Biostats UH-3150: Logistic Regression

Contents

Review

Before beginning this exercise, please review the lecture notes on logistic regression. In particular, you will want to re-read the sections on **Generalized linear models**, **GLM families**, and **ROC and AUC**.

The Wisconsin Breast Cancer Dataset

The following data was obtained from a study where 3D images were taken of a breast mass. The features are measurements of the mass. We will try to create a logistic regression model to predict the **diagnosis*'** (M=malignant and B=benign). For a more detail understanding of the values see the link below:

https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

The file is provided in as a csv file called *cancer_data.csv*. Read it in and call it **cancer_data**.

Note: for very large datasets, you may find it useful to try out the `glimpse()` function in addition to `head()` to take a quick look at your data.

```
# load the dataset and inspect it
cancer_data = read.csv("cancer_data.csv")
str(cancer_data)
```

```
## 'data.frame':    569 obs. of  32 variables:
##  $ id                     : int  842302 842517 84300903 84348301 84358402 843786 844359 84458202 844
##  $ diagnosis              : Factor w/ 2 levels "B","M": 2 2 2 2 2 2 2 2 2 2 ...
##  $ radius_mean            : num  18 20.6 19.7 11.4 20.3 ...
##  $ texture_mean           : num  10.4 17.8 21.2 20.4 14.3 ...
```

```
##  $ perimeter_mean          : num  122.8 132.9 130 77.6 135.1 ...
##  $ area_mean               : num  1001 1326 1203 386 1297 ...
##  $ smoothness_mean         : num  0.1184 0.0847 0.1096 0.1425 0.1003 ...
##  $ compactness_mean        : num  0.2776 0.0786 0.1599 0.2839 0.1328 ...
##  $ concavity_mean          : num  0.3001 0.0869 0.1974 0.2414 0.198 ...
##  $ concave.points_mean     : num  0.1471 0.0702 0.1279 0.1052 0.1043 ...
##  $ symmetry_mean           : num  0.242 0.181 0.207 0.26 0.181 ...
##  $ fractal_dimension_mean  : num  0.0787 0.0567 0.06 0.0974 0.0588 ...
##  $ radius_se               : num  1.095 0.543 0.746 0.496 0.757 ...
##  $ texture_se              : num  0.905 0.734 0.787 1.156 0.781 ...
##  $ perimeter_se            : num  8.59 3.4 4.58 3.44 5.44 ...
##  $ area_se                 : num  153.4 74.1 94 27.2 94.4 ...
##  $ smoothness_se           : num  0.0064 0.00522 0.00615 0.00911 0.01149 ...
##  $ compactness_se          : num  0.049 0.0131 0.0401 0.0746 0.0246 ...
##  $ concavity_se            : num  0.0537 0.0186 0.0383 0.0566 0.0569 ...
##  $ concave.points_se       : num  0.0159 0.0134 0.0206 0.0187 0.0188 ...
##  $ symmetry_se             : num  0.03 0.0139 0.0225 0.0596 0.0176 ...
##  $ fractal_dimension_se    : num  0.00619 0.00353 0.00457 0.00921 0.00511 ...
##  $ radius_worst            : num  25.4 25 23.6 14.9 22.5 ...
##  $ texture_worst           : num  17.3 23.4 25.5 26.5 16.7 ...
##  $ perimeter_worst         : num  184.6 158.8 152.5 98.9 152.2 ...
##  $ area_worst              : num  2019 1956 1709 568 1575 ...
##  $ smoothness_worst        : num  0.162 0.124 0.144 0.21 0.137 ...
##  $ compactness_worst       : num  0.666 0.187 0.424 0.866 0.205 ...
##  $ concavity_worst         : num  0.712 0.242 0.45 0.687 0.4 ...
##  $ concave.points_worst    : num  0.265 0.186 0.243 0.258 0.163 ...
##  $ symmetry_worst          : num  0.46 0.275 0.361 0.664 0.236 ...
##  $ fractal_dimension_worst: num  0.1189 0.089 0.0876 0.173 0.0768 ...
```

```r
head(cancer_data)
```

```
##         id diagnosis radius_mean texture_mean perimeter_mean area_mean
## 1    842302         M       17.99        10.38         122.80    1001.0
## 2    842517         M       20.57        17.77         132.90    1326.0
## 3 84300903         M       19.69        21.25         130.00    1203.0
## 4 84348301         M       11.42        20.38          77.58     386.1
## 5 84358402         M       20.29        14.34         135.10    1297.0
## 6    843786         M       12.45        15.70          82.57     477.1
##   smoothness_mean compactness_mean concavity_mean concave.points_mean
## 1         0.11840          0.27760         0.3001             0.14710
## 2         0.08474          0.07864         0.0869             0.07017
## 3         0.10960          0.15990         0.1974             0.12790
## 4         0.14250          0.28390         0.2414             0.10520
## 5         0.10030          0.13280         0.1980             0.10430
## 6         0.12780          0.17000         0.1578             0.08089
##   symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se
## 1        0.2419                0.07871    1.0950     0.9053        8.589
## 2        0.1812                0.05667    0.5435     0.7339        3.398
## 3        0.2069                0.05999    0.7456     0.7869        4.585
## 4        0.2597                0.09744    0.4956     1.1560        3.445
## 5        0.1809                0.05883    0.7572     0.7813        5.438
## 6        0.2087                0.07613    0.3345     0.8902        2.217
##   area_se smoothness_se compactness_se concavity_se concave.points_se
## 1  153.40      0.006399        0.04904      0.05373           0.01587
## 2   74.08      0.005225        0.01308      0.01860           0.01340
```

```
## 3   94.03        0.006150            0.04006        0.03832            0.02058
## 4   27.23        0.009110            0.07458        0.05661            0.01867
## 5   94.44        0.011490            0.02461        0.05688            0.01885
## 6   27.19        0.007510            0.03345        0.03672            0.01137
##    symmetry_se fractal_dimension_se radius_worst texture_worst perimeter_worst
## 1     0.03003               0.006193        25.38         17.33          184.60
## 2     0.01389               0.003532        24.99         23.41          158.80
## 3     0.02250               0.004571        23.57         25.53          152.50
## 4     0.05963               0.009208        14.91         26.50           98.87
## 5     0.01756               0.005115        22.54         16.67          152.20
## 6     0.02165               0.005082        15.47         23.75          103.40
##    area_worst smoothness_worst compactness_worst concavity_worst
## 1      2019.0           0.1622            0.6656          0.7119
## 2      1956.0           0.1238            0.1866          0.2416
## 3      1709.0           0.1444            0.4245          0.4504
## 4       567.7           0.2098            0.8663          0.6869
## 5      1575.0           0.1374            0.2050          0.4000
## 6       741.6           0.1791            0.5249          0.5355
##    concave.points_worst symmetry_worst fractal_dimension_worst
## 1                0.2654         0.4601                 0.11890
## 2                0.1860         0.2750                 0.08902
## 3                0.2430         0.3613                 0.08758
## 4                0.2575         0.6638                 0.17300
## 5                0.1625         0.2364                 0.07678
## 6                0.1741         0.3985                 0.12440
```

Wow, that's a lot of data! Let's just keep the diagnosis column and the first 8 columns of measurements.

Also, let's shorten the column names for convenience. To do this, use the **sub()** function to replace "_mean" with "", passing the column names of the **cancer_data** df as the object to be operated on.

Hint: It's always good to look up a function in the Help docs when you haven't used it before.

```r
# make the dataset smaller!
cancer_data = cancer_data[,2:10]

# remove trailing "_mean" from the column names
colnames(cancer_data) = sub("_mean","",colnames(cancer_data))

# examine the resulting df
str(cancer_data)
```

```
## 'data.frame':    569 obs. of  9 variables:
##  $ diagnosis    : Factor w/ 2 levels "B","M": 2 2 2 2 2 2 2 2 2 2 ...
##  $ radius       : num  18 20.6 19.7 11.4 20.3 ...
##  $ texture      : num  10.4 17.8 21.2 20.4 14.3 ...
##  $ perimeter    : num  122.8 132.9 130 77.6 135.1 ...
##  $ area         : num  1001 1326 1203 386 1297 ...
##  $ smoothness   : num  0.1184 0.0847 0.1096 0.1425 0.1003 ...
##  $ compactness  : num  0.2776 0.0786 0.1599 0.2839 0.1328 ...
##  $ concavity    : num  0.3001 0.0869 0.1974 0.2414 0.198 ...
##  $ concave.points: num  0.1471 0.0702 0.1279 0.1052 0.1043 ...
```

```r
head(cancer_data)
```

```
##   diagnosis radius texture perimeter   area smoothness compactness concavity
## 1         M  17.99   10.38    122.80 1001.0    0.11840     0.27760    0.3001
```

3

```
## 2            M  20.57    17.77    132.90 1326.0     0.08474      0.07864      0.0869
## 3            M  19.69    21.25    130.00 1203.0     0.10960      0.15990      0.1974
## 4            M  11.42    20.38     77.58  386.1     0.14250      0.28390      0.2414
## 5            M  20.29    14.34    135.10 1297.0     0.10030      0.13280      0.1980
## 6            M  12.45    15.70     82.57  477.1     0.12780      0.17000      0.1578
##    concave.points
## 1         0.14710
## 2         0.07017
## 3         0.12790
## 4         0.10520
## 5         0.10430
## 6         0.08089
```

Whew! That's better.

Let's also **normalize** the data so that the values for each type of measurement are scaled relative to the mean and sd of each column. We will do this using the `scale()` function.

Note: You should use `scale` on the measurement columns only, and recombine them with the diagnosis column as a new data frame (which we will just assign back to the same old variable name). You can exclude the diagnosis column by negating it, or by specifying the desired column range explicitly.

```r
# normalize the data
scaled_data = data.frame(diagnosis = cancer_data$diagnosis,
                         scale(cancer_data[,-1]))
str(scaled_data)
```

```
## 'data.frame':    569 obs. of  9 variables:
##  $ diagnosis     : Factor w/ 2 levels "B","M": 2 2 2 2 2 2 2 2 2 2 ...
##  $ radius        : num  1.096 1.828 1.578 -0.768 1.749 ...
##  $ texture       : num  -2.072 -0.353 0.456 0.254 -1.151 ...
##  $ perimeter     : num  1.269 1.684 1.565 -0.592 1.775 ...
##  $ area          : num  0.984 1.907 1.558 -0.764 1.825 ...
##  $ smoothness    : num  1.567 -0.826 0.941 3.281 0.28 ...
##  $ compactness   : num  3.281 -0.487 1.052 3.4 0.539 ...
##  $ concavity     : num  2.6505 -0.0238 1.3623 1.9142 1.3698 ...
##  $ concave.points: num  2.53 0.548 2.035 1.45 1.427 ...
```

```r
head(scaled_data)
```

```
##   diagnosis     radius     texture  perimeter       area smoothness compactness
## 1         M  1.0960995 -2.0715123  1.2688173  0.9835095  1.5670875   3.2806281
## 2         M  1.8282120 -0.3533215  1.6844726  1.9070303 -0.8262354  -0.4866435
## 3         M  1.5784992  0.4557859  1.5651260  1.5575132  0.9413821   1.0519999
## 4         M -0.7682333  0.2535091 -0.5921661 -0.7637917  3.2806668   3.3999174
## 5         M  1.7487579 -1.1508038  1.7750113  1.8246238  0.2801253   0.5388663
## 6         M -0.4759559 -0.8346009 -0.3868077 -0.5052059  2.2354545   1.2432416
##     concavity concave.points
## 1  2.65054179      2.5302489
## 2 -0.02382489      0.5476623
## 3  1.36227979      2.0354398
## 4  1.91421287      1.4504311
## 5  1.36980615      1.4272370
## 6  0.86554001      0.8239307
```

Visualize the data

We are interested to find variables that will help us predict the diagnosis. Let's make some exploratory plots to visualize how the data are distributed among *malignant* and *benign* outcomes.

First, we will use `ggplot2` to create a boxplot for each variable by diagnosis using the `facet_grid` feature.

In order to do this, we will first have to reshape our matrix/table format to "long" format using the `melt()` function in the `reshape2` package. This produces a data frame with only three columns: "diagnosis", "variable", and "value". Each row contains one measurement, "value", of the type "variable". So essentially, the data from the original df are stacked vertically on top of each other in the long format. Long-format data is often preferred by `ggplot()`.

```r
# install.packages("reshape2") # uncomment to install (only needed once)
library(ggplot2)
library(reshape2)

# melt the dataframe and take a look at it
scaled_data_melt = melt(scaled_data)
```

```
## Using diagnosis as id variables
```

```r
head(scaled_data_melt)
```

```
##   diagnosis variable      value
## 1         M   radius  1.0960995
## 2         M   radius  1.8282120
## 3         M   radius  1.5784992
## 4         M   radius -0.7682333
## 5         M   radius  1.7487579
## 6         M   radius -0.4759559
```

```r
# make multiple boxplots
ggplot(data = scaled_data_melt,
       aes(x = variable, y = value, fill = diagnosis)) +
  geom_boxplot() +
  facet_wrap(~variable, scale="free") +
  scale_fill_manual(values=c("#00BFC4","#F8766D"))
```
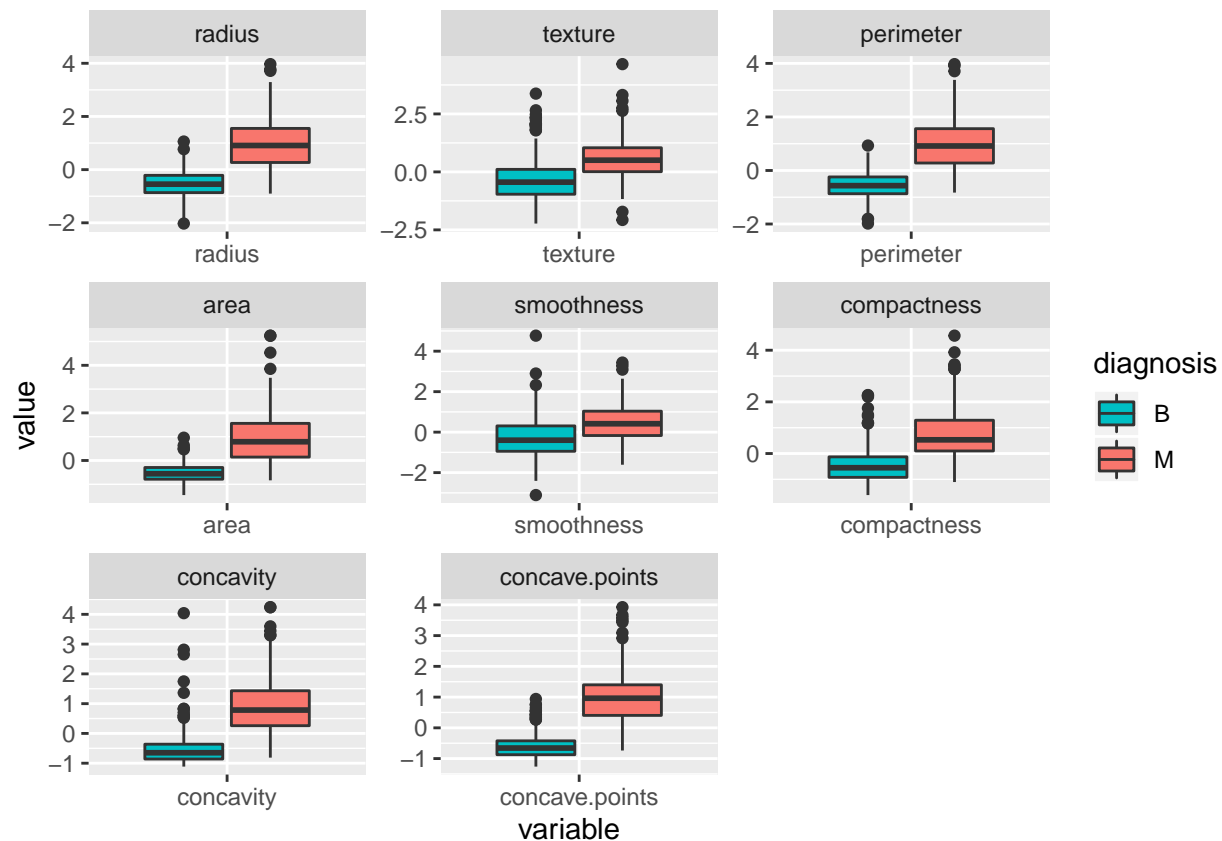
Which of the variables do you think would be most accurate in predicting cancer outcomes on its own? Explain your reasoning.
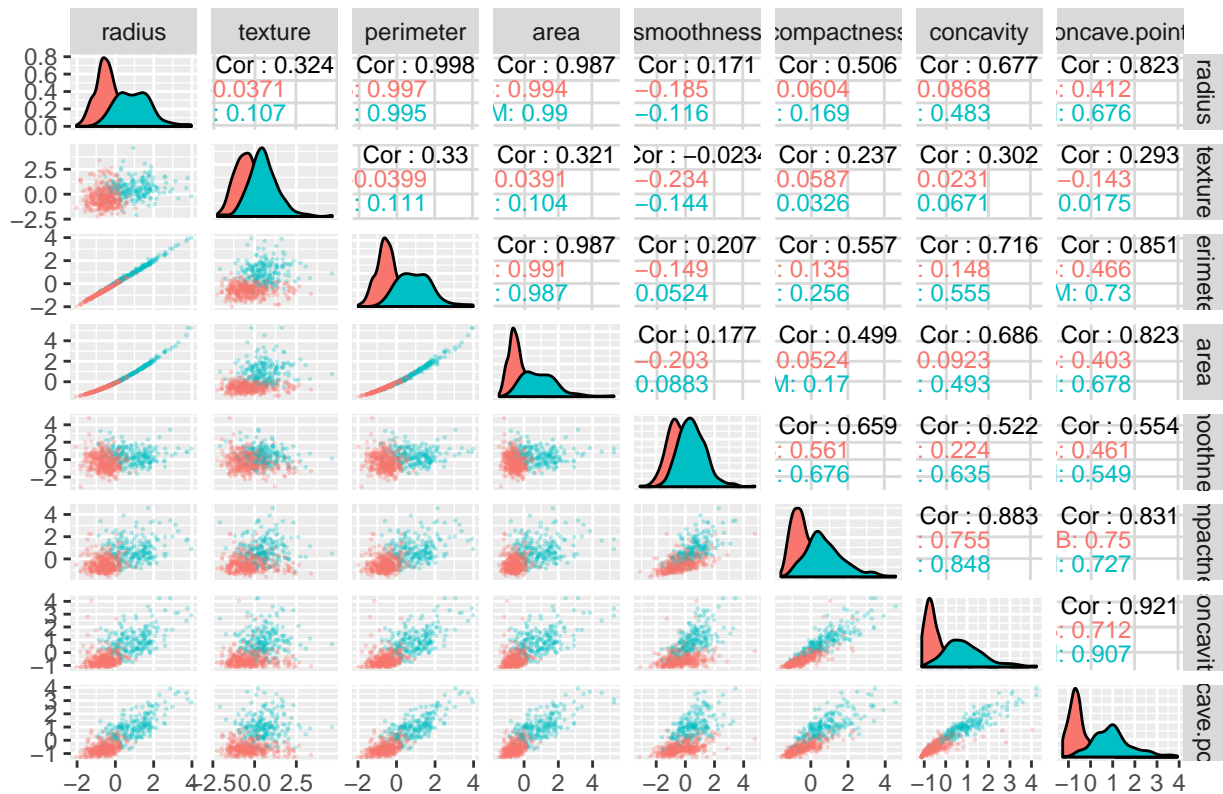

Your answer here.

We can get even more fancy and take a look at how all the variables correlate with each other, and with the diagnosis outcome.

Note: I wanted M to be the warm color and B to be the cool one, but I couldn't figure out how to change the color palette for ggpairs(). So the colors are reversed from the boxplot!

```
#install.packages("GGally")  # only needed once
library(GGally)

ggpairs(scaled_data, columns = 2:9, title = "Diagnosis vs. Features",
        upper = list(continuous = wrap("cor",size = 3)),
        lower = list(continuous = wrap("points",
                                        alpha = 0.3,
                                        size = 0.1)),
        mapping = aes(color = diagnosis))
```

Diagnosis vs. Features

Looking at this pair-wise correlation graph, answer the following questions:

Which variables do you think should provide the most predictive power for the diagnosis outcome? Explain your reasoning.

Your answer here.

Which variables are so highly correlated with each other that they are essentially redundant?

Your answer here.

Remove redundant features

Remove some of the redundant features from the data frame so that you only retain potentially informative variables. Just keep one column from each pair that shows a correlation > 0.9. Save the truncated data back to the `scaled_data` df. You should end up with 5 variables after this step.

```
scaled_data = scaled_data[,c(1:2,3,6:7,9)]
head(scaled_data)
```

```
##   diagnosis    radius     texture smoothness compactness concave.points
## 1         M 1.0960995 -2.0715123  1.5670875   3.2806281      2.5302489
## 2         M 1.8282120 -0.3533215 -0.8262354  -0.4866435      0.5476623
```

```
## 3          M  1.5784992  0.4557859  0.9413821   1.0519999        2.0354398
## 4          M -0.7682333  0.2535091  3.2806668   3.3999174        1.4504311
## 5          M  1.7487579 -1.1508038  0.2801253   0.5388663        1.4272370
## 6          M -0.4759559 -0.8346009  2.2354545   1.2432416        0.8239307
```

What is the motivation for reducing the dimensionality of the dataset?

Your answer here.

Q1: Prepare the data for modeling

To create our predictive models, we will use the `glm()` function, which stands for **generalized linear model**. Since logistic regression uses binary predictors, we will first encode all Benign outcomes as `0` and all Malignant outcomes as `1`.

Recode binary outcomes

First, split the dataframe into two dataframes: `B_all` (all the Benign data) and `M_all` (all the Malignant data), then recode the outcomes as `0` or `1`. You can use the `subset()` function to split the data into two separate data frames based on the diagnosis outcome. Then, just recode the values in each dataframe.

Note: For the sake of clarity, we will create a bunch of separate data frames for this exercise. An alternative would be to recode the original data frame and add some extra columns to label the different subsets.

```
# subset the data based on B or M diagnosis
B_all = subset(scaled_data, diagnosis == "B")
M_all = subset(scaled_data, diagnosis == "M")

# recode the diagnoses as binary outcomes
B_all$diagnosis = 0
M_all$diagnosis = 1
```

Create training and test datasets

For this exercise, we will train several predictive models on part of the data and hold out the rest of the data for testing the models to see which one gives the best predictions. Using hold-out data is a common way to test the quality of predictive models.

To generate the **training** and **test** datasets, we will randomly select 20% of the Benign samples and 20% of the Malignant samples and save them in a data frame called *BM_test*. The remaining samples will be stored in a different data frame called *BM_training*.

Note: there are some data science packages that help with some of the steps below, but we will just do them the old-fashioned way here.

```
# select random samples: 80/20 split
set.seed(2020)  # set a seed for reproducibility
B_sample = sample(1:nrow(B_all),
                  floor(0.2*nrow(B_all)))
set.seed(2020)  # set a seed for reproducibility
M_sample = sample(1:nrow(M_all),
                  floor(0.2*nrow(M_all)))
```

```
# combine the test data into a single data.frame
BM_test = rbind(B_all[B_sample,],
                M_all[M_sample,])

# combine the training data into a single data.frame
# (use negation of test data sets)
BM_train = rbind(B_all[-B_sample,],
                 M_all[-M_sample,])

# make sure the diagnosis variable in each df is a FACTOR
BM_test$diagnosis = as.factor(BM_test$diagnosis)
BM_train$diagnosis = as.factor(BM_train$diagnosis)
```

Q2: Binomial logistic regression

Now let's build separate models for each variable alone using `BM_train`, and then build a model using all of the predictors together. Later, we will compare the individual models to the composite model by evaluating their performance on the held-out test data.

General linear models come in **families** that describe the **link function** and **error distribution** for each model. Review the lecture notes on regression to refresh your memory about this. You should also check out the documentation on `glm()`.

Logistic regression actually uses an interative maximum likelihood estimation procedure. To make sure the models "converge", we will also specify the number of iterations to perform using the `maxit` parameter. (You can test the models without including that parameter and see what happens.)

```
# models with individual predictors
glm_radius      = glm(diagnosis ~ radius,
                      data = BM_train, family="binomial", maxit = 100)

glm_texture     = glm(diagnosis ~ texture,
                      data = BM_train, family="binomial", maxit = 100)

glm_smoothness  = glm(diagnosis ~ smoothness,
                      data = BM_train, family="binomial", maxit = 100)

glm_compactness = glm(diagnosis ~ compactness,
                      data = BM_train, family="binomial", maxit = 100)

glm_concavity   = glm(diagnosis ~ concave.points,
                      data = BM_train, family="binomial", maxit = 100)

# model with linear combination of all predictors (do not include interaction terms)
glm_all =         glm(diagnosis ~ radius + texture + smoothness +
                                  compactness + concave.points,
                      data = BM_train, family="binomial", maxit = 100)
```

Q3: Evaluate model performance

To determine which of the four variables is the most informative predictor, we first need to make diagnostic predictions for the held-out dataset using our different models. Then, we will compare how well they did on

the new data compared with the training data.

Predict using held-out data

To test our models, we will apply the `predict.glm()` function to the **BM_test** dataset. The default prediction `type` uses the scale of the linear predictors. We'll use the scale of the **"response"** variable instead.

Note: The `predict()` function is short-hand for a variety of more specific prediction functions; it will figure out the class of model from the object passed as the first parameter.

First, we will use each predictor individually, and then use all of them together, to generate alternative predictive models:

```
## Individual predictors
cancer_pred_radius  = predict(glm_radius,
                              newdata = BM_test, type="response")

cancer_pred_texture = predict(glm_texture,
                              newdata = BM_test, type="response")

cancer_pred_smooth  = predict(glm_smoothness,
                              newdata = BM_test, type="response")

cancer_pred_compact = predict(glm_compactness,
                              newdata = BM_test, type="response")

cancer_pred_concavity = predict(glm_concavity,
                                newdata = BM_test, type="response")

## All predictors
cancer_pred_all     = predict(glm_all,
                              newdata = BM_test, type="response")
```
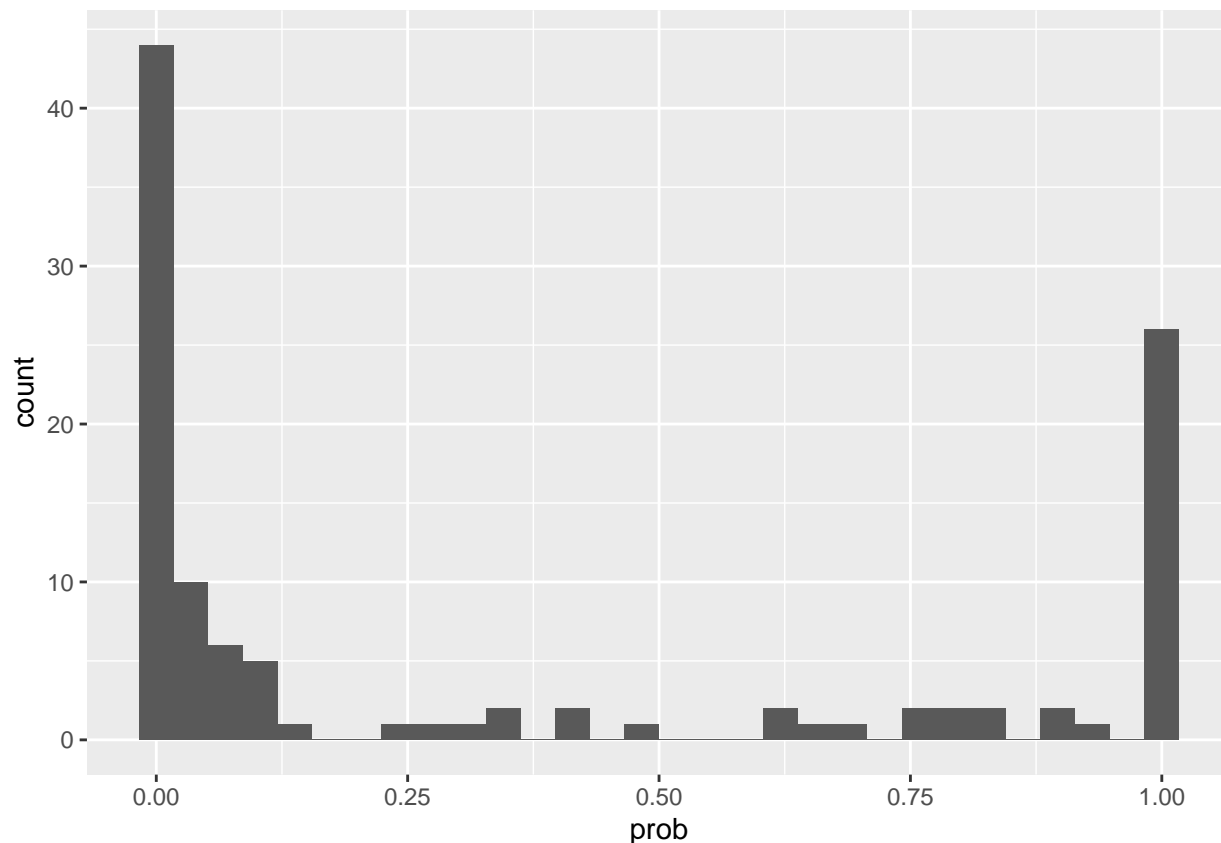
Great! Now we have some predictive models! What's next? Below we will explore some of the different ways we can evaluate the results of our predictions.

Histogram

Let's take a quick look at our predictions for the full model:

```
ggplot(data.frame(prob=cancer_pred_all)) +
  geom_histogram(aes(x=prob))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

The plot shows that the model is pretty confident about most of the test data, but it has a hard time deciding on the diagnosis for some of them.

Confusion matrix

We can look at the **confusion matrix**, which is just a table of true or false positives and negatives at a particular probability cutoff. Check this out for the full prediction model (`cancer_pred_all`) at 20%, 50%, and 80% probability cutoffs.

```
# 20% probability
neg_pos = as.numeric(cancer_pred_all >= 0.2)
table(neg_pos)
```

```
## neg_pos
##  0  1
## 66 47
```

```
table(neg_pos, BM_test$diagnosis)
```

```
##
## neg_pos  0   1
##       0 64   2
##       1  7  40
```

```
# 50% probability
neg_pos = as.numeric(cancer_pred_all >= 0.5)
table(neg_pos)
```

```
## neg_pos
```

```
##  0  1
## 74 39
```

```
table(neg_pos, BM_test$diagnosis)
```

```
##
## neg_pos  0  1
##       0 70  4
##       1  1 38
```

```
# 80% probability
neg_pos = as.numeric(cancer_pred_all >= 0.8)
table(neg_pos)
```

```
## neg_pos
##  0  1
## 82 31
```

```
table(neg_pos, BM_test$diagnosis)
```

```
##
## neg_pos  0  1
##       0 71 11
##       1  0 31
```

How do the false negative and false positives change as the threshold is increased?

_____

Your answer here.

_____

ROC and AUC

Another way we can assess performance is using the **AUC (area under the curve)** of a kind of plot that is strangely named a **ROC (Receiver Operating Characteristic)** plot. An **ROC plot** compares **sensivity (TP rate)** on the y-axis vs. **1-specificity (FP rate)** on the x-axis.

If our predictions are no better than random, then we would have an $AUC = 0.5$. We can visualize this by plotting a line from coordinates `0,0` to `1,1`. A straight line with a slope of 1 represents events that would happen just by chance.

On the other hand, curves that run higher and to the left will have a higher AUC and greater preditive power. So ideally, we will find a good model that has an ROC curve with a much larger area underneath it than the $AUC = 0.5$ line.

To evaluate our models, we will use the `roc()` command in the **AUC** package to compare the **Predicted** values with the **True** values. Once the ROC object is returned, we can get lots of different types of statistics, including **AUC** (by using the `auc()` function), TPR, FPR, specificity, sensitivity, accuracy, etc. and also a plot.

```
# install.packages("AUC")  # uncomment to install (you only need to do this once)
library(AUC)
```

```
## AUC 0.3.0
```

```
## Type AUCNews() to see the change log and ?AUC to get an overview.
```

```r
# set up the plots
par(mfrow=c(2,3))

## individual predictors #########################################################

# compare PREDICTIONS on held-out data with TRUE data from BM_test

# radius
roc_result = roc(cancer_pred_radius, BM_test$diagnosis)
plot(roc_result, main=paste("Radius: AUC = ", signif(auc(roc_result),3), sep=" "))

# texture
roc_result = roc(cancer_pred_texture, BM_test$diagnosis)
plot(roc_result, main=paste("Texture: AUC = ", signif(auc(roc_result),3), sep=" "))

# smoothness
roc_result = roc(cancer_pred_smooth, BM_test$diagnosis)
plot(roc_result, main=paste("Smoothness: AUC = ", signif(auc(roc_result),3), sep=" "))

# compactness
roc_result = roc(cancer_pred_compact, BM_test$diagnosis)
plot(roc_result, main=paste("Compactness: AUC = ", signif(auc(roc_result),3), sep=" "))

# concavity
roc_result = roc(cancer_pred_concavity, BM_test$diagnosis)
plot(roc_result, main=paste("Concavity: AUC = ", signif(auc(roc_result),3), sep=" "))

## combined predictors (all) ####################################################
roc_result = roc(cancer_pred_all, BM_test$diagnosis)
plot(roc_result, main=paste("All: AUC = ", signif(auc(roc_result),3), sep=" "))
```
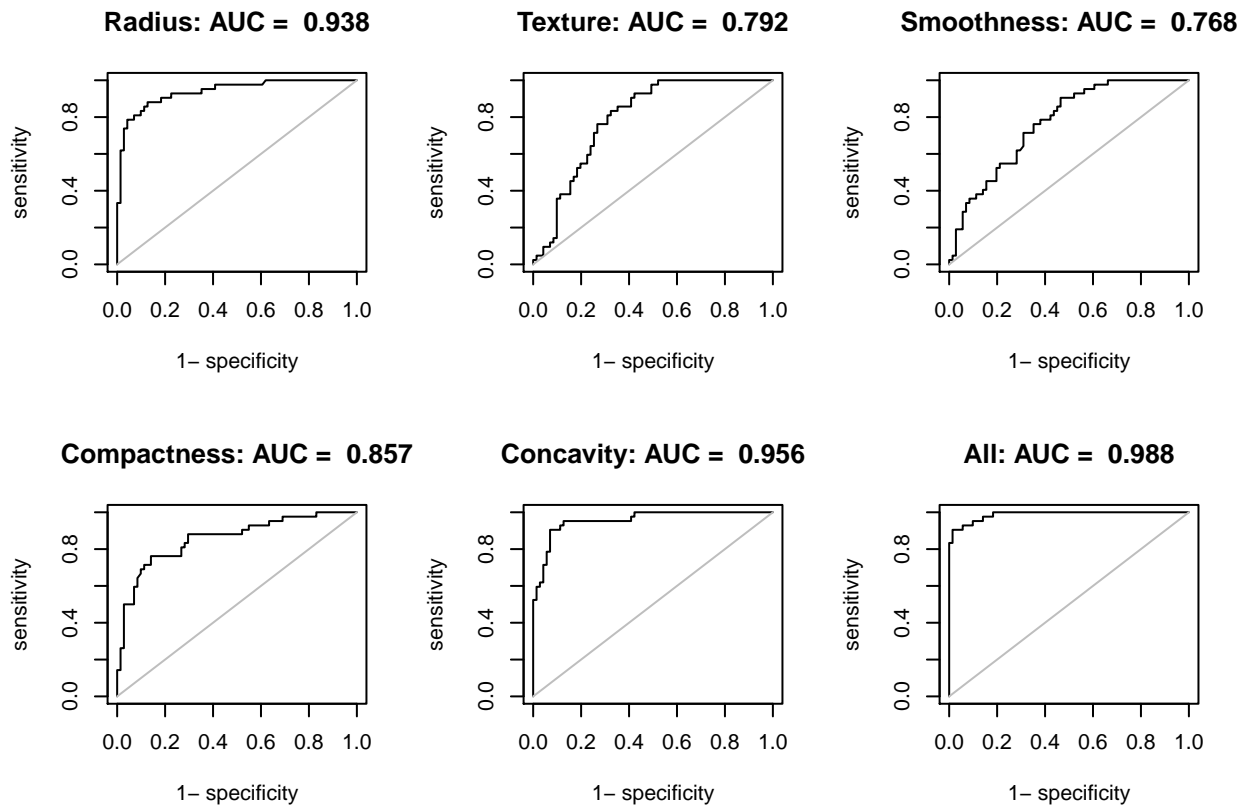
Radius: AUC = 0.938　　**Texture: AUC = 0.792**　　**Smoothness: AUC = 0.768**

Compactness: AUC = 0.857　　**Concavity: AUC = 0.956**　　**All: AUC = 0.988**

Comment on the performance of these models. Which one is best? Explain why you think this is the case.

Your answer here.

Q4: Model refinement

Residuals

Actually, you can look at the quality of the models using the `summary()` function, just like you did with linear regression. Do this with the full model above.

```
summary(glm_all)
```

```
##
## Call:
## glm(formula = diagnosis ~ radius + texture + smoothness + compactness +
##     concave.points, family = "binomial", data = BM_train, maxit = 100)
##
## Deviance Residuals:
##      Min        1Q     Median        3Q        Max
## -2.30219  -0.15319  -0.04559   0.02625    2.73193
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -0.7651     0.2520  -3.037 0.002391 **
```

```
## radius              2.8439        0.6971    4.080 4.51e-05 ***
## texture             1.4463        0.2834    5.103 3.35e-07 ***
## smoothness          0.7445        0.3942    1.889 0.058953 .
## compactness        -0.4563        0.4843   -0.942 0.346130
## concave.points      3.4146        0.9326    3.661 0.000251 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 602.31  on 455  degrees of freedom
## Residual deviance: 127.19  on 450  degrees of freedom
## AIC: 139.19
##
## Number of Fisher Scoring iterations: 8
```

Using the summary of the full model as a guide, now make a model with just the strongest predictors. Compare the difference between keeping two or three predictors in the model.

```
# generate all the objects and plots for models with top 2 or 3 predictors, as above

# set up 2 plots, 1 x 2
par(mfrow=c(1,2))

## top 2 predictors ############################################
glm_top2 = glm(diagnosis ~ radius + texture,
               data = BM_train, family="binomial", maxit = 100)

summary(glm_top2)
```

```
##
## Call:
## glm(formula = diagnosis ~ radius + texture, family = "binomial",
##     data = BM_train, maxit = 100)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1321  -0.3860  -0.1314   0.1084   2.8111
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.6799     0.1693  -4.016 5.92e-05 ***
## radius        3.6481     0.3914   9.320  < 2e-16 ***
## texture       0.9091     0.1783   5.098 3.43e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 602.31  on 455  degrees of freedom
## Residual deviance: 233.03  on 453  degrees of freedom
## AIC: 239.03
##
## Number of Fisher Scoring iterations: 7
```

```
cancer_pred_top2 = predict(glm_top2,
                           newdata = BM_test, type="response")

roc_result = roc(cancer_pred_top2, BM_test$diagnosis)
plot(roc_result, main=paste("Top 2: AUC = ",
                            signif(auc(roc_result),3), sep=" "))


## top 3 predictors #########################################
glm_top3 =glm(diagnosis ~ radius + texture + concave.points,
              data = BM_train, family="binomial", maxit = 100)

summary(glm_top3)
```

```
##
## Call:
## glm(formula = diagnosis ~ radius + texture + concave.points,
##     family = "binomial", data = BM_train, maxit = 100)
##
## Deviance Residuals:
##      Min       1Q    Median       3Q       Max
## -2.28379  -0.15982  -0.05067   0.02941   2.81432
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -0.7157     0.2333  -3.067  0.00216 **
## radius            2.3276     0.4948   4.704 2.55e-06 ***
## texture           1.2965     0.2637   4.917 8.80e-07 ***
## concave.points    3.7061     0.5258   7.049 1.80e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 602.31  on 455  degrees of freedom
## Residual deviance: 131.90  on 452  degrees of freedom
## AIC: 139.9
##
## Number of Fisher Scoring iterations: 8
```
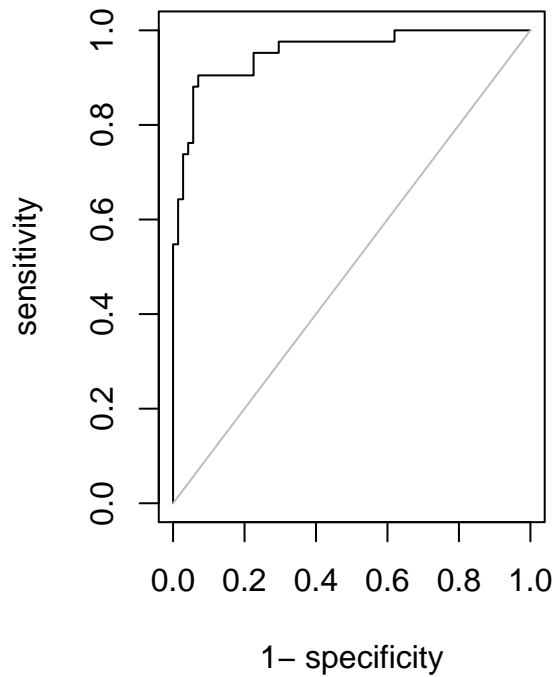
```
cancer_pred_top3 = predict(glm_top3,
                           newdata = BM_test, type="response")

roc_result = roc(cancer_pred_top3, BM_test$diagnosis)
plot(roc_result, main=paste("Top 3: AUC = ",
                            signif(auc(roc_result),3), sep=" "))
```
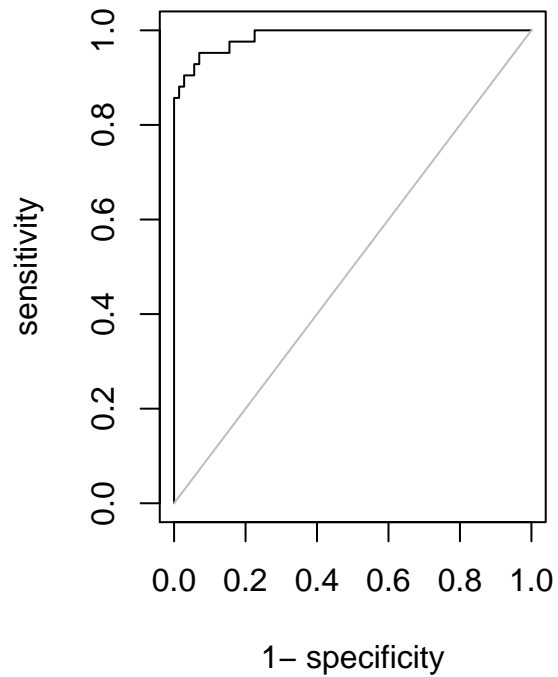
Top 2: AUC = 0.954　　　　**Top 3: AUC = 0.987**


```
## alternative method for plotting with ggplot #########################
# install.packages("ggpubr")  # only need to do this once
library(ggpubr)
```
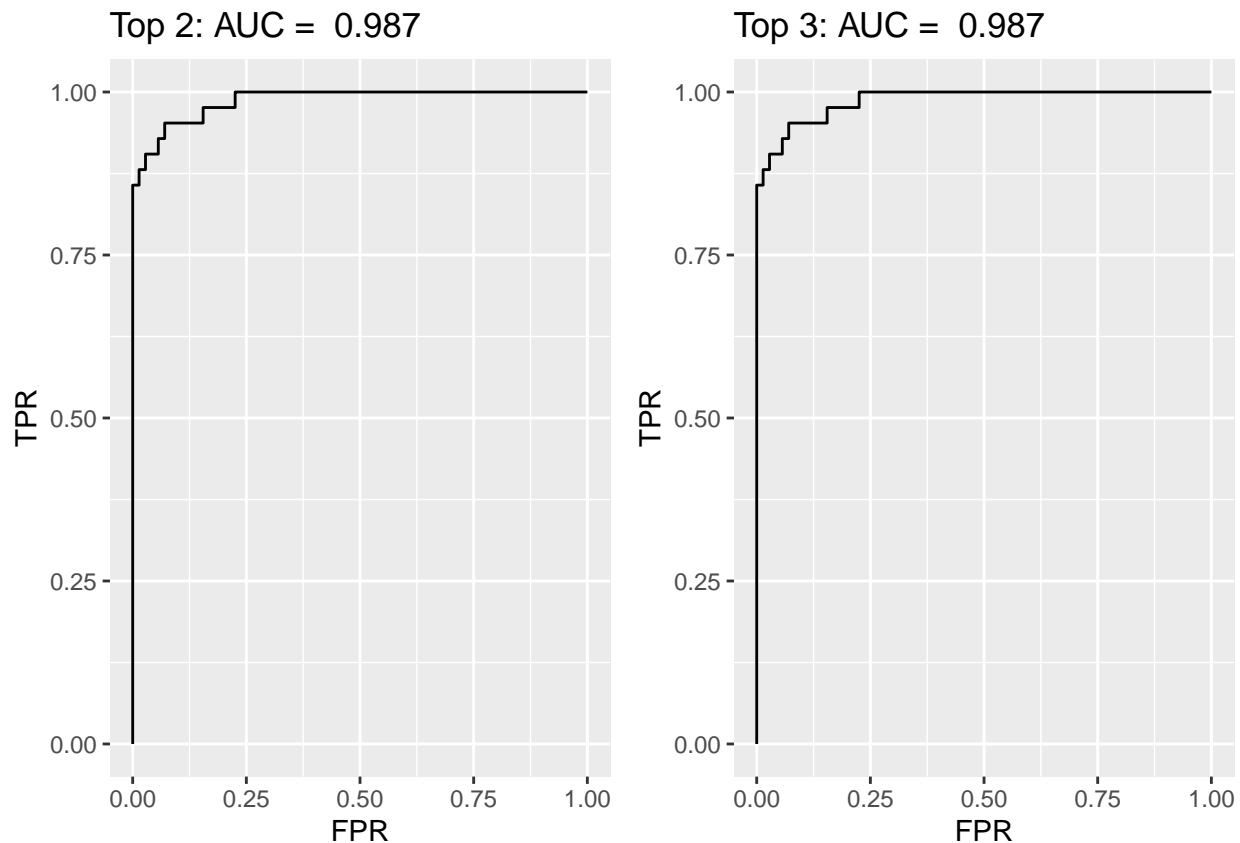
```
## Loading required package: magrittr
```

```
# theme_set(theme_pubr())

t2 = ggplot(data.frame(FPR=roc_result$fpr,
                       TPR=roc_result$tpr)) +
    geom_line(aes(x=FPR, y=TPR)) +
    ggtitle(paste("Top 2: AUC = ", signif(auc(roc_result),3), sep=" "))

t3 = ggplot(data.frame(FPR=roc_result$fpr,
                       TPR=roc_result$tpr)) +
    geom_line(aes(x=FPR, y=TPR)) +
    ggtitle(paste("Top 3: AUC = ", signif(auc(roc_result),3), sep=" "))

ggarrange(t2, t3, nrow = 1, ncol = 2)
```

How does do these reduced models perform relative to the one with all the predictors? Would you rather choose the model with as much non-redundant information as possible, or choose the model with fewer predictors? What are the tradeoffs? Explain your reasoning.

Your answer here.

ANOVA

Just like with linear models, you can also compare two logistic models with the `anova()` function, using the "Chisq" test. Compare the full model with one or more of the individual models, and then compare it with the two reduced models you just created. *Note: Remember that when comparing models, you always want to compare the more complex model to the simpler model, so you have to list the simpler model first.*

```
# simple vs glm_all
anova(glm_radius, glm_all, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: diagnosis ~ radius
## Model 2: diagnosis ~ radius + texture + smoothness + compactness + concave.points
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1       454     261.78
## 2       450     127.19  4   134.59 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
anova(glm_concavity, glm_all, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: diagnosis ~ concave.points
## Model 2: diagnosis ~ radius + texture + smoothness + compactness + concave.points
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1       454     199.96
## 2       450     127.19  4   72.771 5.896e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# glm_top2 and glm_top3 vs. glm_all
anova(glm_top2, glm_all, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: diagnosis ~ radius + texture
## Model 2: diagnosis ~ radius + texture + smoothness + compactness + concave.points
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1       453     233.03
## 2       450     127.19  3   105.84 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
anova(glm_top3, glm_all, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: diagnosis ~ radius + texture + concave.points
## Model 2: diagnosis ~ radius + texture + smoothness + compactness + concave.points
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1       452     131.90
## 2       450     127.19  2   4.7146  0.09467 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

From this analysis, can you decide which model seems best? Explain you reasoning.

Your answer here.

AIC

Akaike's Information Criterion can help with model selection by providing a measure of goodness-of-fit vs. model simplicity, which balances over- and under-fitting. AIC is computed as $-2 * ln(likelihood) + 2 * k$, where k is the number of estimated parameters. AIC is based on information theory, and a lower AIC means that less information is lost, which is better.

AIC is included in the results of the `summary()` function for a general linear model. Compare the AIC from the full model and the alternative models with just the top 2 or 3 predictors. Does this change your viewpoint on which model is best? Which model would you choose based on this criterion?

Note: I don't get the same results every time I knit this document! This is because some of these processes

include some stochasticity. So, you may want to re-run the analysis a few times ... even though the results will vary a little bit, overall the broad picture should be similar.

Your answer here.

Contratulations! You've made it to ** The End.** Yay!!!