National University Ireland, Galway

Logistic Regression

Machine Learning & Data Mining

ROBBIE DEEGAN
12-22-2017

Table of Contents

Overview

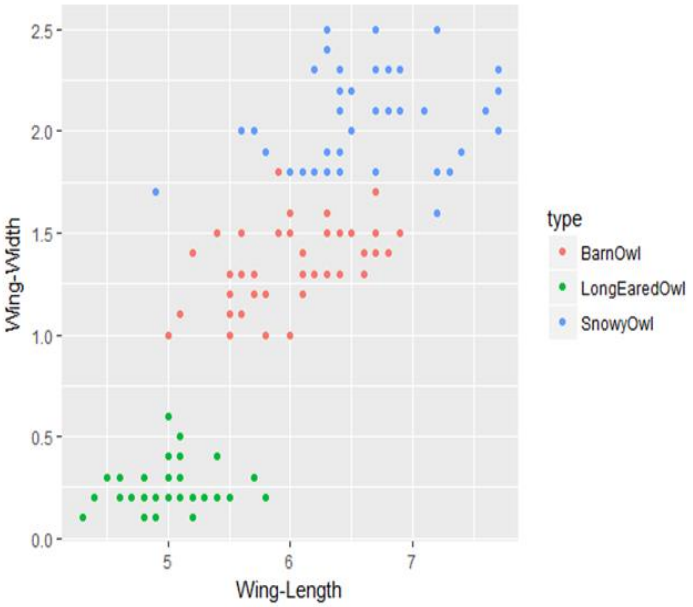
This report will document how I selected, implemented and evaluated a machine learning algorithm, Multinomial Logistic Regression, from scratch. I will carefully articulate the selection, implementation and evaluation of my model. There is an Appendix attached to this report where my code will be laid out. Around my code will be short notes that start with the '#' sign. These notes provide a brief explanation of the code to follow.

Design Decisions

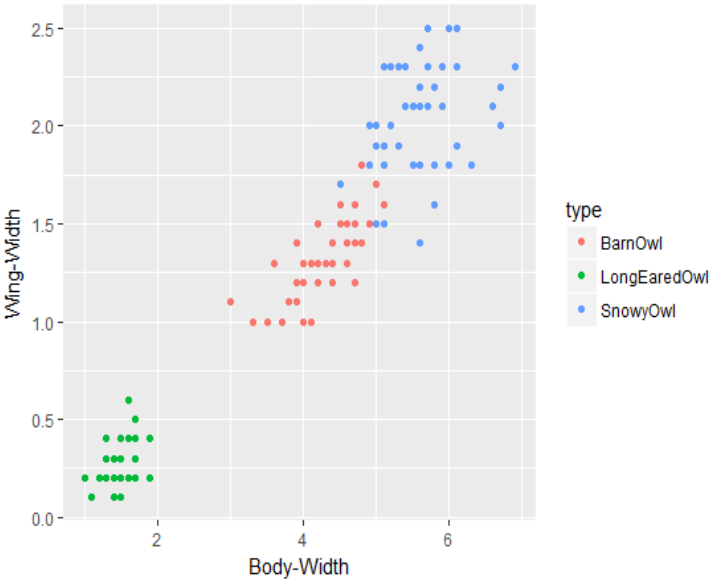
My predictive model was constructed to be able to handle different types of datasets. The ability to plug in different datasets with unique numbers of attributes and training cases. While my report will document results from one specific dataset, owls15, I have constructed the model in a way so that it can reproduce similar results with the above mentioned different datasets. There are three constraints to my model, which must be met in order for the model to accurately make a prediction.

1. All attributes are numeric
2. The last column is the target variable
3. Multinomial (3 or more classes in the predicted variable)

Selection



Choosing which algorithm to use is a critical step in machine learning. Due to the high variety of algorithms available it is critical you carefully select the best algorithm for your data. The dataset given to me by my Professors is "owls.csv". This dataset has information on three types of Owls that are found in Ireland, which are the Barn Owl, Snowy Owl and Long-Earned Owl. The information this dataset includes about the owls are four attributes, body-length, wing-length, body-width and wing-width. The target variable is the type of owl, which is a discrete value. This exploration of the dataset has allowed me to understand that this is a Classification problem. Classification refers to the model trying to predict a discrete valued output (Ng).



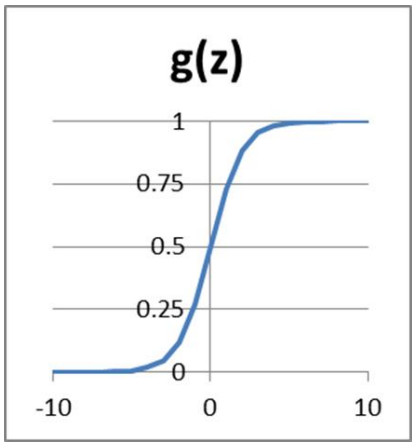
The goal of my classification model is to accurately predict and classify owls. David Wolpert's "no free lunch" theorem is based on the idea that no single machine learning algorithm is always best (Drury). After careful contemplation I decided that multinomial logistic regression was the algorithm I was going to implement. Logistic regression is, "a method for classifying data into discrete outcomes" (Ng). Since Logistic Regression has the word 'regression' in it, it is imperative to note that, "logistic regression is used for classification tasks, not for regression" (Madden).The hypothesis function, $h\theta(x)$, is a calculated probability for our output, y, given our parameters θ and input x (Madden). Using the cost function for $h\theta(x)$ we can optimize the function's parameters (Madden). In order to do this we will use gradient descent, which is an algorithm that minimizes functions given a set of parameter values by using calculus (Madden). The programming language I implemented throughout the entirety of this report is R. R is a popular statistically language that is built for data science (Wickham).

Implementation

Systematically, I will run through the steps I took while creating my machine algorithm model. I began by loading my data onto the R console. Once loaded I labelled the last column in the dataset "type". This reverts back to my design decisions in which I stated that the last column is the target variable. The remaining columns are renamed using the paste function. Each attribute column will be labelled "Predictor_" followed by the number which represents which column it is in the data set. Next I mutated my target variable column, 'type', into a factor of 1, 2 and 3. Followed by an accuracy vector, which is initialized with 0's. After the creation of my accuracy vector I needed to reproduce the Sigmoid function, also known as the Logistic function (Ng).

Either term can be used to describe this function, which is what logistic regression is built on (Ng). The sigmoidal function is calculated as hθ(x) = g(z) = 1/1+e^-z. In this calculation 'e' represents the natural logarithm base and 'z' represents real numbers (Madden).

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$



The Sigmoid function is used specifically for two reasons. The first one being that it will always give an output between 0 and 1 (Ng). Secondly, it gives us a sharp step function where the function rapidly raises from 0 to 1 (Madden). hθ(x) is a calculated probability for our output, y, given our parameters θ and input x (Madden). This allows logistic regression to always output probabilities. In order to generate this model automatically I wrapped the entirety of the function in a for loop. Each iteration of this model was generated using a different set.seed, which resulted in random iterations. Inside the for loop was my initial separation of the data. Dividing the data at random into training and testing sets. The training set had roughly 2/3s of the overall data, and the testing set had the remaining 1/3. Now it was time to create my predictor variables, which for the owls15 dataset are body-length, wing length, body-width and wing-width. It's important to note that I specifically used 'as.matrix' when creating these predictor variables. Matrices allow for easier multiplication between the theta parameters and the x values, which are the feature vectors, in order to find the square error cost function. This was a key step, as I needed to perform matrix multiplication on the variables later on. For matrix multiplication you take the matrix multiplication product for each theta knot, then run them through the sigmoid function but make sure to square the result. Then find the mean of the sigmoid functions^2 by doing1/m * the sum of the sigmoid function^2 <- squared error cost function. After I

created the predictor variables, pred_var and pred_test, I then added ones a column of 1's to both of pred_var and pred_test. This is implemented because we need to set a dummy variable x0 to be multiplied by the Θ0 (which is the intercept of the linear functionΘ.x). Now, it was time to create my response variables. For each of these classes I built a separate logistic classifier by taking a 'one versus all' approach (Ng). This dataset is multinomial so I needed to create a response variable for each class of owl. This means for each 'y' we take the class it represents and set it to 1, while taking the other remaining classes and setting them to 0.

The cost function is crucial to the calculation, and is defined as

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Note: $y = 0$ or 1 always

cost(hθ, (x),y) = -ylog(hθ(x)) - (1-y)log(1- hθ(x)) (Ng).

The average squared error cost function is used to arrive at the optimal set of parameters by minimizing it incrementally in steps (Ng). This cost function is concave, which helps us take a gradient descent approach where adjusting parameters Θ helps us move down the slope to arrive the minimum cost function (Madden). If our cost function was non-convex it would pose potential issues in regards to optimizing at local minimums instead of global minimums (Ng). To initialize the theta variables I set all coefficients to 0. This sets me up to run the cost function. Deriving theta using gradient descent using the optim function. The purpose of optim function is to use derivatives to find the best fitting parameters. The model will minimize a given function, which is very convenient because it will find the best theta values that minimize the cost function (Ng). Finally, I created the new prediction matrix. There are three different classifiers; y1, y2, y3. The output for each of these will be put in y1, y2 and y3. The predication matrix takes columns and goes row by row, and chooses the highest value of that row of values and uses that to predict which class it is for that row. We can interpret these as probabilities for each one belongs to a given class. The final output of our multinomial output, is going to be whichever column of y1, y2, y3 has the highest probability. Even though they're not actually probabilities, they can interpreted so because the values lie between 0 and 1. If the value of y1, y2 and y3 of a test instance is 0.3, 0.5, and 0.8 then we could interpret them as probabilities that they belong to class y1, y2 and y3 respectively which in this case is Barn Owl, Snowy Owl and Long-Earned Owl.

Evaluation

Looping my model allowed me to automate the results, so that my model reproduced the results 10 times. On the left are the overall results of my logistic regression model. I felt as if the final results of my predictive model were above satisfactory. The average accuracy of my model was 93%.

Test	Results
1	0.976
2	0.9574
3	1
4	0.8958
5	0.6458
6	0.9787
7	0.9772
8	0.9069
9	0.95
10	0.9729
Average	0.92607

In addition to these results I measured my classifier's performance with a confusion matrix table. Confusion matrix tables are, "often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. On the left is the confusion generated for my first test, and below on the right is a template that can be referred to.

Prediction/ Actual	1	2	3
1	16	0	0
2	0	13	0
3	1	0	7

Prediction class / True class	A	B	C
A	AA	AB	AC
B	BA	BB	BC
C	CA	CB	CC

Table 3: Statistics of confusion matrices for samples classification.

As you can see my classifier was able to correctly classify my classes in 36 out of the 37 attempts. These are referred to as "True Positives". This is when the model predicts yes and it was actually yes (Madden). In the case of the predicted class being 1 when it was actually 3, that is known as a "False Positive" (Madden).

Code

```
# Load Owls 15
input_csv <- read.csv(file.choose(), sep = ",")

# Use the structure function to understand your dataset
# str(input_csv)

# Rename the last column in the dataset into type
colnames(input_csv)[ncol(input_csv)]<-"type"

#Rename other columns in a format like  Predictor_1, Predictor_2 ...
colnames(input_csv)[-ncol(input_csv)]<-paste("Predictor_",c(1:(ncol(input_csv)-1)),sep = "")

# Turn the last column, type into a factor - 1, 2 and 3
input_csv$type <- factor(input_csv$type)
number_of_classes<-length(levels(input_csv$type))
number_of_features<-ncol(input_csv)-1

# Create an accuracy_vector, which is intialized with 0's
accuracy_vector <- rep(0,10)

#Sigmoid function (logistic function)
Sigmoid <- function(real_number)
{
  sigmoid <- 1/(1+exp(-real_number))
  return(sigmoid)
}
# Randomly divide the file into 2/3 for training, 1/3 for testing
# Loop the formula in a for loop, which reproduces the
for(seed in 1:10)
{

  set.seed(seed)

  index <- sample(1:2, nrow(input_csv), replace = TRUE, prob = c(0.67, 0.33))
  training <- input_csv[index == 1,]
  validation <- input_csv[index == 2,]
  # head(training)
  # head(validation)

  #Predictor variables
  pred_var <- as.matrix(training[1:(ncol(input_csv)-1)])
  pred_test <- as.matrix(validation[1:(ncol(input_csv)-1)])

  #Add ones to pred_var and pred_test
  pred_var <- cbind(rep(1,nrow(pred_var)),pred_var)
  pred_test <- cbind(rep(1,nrow(pred_test)),pred_test)

  #The for loop below carries out the following steps, but for any number of attributes >= 3
  # # Create y1, y2, y3 for owls dataset
  # training$y1 <- ifelse(as.integer(training$type) == 1, 1, 0)
  # training$y2 <- ifelse(as.integer(training$type) == 2, 1, 0)
  # training$y3 <- ifelse(as.integer(training$type) == 3, 1, 0)

validation$y_label<-as.integer(as.factor(validation$type))

class_number <-1
for (class_number in 1:number_of_classes)
  {
  y_variable_name<- paste("y",class_number,sep = "") #y1, y2 or y3
```

```
  #print(y_variable_name)
  assign(y_variable_name, ifelse(as.integer(training$type) == class_number, 1, 0) )
  #print(get(y_variable_name))
  training[ ,number_of_features+class_number+1]<-get(y_variable_name)
  #Creating response variables in matrix form for matrix multiplication later
  #the line below in the for loop is a gneralized version of the following
  # y1 <- as.matrix(training$y1)
  # y2 <- as.matrix(training$y2)
  # y3 <- as.matrix(training$y3)
  assign(y_variable_name,as.matrix(get(y_variable_name)))
  cost_variable<-paste("cost",class_number,sep="")
  assign(cost_variable, function(theta)
  {
    # m = size of data/or number of instances
    instances <- nrow(pred_var)
    # g is the sigmoid function applied to X.theta
    #%*% is matrix multiplacation
    g <- Sigmoid(pred_var %*% theta)
    J <- (1/instances) * sum(((get(paste("y",class_number,sep="")))-g)^2) #squared error cost function that
we're trying to minimize
    return(J)
  })

  #Initializing all coefficients to be 0s . example below
  #initial_theta1 <- rep(0,ncol(pred_var))
  initial_theta_variable_name<-paste("initial_theta",class_number,sep="")
  assign(initial_theta_variable_name,rep(0,ncol(pred_var)))

  #running the cost function with the initial theta coefficients. example below
  #cost1(initial_theta1)
  get(cost_variable)(get(initial_theta_variable_name))

  # Derive theta using gradient descent using optim function , example below
  #  theta_optim1 <- optim(par=initial_theta1,fn=cost1)

  theta_optim_variable_name<-paste("theta_optim",class_number,sep="")
  assign(theta_optim_variable_name, optim(par=get(initial_theta_variable_name),
                                          fn=get(cost_variable)) )

  #Extracting the theta parameters from the optim output, which is a list of 5, example
  #  theta1 <- theta_optim1$par
  theta_variable_name<-paste("theta",class_number,sep = "")
  assign(theta_variable_name,get(theta_optim_variable_name)$par)

# Create prediction columns for each of the classes

  validation[ ,number_of_features+class_number+2]<-
    Sigmoid(pred_test  %*% as.matrix(get(theta_variable_name)))

  }

  # validation$y_label<-as.integer(as.factor(validation$type))
  # validation$y1<- Sigmoid(pred_test %*% as.matrix(theta1))
  # validation$y2<- Sigmoid(pred_test %*% as.matrix(theta2))
  # validation$y3<- Sigmoid(pred_test %*% as.matrix(theta3))
  validation$prediction<- 0
  for(i in 1:nrow(validation))
  {
    class_prob<-validation[i,(number_of_features+3):(number_of_features+3+number_of_classes)]
    validation$prediction[i]<-which(class_prob==max(class_prob))
  }

  accuracy<-sum(validation$y_label==validation$prediction)/nrow(validation)
```

```
  print(accuracy)
  # Use validation

  accuracy_vector[seed]<-accuracy
}

cat("Overall accuracy on 10 different splits of 1/3 and 2/3 = ", mean(accuracy_vector))


#################################
# Evaluation

install.packages("pROC")
library(pROC)
library(caret)
library(ggplot2)
confusionMatrix(validation$prediction, validation$y_label)

plot1 <- ggplot(data = input_csv, mapping = aes(x = Predictor_2, y = Predictor_4, color = type))+
  labs(x = "Wing-Length") + labs(y = "Wing-Width") +
  geom_point()
plot1
plot2 <- ggplot(data = input_csv, mapping = aes(x = Predictor_3, y = Predictor_4, color = type))+
  labs(x = "Body-Width") + labs(y = "Wing-Width") +
  geom_point()
plot2
```

Work Cited

Madden, Michael. “Machine Learning.” Michael Madden - NUI Galway, 1 Sept. 2017, *www.nuigalway.ie/engineering-informatics/information-technology/research/researchtopics/michaelmadden/.*

Ng, Andrew. “Machine Learning.” Coursera, 1 Sept. 2017, www.coursera.org/learn/machine-learning.

Wickham, Hardley. Data Science With R. Oreilly & Associates Inc, 2015.