Logistic Regression in Tensorflow

What is Logistic Regression?

Logistic regression seeks to:

- **Model** the probability of an event occurring, depending on the values of the independent variables, which can be categorical or numerical.
- **Estimate** the probability that an event occurs for a randomly selected observation versus the probability that the event does not occur.
- **Predict** the effect of a series of variables on a binary response variable.
- **Classify** observations by estimating the probability that an observation is in a particular category (such as manual or automatic in our problem.)

Why Is a Using Linear Regression in a Classification Problem So Wrong?

- Binary data does not have a normal distribution, which is a condition needed for most other types of regression.
- Predicted values of the DV can be beyond 0 and 1, which
violates the definition of probability.

- Probabilities are often not linear, such as “U” shapes, where probability is very low or very high at the extremes of x-values.

Some Background

- **Probabilities**: outcomes of interest/all possible outcomes.
- **Odds**: \( \frac{P(\text{Occurring})}{P(\text{Not occurring})} \to \frac{P(\text{Occurring})}{(1-P(\text{Occurring}))} \)
- **Odds Ratio**: Ratio of two odds \( \to \frac{(P(\text{Occurring 1}))/(1-P(\text{Occurring 1}))}{(P(\text{Occurring 0}))/(1-P(\text{Occurring 0}))} \)

Odds Ratio Example

Assume we have two coins: one fair (probability of 0.5 for both sides) and one weighted (probability of 0.7 for heads and 0.3 for tails). What are the odds of getting “heads” on the weighted coin?

```r
p_f_h <- 0.5
p_f_t <- 0.5
p_w_h <- 0.7
p_w_t <- 0.3

odds_f_h <- p_f_h/(1-p_f_h)
odds_w_h <- p_w_h/(1-p_w_h)

odds_ratio <- odds_w_h/odds_f_h

print(odds_ratio)

## [1] 2.333333
```

This means that the odds of getting heads on the weighted coin are 2.33 greater than the fair coin.
The Odds Ratio in Logistic Regression

- The odds ratio for a variable in logistic regression represents how the odds change with a 1 unit increase in that variable holding all other variables constant.
- For example, let’s assume that we predict the probability of someone being positive for diabetes based on his weight. We find out that the odds ratio of weight is 1.25. This means that one kg increase in weight increases the odds for diabetes by 1.25. A 10 kg increase in weight would lead to a 1.25^10 increase of the odds. Two important points to note: 1) this holds true at any point in the weight spectrum. 2) Don’t confuse odds with probability.

Bernoulli Distribution

The dependent variable in logistic regression follows the Bernoulli distribution, having an unknown probability “p.”

- Remember that Bernoulli is only a special case of Binomial distribution where n = 1 (one experiment is conducted) – Success is 1 and failure is 0. The probability of success is “p” and failure q = 1 – p. In logistic regression, we are estimating an unknown “p” for any given linear combination of independent variables. The link between the independent variables and the Bernoulli distribution is called the logit.

Logit

- In logistic regression, we don’t know “p” like we do in Binomial distribution problems. The goal of logistic regression is to estimate “p” for a linear combination of independent variables.
- The natural log of the odds ratio, the logit, results in any value onto the Bernoulli probability distribution
between 0 and 1.
- Logit(p) = ln(p/(1-p)) OR logit(p) = ln(p) – ln(1-p).
- In the logit function, the argument takes values between 0 to 1, but we want the function to take those values. Therefore, we will take the inverse of the logit function: logit^{-1}(a) = 1/(1 + e^{-a}) = e^a /(1 + e^a). This is where “a” is some number and “e” is Euler’s number.
- In our case, “a” will be the linear combination of variables and their coefficients. The inverse-logit will return the probability of being a “1” or in the “event occurs” group.

Let’s illustrate the inverse logit. As you can see, it is the sigmoid function curve.

```r
library(ggplot2)
y <- function(a){
y = 1/(1 + exp(-a))
}
x = seq(-5,5,0.1)
ggplot() + geom_point(aes(x = x, y = y(x)))
```
Estimated Regression Equation

The natural algorithm of the odds ratio is equivalent to a linear function of the independent variables.

\[ \text{logit}(p) = \ln(p/(1-p)) = b_0 + b_1 x \]

The anti-log of the logit function allows us to find the estimation regression equation.

\[ p/(1-p) = e^{(b_0 + b_1 x)} \]
\[ p = (1-p) \times e^{(b_0 + b_1 x)} \]
\[ p = e^{(b_0 + b_1 x)} - p \times e^{(b_0 + b_1 x)} \]
\[ p(1 + e^{(b_0 + b_1 x)}) = p \times e^{(b_0 + b_1 x)} \]
\[ p = e^{(b_0 + b_1 x)}/(1 + e^{(b_0 + b_1 x)}) \]
**Estimated Regression Equation:** \( p^* = \frac{e^{(b_0 + b_1 x)}}{1 + e^{(b_0 + b_1 x)}} \)

**Note About Coefficients:**

- The regression coefficients for logistic regression are calculated using the Maximum Likelihood Estimation or Cross Entropy.

Let’s get our hands dirty and apply the knowledge we acquired in Tensorflow.

Learn more about neural networks in the online course Machine Learning A-Z™: Hands-On Python & R In Data Science. In this course, you will learn how to:

- Work with Deep Learning networks and related packages in R
- Create Natural Language Processing models
- And much more

We will use the ‘cat’ data set from the ‘MASS’ library.

The first thing is to create the data frame:

library(caret)

## Loading required package: lattice

## Warning: Installed Rcpp (0.12.14.5) different from Rcpp used to build dplyr (0.12.11).
## Please reinstall dplyr to avoid random crashes or undefined behavior.

library(MASS)
cat = cats[c("Sex", "Bwt")]
y = with(cat, model.matrix(~ Sex + 0))
x = as.matrix(cat[,2])
trainIndex = createDataPartition(x,
  p=0.7, list=FALSE,times=1)

x_train = as.matrix(x[trainIndex,])
x_test = as.matrix(x[-trainIndex,])
y_train = as.matrix(y[trainIndex,])
y_test = as.matrix(y[-trainIndex,])

Then we will define the placeholders for features and labels:

```r
library(tensorflow)
X <- tf$placeholder(tf$float32, shape(NULL, 1L))
Y = tf$placeholder(tf$float32, shape(NULL, 2L), name = "Y")
```

Having defined the placeholders, we will define the parameters. We will randomly initialize the weights with mean “0” and a standard deviation of “1.” We will initialize bias to “0.”

```r
W = tf$Variable(tf$random_normal(shape(1L,2L), stddev = 1.0), name = "wehghts")
b = tf$Variable(tf$zeros(shape(2L)), name = "bias")
```

Then we will compute the logit.

```r
logits = tf$add(tf$matmul(X, W), b)
pred = tf$nn$sigmoid(logits)
```

The next step is to define the loss function. We will use sigmoid cross entropy with logits as a loss function.

```r
entropy = tf$nn$sigmoid_cross_entropy_with_logits(labels = Y, logits = logits)
loss = tf$reduce_mean(entropy)
```

The last step of the model composition is to define the training op. We will use a gradient descent with a learning rate 0.1 to minimize cost.

```r
optimizer = tf$train$GradientDescentOptimizer(learning_rate = 0.01)$minimize(loss)
init_op = tf$global_variables_initializer()
```

Now that we have trained the model, let’s evaluate it:

```r
correct_prediction <- tf$equal(tf$argmax(pred, 1L), tf$argmax(Y, 1L))
```
accuracy <- tf$reduce_mean(tf$cast(correct_prediction, tf$float32))

Having structured the graph, let’s execute it:

with(tf$Session() %as% sess, {
  sess$run(init_op)
  for (i in 1:1000) {
    sess$run(optimizer, feed_dict = dict(X=x_train, Y=y_train))
  }
  sess$run(accuracy, feed_dict=dict(X = x_test, Y = y_test))
})

## [1] 0.7073171