ES7023 - Assignment 3 : Calibration

Calibration is the process of choosing values for the parameters in a model. For example, a simple and natural model that you might try to apply to the data in Figure 1 is a line. You need a slope and an intercept to define a line, so these two quantities are the parameters that we're trying to specify when we "fit" a line to these data ("fitting" and "calibrating" are synonymous in this context). For a linear model, calibrating the model means answering the question "What slope and what intercept?".

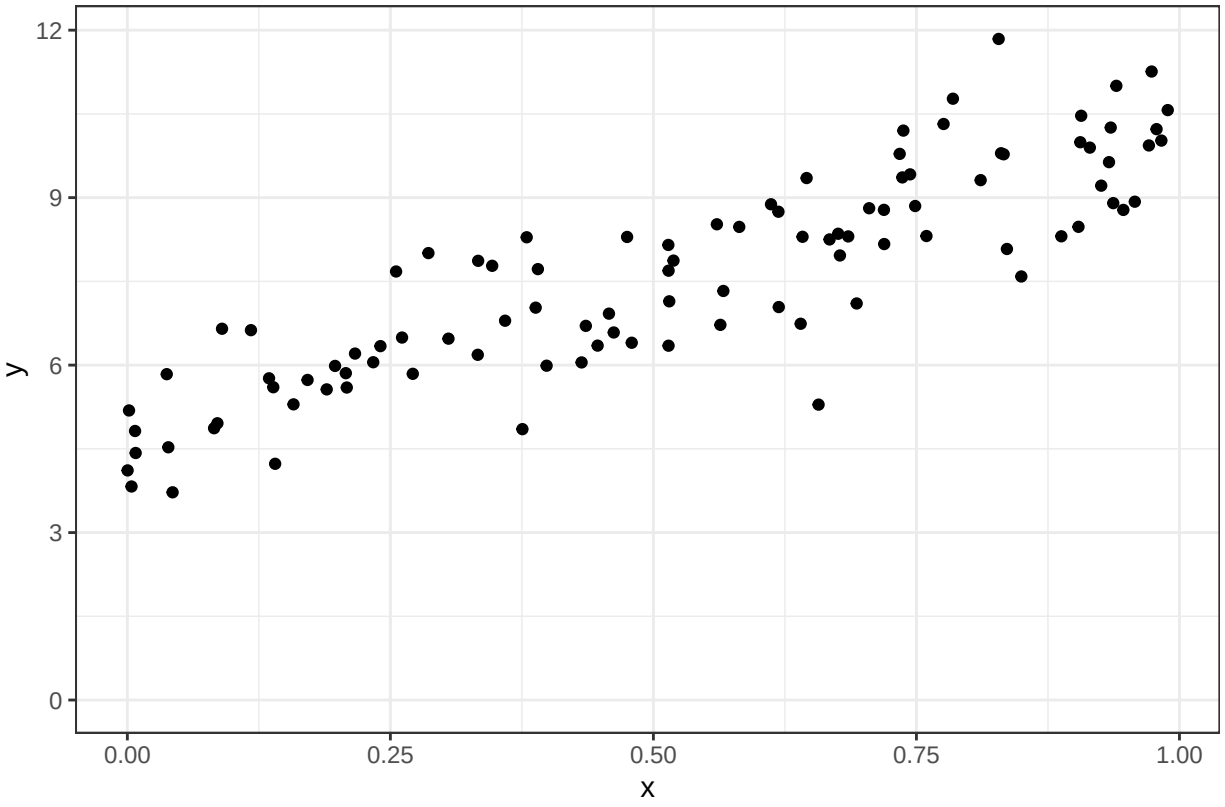

Figure 1: Scatter plot of x and y

To do this we need some measure of goodness-of-fit. A standard approach is to define a loss function $L(Y, \hat{Y})$ and then search for parameters that minimize this function. One attractive loss function (and the one used for linear regression) is the sum of squared errors:

$$L(Y, \hat{Y}) = \sum_{i=1}^{n} (Y - \hat{Y})^2$$

This is attractive because to find the minimum we can take the derivative and set it to zero. For example, take the simple case of a linear regression with one predictor variable:

$$Y = \beta_0 + \beta_1 * X$$

With a least squares loss function we have

$$L(Y, \hat{Y}) = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 * x_i))^2$$

Taking the derivative with respect to β_0 and β_1 and setting equal to zero

$$\frac{dL(Y, \hat{Y})}{d\beta_0} = \sum_{i=1}^{n} 2 * (y_i - (\beta_0 + \beta_1 * x_i)) = 0$$
$$\frac{dL(Y, \hat{Y})}{d\beta_1} = \sum_{i=1}^{n} -2x_i * (y_i - (\beta_0 + \beta_1 * x_i)) = 0$$

You now have two equations and two unknowns (β_0 and β_1; the intercept and slope, respectively), which means that you can algebraically solve for both. Few people (including me) actually remember the algebra, but you can verify that the solutions to these are

$$\hat{\beta_1} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$\hat{\beta_0} = \bar{y} - \hat{\beta_1}\bar{x}$$

where $\bar{x}$ and $\bar{y}$ are the means of x and y respectively. There are many lines close to this one that might look like they fit the data just as well, but these are the exact parameters for the line that minimizes the squared error loss, and that's what we use. Now that you know this formula, you could write a function to return the estimates of β_0 and β_1 for a set of data x and y. For example,

```
# generate some random x and y
x = runif(100)
y = 5+5*x+rnorm(n = length(x),mean=0,sd = 1) # we've specified the beta: b0=5, b1=5


#plot data
#plot(x,y) # this will generate a plot like Figure 1

#function to return the betas
get.beta = function(x,y){
    beta1 = sum((x-mean(x))*(y-mean(y)))/sum((x-mean(x))^2)
    beta0 = mean(y)-beta1*mean(x)
    c(beta0,beta1)
}

get.beta(x,y)
```

```
## [1] 4.989505 5.000252
```

Our function to get the β parameters for single-variate linear regression works! Yay! :)

We can also check against R's built-in linear regression function:

```
lm(y~x)$coef
```

```
## (Intercept)           x
##    4.989505    5.000252
```

Beautiful!

While ordinary least squares (OLS) has numerous advantages, it also has some issues, many of which were discussed in class (e.g. sensitivity to outliers, assumption of normally distributed residuals, etc). Another calibration methods is Maximum Likelihood Estimation (MLE), which we covered briefly in class and will discuss in more detail later in an optional section. It turns out that in simple linear regression the MLE and OLS methods give exactly the same answers, which means we're free to use the computationally much simpler OLS. For other models, such as the one we'll deal with in this assignment, we'll use MLE.

Generalized Linear Models

The model we'll fit in this exercise is an example of what are known as generalized linear models (GLM), which look a lot like regular linear models but involve an extra step. In the model we've been discussing so far, the goal is to find the parameters $\beta_0, \beta_1, ..., \beta_k$ in the linear equation

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_k x_k$$

Sometimes, though, we need to predict some function $f(z)$ rather than z. After the transformation of z by $f()$, the model's errors might not be normally distributed, but have some other recognizable distribution such as a Gamma distribution. Our calibration problem is now focused on finding the β's that give us not the "best" estimate z, but rather the best estimate of $f(z)$.

Logistic Regression

The particular generalized linear model we'll look at in this exercise is called logistic regression, which is a technique used to predict the probability of an occurrence of a binomial outcome, i.e., a response variable with outcomes that can be denoted "true" and "false". For example, predicting whether a patient will or won't have a heart attack, given a set of predictor variables such as age, the presence of a prior surgery, cholesterol, and so on. As discussed in the previous section, these predictors are linearly combined via some coefficients (which our calibration procedure needs to find) $\beta_0, \beta_1, ..., \beta_k$, where k is the number of predictors.

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_k x_k$$

The left hand side, z, is then transformed by a function $f(z)$, which in this case is the logistic function:

$$f(z) = \frac{1}{1 + e^{-z}}$$

The logistic function $f(z)$ maps all possible values of z to values between zero and one, which is a desirable property since the range of probabilities for a binomial outcome is necesarily between zero and 1 (e.g. the probability of getting a heart attack). The exercises will walk you through a series of steps that finds the β's that give the "best" $f(z)$, where "best" means the $f(z)$ with the highest likelihood of producing the data we used to train the model, assuming that said data was generated by some "true" logistic model whose parameters we're not privy to. There's a lot in there, and it's worth thinking about how "best" might not necessarily mean "good".

Those interested more in the math should read on, but at this point you have enough information to start trying the problems.

Maximum Likelighood Estimation and Logistic Regression (recommended but optional)

Maximum likelihood estimation (MLE) is a common way of finding parameters, and it works by maximizing a likelihood function that defines the probability of observing the data given values of the parameters. For

example, in our linear regression model developed previously, we could ask what's the chance of observing the x and y we observed given certain values of β_0 and β_1. If we assume that each sample is independent, this would be proportional to the product of the probabilities of each pair of x and y.

$$\mathcal{L}(x, y \mid \beta_0, \beta_1) = \Pi_{i=1}^{n} Prob(x_i, y_i)$$

Remember that capital pi $\Pi_{i=1}^{n}$ is the notation for the product of a sequence from 1 to n.

The key step in MLE is defining this likelihood function $\mathcal{L}$, which amounts to describing the distribution that you expect the errors to follow. For example, if we assume that errors have a normal (Gaussian) distribution, that is

$$\varepsilon \sim N(0, \sigma^2)$$

$$f(\varepsilon) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\mu)^2}{2\sigma^2}}$$

For the linear regression case we have:

$$\mathcal{L}(x, y \mid \beta_0, \beta_1) = \Pi_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - (\beta_0 + \beta_1 x_i))^2}{2\sigma^2}}$$

Mathematically it is easier to maximize the log of the likelihood, which is equivalent since log is a monotonic function (i.e. if $x > y$, than for any monotonic transformation of x and y, $f(x) > f(y)$). Recall that $log(a * b) = log(a) + log(b)$, so that we can maximize the sum of log probabilities of each pair (x_i, y_i). Computationally this is better than multiplying a lot of small probabilities together, which would quickly approach 0. It is also easier to find the derivative of a sum than a product. The log probability of each (x_i, y_i) is

$$ln(Prob(x_i, y_i \mid \beta_0, \beta_1)) = -\frac{1}{2} ln(2\pi) - \frac{1}{2} ln(\sigma^2) - \frac{(y_i - (\beta_0 + \beta_1 x_i))^2}{2\sigma^2}$$

If you look at this closely, you'll see that if we want to maximize the sum of these probabilities, we will be minimizing the sum of squared differences between our our observations y_i and our estimates $\beta_0 + \beta_1 x_i$. In other words, the Ordinary Least Squares (OLS) solution for the linear regression is equivalent to the MLE estimate assuming normally distributed errors. This is one of the main reasons for using OLS to fit a linear model.

In general, for cases where we don't have a simple linear regression, or where we don't assume a normal distribution for errors, there may not be an exact solution as there is in the case above. In these cases we need some numerical search procedure. For this exercise, you will explore numerical calibration using logistic regression, which is only slightly more complicated than linear regression but does not have an analytical solution.

As explained earlier, logistic regression is very commonly used to model the probability of binary outcomes based on some predictor variables. Since each outcome y_i takes the value of 0 or 1, it can be interpreted as a Bernouilli random variable, whose probability of success (i.e. $y_i = 1$) is a logistic function.

The probability mass function (PMF) –remember the PMF is equivalent to a probability density function (PDF) but for discreet variables– for Bernouilli random variables is:

$$f(y|p) = p^y (1-p)^{1-y} \text{ where } y = 0 \text{ or } 1$$

Replacing the probability of the Bernouilli outcome by the logistic fucntion $f(z)$, we get:

$$f(y|f(z)) = f(z)^y(1 - f(z))^{1-y}$$

Therefore the likelihood of observing all the data is:

$$\mathcal{L}(z) = \Pi_{i=1}^{n} f(z_i)_i^y \, (1 - f(z_i))^{1-y_i}$$

Taking the *log* of the likelihood we get the log-likelihood function

$$\log \mathcal{L}(z) = \sum_{i=1}^{n} -y_i \log \left(f(z_i)\right) + (1 - y_i) \log \left(1 - f(z_i)\right)$$

We can replace $f(z)$ by the full logistic function equation and do some simplifications. Then given a matrix X composed of n observations of k predictor variables (i.e. X_{11} is the 1st observation of the 1st predictor variable, X_{12} is the 1st observation of the 2nd predictor variable etc. . .), and a vector Y of n observations of the response varivale, the log-likelihood function is shown below. Note that there are actually $k + 1$ parameters, since there is also a parameter for the intercept β_0. This is why the index i below goes from 0 to k.

$$\log \mathcal{L}(\beta_0, ..., \beta_k) = -\sum_{i=1}^{n} (1 - Y_i) \sum_{j=0}^{k} \beta_j X_{ij} - \sum_{i=1}^{n} \log \left(1 + e^{-\sum_{j=0}^{k} \beta_j X_{ij}}\right)$$

The parameters $\beta_0, ..., \beta_k$ that maximize the log-likelihood function above, can then be input into the logistic function $f(\beta_0, ..., \beta_k) = \dfrac{1}{1 + e^{-\beta_0, ..., \beta_k}}$. This cuvre represents the "best" logistic model: the one that maximizing the probability of observing our data.

Logistic regression also has the excellent property of having a fairly intuitive interpretation for the parameters it outputs—a rarity among statistical classifiers. Each parameter β_j can be interpreted as the relative change in log odds for a unit change in X_j. The odds of success represent the ratio of probability of success to probability of failure: $\text{Odds}(X_i) = \frac{Pr(Y_i=1|X_i)}{Pr(Y_i=0|X_i)}$.

$$exp(\beta_j) = \frac{\text{Odds}(X_{i,1}, ..., X_{i,j-1}, [X_{i,j} + 1], X_{i,j+1}, ..., X_{i,k})}{\text{Odds}(X_{i,1}, ..., X_{i,j-1}, [X_{i,j}], X_{i,j+1}, ..., X_{i,k})}$$

In other words, if parameter β_1 has a value of 1.2, it means that 1 unit change in X_1 while others predictor variables stay constant produces 1.2 unit change in log of the odd of success. Taking the exponential of β_1, a 1 unit change in X_1 produces an odds change of $e^{1.2} = 3.32$ in the outcome.

Exercises

Problem 1:

In this part of the assignment, you will implement a set of functions to carry out a logistic regression. These functions should match the following interface:

```
# implements log likelihood function
log.likelihood(params, x, y)

# optimizes log likelihood function given a training set
logistic.fit(xtrain, ytrain)

# makes a prediction given a set of parameters and observations
logistic.predict(xtest, fit)
```

We start you out with the first function. If you look closely, you'll see that each line of the function, starting with `Bx.sum = ...`, corresponds to one chunk of the log-likelihood function presented above.

```
log.likelihood = function(params, x, y) {
    # Defines the log-likelihood function for logistic regression
    # Arguments:
    #   params: vector of parameters. Supply k+1 parameters
    #   x: matrix with n observations of k predictor attributes
    #   y: vector of n observations on the binomial response variable
    # Returns:
    #   The negative of the log-likelihood function to be
    #   maximized with the "optim" function in logistic_fit

    x = cbind(rep(1, nrow(x)), x)  # Add column of 1s to get intercept
    Bx.sum = params %*% t(x)   # Multiply parameters by transpose of data
    t1 = sum((1-y)*Bx.sum)      # 1st term
    t2 = sum(log(1+sapply(-Bx.sum, exp, simplify=T)))    # 2nd term
    # Note that the `sapply()` function applies the exponential operator to each indice of -Bx.sum.
    # Look up ?sapply for details
    likelihood = -(t1+t2)
    return(-likelihood)     # return the negative liklihood, since the optim function minimizes
}
```

The second and third functions are left for you to implement. The second should use the `optim()` function in `R` to maximize the log-likelihood. Since `optim()` performs a minimization by default, our like.likelihood function above returns the negative log-likelihood. `optim()` is used as follows:

```
optim(starting values, log-likelihood, data, method='BFGS')
```

where the starting values are a vector of suggested parameter values from which to begin optimizing. These aren't very important here, so just choose something reasonable like zero for all of them. The log-likehood parameter will be the name of the function implemented above (`log.likelihood`), and data will be the training data variables corresponding to each of the parameters. They must be enumerated explicitly, e.g., `optim(..., x=xtrain, y=ytrain)`.

Problem 2:

Once you have the functions working (ask questions in Slack if you aren't getting it to work), create a single vector of length 200, in which the first half is distributed as $N(-5, 5)$, and the second half is $N(4, 8)$. Form a data.frame with this vector. It will be your predictor variable (i.e. X). Note that `log.likelihood` will add a column of ones for the intercept. Create another vector, and assign the first half 0 and the second half 1. This is the response vector (i.e. Y). Run your logistic regression. Plot the data you have created, and then plot the line that is described by the coefficients of your model, i.e., the output of `logistic.fit`. Label the axes appropriately. What does this plot show? Note: you can check that your functions are working right with the built-in `glm` function. Look up the function to see what parameters it takes. The `family` parameter should be set to `binomial('logit')`. The matrix of predictor variables fed into `glm` should *not* include the column of ones we included for our functions—it should just be a vector.

Problem 3:

In this portion of the assignment, you will try out your logistic regression on a real benchmark dataset of much greater complexity. We've provided a data-set of landslide occurence and potential predictor variables. The data is structured as following:

- `slope`: slope angle (degrees).
- `cplan`: plan curvature (rad m^{-1}) expressing the convergence or divergence of a slope and this water flow.

- `cprof`: profile curvature (rad m^{-1}) as a measure of flow acceleration, also know as downslope chanel in slope angle.
- `elev`: normalized elevation (no standard units because it is normalized) as the representation of different altitudinal zones of vegetation and precipitation in the study area.
- `log10_carea`: the decadic logarithm of the catchment area (log10 m2) representing the amount of water flowing towards a location.
- `landslides`: 1 (landslide occured), 0 (no landslide). This is the variable we will try to model and predict.

Use the first quarter of observations as your test set. Give a list of the coefficient estimates and provide a 2x2 table showing how many of your predicted positives/negatives (from the test set) were predicted correctly, and how many incorrectly (that is, the left side of the table will be labeled something like, "Predicted positive/Predicted negative", while the top will be labeled something like "Observed positive/observed negative"). How successful is your model? What are some potential limitations of this model.

If you want to keep playing around with logistic prediction, try predicting heart-attack occurence from the data made available from Hastie and Tibshirani's book, Elements of Statistical Learning. It is a subset of the Coronary Risk-Factor Study, which was conducted in South Africa:
https://web.stanford.edu/~hastie/ElemStatLearn/datasets/SAheart.data
Dataset description here: https://web.stanford.edu/~hastie/ElemStatLearn/datasets/SAheart.info.txt

Problem 4 (for graduate students only):

In this assignment, we learned about logistic regression, one type of generalized linear model (GLM). Learn more about the type of generalized linear models you can fit using the 'glm' function in R.

- Can you use a GLM on the data you selected for the project to answer your research question? If yes, describe a formula and family you would use in the R call. If not, explain why GLM may not be a good structure to use for your analysis.
- Provide a handdrawn sketch of the conceptual model (inputs, parameters, output). Are there any alternative models that other people have used?