Jared Kelnhofer

May 5, 2020

Machine Learning Project 7

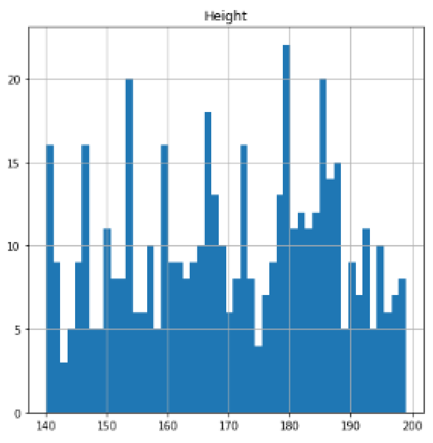
Using Weight, Gender, and Body Mass Index Data to Estimate Height

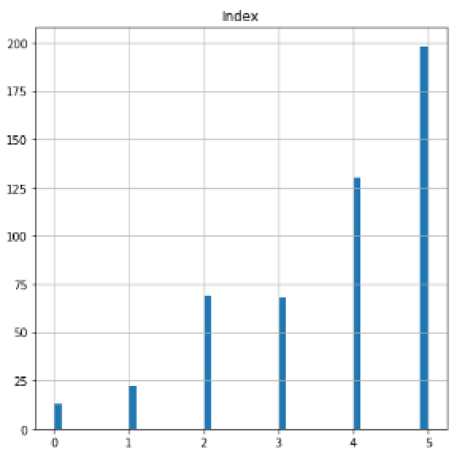
Purpose

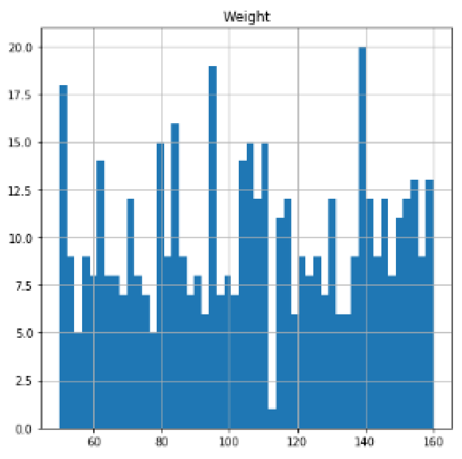
In this project, I will attempt to use machine learning techniques to train a model that can accept 3 features of a person, (Weight, Body Mass Index (BMI,) and Gender) and give a reasonable estimate of that persons height. The challenge with this project is the fact that I have only 500 data instances to work with. This means that pre-processing and careful practices regarding my data are extremely important. One thing that I am excited to learn by doing this project is whether or not a neural network can discover a simple relationship, even with little training data. Because BMI is related to Weight and Height by a simple equation, my hope is that my model will discover this early on and be able to give very good answers despite my small amount of training data.

Phase 1: Acquiring and Exploring the Data

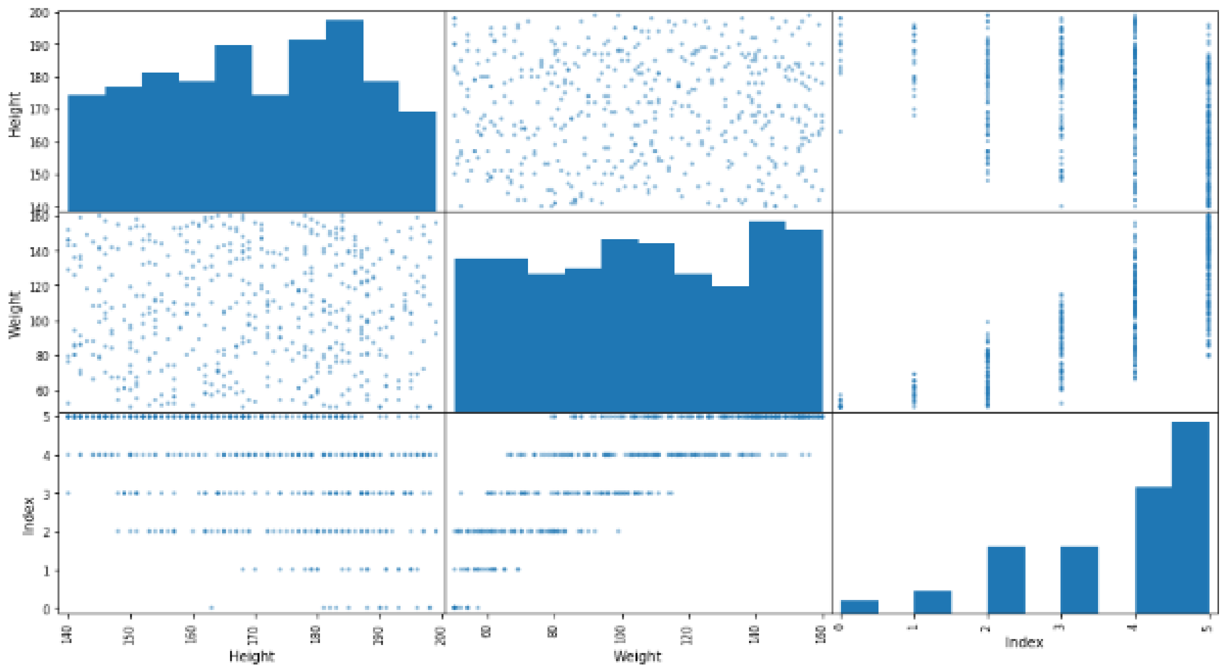
I started my project by using Pandas to import my data into a Dataframe. This allowed me to quickly and easily explore the data and check for missing values. Because I had so little data to worry about, I checked for missing values by eye in about a minute. I also plotted histograms for each numerical feature that showed how often values fell into certain ranges. The "Index" histogram revealed that people were put into one of five categories depending on their BMI. This erased a lot of information that would have helped my model find the relationship between Height, Weight, and BMI.





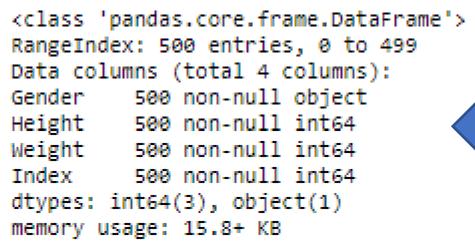


I also plotted a correlation matrix and plotted all numerical features against each other to see what relationships existed in the data. I was extremely surprised at first to see just how unrelated Height and Weight were but remained hopeful that I could use the model to find a relationship between Height, Weight, and Index.



Phase 2: Preparing the Data for Machine Learning

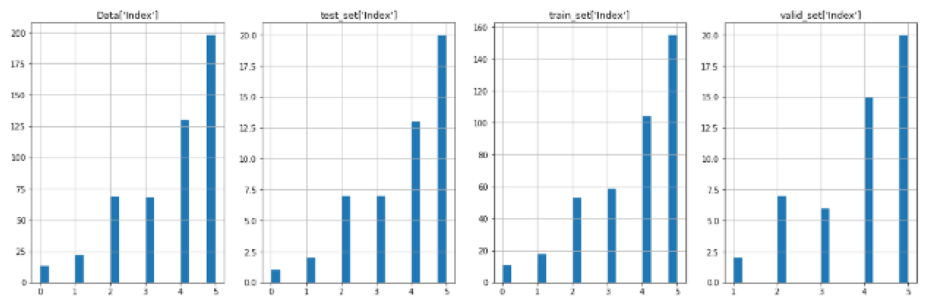
After exploring the data, I knew that there were a number of things to work on before it would be ready for machine learning. I decided to use one-hot encoding to deal with the categorical Gender feature, and also scaled my features into a range between 0 and 1 using the *MinMaxScaler()* class from Scikit-Learn. To keep my small training, validation, and test sets as effective as possible, I implemented stratified splitting with the *StratifiedShuffleSplit()* class in Scikit-Learn. Along the way I was careful to print out information about the subsets, so that I would not make a mistake early on. Keeping track of the data types, shapes, and names of my sets kept me from mixing them up.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 4 columns):
Gender    500 non-null object
Height    500 non-null int64
Weight    500 non-null int64
Index     500 non-null int64
dtypes: int64(3), object(1)
memory usage: 15.8+ KB
```

Here are the columns in my original DataFrame. All but one of the features, (Gender) turned out to be integers. To handle the categorical Gender feature, I decided to go with one-hot encoding by using the handy *get_dummies()* method provided by Pandas.

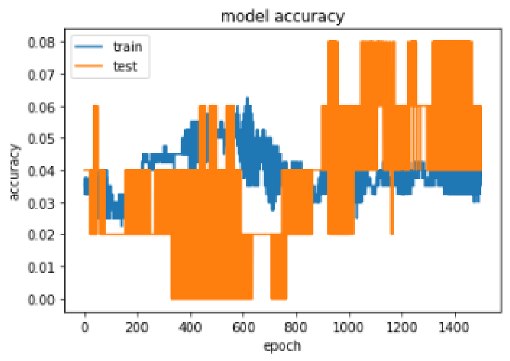
After several bad attempts at using the default Index column, halving the loss on my training set by converting all values in the Index column to the calculated BMI based off of that instance's Height and Weight. I did this after splitting the data because I needed a multi-class feature for stratified splitting.



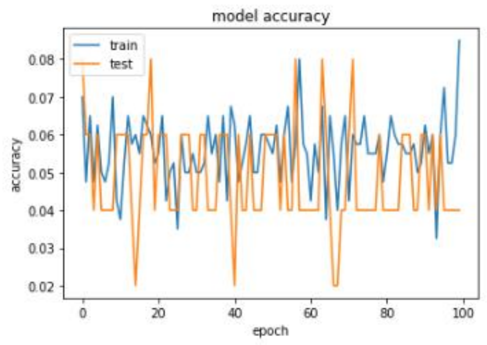
Here are the Index columns of the validation, testing, and training sets after the stratified split, all compared to the Index column of the original dataset.

Phase 3: Preparing and Training a Model

I used Keras for this assignment, and went with a feedforward, densely connected neural network with an input shape of 4, two hidden layers with 10 and 20 nodes respectively, and a final output layer with a single node. I tried much more complex networks but did not notice any improvement. The thing that was most difficult about choosing my network architecture was grasping how exactly a regression model should work. I had to change what loss metric I was using, which took me a while to figure out. I also ended up using a learning rate of 0.1, which is a lot higher than some of the others I tried. The whole process felt much like stumbling around in the dark, but I eventually got my error down to around 110 on average, so that should put me at a +/- 10 cm error. While an error that large is far from ideal, I think that with 500 features to work with its pretty decent. When training my models, I always used callbacks to save the best model, and tested it as a reflection on the entire training process for that particular configuration.

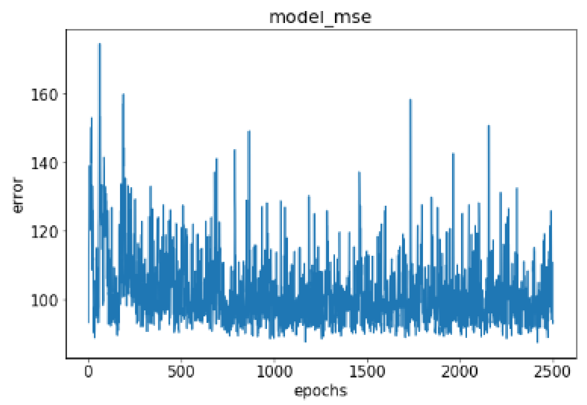


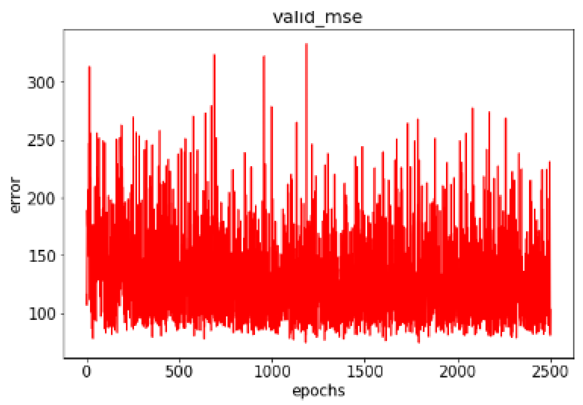
These amazing graphs happened when I was using an incorrect loss function while training my model. I was using *binary_crossentropy*, which I found is inappropriate for regression tasks. I ended up switching the graphs I used from accuracy to loss I figured that out.



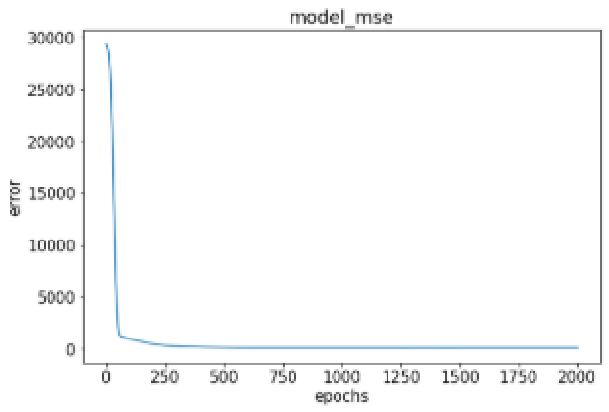
I think that the jumpy behavior of this graph in particular might be due to the fact that the learning rate was set too high. I'm curious about the sort of square wave behavior of the test set as well. One of the things I noticed with these early models is that they didn't usually converge towards a solution, even if I added a lot of epochs to the learning process. They just jumped around.

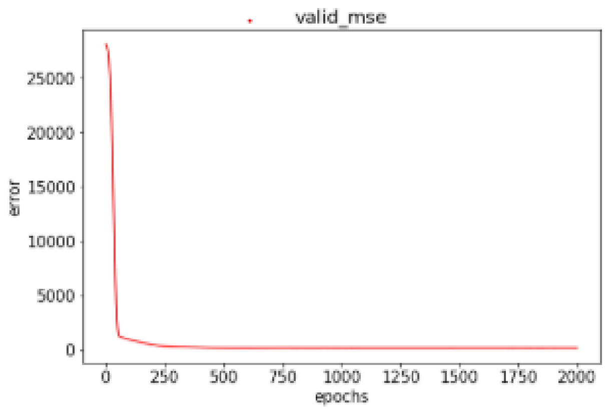
The model that caused these (below) graphs produced one of my lowest errors (105,) but I wasn't too happy with how it was jumping around. This model was created after I switched my scoring metric to MSE, so I attribute the jumpiness to a very high learning rate, and not an incorrect loss function. I was able to lower the learning rate and achieve a much smoother drop in the error, but the error always seemed to end up plateauing, and never could get under 100.





These (below) graphs show the result of a very similar model, but with a much smaller learning rate. I wasn't able to get the error as low as the above model, even after taking the number of epochs to 20,000, but at least this model was not so wild. My error ended up around 130 most of the time.





Conclusion:

In conclusion, this project was tremendously helpful for me. I feel like I have been able to gain some intuition into how neural networks work and are affected by various hyperparameters. I also learned the importance of feature engineering, and the value of providing enough training data to a machine learning algorithms. This project was fun, and I hope that I am able to use it to progress towards a more thorough understanding of machine learning.