Big Data

Coursework

Chronic Kidney Disease

Liam Noonan

S1512127

[Lnoona200@caledonian.ac.uk](mailto:Lnoona200@caledonian.ac.uk)

*“I declare that all work submitted for this coursework is the work*

*of Liam Noonan alone unless stated otherwise.”*

# The Problem

The provided dataset contains 400 medical records, each with 25 values pertaining to patient information such as their age, blood pressure and blood sugar levels etc and finally whether or not the patient has Chronic Kidney Disease defined in a class column. The purpose of this investigation is to determine if there is a relationship between these data points and a patient having Chronic Kidney Disease and if by a process of data pre-processing and the use of a neural network can this be predicted. The neural network implementation used will be the Scikit Python library (Fabian Pedregosa, 2011) which provides plenty of easy to use functionality to create a neural network as well as to help with the training process (Jose Portilla, 2017).

From the provided CSV data (imported using Pandas Python library) 11 of the values are of a Boolean form but represented by strings such as Yes or No with the remaining 14 being numeric. Before pre-processing, 237 of the records are of class ‘ckd’ while the remaining 163 are ‘notckd’. Many values are missing as marked by ‘?’, more prominent in some values than others such as ‘rbc’ which has 152 instances of a null value.

|  |  |
| --- | --- |
| Figure 1 - Distribution of class | Figure 2 - Example of rows missing values |

# Construction and Tuning

## Pre-Processing

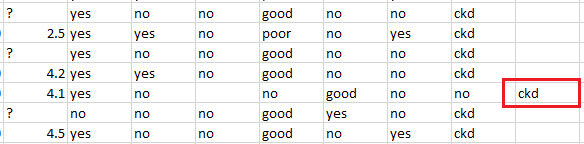
At first glance there are immediately some rows of data that will need to be excluded. Firstly, a number of rows have been shifted so that their class variable is no longer valid. These are dropped by removing the additional column it has created and then by creating a new dataset from the rows that have valid class values. 

Figure 3 - Invalid row due to offset variables

At this stage there are still many rows that are missing values, represented by ‘?’. These values will be replaced using Numpy to assign a NaN value such that they can then be removed using dropNa. This leaves 155 valid rows with complete data. Of the remaining records, more are classed as not having CKD than those who do have it, specifically ‘ckd’:41, ‘notckd’:114. This varies from the original unfiltered dataset which showed a larger number of patients with CKD than without which may have the potential to skew the data.



Figure 4 - Distribution of class after pre-processing

The next step is to attempt some analysis of the remaining data to see if there are any obvious relationships or otherwise – if there are values or columns that could be further eliminated to make the process less complex. Albeit that this is unlikely to make much of a difference due to the relatively small sample size. To accomplish this matplotlib and seaborn libraries were used to generate a heatmap of the variables;

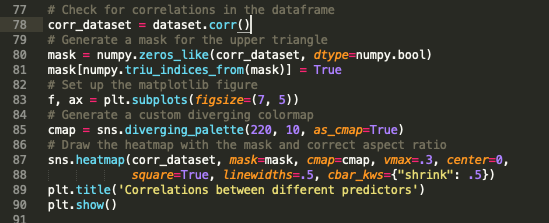


Figure 5 - Example code to create a Heatmap

From the resulting table there wasn’t a great deal to be gleaned. Some relationships showed less correlation than others such as ‘hemo’ and ‘pcv’ but seemingly not to the extent that removing them from the dataframe would produce a better overall result as at this stage limiting the number of complete records any further could also be detrimental to the results.

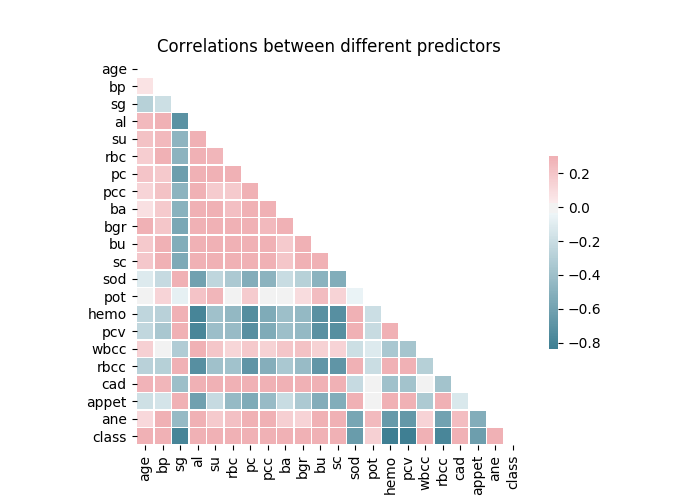


Figure 6 - Heatmap of correlations

## Training

Firstly, to begin creating the training dataset the data frame was separated into one frame containing the attributes and the other containing the class information in order to allow for the training data to have class while the test data does not. These dataframes were then passed to Scikit’s train\_test\_split function which automatically splits the data into training and test sets.

Scikit then recommends using a scalar as part of the pre-processing stage to normalise the training data:

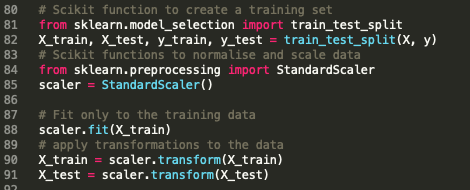


Figure 7 - Scikit pre-processing example

From here, the Scikit Multi-Layer-Perceptron classifier model will be used to create the neural network itself and define the number of hidden layers and iterations. After fitting the data to the model, predictions can be drawn and examined using the metrics library - specifically the classification report and a confusion matrix.

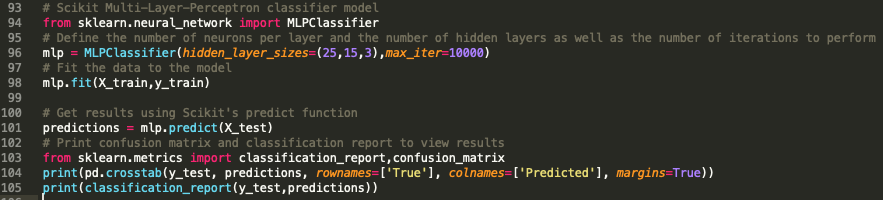


Figure 8 - Use of Scikit Multi-Layer Perceptron Model

# Testing

For the initial test, a neural network with three hidden layers was selected as an arbitrary starting point with the input layer consisting of 25 neurons. 15 and 3 neurons were selected for the other hidden layers respectively with a maximum of 10,000 iterations. This resulted in an accuracy of 100%.

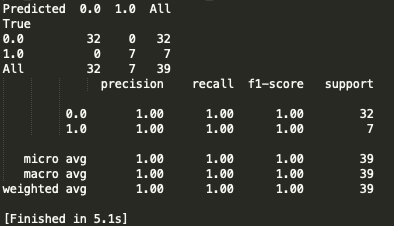


Figure 9 - First Test

This suggested that something might be amiss as it would be unusual to get complete accuracy on the first attempt. However, by repeatedly running tests it can be seen that occasionally the accuracy will drop albeit not by much as seen in Figure 10 which occurred after three consecutive reruns of the program, suggesting that it is in fact working;

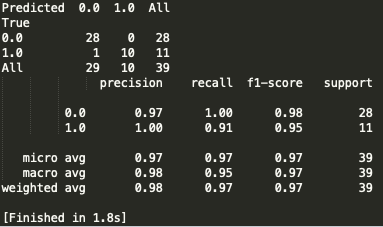


Figure 10 - Less accurate test example

Further tests in varying the number of neurons per hidden layer and the number of hidden layers showed that less layers or less neurons (or both) would negatively affect the accuracy as seen by Figure 11 and 12 which used a neuron configuration of 3,2,1 and 2,1 respectively.

|  |  |
| --- | --- |
| Figure 11 - (3,2,1) Hidden Layer Results | Figure 12 - (2,1) Hidden Layer Results |

# Discussion

Data pre-processing was key to this problem. For starters, without properly encoding the data and dealing with null values the analysis either would not have worked at all or would have returned a meaningless result. With a larger dataset it might have been more effective to remove the values with less correlation as seen in Figure 6, however with the limited number of complete records it made more sense to leave the data as is.

One concern with the overall analysis was that a large proportion of the data was dropped due to null values. In many cases more values could likely have been substituted, for example ‘sc’ was missing on only 17 rows. Given that ‘sc’ had a minimum value of 111 and max of 150 with a mean of 138, these rows likely could have been kept as part of the data frame by substituting some variation of the mean into the missing values.

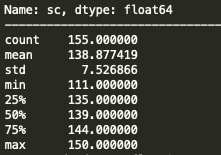


Figure 13 - Distribution of attributes of 'sc'

Furthermore, by dropping the null values in the table, so many rows were eliminated that the overall shape of the data changed to show that less records had Chronic Kidney Disease than those with, which is the opposite of the initial shape whereby more had it than not. This could potentially have some effect on the output as it affects the training set used to teach the neural network how to evaluate the variables to determine a class.

Using the Scikit normalisation and scalar functionality was key to obtaining such a high level of accuracy. For example, upon rerunning the same Hidden Layer configuration as Figure 11 (3,2,1) the following results as shown in Figure 14 are significantly less accurate, proving that it was an effective element of establishing the training sets.

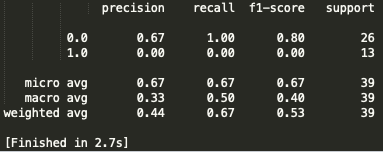


Figure 14 - Results without Scikit normalisation

The use of the Multi-Layer Perceptron classifier model seems to have been fairly effective but this is hard to tell without comparison to other models. This could be an avenue for further research as Scikit provides a selection of different models to choose from which could quite easily be substituted into the same scenario.

To conclude, more time could have been spent on exploring the data as far as looking for correlations and trends using various visualization techniques which could help to identify outliers which to be eliminated in the pre-processing stage or to allow for substitution to better filter the number of variables being used by the neural network. That being said, the completed solution has a very high accuracy and so could be viewed as quite successful based on the data given.

# Appendix

## Code Listings: