R You Ready For Neural Networks?

Prediction of Injury Severity in Car Crash Data

Austin Pickett, Daniele Moro, Rudy Ruiz

April 12, 2018

Introduction

Our group decided to choose R because the other languages had too many letters for us to remember what language we chose. The programming language R is also a language that is heavily used in statistical computing and all the members of our group were very interested in the idea of learning how to make and use a neural network. R is a programming language that makes it relatively easy to build machine learning models by using the myriad of packages available for use, making it an excellent choice for our group. The package we decided to use to build our feed-forward neural network was Keras for R. The purpose of this project is to build a neural network that predicts the injury severity of a person in a car crash based off of 11 attributes of a car crash.

In order to learn more about neural networks and their associated challenges, we initially began our project by creating a digit-recognizer as an introductory learning step. The idea of the digit-recognizer is to see if our neural network can correctly classify hand written digits into the correct digit label. We used the MNIST dataset to train and test our neural network. Our digit-recognizer was able to successfully categorize handwritten images to their respective digit label with an accuracy of 98%. This project taught us how to implement a neural network using a well know example with a plethora of available online tutorials.

After completing the MNIST example, we decided to build a neural network that could operate on a different, more difficult dataset. We chose to use the "NASSCDS: airbags and other influences on accident fatalities" [nas(2018)] dataset because it contained thousands of data points and it we thought that it would be interesting to play with this data. We decided to predict injury severity based off the following attributes: use of airbag, age of passenger, gender of passenger, speed of car, using seatbelt, and more. In this report we will discuss the R language, our project, and our reflections on the language.

About R

The R programming language is built for statistical computing and resembles the Python programming language. R originally comes from the programming language S, which was created in the 1980s. Like Python, R is strongly and dynamically typed, and it is statically scoped. R is an interpreted language and therefore binds variables and allocates memory on the fly. R defines itself as a functional programming language and "allows dynamic creation and manipulation of functions and language objects" [R(2018)]. The strengths of R include its packages, flexibility, and community support. R's shortfalls lie in memory management, efficiency, and security [Willems(2015)]. R has several key characteristics that make it different from other languages which we will discuss in this section.

2.1 Assignments and Names

To assign a value to a variable in R, one must use either "<-" or "=". However, the equals signs does not always work for all assignments, so the safest approach is to always use the arrow operator. R objects have a variety of attributes, and one of these is the "name" attribute. This attribute is used to label and index a given object [R(2018)]. R also reserves the following one letter words for naming variables: c, q, s, t, C, D, F, I, and T. Furthermore, one should not use the dollar sign in any variable names because in R, the "$" is used in the same way as a "." in other languages like Python. Finally, R statements do not need to end with a semicolon (like in Java). Rather, a new line is used to signify the end of a statement.[Cook()]

2.2 Types

The six basic 'atomic' types in R are the following: logical, integer, double, complex, character, or raw. Types such as single numbers and strings are all represented as vectors, which are described in the following section. Other object types include vector object (described in the next section), list objects (generic vectors), symbol objects (refer to other objects), expression objects (a collection of statements), function objects, and a few more. An important concept in R is the Factor, which similar to an Enum in Java. A certain Factor can become one of a finite number of levels or categories. [R(2018)]

2.3 Vectors

Vectors are the primary and perhaps most important object in R. A vector is a collection of "contiguous cells containing data" [R(2018)]. A vector is a collection of data, where every element in this data needs to have the same type. Vectors are defined with the "c" function, with each parameter of the function being an element in the vector (nums <- c(1,2,3,4)), and an elements in the vector are accessed using [x], where x is the index of the element accessed. R is different from other languages in that the vector indexing begins at 1, not 0. Another interesting property of R vectors that makes them unique is that two vectors do not need to be the same size to perform some operation. For example, adding the vectors c(10,20,30) and c(1,2,3,4,5,6) results in the vector c(11,22,33,14,25,36). The shorter vector is recycled repeatedly to make the operation possible. This emphasizes R's "lazy" or "forgiving" approach to programming. Related to vectors are lists and matrices. A list is the same as a vector, but the elements do not need to be the same type. A matrix can be thought of as a two dimensional vector [Cook()].

2.4 R Packages

Another cornerstone of R, and a feature that has allowed R to gain its popularity, are packages. A package is a sharable collection of code, data, documentation, and tests. The place where many valuable packages can be found is called CRAN (Comprehensive R Archive Network) and as of January 2015, CRAN hosted over 6,000 packages. The packages include almost anything an R programmer could ever wish to use, from plotting graphs, to creating confusion matrices, to generating a neural network. Packages are regarded as one of the reasons that R is so successful; they allow any R programmer to very quickly and easily create anything they would like by relying on the work of others. [Wickham()]

2.5 Other Characteristics of R

There are many unique characteristics of R that make it different from other languages. Because we cannot cover all of these differences, we list a few more characteristics that we found particularly important. R functions are very similar to Python functions, with only a minor change in syntax. Although R supports loops such as while loops and for loops, their use is discouraged. Instead, the language promotes loopless loops like the apply function. For example, to multiply every element in the a vector by a constant, one may simply use c(1,2,3,4) * 100. The apply function is used in more complex scenarios, but it essentially allows the programmer to apply some function to every element in a data structure like a data frame, vector, or list. The apply function is truly a family of functions, and it can take time to learn how to use them effectively. [Machlis(2017)]

Project

3.1 Motivation and Background

Neural networks are rapidly expanding in their ability to solve challenging computer science problems. While the idea behind neural networks is relatively simple, implementing a neural network can be difficult. We chose to use the language R for this project which is known to be heavily used in the field of data science in order to learn more about neural networks and the problems they bring. Since we had experience with Python, which is a competitor to R, we felt that we would have an easier time learning and adapting to R. We all hope to be able to work on machine learning in the future and believe that using R for this project will prove to be a valuable first step in learning about this powerful technology. Even though we know we will not be able to produce anything like Google's AlphaGo AI that won against the top Go player in the world[Russell(2017)] we are excited to build a small neural network in R. We chose to produce a neural network to analyze our data instead of using a linear regression because we believed our data to not be linearly separable. Originally, we wanted to have our neural network determine one's survivability in the event of a car crash given certain information, however we adjusted this goal to categorizing the injury severity based on the same information. This change was caused by our discovery that simply too little of our data points were a fatal car crash. As long as our program predicted that one survived every time, it would be right 95% of the time. These results, while accurate, with a 95% baseline were fundamentally uninteresting. Because of this, we chose to predict injury severity instead of dead or alive.

3.2 Procedure

3.2.1 Setting Up The Environment

Integrated development environments are great applications that enhance the ease and ability of programming with a specific programming language. For the language R, RStudio [RStudio()] is the IDE we chose to install. This also required the precompiled binary distribution of the base system and contributed packages of R since RStudio does not come included with them. As the installment of the IDE finalized, it was time to install necessary packages for our project.

The main package required for our project is the "keras" package. In order to get Keras fully functional, it was required to also install Anaconda 3.6 for Python. The "keras" package allowed us to use its built-in neural network functions, models, and optimizers. The packages "caret" and "plotly" were also important to include in our environment. This allowed us to create plots and graphs that allowed us to analyze our data.

3.2.2 Selecting a Data Set

Many datasets are available online for anyone to use. One big source of datasets for the R language can be found at Vincent Bundock's GitHub page [vincent arel bundock(2018)]. This is where the dataset "NassCDS: Airbag and other influences on accidental fatalities:" we used in this report can be found [nas(2018)]. The NassCDS dataset contains variables that correlate with each other when plotted on a graph such as crash speed, injury severity, and airbag usage. This dataset consists of over 26,000 crash cases which makes it a great size for a neural network. This dataset is a prime example on the type of dataset preferred when creating a machine learning algorithm. Each row of the raw data set contained the following data: dvcat (impact speed), weight (observation weights for sampling probabilities), dead (alive or dead), airbag (none or airbag), seatbelt (none or belted), frontal (o for non-frontal impact, and 1 for frontal impact), sex (either male or female), ageOFocc (age of the occupant), yearacc (year of the accident), yearVeh (year of model of vehicle), abcat (airbag deployment), occRole (driver or passenger), injurySeverity (0:none, 1:possible injury, 2:no incapacity, 3:incapacity, 4:killed; 5:unknown, 6:prior death), and caseid.

3.2.3 Cleaning Up The Data

In order to optimize the learning of our neural network, clean up of the NassCDS dataset was required. The variables "weight" and "caseid" were not relevant variables for our neural network. The "weight" variable was designed to account for varying sampling probabilities and the "caseid" variable is simply the case number of the crash. Once the non-relevant variables were deleted from the dataset, merging of some variables are required. The year of the vehicle and the year of the crash variables were merged together into one "age of vehicle" integer variable that shows the age of the vehicle in years. This in return eliminates year values in our data set and simply has one integer value. This process resulted in us using a total of 11 attributes for each car crash.

One major cleanup step that was taken was the re-evaluation of the variable "injury severity". The original dataset considered 7 different ranges of injury severity, ranging from "0 - none" to "6 - prior death". We found that it was logical to merge certain values of injury severity together. The values "6 - prior death" was merged with "4 - death" both ultimately having the same outcome. Another value that was merged was "5 - unknown" with "0 - none" basing the conclusion on both values not having anyone hurt. The last merge that took place was the values of "1 - possibly injured" with "2 - no incapacity", both values having the same idea of no real injury (only bruises and scathing). This resulted in the following injury severities:
0 - none
1 - no incapacity
2 - incapacity
3 - killed

3.2.4 Preparing the Data for the Neural Network

"Injury severity" is the variable that we decided to build our neural network around. When given all other variables, we want the neural network to predict the injury severity. In order to accomplish this, it was necessary to separate the variable "injury severity" into a 1-dimensional vector and leave all other variables in a matrix. The original values in "injury severity" are labeled from 0 to 3. In order to improve the accuracy with our vector, we re-categorized the values in "injury Severity" using the One-Hot Encoding technique [Vasudev(2017)] which prepared the vector for the neural network.

With 26,000 examples in our dataset, a common practice is to split the data up into training and testing data. We split our data into 70% training data and 30% testing data. The separation of data with a random seed of 123 was necessary in order to ensure reproducibility. This finalized the preparation that was needed for our neural network

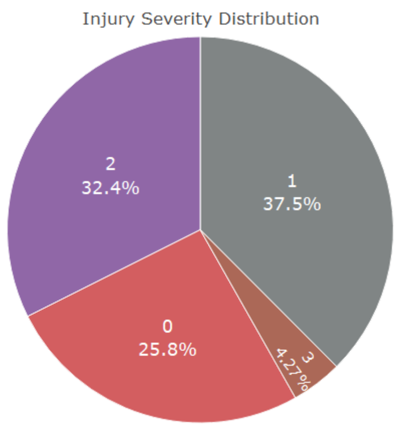

Figure 1: A pie chart showing the distribution of all injury severities

3.2.5 Configuring the Neural Network

The included package "keras" handles the heavy lifting of creating the neural network. As for the model of our neural network, we chose to go with the Sequential Model for a supervised learning neural network. This allowed us to stack layers in our neural network which are linear. This model is simpler than most to use and allows us tailor the amount of layers necessary for our data. The first input layer of the neural network consists of 12 neurons, 11 for the number of features in our training data set and 1 neuron for a bias term. The activation function for this layer was the "relu" function. The output layer consists of 4 neurons, one neuron per label in our model. The amount of neurons for our output was decided when we used the activation function "Softmax". It requires a neuron for each label and is a great activation function for classifier neural networks. No hidden layers were necessary (we tried very many different configurations and came to this one in the end).

Tweaking of the compile settings was also necessary for the neural network. For the loss function, "categorical cross entropy" was necessary because our target data was one-hot encoded and are classes, not labels. The optimizer that we chose for our neural network was the "adam" optimizer. We chose it simply because it is one of the most popular optimizers today. To finish up the compile settings, we simply added the metric of "accuracy" to testa our neural network.

3.2.6 Training and Evaluation of the Neural Network

With the data prepped and our neural network configured properly, it was time to train and evaluate the neural network. The model was trained on our 70% of training data and targeted to our 30% of the injury severity data. The amount of epochs chosen was 50 with a batch size of 100. We found that the accuracy stagnated with any more than 50 epochs. This seemed acceptable with our size of training data.

We then evaluated our model with our 30% of testing data. We used the "evaluate" function to calculate the loss values for our input testing data. Once the evaluation of our data is done, the results are saved into a prediction array for further evaluation. The neural network now finished evaluating our data and saved the results in an array for us to analyze and evaluate on the findings.

3.3 Analyzing the Data

To understand what our neural network is trying to learn, we must first understand the dataset we are using. We used R packages such as Plotly to generate the following graphs that provide meaningful insight on our data [Plotly()].

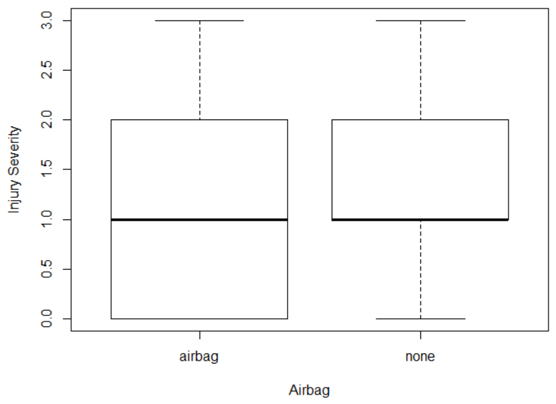

Figure 2: A series of box and whisker plots that show the correlation between the presence of an airbag and the injury severity. Airbags tends to correlate with lower injury severities.

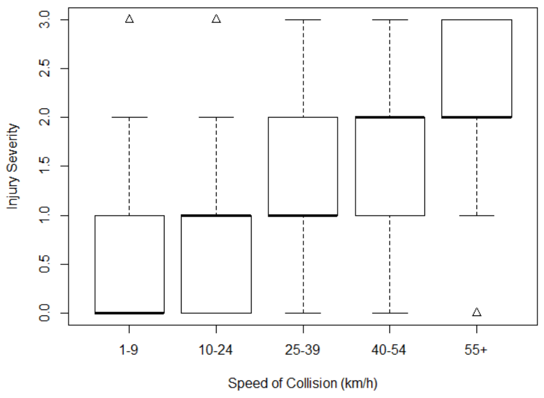

Figure 3: A series of box and whisker plots that show the correlation between the speed of a collision and the injury severity. Higher collision speeds correlate with higher injury severities.

One of the most important aspects of our data is how it is distributed. Figure 1 is a pie chart that concisely shows this distribution of injury severities. We note that this distribution is not uniform: the lower injury severities such as 0 (no injury) and 1 (no incapacity) make up nearly 64% of our data. Only 4.27% of victims of the crashes died. This shows that most car crashes usually do not result in serious injuries. The skewed nature of the data that exhibits a higher proportion of lower injury severities may bias our model towards these lower injury severities.

Figure 2 shows a series of box and whisker plots that show the correlation between the presence of an airbag in a crash and the injury severity. The center box represents the interquartile range, where 50% of the data lies. One can see a that the box plot for the presence of airbags has more data points that experience lower injury severities. This points towards the conclusion that the presence of an airbag may be correlated with lower injury severities. Our model may use this correlation to predict lower injury severities whenever the airbag feature is true.

To better understand the role that speed plays in a crash, we generated Figure 3, which shows the correlation between speed and injury severity through a series of box and whisker plots. The x axis is composed of the speeds, ranging from 1-9 km/h to over 55 km/h. We see a clear trend of higher speeds leading towards higher injury severities. Low speeds between 1 and 24 km/h result in little to no injury, while higher speeds over 40 km/h are very likely to result in a severe incapacity or death.

3.4 Findings and Evaluation

After building and training our model, we collected the predictions for the test data. We obtained two sets of results: one set that contained the 'dead' column in the data, and another set that did not. When 'dead' was included in our data, we obtained an accuracy of 51% on the testing data. When 'dead' was not included, we obtained an accuracy of 47% on the testing data.

To understand the significance of these percentages, we first must understand the baseline accuracy. If our model completely guessed every time it made a prediction, it would guess correctly, and therefore have an accuracy of 25% ((37.5 + 32.4 + 25.8 + 4.27) / 4). However, if our model simply picked injurySeverity X every time, it would get an accuracy of 37%. This is because the category injurySeverity X appears 37% of the time. Therefore, the baseline accuracy for our problem is 37%. Because our accuracies of 47% and 51% are higher than the baseline, we can show that our model does work, and can predict the injury severity of a car crash from limited data.

We do observe that an accuracy of 47% is not much higher than the baseline. However, the task of predicting car crash injury severity is very challenging with limited data per car crash. With only X columns of data, there is no information on details such as what the passengers and driver in the car crashed against. If the car crashed against a bush, the injuries may be much less severe than if the car crashed against an oncoming truck, even if all of the other data such as speed and seatbelt remain the same. This example demonstrated how missing information, as well as an element of chance, is missing from our dataset, making the problem challenging.

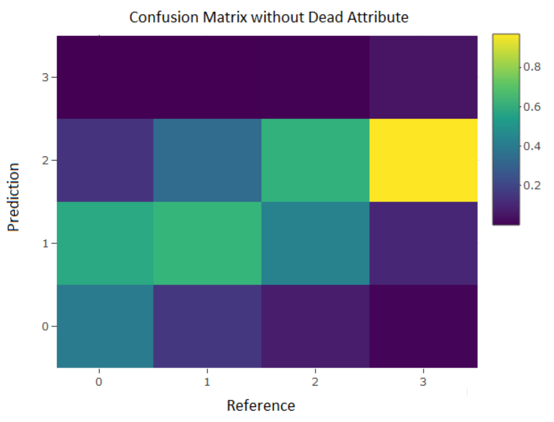

Figure 4: A confusion matrix showing actual injury severities along the X axis, and the model's predicted injury severities along the Y axis. This figure is for the data set with the 'dead' feature.

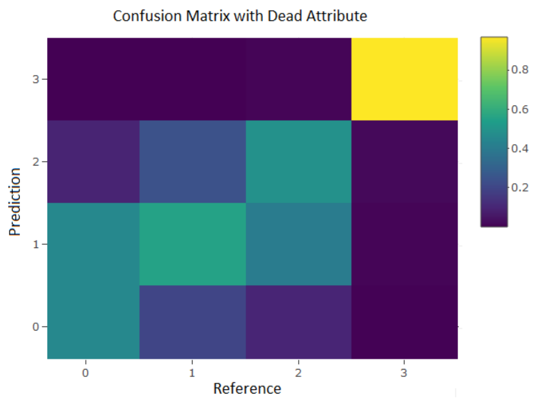

Figure 5: A confusion matrix showing actual injury severities along the X axis, and the model's predicted injury severities along the Y axis. This figure is for the data set with the 'dead' feature.

3.4.1 Confusion Matrices

We further evaluate our model by analyzing the confusion matrices in Figures 4 and 5. The confusion matrices show the correct injury severity (ranging from 0 to 3) for all car crashes in the testing data on the x axis and the corresponding predicted injury severities on the y axis. For a given reference, the corresponding prediction square shows the percentage of times our model predicted that injury severity. The colors represent the concentration of our predictions and each reference column adds up to 100%. When using the dataset with the variable 'dead', Figure 4 shows that our model is able to mostly guess the correct injury severities, but often confuses similar injury severities such as 0 (no injury) and 1 (no incapacity), and 1 and 2 (incapacity). This behavior is reasonable because the exact injury is difficult to predict, but the model can correctly predict a range of injury severity (not severe or very severe). We note that the model predicts injury severity 3 (killed) very often (96%). We reason that this is because the model correctly associated the attribute 'dead' with the injury severity 3. This is not the case when we remove the 'dead' attribute, as is seen on Figure 5.

Figure 5 shows similar patterns as Figure 4, but with the difference that the model almost always predicts the injury severity of 2 when it is truly a 3. We believe that this is because our data is severely skewed or biased. Only about 5% of the people in our dataset are killed, making the model tend to very rarely predict such a rare injury severity. We could fix this behavior by showing the model an equal proportion of all injury severities so that the model is not affected by the skewed data. We predict that this change would improve our accuracy, but implementing this change is beyond the scope and time frame of this project.

Reflections on R

The R language can be daunting at first but once one understands its syntax, it becomes simple and concise. R is nice for novice programmers because it can be dynamically typed and great for advanced programmers for the static typing. The ability to easily cast variables using built in functions is helpful, especially when meeting the criteria for the input of our neural network. Another great thing is the ability to label matrix rows and columns so that referencing specific rows and columns becomes easy. Lastly one of the nicest things about R is the amount of useful packages. R contains many packages for almost every task such creating 3D plots and neural networks design.

R is not all dandelions and daisies. There are some bad things and unusual things about the language. One bad thing we found with R was the syntax of certain operators. For example, the assignment operator is different than most programming languages today and also the piping operator. These unusual operators make it difficult for other programmers new to R to learn basic syntax. Another bad thing about R is the amount of data structures. R contains data structures of type vector, array, factors, lists, and data frames, each having their own specifications on when to use them. When functions require a specific data structure, it can become frustrating to cast one's data structure just to meet the requirement of the function.

For those new to R with the desire to start learning statistical models and neural networks, R is a great language to start the adventure. It contains very helpful documents and many packages to assist the programmer. It holds the hand of novice data scientists through their learning curve and allows the freedom for advanced data scientists to start developing their state-of-the-art neural network. When our next machine learning opportunity arises, we will choose to use R.

References

[nas(2018)] Airbag and other influences on accidental fatalaties. https://rdrr.io/cran/DAAG/man/nassCDS.html, 2018. [Online, accessed 25-February-2018].

[Cook()] John Cook. R language for programmers. https://www.johndcook.com/blog/r_language_for_programmersl. [Online; accessed 8-April-2018].

[Machlis(2017)] Sharon Machlis. Beginner's guide to R: Syntax quirks you'll want to know. https://www.computerworld.com/article/2497319/business-intelligence/business-intelligence-beginner-s-guide-to-r-syntax-quirks-you-ll-want-to-know.html, 2017. [Online; accessed 8-April-2018].

[Plotly()] Plotly. Modern Visualization for the Data Era. https://plot.ly/. [Online; accessed 10-April-2018].

[R(2018)] R. R Language Definition. https://cran.r-project.org/doc/manuals/r-release/R-lang.html, 2018. [Online; accessed 8-April-2018].

[RStudio()] RStudio. Rstudio.

[Russell(2017)] Jon Russell. Google's AlphaGo AI wins three-match series against the world's best Go player. https://techcrunch.com/2017/05/24/alphago-beats-planets-best-human-go-player-ke-jie/, 2017. [Online; accessed 25-February-2018].

[Vasudev(2017)] Rakshith Vasudev. What is one hot encoding? https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f, 2017. [Online, accessed 10-April-2018].

[vincent arel bundock(2018)] vincent arel bundock. R data sets. https://vincentarelbundock.github.io/Rdatasets/, 2018.

[Wickham()] Hadley Wickham. Table of contents R packages. http://r-pkgs.had.co.nz/intro.html. [Online; accessed 8-April-2018].

[Willems(2015)] Karlijn Willems. Choosing R or Python for Data Analysis? An Infographic. https://www.datacamp.com/community/tutorials/r-or-python-for-data-analysis, 2015. [Online; accessed 8-April-2018].