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 = ___
---
---
```

```
## Error: <text>:2:15: unexpected input
## 1: # load the dataset and inspect it
## 2: cancer_data = _
##                  ^
```

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.

```
# make the dataset smaller!
cancer_data = ___

# remove trailing "_mean" from the column names
colnames(___) = ___

# examine the resulting df
---
---
```

```
## Error: <text>:2:15: unexpected input
## 1: # make the dataset smaller!
## 2: cancer_data = _
##                  ^
```

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.*

```
# normalize the data
scaled_data = data.frame(___, ___)

# check it
---
---
```

```
## Error: <text>:2:26: unexpected input
## 1: # normalize the data
## 2: scaled_data = data.frame(_
##                             ^
```

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()`.

```
# 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 = ___
---
```

```
# make multiple boxplots
ggplot(data = ___,
       aes(x = ___, y = ___, fill = ___)) +
  geom_boxplot() +
  facet_wrap(~variable, scale="free") +
  scale_fill_manual(values=c("#00BFC4","#F8766D"))
```

```
## Error: <text>:6:20: unexpected input
## 5: # melt the dataframe and take a look at it
## 6: scaled_data_melt = _
##                       ^
```

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)

# just run this (you don't have to do anything here)
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))
```

```
## Error in ggpairs(scaled_data, columns = 2:9, title = "Diagnosis vs. Features", : object 'scaled_data
```

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.

```
# keep just a subset of non-redundant columns, check the resulting df
scaled_data = ___
---
```

```
## Error: <text>:2:15: unexpected input
## 1: # keep just a subset of non-redundant columns, check the resulting df
## 2: scaled_data = _
##                  ^
```

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(___, ___)
M_all = subset(___, ___)

# recode the diagnoses as binary outcomes (B=0, M=1)
---
---
```

```
## Error: <text>:2:16: unexpected input
## 1: # subset the data based on B or M diagnosis
## 2: B_all = subset(_
##                   ^
```

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: keep 20% for testing
set.seed(___)  # set a seed for reproducibility
B_sample = sample(___,
                  floor(0.2*___))

set.seed(___)  # set a seed for reproducibility
M_sample = sample(___,
                  floor(0.2*___))

# combine the test data into a single data.frame
BM_test = rbind(___, ___)

# combine the training data into a single data.frame (remaining 80%)
# (use negation of test data sets)
BM_train = rbind(___, ___)

# make sure the diagnosis variable in each df is a FACTOR
BM_test$diagnosis = ___
BM_train$diagnosis = ___
```

```
## Error: <text>:2:10: unexpected input
## 1: # select random samples: keep 20% for testing
## 2: set.seed(_
##             ^
```

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(___ ~ ___,
                      data = ___, family="binomial", maxit = 100)

glm_texture     = ___

glm_smoothness  = ___

glm_compactness = ___

glm_concavity   = ___

# model with linear combination of all predictors (do not include interaction terms)
```

```
glm_all = ___
```

```
## Error: <text>:2:23: unexpected input
## 1: # models with individual predictors
## 2: glm_radius      = glm(_
##                          ^
```

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(___,
                                newdata = ___, type="___")

cancer_pred_texture   = ___

cancer_pred_smooth    = ___

cancer_pred_compact   = ___

cancer_pred_concavity = ___

## All predictors combined
cancer_pred_all       = ___
```

```
## Error: <text>:2:33: unexpected input
## 1: ## Individual predictors
## 2: cancer_pred_radius    = predict(_
##                                    ^
```

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))
```

```
## Error in data.frame(prob = cancer_pred_all): object 'cancer_pred_all' not found
```

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)
table(neg_pos, BM_test$diagnosis)

# 50% probability
---
---
---

# 80% probability
---
---
---
```

```
## Error: <text>:7:1: unexpected input
## 6: # 50% probability
## 7: _
##    ^
```

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). Other functions that can be used on the object include: specificity, sensitivity, accuracy, roc, and also plot.

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

```
# set up the plots (2 rows by 3 columns)
par(___))

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

# compare PREDICTIONS on held-out data with TRUE data from BM_test (don't forget
# to change the plot titles, and double-check the data you're using for each one!)

# 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 = ___
plot(___)

# smoothness
---
---

# compactness
---
---

# concavity
---
---

## combined predictors (all) ######################################################
---
---
```

```
## Error: <text>:5:5: unexpected input
## 4: # set up the plots (2 rows by 3 columns)
## 5: par(_
##        ^
```

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 for glm_all
---
```

```
## Error: <text>:2:1: unexpected input
```

```
## 1: # summary for glm_all
## 2: _
##    ^
```

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

## top 2 predictors ################################################
glm_top2 =glm(___)

summary(___)

cancer_pred_top2 = predict(___)

roc_result = ___
plot(___)

## top 3 predictors ################################################
# do all the same stuff you did above

glm_top3 = ___

...
```

```
## Error: <text>:4:15: unexpected input
## 3: ## top 2 predictors ################################################
## 4: glm_top2 =glm(_
##                  ^
```

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(___, ___, test = "Chisq")
anova(___, ___, test = "Chisq")
...

# glm_top2 and glm_top3 vs. glm_all
---
---
```

```
## Error: <text>:2:7: unexpected input
```

```
## 1: # simple vs glm_all
## 2: anova(_
##          ^
```

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!!!