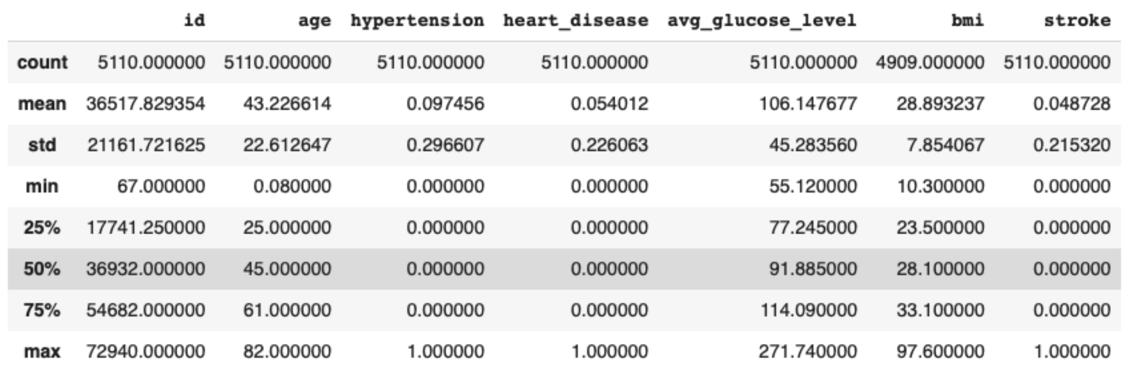
CS4287 Assignment 1 Report

Dylan Kearney - 18227023
Cyiaph McCann - 17233453

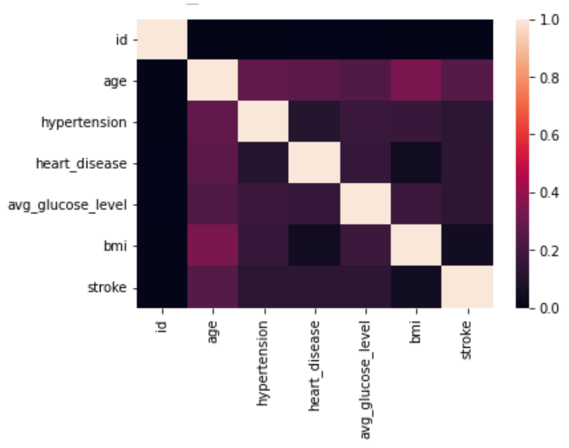
The Data Set: We used a stroke prediction data set for our MLP. This data set contains attributes such as gender, age, hypertension, heart disease, glucose level, work type, ever married and BMI.

Visualization of key attributes:



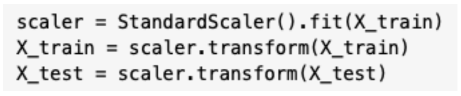
	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	4909.000000	5110.000000
mean	36517.829354	43.226614	0.097456	0.054012	106.147677	28.893237	0.048728
std	21161.721625	22.612647	0.296607	0.226063	45.283560	7.854067	0.215320
min	67.000000	0.080000	0.000000	0.000000	55.120000	10.300000	0.000000
25%	17741.250000	25.000000	0.000000	0.000000	77.245000	23.500000	0.000000
50%	36932.000000	45.000000	0.000000	0.000000	91.885000	28.100000	0.000000
75%	54682.000000	61.000000	0.000000	0.000000	114.090000	33.100000	0.000000
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.600000	1.000000

We added a heatmap graph in order to better understand the correlation between the different variables in the dataset. Each value in the data set is plotted and compared against itself and every other value listed. The colours in the chart depict the likelihood of having a stroke which is returned as a percentage. For example, the combination of the age and BMI attributes are most likely to be the main factors of having a stroke having a colour that is closer to pink which in this case is the lighter shade indicating around a 40% chance that this person has had a stroke.



Pre-processing / normalization

To normalize the values in our data set we calculated the distance of each value in the data set to the mean value and the divide that by the standard deviation $(x - \mu) / \delta$.
This standardizes our data. The data is normalized using the 'sklearn' libraries StandardScalar function as seen below with the .fit() and .transform() functions.



```
scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Network Structure

We created the Keras model by adding layers, in our case we added 2 hidden layers. Our network is also designed using the sequential model as we wanted a traditional feedforward MLP. Both hidden layers are made up of 10 neurons each and for our model, we chose to use the ***relu*** activation function, for the output layer we chose the sigmoid activation function.

The Cost/Loss/Error/Objective Function

We initially chose binary-cross-entropy as our loss function in our neural network because our label data is binary. The stroke prediction outcome returned either a 1 or a 0 indicating that this person has had a stroke or hasn't.
Upon further testing, we discovered that Mean Squared Error (MSE) was also an applicable option as it returned a lower loss rate.

The Optimizer

We chose to use the Adam optimization algorithm for training our models. Adam is a replacement optimizer for stochastic gradient descent that combines the Adaptive Gradient Algorithm and also Root Mean Square Propagation.

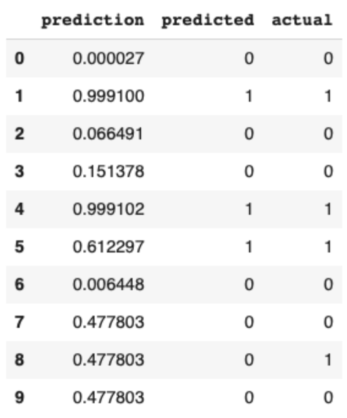
https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

Cross Fold Validation

Cross-validation is a resampling technique for evaluating machine learning models on a small sample of data. We separated our Stroke prediction dataset into X_train, y_train, X_test and y_test. We fitted the model using the training sets and evaluated the model using the test sets.

Results

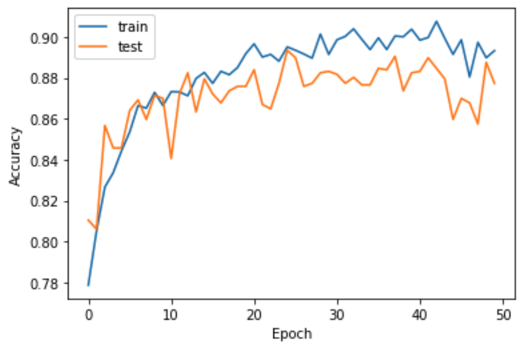
The results of our MLP are based on a threshold of 0.5 due to the activation function of the output layer being sigmoid. This means that any value that is > 0.5 the model will predict that the person will likely have a stroke. Below is the output our model has produced; we can see that for row 8 the prediction fell short of 0.5 and therefore our model did not predict a stroke when the actual result should have been a positive result.

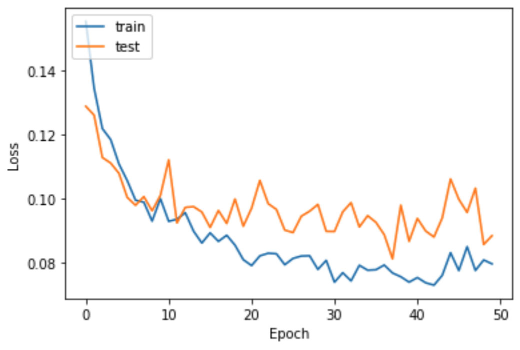


	prediction	predicted	actual
0	0.000027	0	0
1	0.999100	1	1
2	0.066491	0	0
3	0.151378	0	0
4	0.999102	1	1
5	0.612297	1	1
6	0.006448	0	0
7	0.477803	0	0
8	0.477803	0	1
9	0.477803	0	0

```
True      2572
False      345
```

From the results table above we can easily work out how well our model performed by checking how many results it got right and how many of them were wrong. In our case, our model correctly predicted 2572 cases and 345 of them were incorrectly predicted. We can visualize the progressing of the predictions over each epoch as seen in the graph below (left), this shows how the accuracy of our model changed over time. The other graph (right) illustrates how the loss changed over each epoch.





Evaluation of results

Loss Function:

1. Binary_crossentropy: 89% accuracy, 25% loss
2. MSE: 89% accuracy, 8% loss

We decided to use MSE as there was a significantly lower loss percentage compared to binary_crossentropy.

Optimizers:

1. Adam -> ~86.5% accuracy, 28% loss
2. SGD -> ~77.6% accuracy, 47% loss
3. RMSProp -> ~86.6% accuracy, 31% loss
4. Adamax -> ~82% accuracy, 40% loss

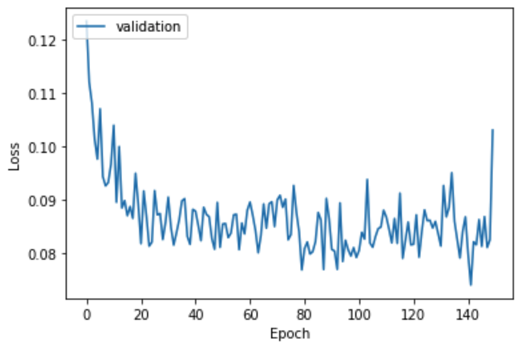
We chose Adam as our optimizer as it returned the best results with minimal loss.

Epochs:
We looked at how many epochs would return the best results, attempting to find the sweet spot in order to stay between underfitting and overfitting. After trailing many different numbers of epochs we found that around epoch number 50, the loss and accuracy of the

model begins to improve at a much slower rate. As a result, we decided to set the number of epochs to 40.

Batch Size:
We tried increasing the batch size and quickly noticed the accuracy reducing. After multiple attempts with different batch sizes, we found that setting it to 16 allowed us to speed up our computation while maintaining good accuracy.

Checking for overfitting and underfitting:
We used EarlyStopper to choose a good stopping point for the model. To confirm our selection, we executed our model with a large number of epochs and we visualized the results to see where the best point on the curve is. We made a similar MLP model with extra epochs and the results showed that around 50 epochs returned the best results.



Impact of varying a hyperparameter(s)

1. We experimented with changing the size of the training and test sets, changing them from 0.3 to 0.2 and we found that the accuracy went down as the model was overfitting.
2. After Implementing SMOTE to fix the imbalances in our data we tested our model by reducing the number of hidden layers down to just one layer. This resulted in a loss of ~50% indicating that our model was underfitting.
3. We added back in our initial second layer and the loss was reduced to ~40% giving us an improvement of about 10%.
4. We also experimented with the number of neurons per layer by increasing and decreasing the amount per layer, we found that having 30 neurons in the first layer and 15 neurons in the second layer gave us the best results for our model.
5. We changed the activation functions around using different ones such as signum and relu. Our conclusion was to apply softmax to the input layer, relu to both hidden layers and then to apply sigmoid to the output layer. This combination gave us the best results for our MLP

Softmax: Softmax is similar to sigmoid. Softmax returns the probability of each class. Z represents the values of neurons. Exponential acts as a non-linear function and then these values are divided by the sum of exponential values.

$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

Relu: Relu is a linear function that will output the input directly if it is positive otherwise it will output zero.

Sigmoid: Sigmoid is used to add non-linearity, it will determine which values to pass as output and which values not to pass.

Linear Regression
We attempted to apply linear regression to our data set to check to see if the data is linearly predictable. We only got around a 9% accuracy, so we decided that it wasn't a good choice to use it.

Findings:
Below are some of the findings and learnings we gained with this project.

- Increasing batch size reduces the computation time of the MLP
- Our model is dealing with a binary classification problem so we decided to use either MLP or binary_crossentropy.
- We were initially getting 0.953 accuracies consistently and without any change, even after altering our MLP to one hidden layer the result would stay the same, we investigated this and discovered that there was an imbalance in our dataset more specifically in the stroke column. Out of the 5000+ rows of data, there were approximately 250 rows in the data with strokes. In order to deal with this, we employed SMOTE to refit the data and to attempt to balance it. SMOTE or Synthetic Minority Oversampling Technique is used to increase the number of cases in our dataset. It creates new instances from the existing minority positive stroke results.
- As we approached the end of this project we used the MSE loss function and it reduced the loss of our MLP from around 25% down to 9%.