XDAS-2020 HW 11: Logistic Regression

Angelika Amon [YOUR NAME HERE]

17 November, 2020

Contents

The Wisconsin Breast Cancer Dataset

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**.

Q0: Prepare the data

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**.

```
# load the dataset and inspect it
```

Wow, that's a lot of data! Let's just keep the `diagnosis` column and the **first 8 columns of measurements**. Save the result in the same data frame.

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!

# remove trailing "_mean" from the column names with sub function

# examine the resulting df
```

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. Save the result at **scaled_data**.

Note: You should use `scale` on the measurement columns only, and recombine them with the diagnosis column as a new data frame. You can exclude the diagnosis column by negating it, or by specifying the desired column range explicitly.

```
# normalize the data


# examine the resulting data frame
```

Q1: 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.

Q1a: Boxplots

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 = scaled_data_melt,
       aes(x = ... )) +
  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.

Q1b: Correlogram

We can get even more fancy and take a look at how all the variables correlate with each other, and with the diagnosis outcome, using the `ggpairs` function from the `GGally` package. Below, the distributions on the diagonal give essentially the same information as the boxplots, but we also get visual and quantitative information on correlations between all pairwise combinations of the features.

Note: I want M to be the warm color and B to be the cool one for consistency with the boxplot. I wasn't sure how to do this so I did it by brute force below using a solution I found on the web (the other solutions looked a lot more complicated to me). Maybe someone can find a better way?

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

p = 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

# change color scheme for each plot in the `ggmatrix` object created by `ggpairs`
# so B = teal and M = reddish
for(i in 1:p$nrow) {
  for(j in 1:p$ncol){
    p[i,j] <- p[i,j] +
        scale_fill_manual(values=c("#00BFC4","#F8766D")) +
        scale_color_manual(values=c("#00BFC4","#F8766D"))
  }
}
## Error in eval(expr, envir, enclos): object 'p' not found

p
## Error in eval(expr, envir, enclos): object 'p' not found
```

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

Your answer here.

Q2: Prepare the data for modeling

Q2a: Remove redundant features

Below you will 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.
  - ***NOTE: For the purposes of this exercise, please make sure you keep `concave.points` instead of `concavity` when choosing to remove one of these.***
- Save the truncated data back to the `scaled_data` df.

You should end up with 5 variables after this step.

```
# reduce the number of features


# check out the resulting data frame
```

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

Your answer here.

Q2b: Recode binary outcomes

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`.

- First, split the data into two separate data frames based on the diagnosis outcome: `B_all` (all the Benign data) and `M_all` (all the Malignant data)
- Then, recode the `diagnosis` values in each dataframe as `0` (benign) or `1` (malignant)

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



# recode the diagnoses as binary outcomes
```

Q2c: 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*. Follow these steps:

- Use the `sample` function to sample 20% of the data from each data frame

- Combine those into a single data frame for testing later
- Combine the rest of the data into a single data frame for training
- Make sure the `diagnosis` column in each matrix is a **factor**

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.

```r
# select random samples: 80/20 split
set.seed(2020)  # set a seed for reproducibility
B_sample = sample(___,
                  floor(0.2*nrow(B_all))) # sample an integer number of rows

set.seed(2020)  # set a seed for reproducibility
M_sample = ...

# combine the test data into a single data.frame
BM_test = ...

# combine the training data into a single data.frame
# (use negation of test data sets)
BM_train = ...

# make sure the diagnosis variable in each df is a FACTOR
```

```
## Error: <text>:3:19: unexpected input
## 2: set.seed(2020)  # set a seed for reproducibility
## 3: B_sample = sample(_
##                      ^
```

Q3: 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 iterative ***maximum likelihood estimation (MLE)*** 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.)

Below, use `glm` to build separate models using individual features, and then using all the features in your reduced dataset. The arguments you will need are:

- a formula for the linear model
- data = training dataset
- family = family of link functions (use binomial)
- maxit = number of iterations (use 100)

```r
# models with individual predictors
glm_radius      = glm(___ ~ ___,
                      data = ___, family= ___, maxit = ___)

glm_texture     =
glm_smoothness  = 


glm_compactness = 


glm_concave.points   = 


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

# what kind of object is `glm_all`?
```

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

Q4: 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.

Q4a: Predict using held-out data

To test our models, we will apply the `predict.glm()` function to the **BM_test** dataset and see what outcomes are predicted for the held-out data. The default prediction `type` uses the scale of the linear predictors. We'll use the scale of the **"response"** variable instead.

Note: You can just use the `predict()` function, which 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 feature individually, and then use all of them together, to generate alternative predictions. The arguments you will need are:

- a glm object
- newdata = the held-out test data to use for prediction
- type = "response"

Take a look at one or more of the created objects to see what kind of output you have created for the predictions.

```
## Individual predictors
cancer_pred_radius  = predict(___,
                              newdata = ___, type= ___)

cancer_pred_texture = 


cancer_pred_smooth  = 


cancer_pred_compact = 

cancer_pred_concave.points =


## All predictors
cancer_pred_all     =

# what kind of object is `cancer_pred_all`?
```

```
## Error: <text>:2:31: 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.

Q4b: Histogram

Take a look at the predictions for the **full model** (`cancer_pred_all`) by plotting the probability of classifying the held-out 3D image data as benign (0) or malignant (1).

```
# histogram of predicted outcomes
```

What does this plot tell you about the predictions?

Your answer here

Q4c: Confusion matrix

We can look at the **confusion matrix**, which is just a table comparing predicted vs. actual outcomes at a particular probability cutoff (it's like a contingency table but not quite the same since we are not looking at observed vs. expected, but actual vs. predicted). This will tell you the proportion of TP and FP, from which you can compute lots of statistics like precision, accuracy, TP rate, FP rate, etc. A quick guide to confusion matrices may be found **here**.

Check this out for the full prediction model (`cancer_pred_all`) at 20%, 50%, and 80% probability cutoffs. For each cutoff, do the following:

- Subset the predicted data in `cancer_pred_all` to select everything above the cutoff (using a logical operation) and save the result as `pred.neg_pos`.
- Check the frequency table to look at the number of FALSE and TRUE you get.
- Optional: convert the TRUE / FALSE to 1's and 0's or "M" and "B".
- Make a **confusion matrix** showing the proportion of the actual vs. predicted diagnoses (from the original `BM_test` data frame) and print the output.
  - Note that `pred.neg_pos` retains the same indexes as the original row indexes from the `BM_test` data frame, which is how they get matched in the `table` function.

Note: `table` will interpret one or more objects passed to it as as factors (the categories will be coerced into factors if not explicitly specified already). Use the diagnoses from the `BM_test` data as the first factor, and the filtered predictions as the second factor.

```
# 20% probability ========================================================== #
# subset results with prob 20% or above
pred.neg_pos = ___
```

```r
# optional: covert TRUE / FALSE to something else


# check the frequency of negative and positive predictions


# contingency table
table(___, ___,
      dnn = c("Actual", "Predicted")) # dnn adds dimension names

# 50% probability ========================================================== #




# 80% probability ========================================================== #

## Error: <text>:3:16: unexpected input
## 2: # subset results with prob 20% or above
## 3: pred.neg_pos = _
##                   ^
```

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

Your answer here.

Q4d: 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 **Actual** 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.

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

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

## individual predictors ##############################################################
```

```r
# compare PREDICTIONS on held-out data with ACTUAL data from BM_test

# radius
type = "Radius"
roc_result = roc(___, ___)
plot(___, main=paste(type, ": AUC = ", signif(auc(roc_result),3), sep=""))

# texture
type = "Texture"
roc_result = ...
plot(...)

# smoothness
type = "Smoothness"
roc_result = ...
plot(...)

# compactness
type = "Compactness"
roc_result = ...
plot(...)

# concave.points
type = "ConcavePoints"
roc_result = ...
plot(...)

## combined predictors (all) #####################################################
type = "All"
roc_result = ...
plot(...)
```

```
## Error: <text>:13:18: unexpected input
## 12: type = "Radius"
## 13: roc_result = roc(_
##                      ^
```

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

Your answer here.

Which features look most informative for prediction?

Your answer here.

Q4e: Regression coefficients

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
```

Which features look most informative for prediction? Look at the AUC for the individual predictors and the regression coefficients for each of them in the full model. Would you make the same conclusions about which features are most useful for prediction by looking at these different pieces of information? Explain your reasoning.

Your answer here.

Q5: Model refinement and evaluation

Q5a: Make reduced models

Using the AUC for the individual models and the summary of the full model as a guide, make two models that keep the three strongest predictors based on (1) the AUC, and (2) the regression coefficients. Then, plot the AUC's for the two new models to see how they compare.

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

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

## AUC top 3 predictors ================================================== #
glm_top3_AUC =glm(...)
## Error in eval(expr, envir, enclos): '...' used in an incorrect context
summary(glm_top3_AUC)
## Error in summary(glm_top3_AUC): object 'glm_top3_AUC' not found

cancer_pred_top3_AUC = predict(...)
## Error in eval(expr, envir, enclos): '...' used in an incorrect context

roc_top3_AUC = roc(...)
## Error in roc(...): could not find function "roc"

plot(roc_top3_AUC, main=paste("Top 3 AUC: AUC = ",
                              signif(auc(roc_top3_AUC),3), sep=" "))
## Error in plot(roc_top3_AUC, main = paste("Top 3 AUC: AUC = ", signif(auc(roc_top3_AUC), : object 'ro


## REG top 3 predictors ================================================== #
glm_top3_REG =glm(...)
## Error in eval(expr, envir, enclos): '...' used in an incorrect context
summary(glm_top3_REG)
## Error in summary(glm_top3_REG): object 'glm_top3_REG' not found

cancer_pred_top3_REG = predict(...)
## Error in eval(expr, envir, enclos): '...' used in an incorrect context
```

```r
roc_top3_REG = roc(...)
## Error in roc(...): could not find function "roc"

plot(roc_top3_REG, main=paste("Top 3 REG: AUC = ",
                              signif(auc(roc_top3_REG),3), sep=" "))
## Error in plot(roc_top3_REG, main = paste("Top 3 REG: AUC = ", signif(auc(roc_top3_REG), : object 'ro


## alternative method for plotting with ggplot  =========================== #
# install.packages("ggpubr")  # only need to do this once
library(ggpubr)

t3_auc = ggplot(data.frame(FPR=roc_top3_AUC$fpr,
                           TPR=roc_top3_AUC$tpr)) +
  geom_line(aes(x=FPR, y=TPR)) +
  ggtitle(paste("Top 3 AUC: AUC = ", signif(auc(roc_top3_AUC),3), sep=" "))
## Error in data.frame(FPR = roc_top3_AUC$fpr, TPR = roc_top3_AUC$tpr): object 'roc_top3_AUC' not found

t3_reg = ggplot(data.frame(FPR=roc_top3_REG$fpr,
                           TPR=roc_top3_REG$tpr)) +
  geom_line(aes(x=FPR, y=TPR)) +
  ggtitle(paste("Top 3 REG: AUC = ", signif(auc(roc_top3_REG),3), sep=" "))
## Error in data.frame(FPR = roc_top3_REG$fpr, TPR = roc_top3_REG$tpr): object 'roc_top3_REG' not found

ggarrange(t3_auc, t3_reg, nrow = 1, ncol = 2)
## Error in ggarrange(t3_auc, t3_reg, nrow = 1, ncol = 2): object 't3_auc' not found
```

Which model is better? 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.

Q5b: 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.

```r
# simple vs glm_all



# glm_top3_AUC and glm_top3_REG vs. glm_all
```

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

Your answer here.

Q5c: Akaike's Information Criterion (AIC)

Akaike's Information Criterion (AIC) provides a measure of the relative quality of different statistical models. AIC can help with model selection by balancing goodness-of-fit vs. model simplicity. AIC rewards smaller residual errors, but it also penalizes the addition of more predictors. This tradeoff helps to avoid over- and under-fitting.

AIC is based on information theory is computed as $-2 * ln(likelihood) + 2 * k$, where k is the number of estimated parameters. A lower AIC means that less information is lost, which is better.

You might have noticed that the AIC is included in the results of the `summary()` function for a general linear model, and can be accessed directly using the `aic` attribute (so, you can just write `summary(model)$aic` to get the AIC for any model).

```
# extract aic values for different models
```

Compare the AIC from the full model and the alternative models to see how AIC changes as more predictors are added.

Does this change your viewpoint on which model is best? Which model would you choose based on this criterion?

Your answer here.

Q5d: Stepwise regression

Instead of comparing all of these models by hand, which we just saw can get a bit tedious (!), you can use stepwise regression instead, which tests the effect of adding and removing predictors. This can help you find the simplest model that gives the smallest AIC value.

Use the `step` function on the full model to find the best model according to this method.

```
# perform step-wise model evaluation
```

What is the best model based on the step-wise regression?

Your answer here.

Compare the optimized model with the full model

- Make a model with the top 4 predictors.
- Perform a Chi-square test against the full model.
- Compare the model summaries. (Notice that all the coefficients change a little bit...)
- Plot the AUC's side by side.

```
## model with top 4 predictors ============================================ #
# generate model
glm_top4 = glm(...)
## Error in eval(expr, envir, enclos): '...' used in an incorrect context
```

```
# make predictions
cancer_pred_top4 = predict(...)
## Error in eval(expr, envir, enclos): '...' used in an incorrect context
# get the roc
roc_result_top4 = roc(...)
## Error in roc(...): could not find function "roc"

# anova chi-square ========================================================= #


# model summaries ========================================================== #



# AUC plots ================================================================ #
par(mfrow=c(2,3))
plot(roc_result_top4, main=paste("Top4: AUC = ", signif(auc(roc_result_top4),3), sep=""))
## Error in plot(roc_result_top4, main = paste("Top4: AUC = ", signif(auc(roc_result_top4), : object 'r
plot(roc_result, main=paste("All: AUC = ", signif(auc(roc_result),3), sep=""))
## Error in plot(roc_result, main = paste("All: AUC = ", signif(auc(roc_result), : object 'roc_result'
```

Note: If you don't set a seed in making the held-out data, then you won't get exactly the same results every time you knit this document. Even though the results will vary a little bit, overall the broad picture should be similar because you are running a bunch of iterations to get convergence. An alternative to just using one held-out test set is to use k-fold cross-validation, but we won't get into that here.

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