Lab 3: Logistic Regression

PSTAT 131/231, Spring 2018

Learning Objectives

- Theoretical background of logistic regression model
  - The reason that we use link functions
  - Log-odds
  - Model assumptions
- Fit logistic model using `glm()`
  - Use `glm()` to fit the model
  - Coefficients interpretation
  - Use `predict()` to obtain $p(X)$
  - Construct confusion matrix
- Receiver Operating Characteristic (ROC) curve
  - Use `prediction()` to transform input data into a standardized format
  - Use `performance()` to get FPR and FNR
  - Construct ROC curve
  - Compute the Area under the Curve (AUC)

---

1. Obtain dataset *Discrim* and raise several questions

(a) Dataset description

Discrim is a simulated dataset containing $n = 28$ job interview outcomes of a company on $p = 4$ features.

- `HIRING`: response variable with two levels, "1" stands for YES and "0" for NO
- `EDUCATION`: years of college education, three values are available
- `EXPERIENCE`: years of working experience
- `GENDER`: "1" for MALE and "0" for FEMALE

```r
# Read the txt file from your current working directory
Dis = read.table("Discrim.txt", header=T)
# Convert Dis into a data frame
Dis = as.data.frame(Dis)
str(Dis)
```

```
## 'data.frame':    28 obs. of  4 variables:
##  $ HIRING    : int  0 0 1 1 0 1 0 0 0 1 ...
##  $ EDUCATION : int  6 4 6 6 4 8 4 4 6 8 ...
##  $ EXPERIENCE: int  2 0 6 3 1 3 2 4 1 10 ...
##  $ GENDER    : int  0 1 1 1 0 0 1 0 0 0 ...
```

Since this dataset contains only 28 observations, therefore this time we do not split it into training and test sets. We will fit the logistic regression model on the whole data (training set) and create a small new dataset in order to make predictions (test set)[1]. Notice that `HIRING` and `GENDER` are integer-value coded, therefore we have to discretize them.

[1] If the sample size of a dataset is too limited to be divided, you can use the whole dataset as the training, and construct an artificial test set.

```
# install.packages("dplyr")
library(dplyr)
Dis = Dis %>%
  mutate(HIRING=as.factor(ifelse(HIRING==0,"No", "Yes"))) %>%
  mutate(GENDER=as.factor(ifelse(GENDER==0,"F", "M")))
str(Dis)
```

```
## 'data.frame':    28 obs. of  4 variables:
##  $ HIRING    : Factor w/ 2 levels "No","Yes": 1 1 2 2 1 2 1 1 1 2 ...
##  $ EDUCATION : int  6 4 6 6 4 8 4 4 6 8 ...
##  $ EXPERIENCE: int  2 0 6 3 1 3 2 4 1 10 ...
##  $ GENDER    : Factor w/ 2 levels "F","M": 1 2 2 2 1 1 2 1 1 1 ...
```

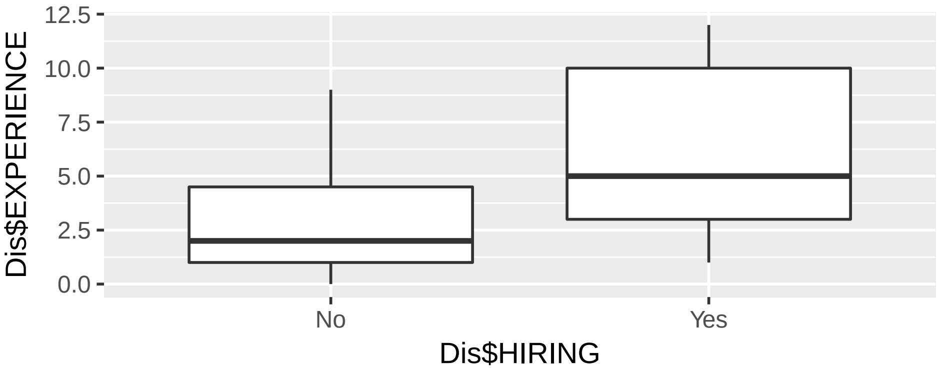
Let's check some explanatory analysis on the dataset.

```
table(Dis$GENDER,Dis$HIRING)
```

```
    No Yes
  F 12   3
  M  7   6
```

- Among 15 *FEMALE* applying, 3 have been hired (20%)
- Among 13 *MALE* applying, 6 have been hired (46.15%)

```
# install.packages("ggplot2")
library(ggplot2)
# Use qplot to make a boxplot of EXPERIENCE vs GENDER
qplot(Dis$HIRING, Dis$EXPERIENCE, data = Dis, geom = "boxplot")
```



- Among 28 candidates, those who got hired tend to have higher *EXPERIENCE* values.

(b) Interesting questions

Based on the dataset, we may pose some intriguing questions like

- Why is a logistic regression model better than a linear one?

- What is the probability of being hired given some features of candidates (`EDUCATION`, `EXPERIENCE` and `GENDER` of a candidate)?

- Does each predictor actually have impact on the estimated probabilities in the logistic model?
- Can we formally verify (with a test) whether there is evidence of discrimination based on Gender?
-

2. Logistic Regression

(a) Review the theoretical background

If we are dealing with a binary classification problem, how should we model the relationship between $p(X) = Pr(Y = 1|X_1, ..., X_p)$ and $X_i's$? (Notice that we are using the generic 0/1 coding for the response) One possible answer may be estimating $p(X) = Pr(Y = 1|X_1, ..., X_p)$ by linear model, that is

$$p(X) = \beta^T X = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p$$

However we see the problem with this approach: the left-hand-side, $p(X) = Pr(Y = 1|X_1, ..., X_p)$, must fall between 0 and 1 because it's intrinsically a probability; yet if we were to predict for the right-hand-side, we would get values bigger than 1 or less than 0. Any time a straight line is fit to a binary response that is coded as 0 or 1, in principle we can always predict $p(X) < 0$ for some values of X and $p(X) > 1$ for others (unless the range of X is limited).

To avoid this problem, we must model $p(X)$ using a function that gives outputs between 0 and 1 for all values of X. In logistic regression, we use the logistic function

$$p(X) = \frac{e^{\beta^T X}}{1 + e^{\beta^T X}} = \frac{e^{\beta_0 + \beta_1 X_1 + ... + \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X_1 + ... + \beta_p X_p}}$$

Notice the right-hand-side fraction now yields probability strictly between 0 and 1. It is never below 0 or above 1. In addition, the logistic function will always produce an S-shaped curve, and so regardless of the value of $X_i's$, we will obtain a sensible prediction. We also see that the logistic model is better able to capture the range of probabilities than is the linear regression model in the binary classification setting.

After a bit of manipulation and taking the logarithm, we arrive at

$$log\Big(\frac{p(X)}{1 - p(X)}\Big) = \beta^T X$$

The left-hand-side is called the **log-odds** or **logit**, which is the link function in a logistic regression.

(b) Build and summarise a logistic regression model

- **`glm()`** is used to fit generalized linear models. The usage of `glm()` is pretty much like that of `lm()` with one more necessary argument `family`. Specifying **`family=binomial`** produces a logistic regression model. By default, `family=binomial` uses logit as its link function. More options such as probit, log-log link are also available. As described previously, `HIRING` is our response and `EDUCATION`, `EXPERIENCE` and `GENDER` are predictors.
- **`summary()`** is a generic function that is used to produce result summaries of various model fitting functions. We can call the `summary()` of our glm object after fitting it and expect several things to be reported:
  - *Call*: this is R reminding us what the model we ran was, what options we specified, etc

- *Deviance residuals*: measures of model fit. This part of output shows the distribution of the deviance residuals for individual cases used in the model

- *Coefficients*: shows the coefficients, their standard errors, the Z-statistic (sometimes called a Wald Z-statistic), and the associated p-values

- *Fit indices*: goodness-of-fit measures including the null and deviance residuals, and the AIC.

```r
# Specify ’family=binomial’ is important!
glm.fit = glm(HIRING ~ EDUCATION + EXPERIENCE + GENDER,
              data=Dis, family=binomial)
# Summarize the logistic regression model
summary(glm.fit)
```

```
##
## Call:
## glm(formula = HIRING ~ EDUCATION + EXPERIENCE + GENDER, family = binomial,
##     data = Dis)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4380  -0.4573  -0.1009   0.1294   2.1804
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -14.2483     6.0805  -2.343   0.0191 *
## EDUCATION     1.1549     0.6023   1.917   0.0552 .
## EXPERIENCE    0.9098     0.4293   2.119   0.0341 *
## GENDERM       5.6037     2.6028   2.153   0.0313 *
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 35.165  on 27  degrees of freedom
## Residual deviance: 14.735  on 24  degrees of freedom
## AIC: 22.735
##
## Number of Fisher Scoring iterations: 7
```

(c) Interpret coefficients

In above results, Both `EXPERIENCE` and `GENDERM` are statistically significant at level 0.01. Let’s take a look at the interpretation of the model coefficients. The logistic regression coefficients, if logit link function is used, give the change in the log odds of the outcome for a one unit increase in a predictor variable, while others being held constant. Therefore:

- The variable `EXPERIENCE` has a coefficient 0.9098. For every one unit change in `EXPERIENCE`, the log odds of getting hired (versus not-hired) increases by 0.9098, holding other variables fixed

- The variable `EDUCATION` has a coefficient 1.1549. For a one unit increase in `EDUCATION`, the log odds of being hired increases by 1.1549, holding other variables fixed

- The indicator variable for `GENDERM` has a slightly different interpretation. The variable `GENDERM` has a coefficient 5.6037, meaning that the indicator function of `MALE` has a regression coefficient 5.6037. That being said, the gender `MALE` versus `FEMALE`, changes the log odds of getting hired by 5.6037.

(d) Construct confusion matrix for the training data

- As a reminder from Lab03, we discussed how to use **`predict()`** function to get the actual class predictions of a classification tree. There, we specified **`type="class"`** because the response variable `High` was binary. However, in the context of a logistic regression problem, even though we still have a binary response (like `HIRING`), we are more inclined to use **`type="response"`** in `predict()` function to get a sequence of probabilities. To predict the classes of the response from the sequence of probabilities, we will select the best threshold value in (3d). In the lecture, Prof. Oh showed how to use `fitted()` function to get the model's estimated hiring probabilities. Notice that both ways (`fitted()` and `predict()`) are good to go, commonly you have to make it clear that **`type="response"`**.

```
# Specify type="response" to get the estimated probabilities
prob.training = predict(glm.fit, type="response")

# Round the results to a certain number of decimal places by round(). For instance,
# we can round prob.training to 2 decimal places
round(prob.training, digits=2)
```

```
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 0.00 0.02 0.98 0.73 0.00 0.09 0.10 0.00 0.00 0.98 0.10 0.39 0.00 0.28 0.63
##   16   17   18   19   20   21   22   23   24   25   26   27   28
## 0.02 0.64 0.31 0.04 0.04 0.01 0.15 0.99 0.19 0.02 0.31 0.99 0.97
```

- We want to construct the confusion matrix for the training data. The true labels of `HIRING` (either No or YES) are available in the dataset, but how do we obtain the predicted labels from the model's estimated hiring probabilities, i.e., from `prob.training`?

  The answer is a two-step approch: pick the threshold value in order to assign labels, then assign the labels. Some natural questions arise, for instance, is 0.5 always our best choice? Let's keep this question in mind for a while and use $p_{threshold} = 0.5$ first as an example to complete the confusion matrix construction.

```
# Save the predicted labels using 0.5 as a threshold
Dis = Dis %>%
  mutate(predHIRING=as.factor(ifelse(prob.training<=0.5, "No", "Yes")))
# Confusion matrix (training error/accuracy)
table(pred=Dis$predHIRING, true=Dis$HIRING)
```

```
##      true
## pred  No Yes
##   No  18   2
##   Yes  1   7
```

  - Out of 28 cases, the model classifies $18 + 7 = 25$ correctly (89.29%)
  - Out of 19 NOT HIRED, the model classifies 18 correctly (94.74%)
  - Out of 9 HIRED, the model classifies 7 correctly (77.78%)

(e) Estimate hiring probabilities for the new data

To have a better understanding of the model, let's predict the hiring probabilities for the new cases defined in below as `test`.

```
# Create a new 'test' set for prediction
test = data.frame("EDUCATION"=c(4,4,4,4,8,8,8,8),
                  "EXPERIENCE"=c(6,6,10,10,6,6,10,10),
                  "GENDER"=c("F","M","F","M","F","M","F","M"))

# Predict the hiring probabilities and round the predict results to 5 decimals
prob.test = round(predict(glm.fit, test, type="response"),digits=5)
test = test %>%
  mutate(Probability=prob.test)
test
```

```
##   EDUCATION EXPERIENCE GENDER Probability
## 1         4          6      F     0.01522
## 2         4          6      M     0.80753
## 3         4         10      F     0.37046
## 4         4         10      M     0.99378
## 5         8          6      F     0.61061
## 6         8          6      M     0.99766
## 7         8         10      F     0.98352
## 8         8         10      M     0.99994
```

Several conclusion can be drawn here. For example, when `EXPERIENCE` is low, `MALE` have much higher probabilities of being hired. Differences diminish for higher levels of `EDUCATION` and `EXPERIENCE`.

(f) More analysis for fun

- *MALE* group:

  ```
  table(pred=Dis$predHIRING[Dis$GENDER=="M"],true=Dis$HIRING[Dis$GENDER=="M"])
  ```

  ```
       true
  pred  No Yes
    No   6   1
    Yes  1   5
  ```

- *FEMALE* group

  ```
  table(pred=Dis$predHIRING[Dis$GENDER=="F"],true=Dis$HIRING[Dis$GENDER=="F"])
  ```

  ```
       true
  pred  No Yes
    No  12   1
    Yes  0   2
  ```

3. Construct ROC curve and compute AUC

(a) Types of errors

False positive rate: The fraction of negative examples that are classified as positive. In our analysis, *FPR* is the portion of no-hiring candidates (No) that are classified as hiring (Yes).

False negative rate: The fraction of positive examples that are classified as negative. In our analysis, *FNR* is the portion of hiring candidates (Yes) that are classified as not hiring (No).

We can change the two error rates by changing the threshold from 0.5 to some other value in [0, 1]:

$$P(\text{HIRING=Yes}|\text{EXPERIENCE,EDUCATION,GENDER}) \geq \text{threshold}$$

and vary *threshold*.

(b) Construct ROC curve

The *ROCR* package can be used to produce ROC curves, like those you have seen in lecture. Specifically, **prediction()** and **performance()** are the most important two functions for generating a ROC curve. Every classifier evaluation using ROCR starts with creating a prediction object.

- **prediction()** is used to transform the input data (which can be in vector, matrix, data frame, or list form) into a standardized format. The first argument is the predicted probabilities obtained from the training set, the second is true labels.
- **preformance()** is to perform all kinds of predictor evaluations. Here, specifically, we use it to get the True Positive Rate by writing measure=tpr and False Positive Rate by measure=fpr.

Firstly, we transform the predicted probabilities for the training set into a standardized format used for ROC curve. The predicted probabilites are saved as prob.training in section (2d).

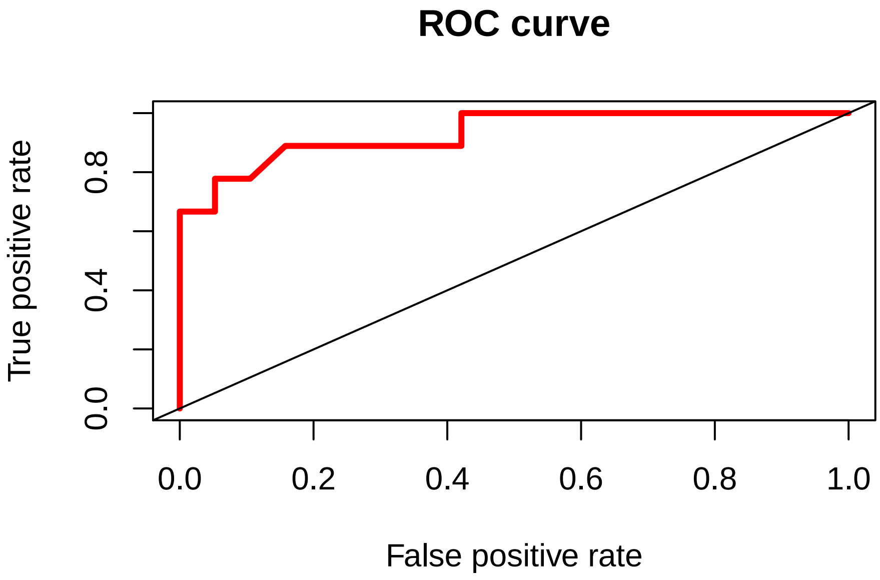
```
# install.packages("ROCR")
library(ROCR)

# First arument is the prob.training, second is true labels
pred = prediction(prob.training, Dis$HIRING)
```

Secondly, we calculate the True Positive Rate and False Positive Rate by performance().

```
# We want TPR on the y axis and FPR on the x axis
perf = performance(pred, measure="tpr", x.measure="fpr")
```

Lastly, plot the object you obtained above, and you will have the ROC curve!

```
plot(perf, col=2, lwd=3, main="ROC curve")
abline(0,1)
```



If you want to practice generating ROC curve manually, use the following code.

```
tpr = performance(pred, "tpr")@y.values[[1]]
fpr = performance(pred, "fpr")@y.values[[1]]
```

```
plot(fpr, tpr, type="l", col=3, lwd=3, main="ROC curve")
abline(0,1)
```

(c) Compute AUC

Often we use the **AUC** or **area under the curve** to summarize the overall performance of a model. Higher AUC is better. We can use `performance()` again to compute AUC as follows:

```
# Calculate AUC
auc = performance(pred, "auc")@y.values
auc
```

```
## [[1]]
## [1] 0.9327485
```

(d) Determine the best threshold value

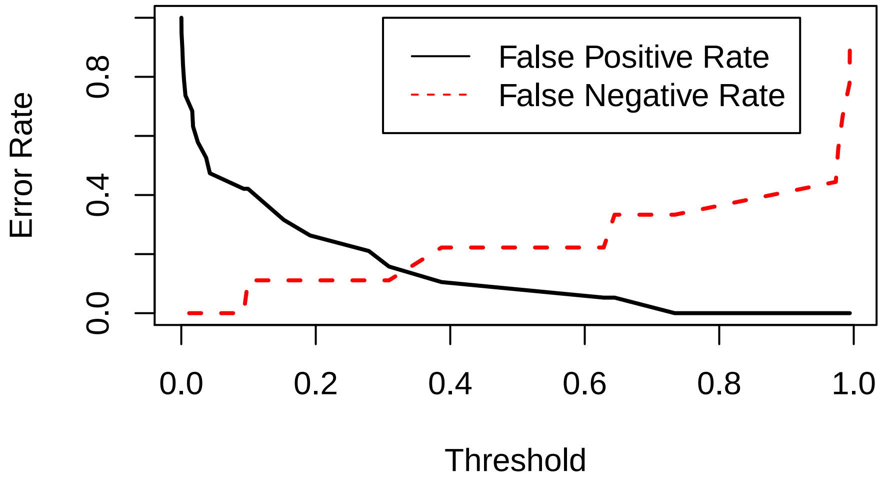
We want to control **False Negative Rate** and **False Positive Rate** to be as small as possible at the same time. Therefore, we'd like to choose probability threshold that is closest to $(FPR, FNR) = (0, 0)$. There are multiple ways to determine which threshold value corresponds to the smallest distance from (FPR, FNR) to $(0, 0)$. One possible choice is to calculate the euclidean distance between each point of (FPR, FNR) and $(0, 0)$.

- Obtain FPR and FNR from `performance()` output:

  ```
  # FPR
  fpr = performance(pred, "fpr")@y.values[[1]]
  cutoff = performance(pred, "fpr")@x.values[[1]]
  # FNR
  fnr = performance(pred,"fnr")@y.values[[1]]
  ```

- Plot FPR and FNR versus threshold values using `matplot()`:

  ```
  # Plot
  matplot(cutoff, cbind(fpr,fnr), type="l",lwd=2, xlab="Threshold",ylab="Error Rate")
  # Add legend to the plot
  legend(0.3, 1, legend=c("False Positive Rate","False Negative Rate"),
         col=c(1,2), lty=c(1,2))
  ```



- Calculate the euclidean distance between (FPR,FNR) and (0,0)

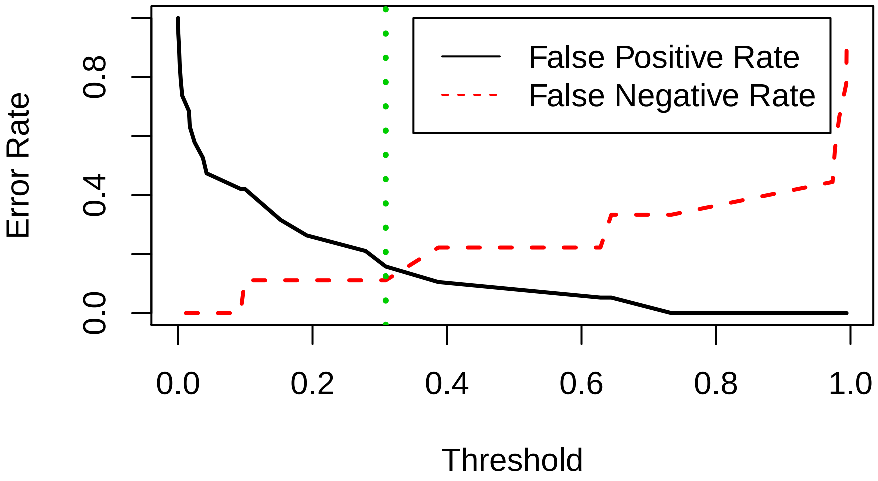
```
rate = as.data.frame(cbind(Cutoff=cutoff, FPR=fpr, FNR=fnr))
rate$distance = sqrt((rate[,2])^2+(rate[,3])^2)
```

- Select the probability threshold with the smallest euclidean distance

```
index = which.min(rate$distance)
best = rate$Cutoff[index]
best
```

```
## [1] 0.3088615
```

```
# Plot
matplot(cutoff, cbind(fpr,fnr), type="l",lwd=2, xlab="Threshold",ylab="Error Rate")
# Add legend to the plot
legend(0.35, 1, legend=c("False Positive Rate","False Negative Rate"),
       col=c(1,2), lty=c(1,2))
# Add the best value
abline(v=best, col=3, lty=3, lwd=3)
```



Therefore, our best cutoff value is 0.3089 which corresponds to the smallest Euclidean distance 0.1931. That being said, hiring probabilities less than 0.3089 should be predicted as **No** and higher than 0.3089 as *Yes*.

Your turn

- Compute the odds $\frac{\widehat{p(X)}}{1-\widehat{p(X)}}$ for *MALE* and *FEMALE* separately

```
prob.training
```

- Make boxplot of the odds versus each gender
- Compare the difference between the median odds for *MALE* and *FEMALE*
- State in plain words what the interpretation of odds are