 0.12858 1.251 0.211086   
## mitoses 0.75908 0.31918 2.378 0.017397 \*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 706.026 on 544 degrees of freedom  
## Residual deviance: 73.771 on 535 degrees of freedom  
## AIC: 93.771  
##   
## Number of Fisher Scoring iterations: 8

Looking at our summary, we can se we have a handful of insignificant variables. You may also note that this summary is missing a lot of things like R2. You can look into it, but the way we evaluate this binomial models and their non-linearity doesn’t care about normal distribution or residuals in the same way. The deviance values are the residuals as the err from the model in a non-linear fashion, so they’re the indicator of fit.

We’ve got a couple assumptions we can check to to see how well our model is performing, such as VIF and AIC to make improvements.

# Check for Collinearity

It’s been a couple weeks since we’ve looked at VIF, but a value of 1-5 is okay, but anything higher is problematic.

vif(dt.model)

## clump\_thickness cell\_size cell\_shape marginal\_adhesion   
## 1.480924 3.716308 3.626329 1.341399   
## se\_cell\_size bare\_nucleoli bland\_chromatin normal\_nucleoli   
## 1.406589 1.156212 1.270735 1.327831   
## mitoses   
## 1.134210

I don’t see any problems, but we do have a couple that are higher than the others.

# StepAIC to Clean Model

It’s also been awhile since we’ve used stepAIC. We’ll be doing this to remove unnecessary variables to simplify and increase the statistical power of our model. The curse of dimensionality never goes away, so having an effective model with as few dimensions as possible translates to more efficiency without sacrificing accuracy.

stepAIC(dt.model, dirrection = 'both')

## Start: AIC=93.77  
## Class ~ clump\_thickness + cell\_size + cell\_shape + marginal\_adhesion +   
## se\_cell\_size + bare\_nucleoli + bland\_chromatin + normal\_nucleoli +   
## mitoses  
##   
## Df Deviance AIC  
## - cell\_size 1 73.834 91.834  
## - cell\_shape 1 74.252 92.252  
## - se\_cell\_size 1 74.356 92.356  
## - normal\_nucleoli 1 75.396 93.396  
## <none> 73.771 93.771  
## - mitoses 1 78.943 96.943  
## - bland\_chromatin 1 79.500 97.500  
## - marginal\_adhesion 1 87.286 105.286  
## - clump\_thickness 1 92.851 110.851  
## - bare\_nucleoli 1 95.432 113.432  
##   
## Step: AIC=91.83  
## Class ~ clump\_thickness + cell\_shape + marginal\_adhesion + se\_cell\_size +   
## bare\_nucleoli + bland\_chromatin + normal\_nucleoli + mitoses  
##   
## Df Deviance AIC  
## - se\_cell\_size 1 74.374 90.374  
## - cell\_shape 1 74.430 90.430  
## - normal\_nucleoli 1 75.398 91.398  
## <none> 73.834 91.834  
## - mitoses 1 78.943 94.943  
## - bland\_chromatin 1 79.565 95.565  
## - marginal\_adhesion 1 87.483 103.483  
## - clump\_thickness 1 92.854 108.854  
## - bare\_nucleoli 1 95.550 111.550  
##   
## Step: AIC=90.37  
## Class ~ clump\_thickness + cell\_shape + marginal\_adhesion + bare\_nucleoli +   
## bland\_chromatin + normal\_nucleoli + mitoses  
##   
## Df Deviance AIC  
## - cell\_shape 1 75.286 89.286  
## - normal\_nucleoli 1 76.369 90.369  
## <none> 74.374 90.374  
## - mitoses 1 79.880 93.880  
## - bland\_chromatin 1 81.204 95.204  
## - marginal\_adhesion 1 90.404 104.404  
## - clump\_thickness 1 94.002 108.002  
## - bare\_nucleoli 1 96.433 110.433  
##   
## Step: AIC=89.29  
## Class ~ clump\_thickness + marginal\_adhesion + bare\_nucleoli +   
## bland\_chromatin + normal\_nucleoli + mitoses  
##   
## Df Deviance AIC  
## <none> 75.286 89.286  
## - normal\_nucleoli 1 79.071 91.071  
## - mitoses 1 81.301 93.301  
## - bland\_chromatin 1 85.226 97.226  
## - marginal\_adhesion 1 95.836 107.836  
## - bare\_nucleoli 1 105.699 117.699  
## - clump\_thickness 1 110.077 122.077

##   
## Call: glm(formula = Class ~ clump\_thickness + marginal\_adhesion + bare\_nucleoli +   
## bland\_chromatin + normal\_nucleoli + mitoses, family = binomial(link = "logit"),   
## data = dt.train)  
##   
## Coefficients:  
## (Intercept) clump\_thickness marginal\_adhesion bare\_nucleoli   
## -11.5597 0.7669 0.6568 0.5325   
## bland\_chromatin normal\_nucleoli mitoses   
## 0.4898 0.2193 0.7898   
##   
## Degrees of Freedom: 544 Total (i.e. Null); 538 Residual  
## Null Deviance: 706   
## Residual Deviance: 75.29 AIC: 89.29

So we can see it *step* through a few iterations to generate the lowest AIC possible. This in turn gives us a new model to try which will have fewer variables but still keeping a majority of the accuracy and significance of the model, if not improving it outright.

We can run it again just to make sure we have the best fit for our model!

dt.model2 <- glm(formula = Class ~ clump\_thickness + marginal\_adhesion + bare\_nucleoli +   
 bland\_chromatin + normal\_nucleoli + mitoses, family = binomial(link = "logit"),   
 data = dt.train)  
  
stepAIC(dt.model2, dirrection = 'both')

## Start: AIC=89.29  
## Class ~ clump\_thickness + marginal\_adhesion + bare\_nucleoli +   
## bland\_chromatin + normal\_nucleoli + mitoses  
##   
## Df Deviance AIC  
## <none> 75.286 89.286  
## - normal\_nucleoli 1 79.071 91.071  
## - mitoses 1 81.301 93.301  
## - bland\_chromatin 1 85.226 97.226  
## - marginal\_adhesion 1 95.836 107.836  
## - bare\_nucleoli 1 105.699 117.699  
## - clump\_thickness 1 110.077 122.077

##   
## Call: glm(formula = Class ~ clump\_thickness + marginal\_adhesion + bare\_nucleoli +   
## bland\_chromatin + normal\_nucleoli + mitoses, family = binomial(link = "logit"),   
## data = dt.train)  
##   
## Coefficients:  
## (Intercept) clump\_thickness marginal\_adhesion bare\_nucleoli   
## -11.5597 0.7669 0.6568 0.5325   
## bland\_chromatin normal\_nucleoli mitoses   
## 0.4898 0.2193 0.7898   
##   
## Degrees of Freedom: 544 Total (i.e. Null); 538 Residual  
## Null Deviance: 706   
## Residual Deviance: 75.29 AIC: 89.29

summary(dt.model2)

##   
## Call:  
## glm(formula = Class ~ clump\_thickness + marginal\_adhesion + bare\_nucleoli +   
## bland\_chromatin + normal\_nucleoli + mitoses, family = binomial(link = "logit"),   
## data = dt.train)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -4.0865 -0.0991 -0.0401 0.0064 2.0069   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -11.5597 1.5623 -7.399 1.37e-13 \*\*\*  
## clump\_thickness 0.7669 0.1695 4.524 6.08e-06 \*\*\*  
## marginal\_adhesion 0.6568 0.1679 3.912 9.16e-05 \*\*\*  
## bare\_nucleoli 0.5325 0.1183 4.501 6.76e-06 \*\*\*  
## bland\_chromatin 0.4898 0.1696 2.888 0.00387 \*\*   
## normal\_nucleoli 0.2193 0.1171 1.873 0.06113 .   
## mitoses 0.7898 0.3135 2.519 0.01176 \*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 706.026 on 544 degrees of freedom  
## Residual deviance: 75.286 on 538 degrees of freedom  
## AIC: 89.286  
##   
## Number of Fisher Scoring iterations: 8

With an AIC of 89.286, it looks like we have the best model out of our variables. A quick summary shows that all of our variables are significant with the near exception of normal\_nucleoli. However, that variable may have a relationship with another that *is* significant. Probably why that one survived the cut.

# Predictions!

Now we have an optimized model, we can run it through its paces. We’re really just seeing if the model is accurate or not with known factors. Based on the trend of the model, how well does that predict on the training set?

## Training Prediction

# Predictions from training set  
dt.train.p <- predict(dt.model2, type = 'response', dt.train)  
# Factorizing the results to B or M based on the cutoff of 50% probability  
dt.train.p.f <- factor(dt.train.p >= cutoff, labels = c('B', 'M'))  
  
# Making a confusion matrix from our dependent variable an our results  
dt.train.cm <- confusionMatrix(dt.train$Class, dt.train.p.f)  
dt.train.cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction B M  
## B 348 6  
## M 7 184  
##   
## Accuracy : 0.9761   
## 95% CI : (0.9596, 0.9872)  
## No Information Rate : 0.6514   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.9475   
##   
## Mcnemar's Test P-Value : 1   
##   
## Sensitivity : 0.9803   
## Specificity : 0.9684   
## Pos Pred Value : 0.9831   
## Neg Pred Value : 0.9634   
## Prevalence : 0.6514   
## Detection Rate : 0.6385   
## Detection Prevalence : 0.6495   
## Balanced Accuracy : 0.9744   
##   
## 'Positive' Class : B   
##

There’s a lot of data to unpack here. The important bits are a very respectable 97% accuracy with 7 false negative and 6 false positives. An accuracy this high is at risk of being an overfit and only testing it will prove otherwise.

## Testing Prediction

So if the model is overfit to the training data (a poor random sampling), we’ll see here. Otherwise, the binomial logistic regression model is a great fit for this data!

dt.test.p <- predict(dt.model2, type = 'response', dt.test)  
  
dt.test.p.f <- factor(dt.test.p >= cutoff, labels = c('B', 'M'))  
  
dt.test.cm <- confusionMatrix(dt.test$Class, dt.test.p.f )  
dt.test.cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction B M  
## B 85 4  
## M 2 46  
##   
## Accuracy : 0.9562   
## 95% CI : (0.9071, 0.9838)  
## No Information Rate : 0.635   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.9047   
##   
## Mcnemar's Test P-Value : 0.6831   
##   
## Sensitivity : 0.9770   
## Specificity : 0.9200   
## Pos Pred Value : 0.9551   
## Neg Pred Value : 0.9583   
## Prevalence : 0.6350   
## Detection Rate : 0.6204   
## Detection Prevalence : 0.6496   
## Balanced Accuracy : 0.9485   
##   
## 'Positive' Class : B   
##

Again we have a 95% accuracy, which is quite high. The only thing I noticed here is that there is a poorer performance for predicting B, while predicting M remained relatively the same from training and testing. This could just be due to the small sample sizes.

## Predicting with ROC

I don’t think we’ve messed with Receiver Operating Characteristic (ROC) so far in this class. Super helpful name too - doesn’t tell me much about itself! The ROC curve is a test of sensitivity and selectivity, basically the true positive and true negative rate and the closer it is to 1, the more accurate your model is. For instance a 100% rate for both would be an “L” rotated 90*, or a right angle from 90* to 180\*. A plot will make more sense of this…

### Train ROC

So we create our roc using our dependent variable of the applicable data set and the predictions. The 2nd bit below gives us values of the specificity and selection.

dt.train.roc <- roc(dt.train$Class, dt.train.p)

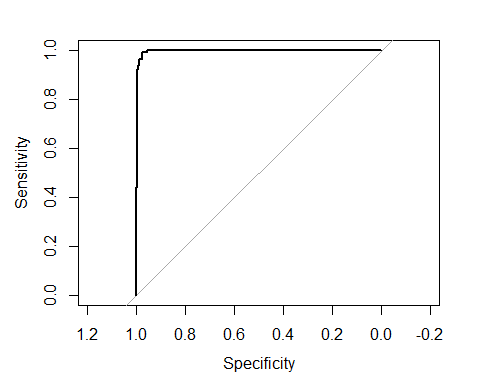
## Setting levels: control = B, case = M

## Setting direction: controls < cases

dt.train.roc

##   
## Call:  
## roc.default(response = dt.train$Class, predictor = dt.train.p)  
##   
## Data: dt.train.p in 354 controls (dt.train$Class B) < 191 cases (dt.train$Class M).  
## Area under the curve: 0.9968

plot(dt.train.roc)



dt.train.roc.c <- coords(roc=dt.train.roc, x = 'best', best.method = 'closest.topleft')  
dt.train.roc.c

## threshold specificity sensitivity  
## 1 0.219914 0.9745763 0.9895288

So, that looks kind of like what I was describing with a rounded corner to show it’s not quite perfect with our few false predictions.

### Test ROC

Now do test!

dt.test.roc <- roc(dt.test$Class, dt.test.p)

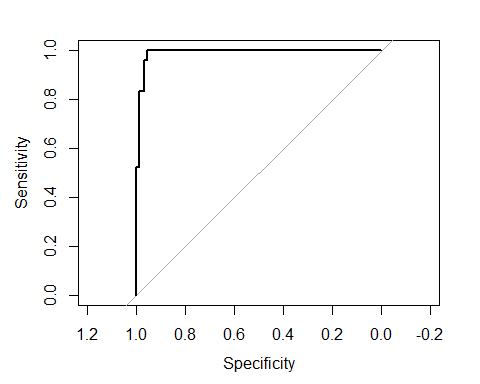
## Setting levels: control = B, case = M

## Setting direction: controls < cases

dt.test.roc

##   
## Call:  
## roc.default(response = dt.test$Class, predictor = dt.test.p)  
##   
## Data: dt.test.p in 89 controls (dt.test$Class B) < 48 cases (dt.test$Class M).  
## Area under the curve: 0.9904

plot(dt.test.roc)



dt.test.roc.c <- coords(roc=dt.test.roc, x = 'best', best.method = 'closest.topleft')  
dt.test.roc.c

## threshold specificity sensitivity  
## 1 0.2443108 0.9550562 1

Ohh, neat. We have a perfect sensitivity of 1, meaning that our model got all the true positives right. specificity isn’t quite there yet, meaning we have a few false negatives… In this case, with B being the positive, a false negative means a misdiagonsis of a Benign cancer actually being Malignant. So not good. We could rearrange all this, but I’m okay with the connotation that Benign is Positive.

## ROC Predictions

Why did we do that? Performing ROC gave us s a new cutoff value that is more accurate than 0.5 that we chose. This is the threshold that is from the roc coords function. I think this directly translates that anything below that threshold is negative, or M.

### Train ROC Predictions

With our seemingly more accurate threshold, let’s put it to task.

dt.train.roc.p <- factor(dt.train.p >= as.numeric(dt.train.roc.c[1]), labels = c('B', 'M'))  
  
dt.train.roc.cm <- confusionMatrix(dt.train$Class, dt.train.roc.p)  
dt.train.roc.cm

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction B M  
## B 345 9  
## M 2 189  
##   
## Accuracy : 0.9798   
## 95% CI : (0.9642, 0.9899)  
## No Information Rate : 0.6367   
## P-Value [Acc > NIR] : < 2e-16   
##   
## Kappa : 0.956   
##   
## Mcnemar's Test P-Value : 0.07044   
##   
## Sensitivity : 0.9942   
## Specificity : 0.9545   
## Pos Pred Value : 0.9746   
## Neg Pred Value : 0.9895   
## Prevalence : 0.6367   
## Detection Rate : 0.6330   
## Detection Prevalence : 0.6495   
## Balanced Accuracy : 0.9744   
##   
## 'Positive' Class : B   
##

We have a 97.9% accuracy. This is only 0.3% better than the 50% threshold. Still, it is an improvement.

### Test ROC Predictions

Now with the test set… The ROC results were very good for this.

dt.test.roc.p <- factor(dt.test.p >= as.numeric(dt.test.roc.c[1]), labels = c('B', 'M'))  
dt.test.roc.cm <- confusionMatrix(dt.test$Class, dt.test.roc.p)  
dt.test.roc.cm

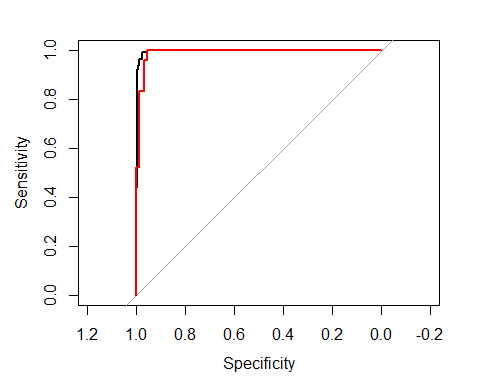
## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction B M  
## B 85 4  
## M 0 48  
##   
## Accuracy : 0.9708   
## 95% CI : (0.9269, 0.992)  
## No Information Rate : 0.6204   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.9371   
##   
## Mcnemar's Test P-Value : 0.1336   
##   
## Sensitivity : 1.0000   
## Specificity : 0.9231   
## Pos Pred Value : 0.9551   
## Neg Pred Value : 1.0000   
## Prevalence : 0.6204   
## Detection Rate : 0.6204   
## Detection Prevalence : 0.6496   
## Balanced Accuracy : 0.9615   
##   
## 'Positive' Class : B   
##

Better than the previous test predictions, just like train was better than the first.

### ROC Plots

So the beauty of ROC is in plotting them for a quick comparison of the most efficient models.

plot(dt.train.roc)  
plot(dt.test.roc, add=TRUE, col='red')

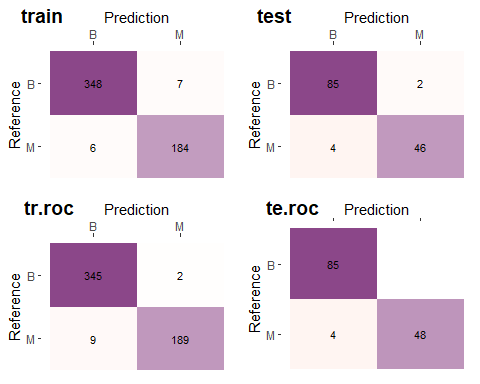
 We can handily see that the black line, or our training ROC is the more accurate one.

# Confusion Matrices

So I went a little crazy - all of our confusion matrix summaries are in line above, with pretty solid conclusions after them. I wanted to try to figure out how to plot this mess. I found a solution and it is anything but elegant, but at least it’s descriptive.

library(ggplot2)   
library(ggpubr)   
library(likert)  
  
dt.train.cm.data <- as.data.frame(dt.train.cm$table) # extract the confusion matrix values as data.frame  
dt.train.cm.stats <- data.frame(dt.train.cm$overall) # confusion matrix statistics as data.frame  
dt.train.cm.stats$dt.train.cm.overall <- round(dt.train.cm.stats$dt.train.cm.overall,2) # round the values  
dt.train.cm.data$diag <- dt.train.cm.data$Prediction == dt.train.cm.data$Reference # Get the Diagonal  
dt.train.cm.data$ndiag <- dt.train.cm.data$Prediction != dt.train.cm.data$Reference # Off Diagonal   
dt.train.cm.data[dt.train.cm.data == 0] <- NA # Replace 0 with NA for white tiles  
dt.train.cm.data$Reference <- reverse.levels(dt.train.cm.data$Reference) # diagonal starts at top left  
dt.train.cm.data$ref\_freq <- dt.train.cm.data$Freq \* ifelse(is.na(dt.train.cm.data$diag),-1,1)  
  
dt.train.cm.plot <- ggplot(data = dt.train.cm.data, aes(x = Prediction , y = Reference, fill = Freq))+  
 scale\_x\_discrete(position = "top") +  
 geom\_tile( data = dt.train.cm.data,aes(fill = ref\_freq)) +  
 scale\_fill\_gradient2(guide = FALSE ,low="red3",high="orchid4", midpoint = 0,na.value = 'white') +  
 geom\_text(aes(label = Freq), color = 'black', size = 3)+  
 theme\_bw() +  
 theme(panel.grid.major = element\_blank(), panel.grid.minor = element\_blank(),  
 legend.position = "none",  
 panel.border = element\_blank(),  
 plot.background = element\_blank(),  
 axis.line = element\_blank(),  
 )  
  
dt.test.cm.data <- as.data.frame(dt.test.cm$table) # extract the confusion matrix values as data.frame  
dt.test.cm.stats <- data.frame(dt.test.cm$overall) # confusion matrix statistics as data.frame  
dt.test.cm.stats$dt.test.cm.overall <- round(dt.test.cm.stats$dt.test.cm.overall,2) # round the values  
dt.test.cm.data$diag <- dt.test.cm.data$Prediction == dt.test.cm.data$Reference # Get the Diagonal  
dt.test.cm.data$ndiag <- dt.test.cm.data$Prediction != dt.test.cm.data$Reference # Off Diagonal   
dt.test.cm.data[dt.test.cm.data == 0] <- NA # Replace 0 with NA for white tiles  
dt.test.cm.data$Reference <- reverse.levels(dt.test.cm.data$Reference) # diagonal starts at top left  
dt.test.cm.data$ref\_freq <- dt.test.cm.data$Freq \* ifelse(is.na(dt.test.cm.data$diag),-1,1)  
  
dt.test.cm.plot <- ggplot(data = dt.test.cm.data, aes(x = Prediction , y = Reference, fill = Freq))+  
 scale\_x\_discrete(position = "top") +  
 geom\_tile( data = dt.test.cm.data,aes(fill = ref\_freq)) +  
 scale\_fill\_gradient2(guide = FALSE ,low="red3",high="orchid4", midpoint = 0,na.value = 'white') +  
 geom\_text(aes(label = Freq), color = 'black', size = 3)+  
 theme\_bw() +  
 theme(panel.grid.major = element\_blank(), panel.grid.minor = element\_blank(),  
 legend.position = "none",  
 panel.border = element\_blank(),  
 plot.background = element\_blank(),  
 axis.line = element\_blank(),  
 )  
  
dt.train.roc.cm.data <- as.data.frame(dt.train.roc.cm$table) # extract the confusion matrix values as data.frame  
dt.train.roc.cm.stats <- data.frame(dt.train.roc.cm$overall) # confusion matrix statistics as data.frame  
dt.train.roc.cm.stats$dt.train.roc.cm.overall <- round(dt.train.roc.cm.stats$dt.train.roc.cm.overall,2) # round the values  
dt.train.roc.cm.data$diag <- dt.train.roc.cm.data$Prediction == dt.train.roc.cm.data$Reference # Get the Diagonal  
dt.train.roc.cm.data$ndiag <- dt.train.roc.cm.data$Prediction != dt.train.roc.cm.data$Reference # Off Diagonal   
dt.train.roc.cm.data[dt.train.roc.cm.data == 0] <- NA # Replace 0 with NA for white tiles  
dt.train.roc.cm.data$Reference <- reverse.levels(dt.train.roc.cm.data$Reference) # diagonal starts at top left  
dt.train.roc.cm.data$ref\_freq <- dt.train.roc.cm.data$Freq \* ifelse(is.na(dt.train.roc.cm.data$diag),-1,1)  
  
dt.train.roc.cm.plot <- ggplot(data = dt.train.roc.cm.data, aes(x = Prediction , y = Reference, fill = Freq))+  
 scale\_x\_discrete(position = "top") +  
 geom\_tile( data = dt.train.roc.cm.data,aes(fill = ref\_freq)) +  
 scale\_fill\_gradient2(guide = FALSE ,low="red3",high="orchid4", midpoint = 0,na.value = 'white') +  
 geom\_text(aes(label = Freq), color = 'black', size = 3)+  
 theme\_bw() +  
 theme(panel.grid.major = element\_blank(), panel.grid.minor = element\_blank(),  
 legend.position = "none",  
 panel.border = element\_blank(),  
 plot.background = element\_blank(),  
 axis.line = element\_blank(),  
 )  
  
dt.test.roc.cm.data <- as.data.frame(dt.test.roc.cm$table) # extract the confusion matrix values as data.frame  
dt.test.roc.cm.stats <- data.frame(dt.test.roc.cm$overall) # confusion matrix statistics as data.frame  
dt.test.roc.cm.stats$dt.test.roc.cm.overall <- round(dt.test.roc.cm.stats$dt.test.roc.cm.overall,2) # round the values  
dt.test.roc.cm.data$diag <- dt.test.roc.cm.data$Prediction == dt.test.roc.cm.data$Reference # Get the Diagonal  
dt.test.roc.cm.data$ndiag <- dt.test.roc.cm.data$Prediction != dt.test.roc.cm.data$Reference # Off Diagonal   
dt.test.roc.cm.data[dt.test.roc.cm.data == 0] <- NA # Replace 0 with NA for white tiles  
dt.test.roc.cm.data$Reference <- reverse.levels(dt.test.roc.cm.data$Reference) # diagonal starts at top left  
dt.test.roc.cm.data$ref\_freq <- dt.test.roc.cm.data$Freq \* ifelse(is.na(dt.test.roc.cm.data$diag),-1,1)  
  
dt.test.roc.cm.plot <- ggplot(data = dt.test.roc.cm.data, aes(x = Prediction , y = Reference, fill = Freq))+  
 scale\_x\_discrete(position = "top") +  
 geom\_tile( data = dt.test.roc.cm.data,aes(fill = ref\_freq)) +  
 scale\_fill\_gradient2(guide = FALSE ,low="red3",high="orchid4", midpoint = 0,na.value = 'white') +  
 geom\_text(aes(label = Freq), color = 'black', size = 3)+  
 theme\_bw() +  
 theme(panel.grid.major = element\_blank(), panel.grid.minor = element\_blank(),  
 legend.position = "none",  
 panel.border = element\_blank(),  
 plot.background = element\_blank(),  
 axis.line = element\_blank(),  
 )  
ggarrange(dt.train.cm.plot, dt.test.cm.plot, dt.train.roc.cm.plot, dt.test.roc.cm.plot + rremove("x.text"),   
 labels = c("train", "test", "tr.roc", "te.roc"),  
 ncol = 2, nrow = 2)

## Warning: Removed 1 rows containing missing values (geom\_text).

 The way this came out is True Positive:False Positive on top and False Negative: True Negative on bottom. I wanted to make it more descriptive, but I ran out of brain power.

We can see that both the models with the ROC suggested threshold are more accurate overall and in the case of cancer, that’s most definitely a good thing.

# Conclusions

That was a lot. We made our binomial regression model, trimmed it down a bit with stepAIC to make it as efficient as possible while still utilizing the most variables possible. Turns out, that resulted in a highly accurate model with a 50% certainty threshold. After running ROC on our predictions, we found more accurate thresholds which ended up increasing accuracy on both our training and test data sets.

Overall, I’d say this data set was specifically selected for it’s extremely efficient logistic regression model with minimal effort. There’s a lot more that we could have gone into to fine-tune a logistic regression model, but it was absolutely not necessary to produce a very good model for our purposes.