Peter Becich

Math 32, 14D

December 11th, 2011

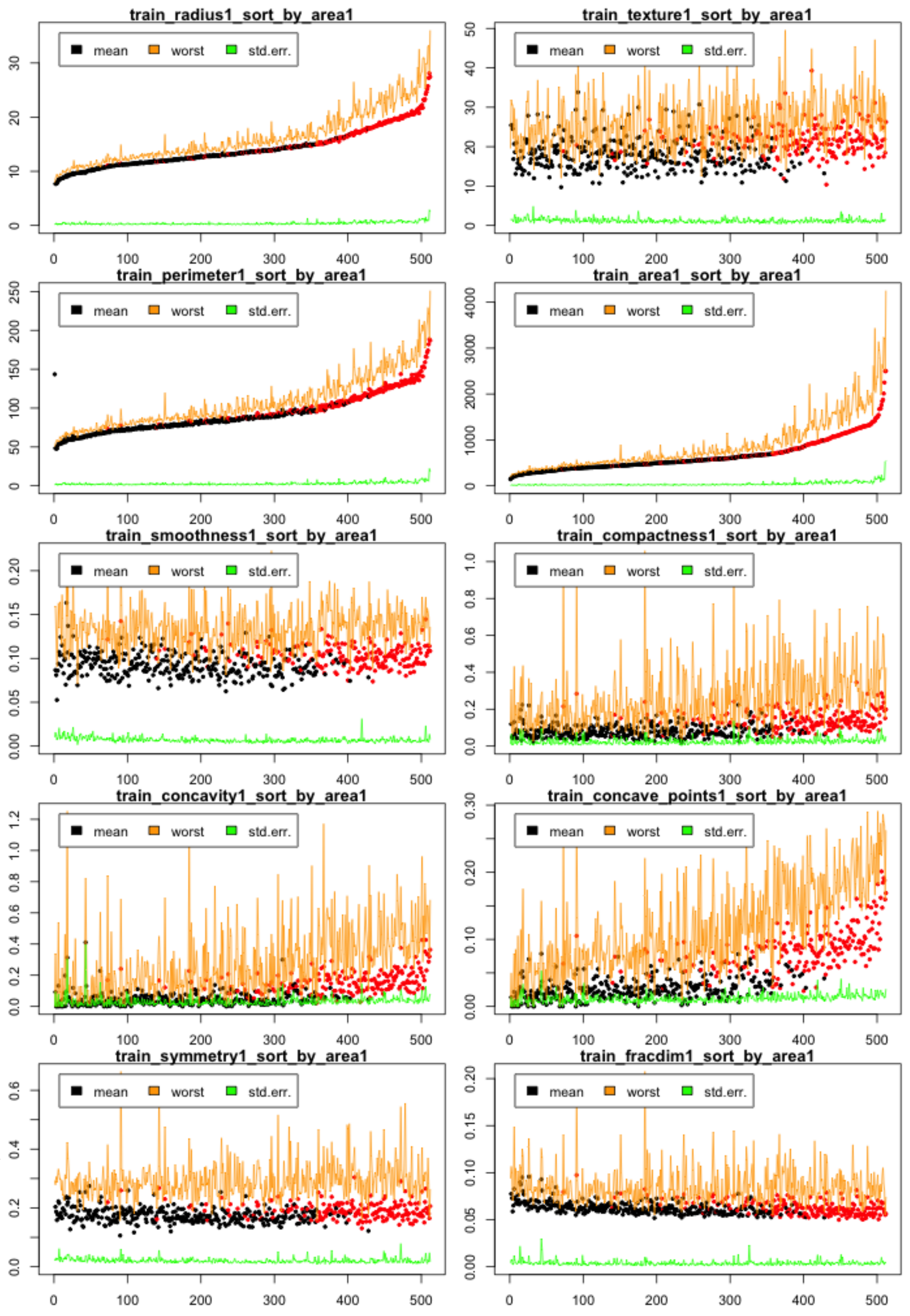
Wisconsin Breast Cancer Data

Brief introduction

Ten types of predictor variables are provided, each with three pieces of data: the average, the standard error, and the worst case. 30 total predictor data are provided for 569 patients with breast cancer. I've attempted to make a binary classification model that will predict the outcome of either *malignant* or *benign* with the provided predictors.

Note: List of written R functions on second-to-last page

Exploratory data analysis



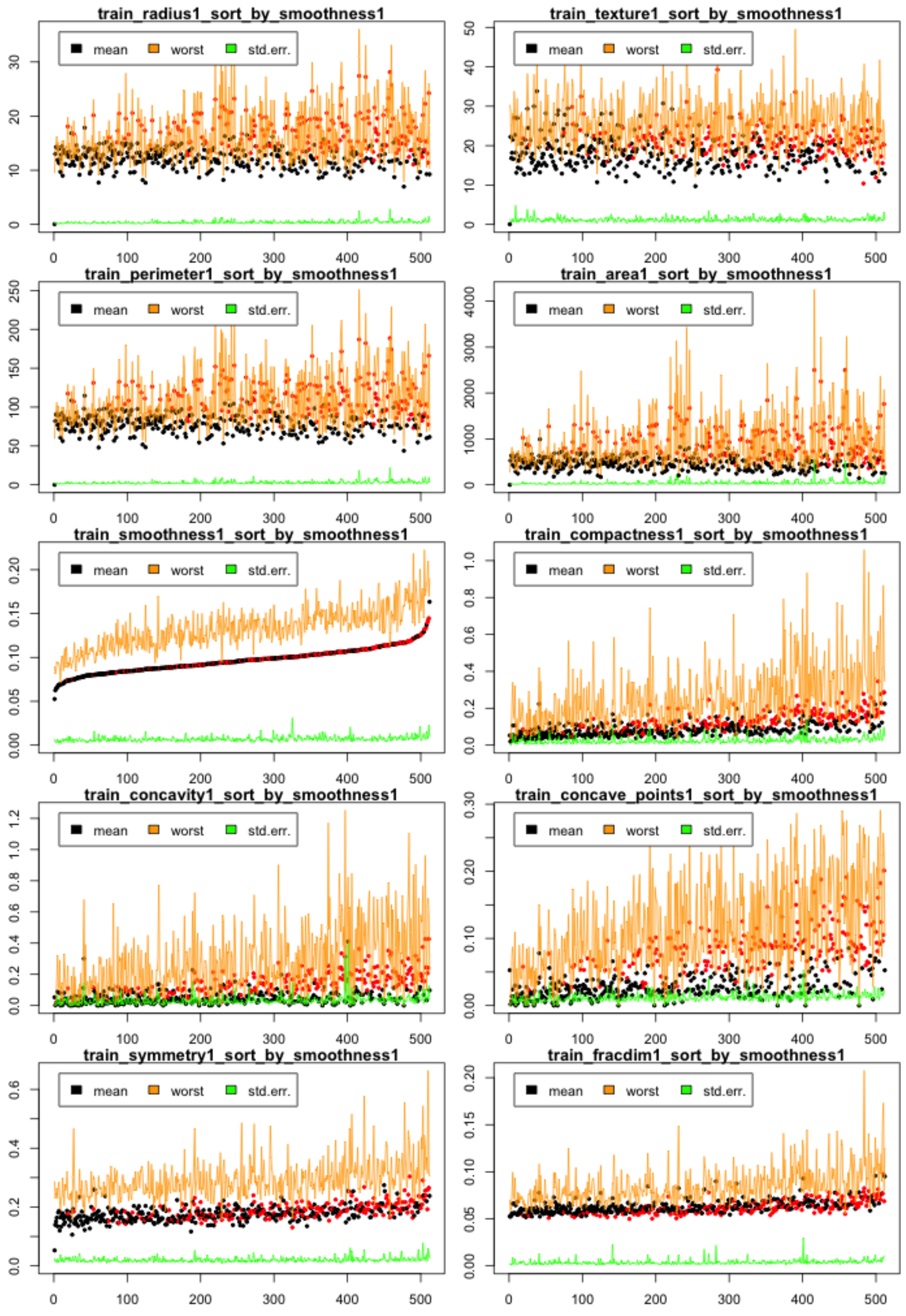
I found it useful to visualize the relationships between predictor variables with plots. These ten plots are sorted by the average area per cell per patient, *area1*. The plots may lead to the same conclusions that covariance or correlation calculations would have yielded, but seeing the information has helped me to make sense of it. Ultimately, this visual EDA was not used to decide on a model, which was done by a brute force function. This work could be used to improve the efficiency of the brute force function with knowledge of which predictors should not be rigorously tested.

Predictors from a tumor that was ultimately benign are colored black, while those that were ultimately malignant are red. The separation of the black and red in sorted average cell area infers a strong correlation between area and malignancy. Note that red and black points have the same x coordinate in every plot; only the y coordinate is affected by the predictor being plotted.

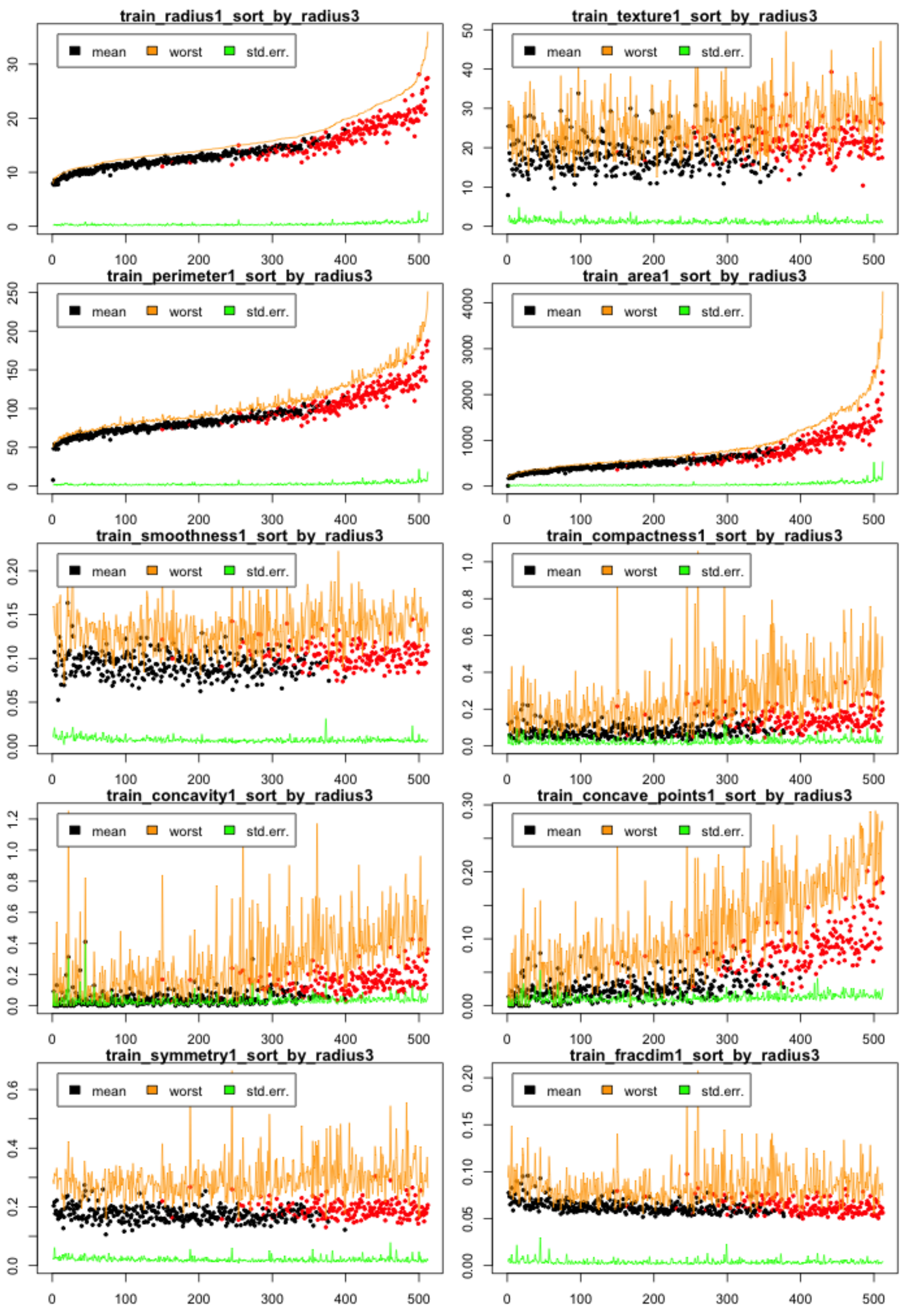
After being ordered from least to greatest, the plot of *area1* has a strictly positive slope. Logically, the average cell radius and perimeter increase with average area. One of the less obvious discoveries from this analysis shows that the average number of concave points per cell per patient tends to increase as average area per cell per patient increases. Conversely, average symmetry does not visibly correlate with increasing average area.

Worst lines of all predictors, in orange, sorted by an average predictor, area1, do not necessarily yield any useful information. It would be impossible for the worst line to be below the average line of any plot.

These plots are sorted by the ordered average smoothness per cell, per patient. Ordering by this predictor leads some other predictors to increase some, albeit while diverging. The information is not useful though because there isn't a strong separation of the malignant data points from the benign; average fractal dimension may tend to increase as average smoothness increases, but these increased data points are just as often malignant as benign, for example. This also means that the worst lines do not yield any useful information.



These plots are sorted by worst radius. The worst line in train_radius1_sort_by_radius3 has a strictly positive slope. Sorting by worst radius *does* visibly separate the malignant from the benign, but worst radius closely correlates with average radius, so the information taken from these plots is not exactly new.



Code structure

Ten-fold cross validation

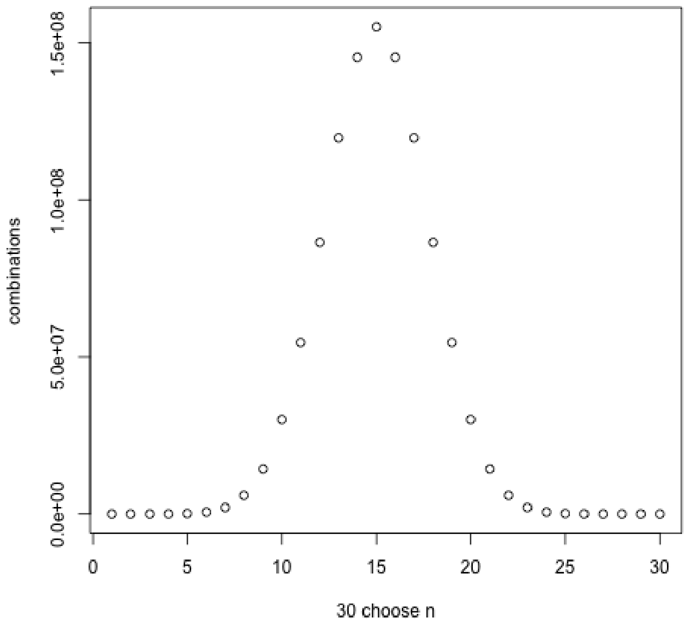
xtest and xtrain are created from a shuffled copy of the cancer data, ten times in total during the ten-fold cross validation. This is very similar to discussion 11. The model in mod.R is fit to xtrain during each cross validation. The predictions are then made for both xtrain and xtest, and the confusion matrices calculated. The confusion matrices are tallied up, and the percentage of data points in either *True Positive* or *True Negative,* out of the sum of each matrix, is calculated for both matrices; these are the percentages of accuracy.

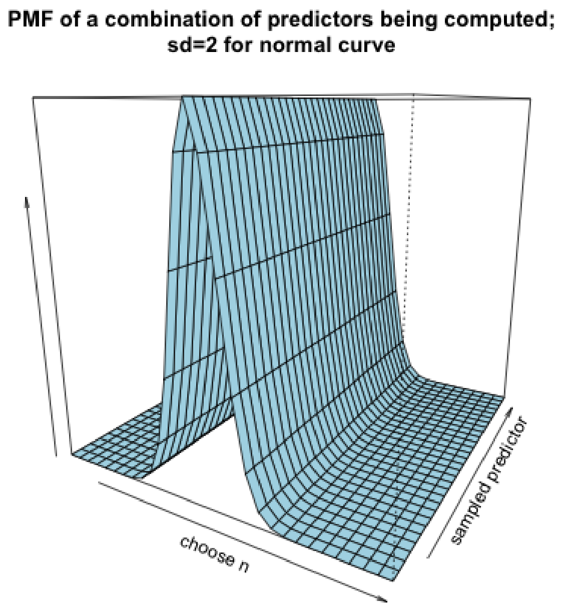
In *main.R*, ten-fold cross validation is used to check randomly generated models, a process described below. *single_mod_main.R* is used to run individual models through ten-fold cross validation.

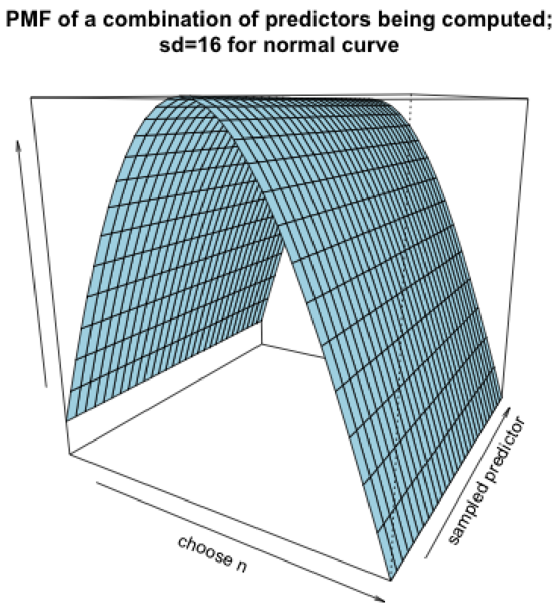
Brute force model selection

One direction I pursued in this project was a brute force search for the most accurate model. Accuracy was judged by the results of ten-fold cross validation. Performing a ten-fold cross validation on every combination of 30 choose 1:30 predictors is clearly computationally prohibitive, so I created my own sampling function to use available computing time as efficiently as possible.

There are over 1.07 x 10^9 possible combinations of the 30 predictors provided.







The volume under both surfaces is 1

I came up with a way of sampling from the set of possible combinations without calculating all the combinations beforehand, which is also computationally prohibitive. To randomly choose one set of predictors, first the number of predictors to use (n) is sampled from a normal distribution. If n is not in [1,30], n is sampled again. n predictors are then sampled, without replacement, from a uniform distribution (domain [1:30]).

The proportion of computing time devoted to the impossible task of calculating all combinations near the median of the domain of n, [1,30], can be controlled by adjusting the standard deviation of the normal distribution that n is sampled from ("How much computing time to throw into the bottomless pit?"). A normal distribution with a standard deviation of 16 devotes more time to combinations of n = 1, and n=30, for example.

The mean of the normal distribution is 15, which was an error caught too late to fix. The correct mean to evenly sample from the domain [1,30] would have been 15.5. Redundancy in chosen models was not checked for, as I didn't not have enough time to implement a check that was faster than ten-fold cross validation.

The brute force function takes available computing time as an input. Using the computing time of one ten-fold cross validation, a realistic number of random models to generate and cross validate is estimated.

I ran this function on one node of the Evolution cluster, and on my own computer. Each instance of the program sampled n from a normal distribution of a different standard deviation. These standard deviations were [2:20], at increments of 2. Large standard deviations essentially flattened the normal curve.

The main routine of my program uses the brute force function to generate these models, first. The brute force function does not *fit* the model. The vast majority of the computing time is spent cycling through every model generated, and performing 10-fold cross validation. The model, confusion matrices, and percentages of accuracy are stored in a large matrix. This matrix is written to a CSV file. If any model that underwent 10-fold cross validation had 0 false negatives in either the test or train confusion matrices, these optimal models are written to a separate CSV file.

Chosen model

The best model I was able to find with the brute force function was a model of *every* predictor except average perimeter and area, and worst area.

```
glm(form, family=binomial(link=logit),data=cbind(diag.in,predict.in))
```

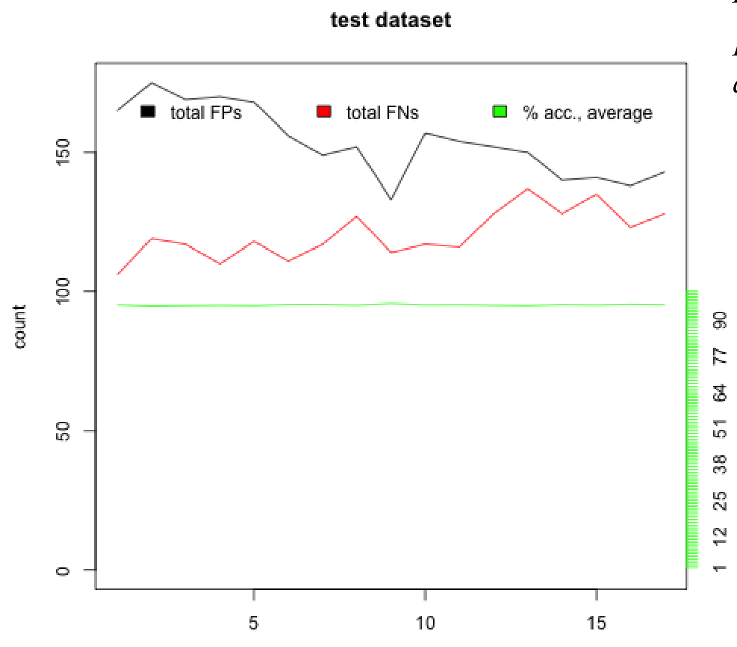
```
form ← radius1 + texture1 + smoothness1 + compactness1 + concavity1 +
concave_points1 + symmetry1 + fracdim1 + radius2 + texture2 + perimeter2 + area2 +
smoothness2 + compactness2 + concavity2 + concave_points2 + symmetry2 + fracdim2 +
radius3 + texture3 + perimeter3 + smoothness3 + compactness3 + concavity3 +
concave_points3 + symmetry3 + fracdim3
```

This seems counter to the visual EDA that appears so in favor of the removed predictors. This model was chosen from the CSV file of 0 FN models, and had the highest test dataset percentage accuracy of all models with 0 FNs.

This model gave 0 FN for the ten-fold cross validation that it underwent in the main routine. This did not guarantee that it would give 0 FN when undergoing another ten-fold cross validation. Running the model through the *single.mod.main* function in *single_mod_main.R* produced these results:

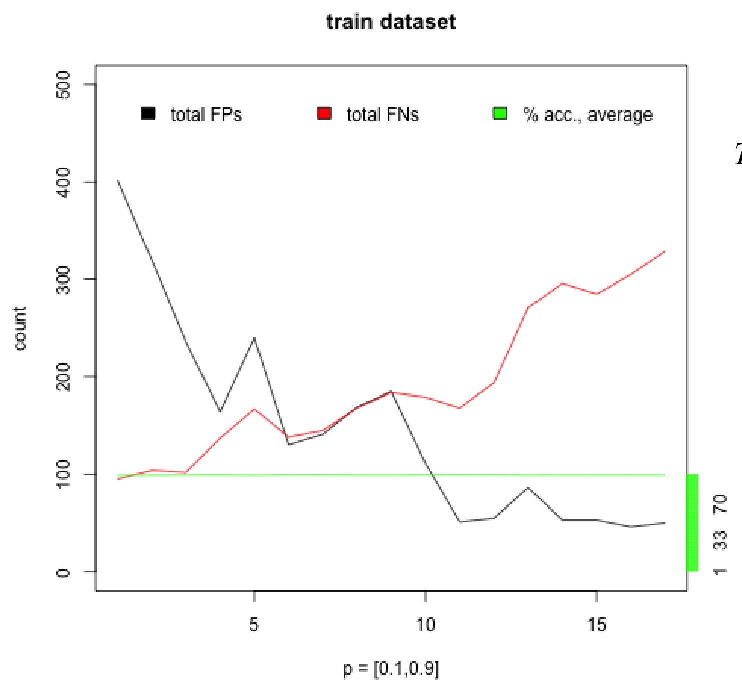
```
[1] "xtrain.confuse.cumulative"

     [,1] [,2]

[1,] 1883   64

[2,]   27 3156

[1] "98.23 % correct"

[1] "xtest.confuse.cumulative"

     [,1] [,2]

[1,]  207    6

[2,]    3  344

[1] "98.39 % correct"

[1] "sum of xtest.confuse.cumulative:  560   |  sum of xtrain.confuse.cumulative:
5130"
```

The next stage of the project was to adjust the threshold level *p*, to try and get the false negative count for both the test and train datasets down to zero, for every cross validation. A rigorous testing function designed to show the effect of threshold adjustments, *single.mod.main.repetitive*, runs the model through 10 ten-fold cross validations for every value of p, for a range of p. For each of the 10 ten-fold CVs, the entire dataset is reshuffled, re-randomizing test and train boundaries. Percentages of accuracy are averaged, and FNs are summed, for every value of p. Ideally, this would give away a *p* value where FNs are minimized.

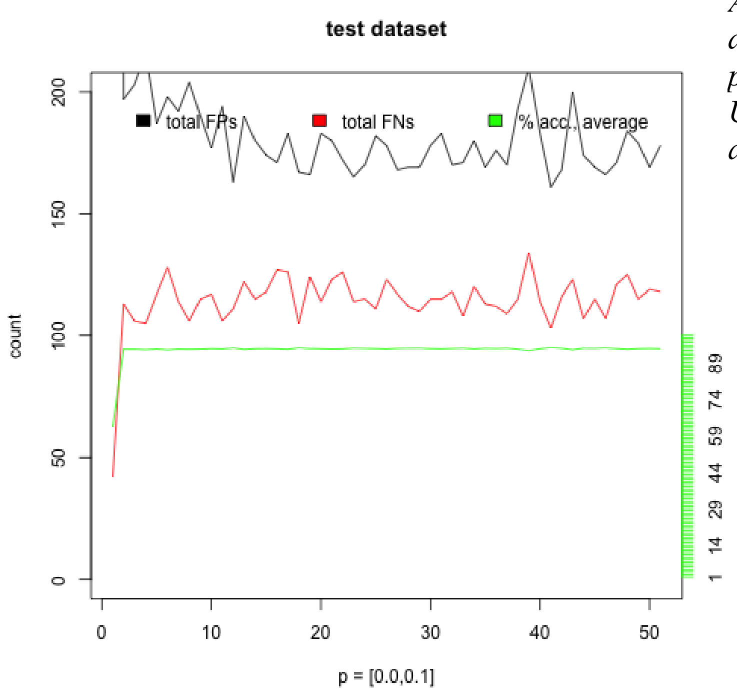


FN count tends to decrease as p decreases;

FP count tends to increase, keeping % accuracy roughly the same.

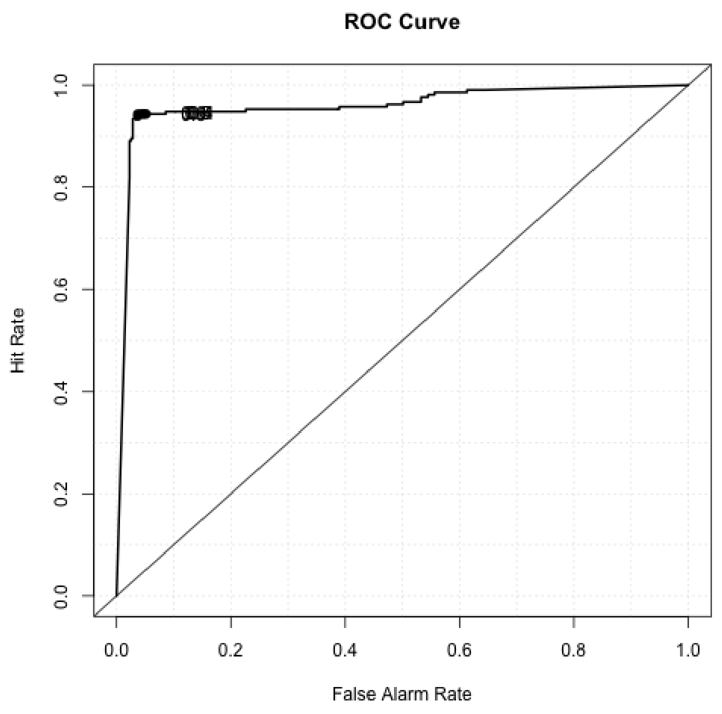


The same trend



As p approaches 0, average % accuracy drops drastically. When p = 0, all tumors are predicted to be positive (malignant). Understandably, FNs drop to zero when there are no negatives.

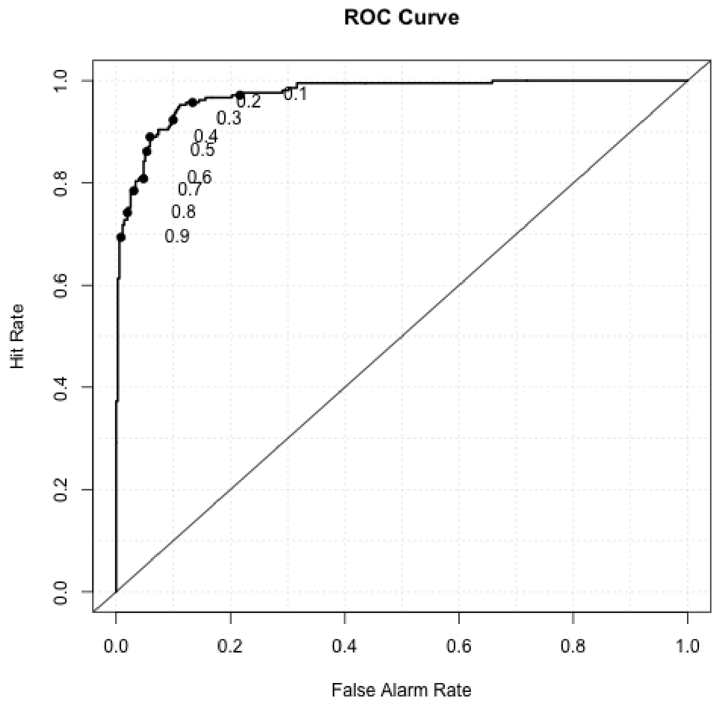
```
> out<-single.mod.main(best=best,verbose=1,thresh.alt=1,p=0.01,confuse.data.out=0)
> roc.plot(x=out[,2],pred=out[,1])
...
[1] "xtrain.confuse.cumulative"
     [,1] [,2]
[1,] 1898  106
[2,]   11 3115
[1] "97.72 % correct"
[1] "xtest.confuse.cumulative"
     [,1] [,2]
[1,]  199   20
[2,]   12  329
[1] "94.29 % correct"
```



With a p value of 0.01, FNs are still unavoidable with this model.

For comparison, the simple model of average area, texture, and perimeter does not work nearly as well, at least with the same p value of 0.01.

```
> out<-single.mod.main(best=c(1,2,3),verbose=1,thresh.alt=1,p=0.01,0)
[1] "xtrain.confuse.cumulative"
     [,1] [,2]
[1,] 1903 2009
[2,]    8 1210
[1] "60.68 % correct"
[1] "xtest.confuse.cumulative"
     [,1] [,2]
[1,]  208  219
[2,]    1  132
[1] "60.71 % correct"
```



The ROC plot cannot be used as a *response variable* to adjustments in the thresholding function. It will not change, aside from minor changes by reshuffling the dataset with each set of ten-fold cross validation.

With a p value of 0.4, this alternative model has higher average % accuracies, but FNs and FPs remain unpredictable. As the ROC plot does not take thresholding into consideration, there is no need to plot the ROC twice.

```
[1] "xtrain.confuse.cumulative"
     [,1] [,2]
[1,] 1703  201
[2,]  207 3019
[1] "92.05 % correct"
[1] "xtest.confuse.cumulative"
     [,1] [,2]
[1,]  185   23
[2,]   25  327

[1] "91.43 % correct"
```

Conclusion, and next steps

To take this project further, many models tested by *main.R* could go under the same adjustments and scrutiny that the chosen model went through; adjusting the threshold p value, namely. The chosen model was simply at the top of the list of models generated by *main.R*, with the time I had available.

Considering how much time the threshold adjusting routine takes, about three minutes, this is impractical to do for *every* model that goes through ten-fold cross validation in the *main.R* routine.

The brute force function could be made more efficient by taking prior knowledge into consideration when sampling predictors. I had no knowledge of *how many predictors* would most likely yield the best model. The brute force function, using a normal curve of a small standard deviation, distributes computing time roughly evenly to all possible combinations of predictors that could be generated; 30 choose 15 predictors receives more computing time that 30 choose 2.

Ultimately, the chosen model had 27 predictors.

This model responded well to adjustments in the threshold p value. However, the expected value of the sum of false negatives from a ten-fold cross validation is still greater than one. Occasionally, the model will go through ten-fold cross validation without producing a single false negative. This is not the norm, but the exception.

Resources:

Machine Learning for Cancer Diagnosis and Prognosis
http://pages.cs.wisc.edu/~olvi/uwmp/cancer.html#diag
Some background on the computer vision program used to collect the data.

Wisconsin Diagnostic Breast Cancer Source Information
http://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.names

R 3D plotting tutorial
http://www.stat.ucl.ac.be/ISpersonnel/lecoutre/stats/fichiers/_gallery.pdf

Fawcett: An introduction to ROC analysis

Functions:

- func.R
    - source("sort_plots.R") beforehand
    - plot.all.sorted.by.all(arg)
        - plot every predictor sorted by either mean, standard error, or worst of every predictor
        - arg is 1, 2, or 3 for mean, standard error, or worst
        - one plot per PNG
        - generates 100 plots
    - plot.all.sorted.by.all.window(arg)
        - plot.all.sorted.by.all with 10 plots per PNG
    - various functions for shuffling and sorting data
    - color.finder(vector, index)
        - checks if a data point is malignant or benign, and returns red or black
    - sample.patient.match(a,b)
        - checks for duplicate patients in test and train datasets
    - narrow(predicted)
        - takes predicted values and narrows them to domain [0,1]
        - 1/(1+exp(-predicted))
    - thresh(predicted)
        - threshold function with p=0.5
    - thresh.alt(predicted, p)
        - custom threshold function
- main.R
    - main routine for brute force function

    - source(“main.R”)
    - adjust hours to run and normal distribution std. dev. at top
- mod.R
    - buildMod(diag.in.predict.in)
        - for creating a model yourself
    - buildMod.by.index(diag.in, predict.in, predict.indices)
        - for building a model with predictors of particular indices
        - for brute force function
    - brute.force.search(diag.in, predict.in, maxdays, choosesd)
        - for main.R
        - returns a matrix of predictors
- single_mod_main.R
    - single.mod.main.R(best, verbose, thresh.alt, p, confuse.data.out)
        - for putting a single model through ten-fold cross validation
        - best – predictor indices
        - verbose – print info for every cross-validation
        - thresh.alt – use custom p value; 1 or 0
        - p – if thresh.alt is 1, custom p value; must be entered, regardless
        - confuse.data.out
            - 1 to return confusion matrix data, for single.mod.main.repetitive
            - 0 to return matrix for creating ROC plot
    - single.mod.main.repetitive(best, runs)
        - shows how well a model improves with different p threshold values
        - adjust pstart, pfinish, and pinc in file
        - best – predictor indices
        - runs – how many times to run a ten-fold cross validation on every value of p
- confuse.R
- sort_plots.R
    - the name is misleading; it's just for generating random xtest and xtrain datasets