https://rpubs.com/aelhabr/logistic-regression-tutorial

# Logistic Regression Tutorial (By Example)

*Tony*

*2017-11-14*

Perhaps the quickest manner of getting an understanding of logistic regression is by comparing it directly to linear regression. We'll do exactly that. First, we'll look at theory, then we'll move on to implementation. [1]

## Logistic Regression Theory

Both linear regression and logisitic regression are types of **generalized linear models** (GLMs). [2] Mathematically, GLMs can be expressed as

$$Y \mid X = x \sim N(\beta_0 + \beta_1 x_1 + \ldots + \beta_{p-1} x_{p-1},\ \sigma^2)$$

Y|X=x∼N(β0+β1x1+…+βp−1xp−1, σ2)

The logistic regression framework's inheritance of the GLM form is easy to see when observing the generalized equations for linear and logisitic regression. Recall that linear regression models are defined by the equation

$$Y = \beta_0 + \beta_1 x_1 + \ldots + \beta_q x_q + \epsilon, \quad \epsilon \sim N(0, \sigma^2)$$

Y=β0+β1x1+…+βqxq+ϵ,  ϵ∼N(0,σ2)

which calculates the response $Y$Y directly. Logisitc regression is defined in a similar manner

$$\log\left(\frac{p(x)}{1 - p(x)}\right) = \beta_0 + \beta_1 x_1 + \ldots + \beta_{p-1} x_{p-1}$$

log(p(x)1−p(x))=β0+β1x1+…+βp−1xp−1

However, in contrast to linear regression, note that the calculation of the response is not direct.

The side of the equation with the response variable is known as the **log odds**. [3]

In a binary context, the odds are the probability for a "positive" event $(Y = 1)$ (Y=1) divided by the probability of a "negative" event $(Y = 0)$ (Y=0). [4]

$$\frac{p(x)}{1 - p(x)} = \frac{P[Y = 1 \mid X = x]}{P[Y = 0 \mid X = x]}$$

p(x)1−p(x)=P[Y=1|X=x]P[Y=0|X=x]

The logistic regression equation guarantees that a value between $0$0 and $1$1 is calculated. This is evident the when the inverse logit transformation is applied, which results in a "direct" probability prediction.

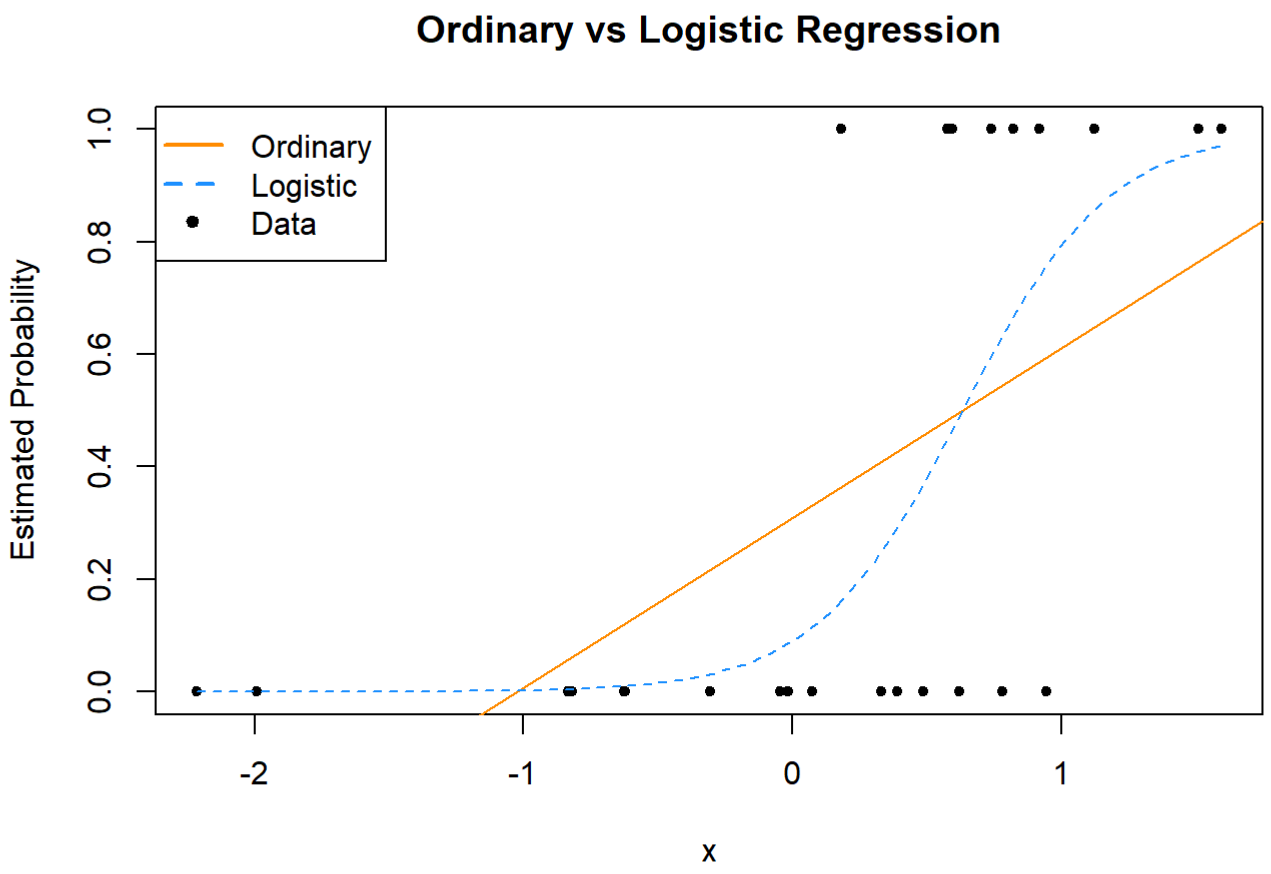
$$p(x_i) = P[Y_i = 1 \mid X_i = x_i] = \frac{e^{\beta_0 + \beta_1 x_{i1} + \cdots + \beta_{p-1} x_{i(p-1)}}}{1 + e^{\beta_0 + \beta_1 x_{i1} + \cdots + \beta_{p-1} x_{i(p-1)}}}$$

p(xi)=P[Yi=1|Xi=xi]=eβ0+β1xi1+⋯+βp−1xi(p−1)1+eβ0+β1xi1+⋯+βp−1xi(p−1)

Note that this is prediction of ***probability***, not a numerical value. This probability value must be translated to a categorical prediction.

## Theory Summary

So, what does this all mean? Put simply, linear regression should be used to predict a **quantitative**(i.e. numerical) response variable, while logisitic regression should be used to predict a **qualitiative**(i.e. categorical) response variable. (More generally, predicting a categorical response variable is known as **classification**.) Visually, the linear model generates a straight line, and the logisitic model generates an “S” curve.



## Logistic Regression Implementation in R

For this tutorial, we will use the `Default` dataset from the `ISLR` package.

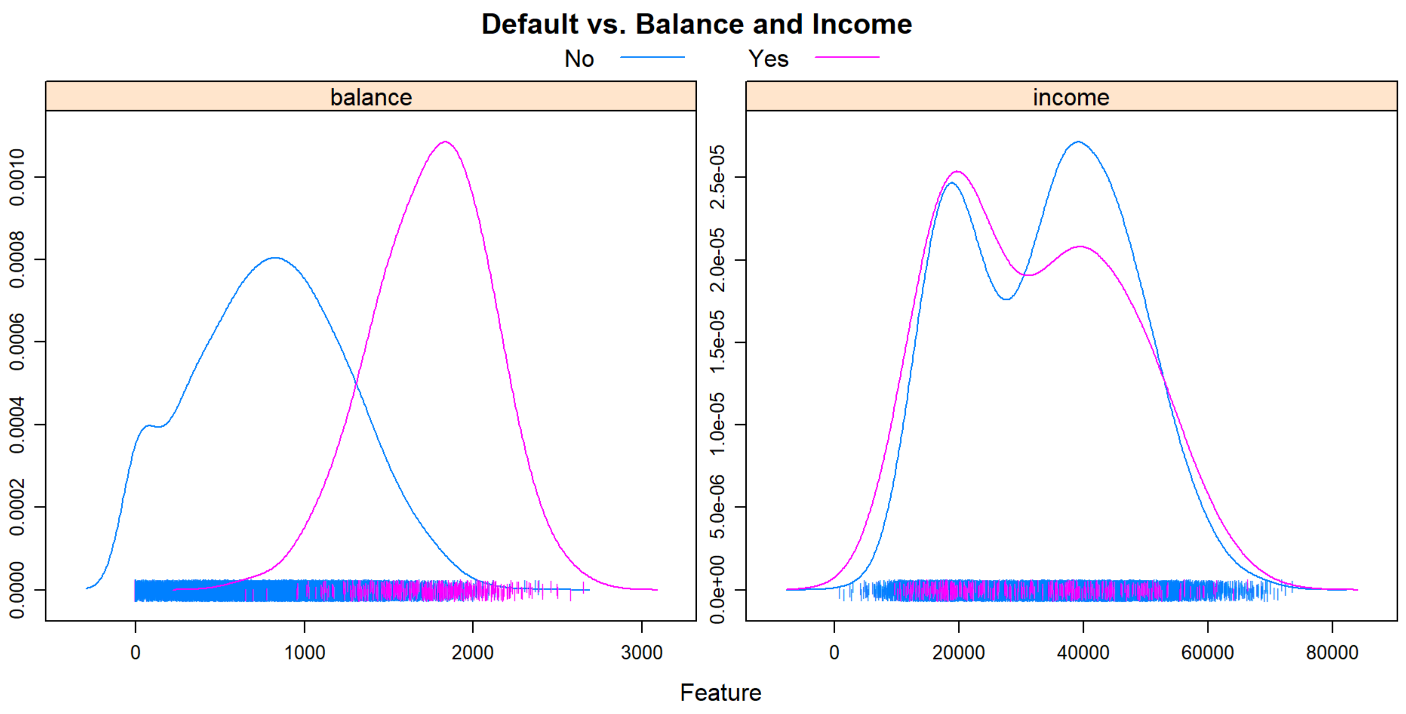
```r
library("ISLR")
library("tibble")
as_tibble(Default)
```

```
## # A tibble: 10,000 x 4
##    default student   balance    income
##     <fctr>  <fctr>     <dbl>     <dbl>
##  1      No      No  729.5265 44361.625
##  2      No     Yes  817.1804 12106.135
##  3      No      No 1073.5492 31767.139
##  4      No      No  529.2506 35704.494
##  5      No      No  785.6559 38463.496
##  6      No     Yes  919.5885  7491.559
##  7      No      No  825.5133 24905.227
##  8      No     Yes  808.6675 17600.451
##  9      No      No 1161.0579 37468.529
## 10      No      No    0.0000 29275.268
## # ... with 9,990 more rows
```

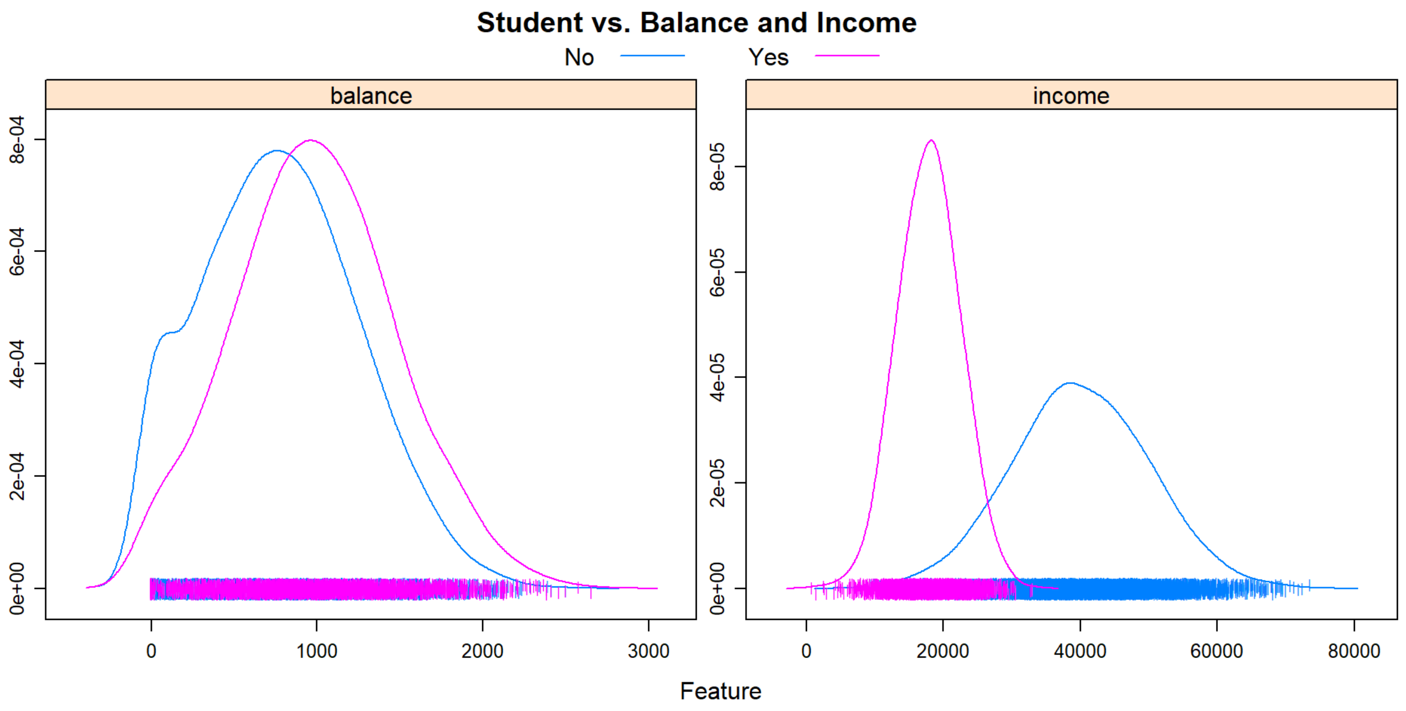
Our goal is to properly classify people who have defaulted based on student status, credit card balance, and income.

## Data Inspection

It is always a good idea to visualize our data before trying to build a model for it. For example, density plots are useful for identifying the distribution of the predictors relative to one another and to the response variable.



Observing that the distributions of `"No"` and `"Yes"` for `default` given `income` do not differ much, we might believe that `income` will not be particularly useful for our model. On the other hand, there seems to be a big difference in the `balance` distributions at a value around 1400 for the two categories of `default`.



We can also observe the distributions of `income` and `balance` with the other predictor `student`, which is qualitative (like `default`). These density plots indicate that students have much less income than the rest of the population.

## Logistic Regression Model Creation

Although we could do more to inspect the data, let's go ahead and create logistic regression model. To implement good modeling practices, we'll create training and testing splits in an attempt to avoid under/over-fitting when performing regression.

```r
# Split into train/test splits first.
set.seed(42)
default_idx <- sample(nrow(Default), ceiling(nrow(Default) / 2))
default_trn <-  Default[default_idx, ]
default_tst <- Default[-default_idx, ]

# Create the model.
model_glm <- glm(default ~ balance, data = default_trn, family = "binomial")
```

Creating a logistic regression model should look very similar to creating a linear regression model. However, instead of `lm()` we use `glm()`. Also, note that we must specify `family = "binomial"` for a binary classification context. (Actually, calling `glm()` with `family = "gaussian"` would be equivalent to `lm()`.)

Before making any predictions, let's briefly examine the model with `summary()`. Among other things, it is important to see what coefficient values have been estimated for our model.

```r
summary(model_glm)
```

```
##
## Call:
## glm(formula = default ~ balance, family = "binomial", data = default_trn)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2227  -0.1560  -0.0634  -0.0237   3.7286
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -10.452183   0.500042  -20.90   <2e-16 ***
## balance       0.005368   0.000306   17.54   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1470.42  on 4999  degrees of freedom
## Residual deviance:  830.61  on 4998  degrees of freedom
## AIC: 834.61
##
## Number of Fisher Scoring iterations: 8
```

Like the `summary` values calculated for a linear regression model, we get quantile information regarding residuals, as well significance estimates for our predictors. The interpretation of p-values in the logisitic regression framework is the same as that for linear regression models. (So the rule of "siginifcant" predictors being associated with `p-value`s less than 5 % also holds.)

However, note that we get a `z value` instead of a `t value`. Without getting too much into the theory regarding this difference, one should understand that this meaning of this value is analogous to that of a `t value`. [5] Also, note that we see `Null deviance`, `AIC`, and `Number of Fisher Scoring iterations` instead of `Residual standard error`, `Multipe R-squared`, `Adjusted R-squared`, and `F-statistic`.

The logistic regression diagnostic values generated by the `summary()` call are typically **not** used directly to interpret the "goodness of fit" of a model.

## Logistic Regression Model Prediction

However, before looking more closely at model diagnostics that are more suitable for logistic regression, we should first understand how to use the `predict()` function with `glm()`. In order to return probabilities, we must specify `type = "response"`. (The default setting is `type = "link"`, which corresponds to the log odds value.)

```r
head(predict(model_glm, type = "response"))
```

```
##         9149         9370         2861         8302         6415
## 9.466353e-04 3.298300e-01 7.437969e-03 8.170105e-05 1.183661e-04
##         5189
## 6.357530e-04
```

As mentioned before, these predicted values are probablliites, *not* classifications. We must "manually" convert the probabilities to classifications. Traditionally, a midpoint value such as 0.5 is used to "categorize" the probabilities. (This is actually equivalen to specifyng `type = "link"` and using a threshhold value of $0$0.)

```
trn_pred <- ifelse(predict(model_glm, type = "response") > 0.5, "Yes", "No")
head(trn_pred)
```

```
## 9149 9370 2861 8302 6415 5189
## "No" "No" "No" "No" "No" "No"
```

## Logistic Regression Model Evaluation

Probably the most common thing that is done to evaluate a classificiation models is to compare the actual response values with the predicted ones using a cross-table, which is often called a **confusion matrix**. This matrix can be generated with the base `table()` function.

```
# Making predictions on the train set.
trn_tab <- table(predicted = trn_pred, actual = default_trn$default)
trn_tab
```

```
##          actual
## predicted   No  Yes
##       No  4815  122
##       Yes   17   46
```

```
# Making predictions on the test set.
tst_pred <- ifelse(predict(model_glm, newdata = default_tst, type = "response") > 0.5, "Yes", "No"
)
tst_tab <- table(predicted = tst_pred, actual = default_tst$default)
tst_tab
```

```
##          actual
## predicted   No  Yes
##       No  4817  113
##       Yes   18   52
```

Perhaps unsurprisingly, the most common metrics for evaluating logistic regression models are **error rate** and **accuracy** (which is simply the additive inverse of the error rate). These metrics can be calculated directly from the confustion matrix.

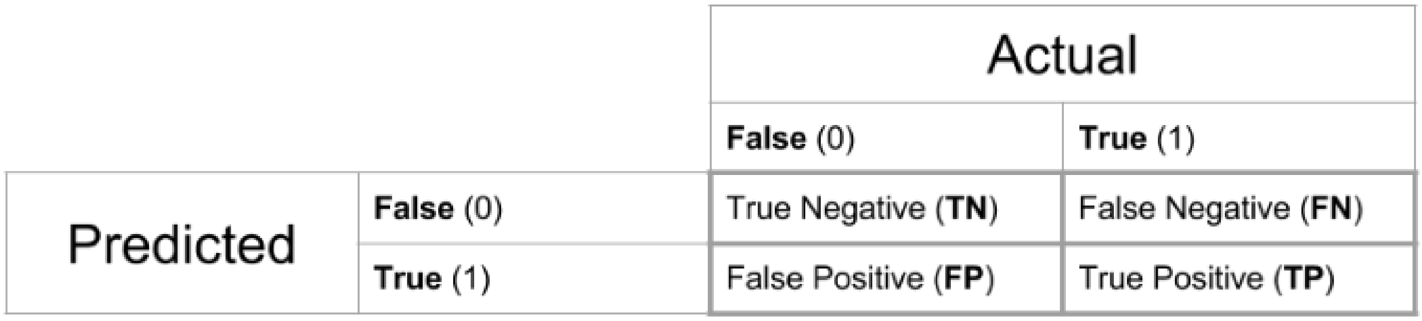
```
calc_class_err <- function(actual, predicted) {
  mean(actual != predicted)
}
```

```
calc_class_err(actual = default_trn$default, predicted = trn_pred)
```

```
## [1] 0.0278
```

```
# Test error rate should be close to train error rate if model is fitted properly.
calc_class_err(actual = default_tst$default, predicted = tst_pred)
```

```
## [1] 0.0262
```



| | | Actual | |
|---|---|---|---|
| | | **False** (0) | **True** (1) |
| Predicted | **False** (0) | True Negative (**TN**) | False Negative (**FN**) |
| | **True** (1) | False Positive (**FP**) | True Positive (**TP**) |

Now, let's consider the confusion matrix in more detail now. The names **true positive (TP)**, **true negative (TN)**, **false positive (FP)**, and **false negative (FN)** are often used to reference the four cells of the confustion matrix.

$$\text{Sens} = \text{True Positive Rate} = \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Sens=True Positive Rate=TPP=TPTP + FN

$$\text{Spec} = \text{True Negative Rate} = \frac{\text{TN}}{\text{N}} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

Spec=True Negative Rate=TNN=TNTN + FP

$$\text{Prev} = \frac{\text{P}}{\text{Total Obs}} = \frac{\text{TP} + \text{FN}}{\text{Total Obs}}$$

Prev=PTotal Obs=TP + FNTotal Obs

Calculations of metrics such as **sensitivity**, **specificity**, and **prevalance** are derived from the confusion matrix. The importance of these (and other) metrics is dependent on the nature of the data (e.g. lower values may be acceptable if the data is deemed difficult to predict), as well as the tolerance for the type of misclassification. For example, we may want to bias our predictions for classifying defaults such that we are more likely to predict a default when one does not occur. We must carefully identify whether we want to prioritize sensitivity or specificity.

We can get the values of sensitivity, specificity, prevalance, etc. easily for our predictions using the `confusionMatrix()` function from the `caret` package. [6]

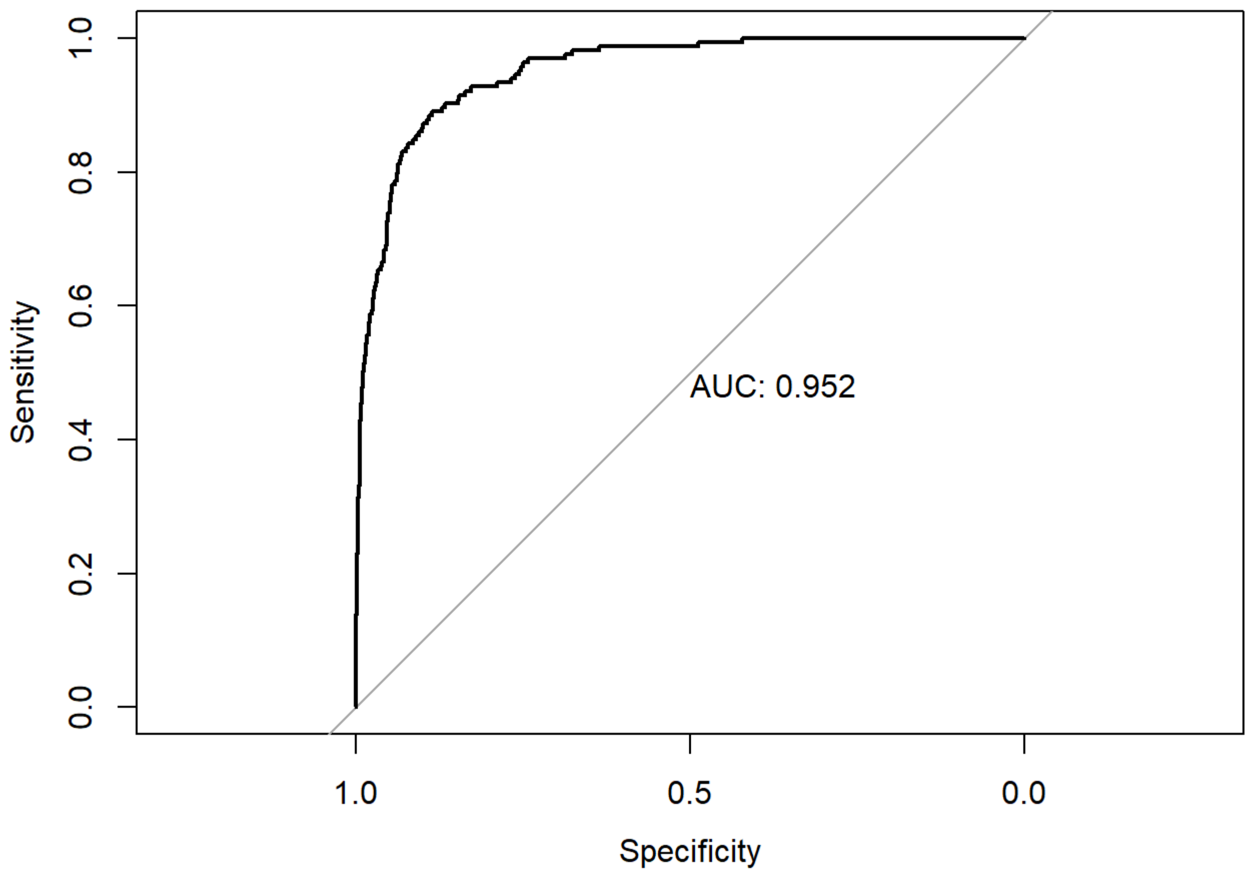
```r
library("caret")
confusionMatrix(trn_tab, positive = "Yes")
```

```
## Confusion Matrix and Statistics
##
##          actual
## predicted   No  Yes
##       No  4815  122
##       Yes   17   46
##
##                Accuracy : 0.9722
##                  95% CI : (0.9673, 0.9766)
##     No Information Rate : 0.9664
##     P-Value [Acc > NIR] : 0.01101
##
##                   Kappa : 0.387
##  Mcnemar's Test P-Value : < 2e-16
##
##             Sensitivity : 0.2738
##             Specificity : 0.9965
##          Pos Pred Value : 0.7302
##          Neg Pred Value : 0.9753
##              Prevalence : 0.0336
##          Detection Rate : 0.0092
##    Detection Prevalence : 0.0126
##       Balanced Accuracy : 0.6351
##
##        'Positive' Class : Yes
##
```

Now, let’s consider another metric, unrelated to the confusion matrix. Remember where we chose the value $0.5$0.5as a threshhold for classification? How do we know that $0.5$0.5 value is the “optimal” value for accuracy. In reality, other cutoff values may be better (although $0.5$0.5 will tend to be the best value if all model assumptions are true and the sample size is reasonably large).

The **ROC curve** (receiver operating characteristic curve) illustrates the sensitivity and specificity for all possible cutoff values. We can use the `roc()` function from the `pROC` package to generate the ROC curve for our predictions.

```r
library("pROC")
test_prob <- predict(model_glm, newdata = default_tst, type = "response")
test_roc <- roc(default_tst$default ~ test_prob, plot = TRUE, print.auc = TRUE)
```



```
as.numeric(test_roc$auc)
```

```
## [1] 0.9515076
```

In general, we would like the curve to “hug” the right and upper borders of the plot (indicating high sensitivity and specificity). The **AUC** (area under the curve) is used to quantify the visual profile of the ROC.

## “Simple” Logistic Regression Model Alernatives

### A Single-Variable Threshhold Classifier Model

Based on the density plots, we might have reasonably created a very simplistic model using only the `balance` predictor. In particular, we could create a model based on the rule

$$\hat{C}(\text{balance}) = \begin{cases} \text{Yes} & \text{balance} > 1400 \\ \text{No} & \text{balance} \leq 1400 \end{cases}$$

C^(balance)={Yesbalance>1400Nobalance≤1400

```
simple_class <- function(x, boundary, above = "Yes", below = "No") {
  ifelse(x > boundary, above, below)
}
```

```
tst_pred_basic <- simple_class(x = default_tst$balance, boundary = 1400)
```

Although this appears to be a reasonable model, it turns out that its prediction accuracy is actually worse than what we would calculate if we simply predicted that no one defaulted!

```
tst_pred_all_no <- simple_class(x = default_tst$balance, boundary = 1400, above = "No", below = "N
o")
```

(However, predicting all `"No"` s gives worst prediction accuracy than that for our logistic regression model on the test set. We get an error rate of `r calc_class_err(actual = default_tst$default, predicted = tst_pred)` for our model. Here, we
caclulate `r calc_class_err(actual = default_tst$default, predicted = tst_pred_all_no)` when predicting all `"No"` s, and `r calc_class_err(actual = default_tst$default, predicted = tst_pred_basic)` when using $1400$1400 as a cutoff value for our model based on a one-variable classifier.)

```
calc_class_err(actual = default_tst$default,
               predicted = tst_pred)
```

```
## [1] 0.0262
```

```
calc_class_err(actual = default_tst$default,
               predicted = tst_pred_basic)
```

```
## [1] 0.0994
```

```
calc_class_err(actual = default_tst$default,
               predicted = tst_pred_all_no)
```

```
## [1] 0.033
```

## Linear Regression Model

Of course, (even after reviewing the theory) we may return to the inevitable question **"Why not linear regression?"** The main issue, in practice, involves how to interpret the linear regression prediction values as probabilities and, subsequently, how to convert the numerical values to categories. To better understand this issue, let's go ahead and try a linear regression model. (Here, we create a model using all predictors, not just `balance` .)

```
##
## Call:
## lm(formula = default ~ ., data = default_trn_lm)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.23775 -0.07021 -0.02729  0.02013  0.98785
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -6.074e-02  1.202e-02  -5.052 4.53e-07 ***
## studentYes  -2.244e-02  8.131e-03  -2.759  0.00582 **
## balance      1.304e-04  5.033e-06  25.908  < 2e-16 ***
## income      -2.638e-07  2.741e-07  -0.963  0.33583
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1692 on 4996 degrees of freedom
## Multiple R-squared:  0.1193, Adjusted R-squared:  0.1187
## F-statistic: 225.5 on 3 and 4996 DF,  p-value: < 2.2e-16
```

For this case, if we create a linear regression model using all of the predictors, then all of the predicted probabilities are below 0.5. Thus, if we were to classify every observation below 0.5 as `"No"`, then all of our predictions would be `"No"`. Additionally, because the linear model returns predictions with values less than $0$0, we would have to figure out how to convert these values to categories.

```
all(predict(model_lm) < 0.5)
```

```
## [1] TRUE
```

```
any(predict(model_lm) < 0)
```

```
## [1] TRUE
```

Although we could classify the "upper half" of values into the positive category and the "lower half" of values to the negative category (in order to avoid the issue of interpreting the linear regressions model's numerical predictions), but we would run into another issue in the case that there are more than two categories for the response variable. Encoding of the different categories to numerical values can become subjective and bias the classifications in such a case. For example, we might encode different ethnicities as numerical values for a problem in which we are trying to predict a person's heritage, but who is to say one ethnicity is "closer" to another? (Are 1, 2, and 3 proper values to encode Asian, Hispanic, and European? Order matters!)

Anyways, it should be easy to see why the logistic regression framework is more appropriate for classification problems.

# Conclusion

That's it for now. There is certainly much more we could cover, both in terms of theory and implementation.

Notably, we haven't discussed a case where there are more than two categories for the response variable. In that case, we might use **multinomial logistic regression**. Probably the most commonly-used implementation is the `multinom()` function from the `nnet` package. However, we will leave this topic for another time.

# Appendix

GLMs have three components:

- A **distribution** of the response conditioned on the predictors.
- A **linear combination** of the $p-1$ predictors.
- A **link** function that defines how the linear combination of the predictors is related to the mean of the response conditioned on the predictors.

For reference purposes, the following table summarizes the differences between linear and logistic regression with respect to the broader GLM framework.

| | **Linear Regression** | **Logistic Regression** |
|---|---|---|
| **Distribution of** $Y \mid X = x$ | $N(\mu(x), \sigma^2)$ | $\text{Bern}(p(x))$ |
| **Distribution Name** | Normal | Bernoulli (Binomial) |
| $E[Y \mid X = x]$ | $\mu(x)$ | $p(x)$ |
| **Support** | Real: $(-\infty, \infty)$ | Integer: $0, 1$ |
| **Usage** | Numeric Data | Binary (Class: Yes/No) Data |
| **Link Name** | Identity | Logit |
| **Link Function** | $\eta(x) = \mu(x)$ | $\eta(x) = \log(\frac{p(x)}{1-p(x)})$ |
| **Mean Function** | $\mu(x) = \eta(x)$ | $p(x) = \frac{e^{\eta(x)}}{1+e^{\eta(x)}} = \frac{1}{1+e^{-\eta(x)}}$ |

Note that logistic regression models do not make the same assumption that linear regression makes, including:

- Linear relationship (between the response and the predictors)
- Multivariate normality (of the predictors)
- No or little multicollinearity (of the predictors)
- No auto-correlation
- Homoscedasticity

---

1. The following sites were used as resources for this tutorial:
   - https://daviddalpiaz.github.io/r4sl/index.html
   - http://daviddalpiaz.github.io/appliedstats/

   Some code and phrasing is taken directly from these resources.↩
2. See the appendix for more details regarding the theory discussed here.↩
3. The log odds are the logit transform applied to $p(x)$.↩
4. This means that when the odds are $1$, the two events have equal probability. Moreover, odds greater than $1$ indicate higher probabiliity for the positive event, while the converse is true for odds less than $1$.↩
5. (Recall that the $t$-value is calculated by a $t$-test to affirm (or deny) the null hypothsesis for a given predictor.)↩
6. Note that we could do a lot more with the `caret` package, which is perhaps R's most powerful machine learning package.↩