Machine Learning Classification

Andrew Wang

11/7/2020

Contents

Abstract

The purpose of this paper is to conduct a K-nearest Neighbor classification on a set of climate model simulation data. I conduct standard data cleaning procedures on the data, before filtering and splitting it for KNN classification. The machine learning model was able to predict the outcome with a 89.6% accuracy rate, and most misses were false positives, in which the model predicted the simulation would succeed where it actually failed.Out of 163 testing cases, the model was able to predict 146 of them correctly.

Introduction

In this report I will be using the K-nearest Neighbor algorithm for dataset classification on a set of climate model simulation data (Source: Lucas and Zhang (2013)). The primary purpose is to use the KNN algorithm on the dataset, but I also employ an assortment of plots to explain the data and give an overview of trends in the data. First, I will provide some background information about the dataset, its content, and its collection.

Climate Model Simulation Crashes Data Set

This dataset is from the UCI Machine Learning Repository, and was donated to the repository in 2013 by individuals at the Lawrence Livermore National Laboratory (LLNL). This data was constructed using LLNL's unerctainty quantification (UQ) pipeline. The purpose of this data was to analyze the causes of failure in climate models. The data consists of 540 simulations (rows) with 20 variables (columns). They are as follows:

- Column 1: Latin hypercube study
- Column 2: simulation ID
- Columns 3-20: values of 18 climate model parameters
- Column 21: simulation outcome

I will now go into some more detail regarding the specific methods used for collecting the data used and how the variables are portrayed.

Latin Hypercube Studies The first column of the dataset describes the latin hypercube study which the observation belongs to. The value can be between 1 and 3, as 3 Latin hypercube studies were conducted to produce this data. The Latin hypercube is a method of sampling which seeks to recreate input distribution when selecting from a smaller sample size. Many sampling methods require large samples for a reflective distribution, but the Latin hypercube solves this issue by grouping selection. It splits the cumulative curve of the graph, a frequency distribution of data on the graph, into equal parts, and then takes a sample from each interval. By doing this, it can ensure that samples are taken evenly across the entire interval, while still maintaining randomness within that interval. The Latin hypercube is meant to be a generalization of the Latin square concept, in which there is one sample per row and column.

Simulation ID This is a very simple column, which just portrays the ID, identification number, for each observation, from a range of 1 to 540.

Climate Model Parameters There are 18 parameters used to determine the cause of failure in this dataset. They can generally be separated into two categories, those which measure viscosity and those which measure diffusion (diffusivity). Parameters 2-6 are all measures which contribute to the overall viscosity of the observation, giving in parameter 1. All other parameters are various measures of diffusivity, such as equatorial diffusivity, maximum PSI induced diffusivity, and Banda Sea diffusivity. All the parameters are scaled, and measuring from a range of 0 to 1.

Simulation Outcome The last column of the dataset is the simulation outcome, whether or not the simulation succeeded or crashed (failed). A 0 means the simulation crashed, while a 1 means the simulation succeeded.

Methods

Now I will go over the methods and code used to produce my results. There is a lot of setup and data organization to produce the results, and I will go over all of that in this section.

Setup In this section, I am going to review basic environment clean-up and imported packages, along with cleaning the data and making sure everything is ready for analysis. First, I clean up and set up the R environment with the working directory. Keep in mind that your directory structure is likely different from mine, and adjust accordingly. Furthermore, I turn warnings off in the effort of saving paper.

```
# Andrew Wang
# November 7
# Machine Learning Classification
#
# clean up and setup
rm(list=ls()) # clean up any old stuff in R
```

```
setwd("C:/Users/hyper/OneDrive/Desktop/Desktop Folders/Programming/R/Assignments/Week 10") # go to this
#load up myfunctions.R
source("C:/Users/hyper/OneDrive/Desktop/Desktop Folders/Programming/R/myfunctions.R")

options(warn = -1)
```

After this is done, I import the packages I'm using. Keep in mind that you will likely need to install these packages first before using them.

```
library(class)
library(ggvis)
library(gmodels)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine
```

Finally, we can import our .csv file, and assign it to a variable. I then run some basic observations on the dataset as a starting point, in order to get a basic overview of the data we have. In the effort of saving paper, I have commented out the str() and summary() functions here because they have a large amount of output.

```
climate <- read.csv("pop_failures.csv")
View(climate)
#dim(climate)
#glimpse(climate)
#str(climate)
print(sum(is.na(climate))) #no missing data)
```

```
## [1] 0
```

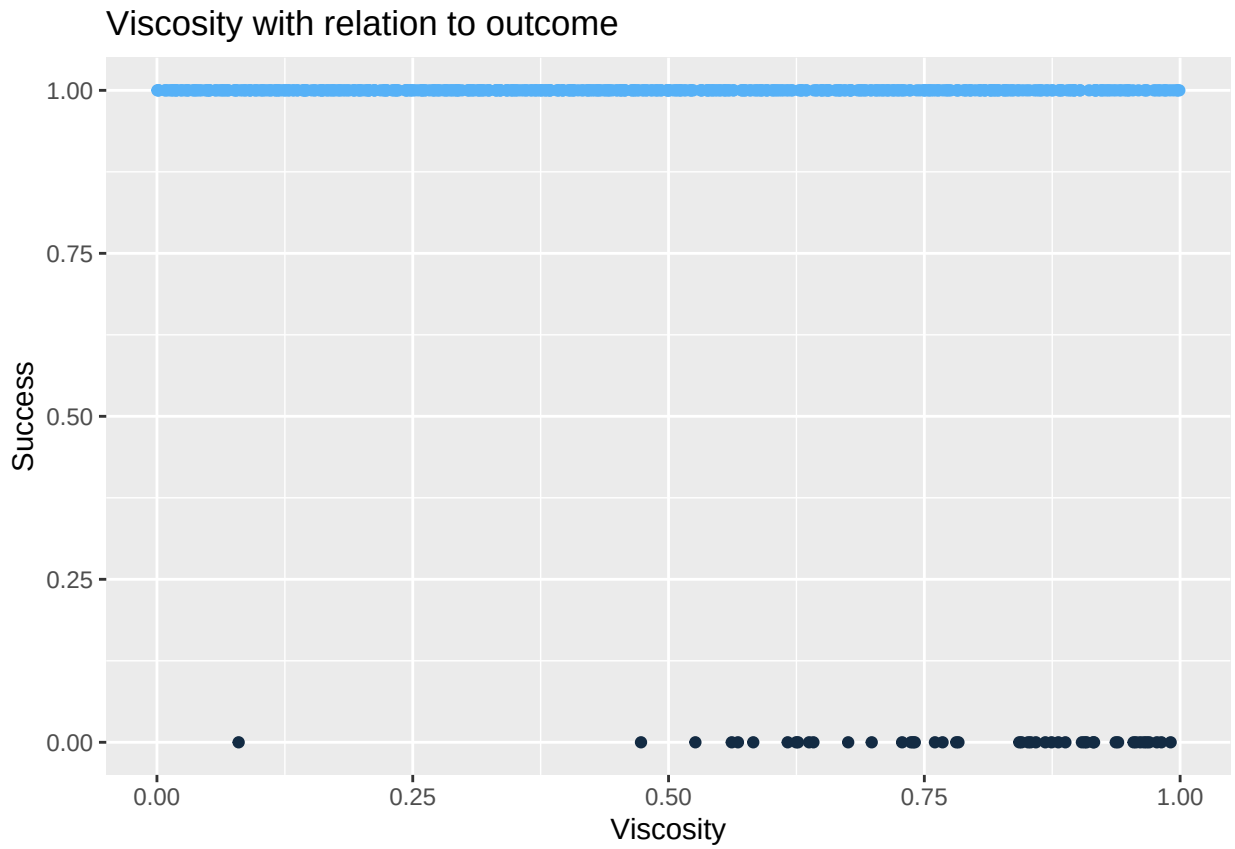
Notice, there are no missing values.

Next, I need to convert the data types of a few columns into numbers, so that when they are analyzed by plot functions and KNN classification they are recognizable. I first convert the existing character data into a factor, and then convert it into a numeric. By doing this, I can avoid creating NAs through coercion.

```
#converting a few chr types to nums
climate$vertical_decay_scale <- as.numeric(as.factor(climate$vertical_decay_scale))
climate$convect_corr <- as.numeric(as.factor(climate$convect_corr))
climate$bckgrnd_vdc1 <- as.numeric(as.factor(climate$bckgrnd_vdc1))
```

Basic plotting for viscosity and diffusivity Before doing the KNN classification, I wanted to get some basic plotting done for the two parameter groups and see if there were any generalizable trends in the data. First I preform a scatterplot for viscosity with relation to outcome.

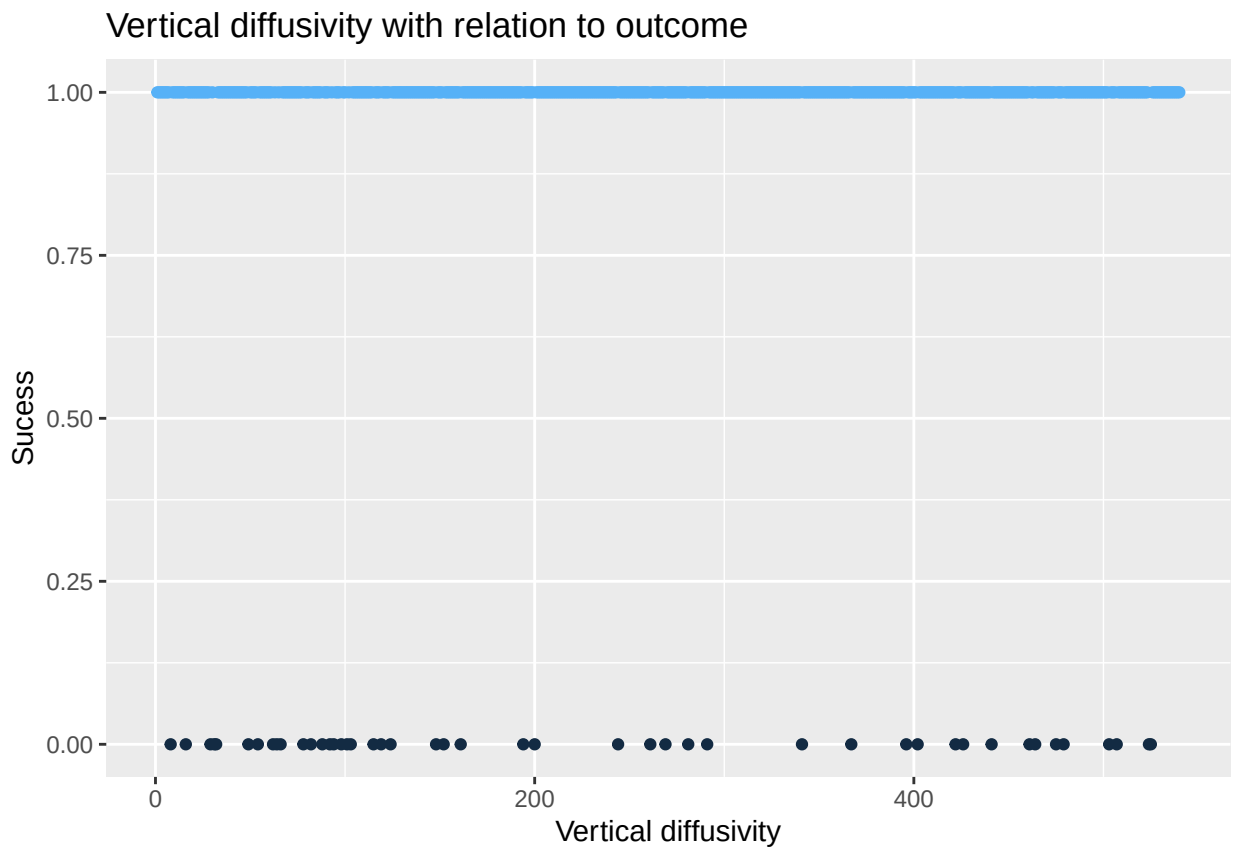
```
#basic plot
plot1 <- ggplot(climate, aes(x = vconst_corr, y = outcome, color = outcome)) +
  geom_point() +
  guides(color = FALSE) +
  labs(title = "Viscosity with relation to outcome", x = "Viscosity", y = "Success")
plot1
```



Notice a general trend: as viscosity increases so does the frequency of failures. This is possibly because of the difficulty computers have in simulating liquid's iteractions and properties, as reflected in modern-day difficulties in simulating water.

Next, I conduct the same procedure for the base background vertical diffusivity.

```
#basic plot 2
plot2 <- ggplot(climate, aes(x = bckgrnd_vdc1, y = outcome, color = outcome)) +
  geom_point() +
  guides(color = FALSE) +
  labs(title = "Vertical diffusivity with relation to outcome", x = "Vertical diffusivity", y = "Sucess
plot2
```



Looking at this plot, we can instead tell that there is no trend among data. The failures are roughly evenly spaces and therefore no particular relationship can be discerned.

KNN Classification For my final method, I am going to conduct a KNN classification on the dataset. By doing so, I apply the KNN machine learning algorithm and see how well it is able to learn patterns and trends in the data. First, I need to set the seed.

```
#set seed
set.seed(1234)
```

Next, I need to split the data into two groups, one for training and one for testing. The algorithm will learn trends off the training set, which is usually 60-70% of the data, and then predict outcomes for the test set, which is used to check the accuracy of the model. First, I create a new variable entirely composed of 1s and 2s, this is how I can assign rows to a testing or training set. I make the probability 70-30, meaning 70% of the instances will have a 1 while 30% of the instances will have a 2. Next, I create two new variables to store subsets of the climate data based on the `ind` value calculated. These two new variables hold the entirety of the set, as they are meant to be data subsets. Finally, I create two more variables to only store the outcome, which is in column 21. I create these two final variables so that the algorithm can check its predicted outcomes with the actual outcomes.

```
# data preparation and splitting code
#randomized index
ind <- sample(2, nrow(climate), replace = TRUE, prob = c(0.7, 0.3))
# see webinar notes for specifications
#view(ind)
trainingSet <- climate[ind == 1, 1:21]
```

```
#view(trainingSet)
testingSet <- climate[ind == 2, 1:21]
#view(testingSet)
trainingLabels <- climate[ind == 1, 21]
#view(trainingLabels)
testingLabels <- climate[ind == 2, 21]
#view(testingLabels)
```

With all the preparation complete, I can now conduct use KNN on the data subsets. While machine learning is a complex idea, many programs simply have a function to call to complete the machine learning sequence. I use the `knn()` function to do this. I set the train argument to the trainingSet, the test argument to the testingSet, the answer key (cl) to trainingLabels, and k (the number of neighbors considered) to 0, after some testing. Below is a table for the values which I tested for k, before settling on the one with the least number of missed guesses

Table 1: K value breakdown

kValues	missValues
1	31 misses
2	33 misses
3	19 misses
4	18 misses
5	19 misses
6	19 misses
7	17 misses
8	18 misses
9	17 misses
10	Does not function properly

```
#KNN ML technique
predictionSet <- knn(train = trainingSet, test = testingSet, cl = trainingLabels, k = 9)
predictionSet
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## Levels: 0 1
```

With my prediction set created, I now want to create a dataframe out of the two sets. By doing this, I can now compare the two in any future plotting.

```
#merge data
mergedData <- data.frame(testingLabels, predictionSet)
dim(mergedData)
```

```
## [1] 163   2
```

```
view(mergedData)
```

Next, I want to add all the other columns into my testing set, and I do so using the `cbind()` function. By doing this, I maintain the number of observations (rows), but add in all the already existing data from the parent dataset. First, I store all the names of the testingSet, the rows which were randomly selected to be part of the testing set in the KNN classification. Next, I bind together the testingSet and the mergedData, creating my final data dataset. Finally, I put the column names back onto the final dataset, by using the names I had stored earlier along with two new column names for the added columns.

```
#create the final data
names <- colnames(testingSet) #stores the column names for reference in line 97
finalData <- cbind(testingSet, mergedData)
dim(finalData)
```

```
## [1] 163  23
```

```
names(finalData) <- c(names, "Outcome", "Predicted Outcome")
```

For my final piece of code, I create a crosstable out of my outcome rows, classified by my testingSet and predictionSet datasets.

```
#crosstable
CrossTable(x = testingLabels, y = predictionSet, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  163
##
##
##                | predictionSet
## testingLabels |         0 |         1 | Row Total |
## --------------|-----------|-----------|-----------|
##             0 |         0 |        16 |        16 |
##               |     0.000 |     1.000 |     0.098 |
##               |     0.000 |     0.099 |           |
##               |     0.000 |     0.098 |           |
## --------------|-----------|-----------|-----------|
##             1 |         1 |       146 |       147 |
##               |     0.007 |     0.993 |     0.902 |
##               |     1.000 |     0.901 |           |
##               |     0.006 |     0.896 |           |
## --------------|-----------|-----------|-----------|
##  Column Total |         1 |       162 |       163 |
```

```
##               |     0.006 |     0.994 |           |
## --------------|-----------|-----------|-----------|
##
##
```

There were a total of 17 misses, meaning the machine learning model was able to predict the outcome with a 89.6% accuracy rate. Most of the misses were false positives, in which the model predicted the simulation would succeed where it actually failed. Strangely, the model was unable to predict any failures from the testing set, and this alone accounted for 94% of the failures. Out of 163 testing cases, the model was able to predict 146 of them correctly.

References

Lucas, Klein, D. D., and Y Zhang. 2013. *Failure Analysis of Parameter-Induced Simulation Crashes in Climate Models. Clustering Algorithms.* https://gmd.copernicus.org/articles/6/1157/2013/gmd-6-1157-2013-discussion.html.