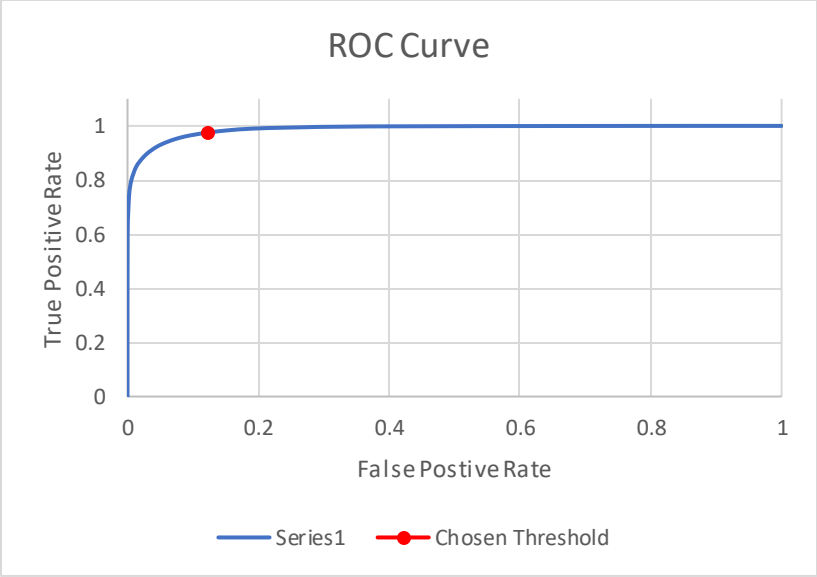
Homework 2 – Bone – James Sheridan 17463312

I wrote the classifier primarily using PyTorch but I also made use of some sklearn features for preprocessing and metrics. Upon loading in the data, it is split into training and validation sets (80/20 split). At first I tried normalizing the data by simply dividing every value by ten so that all values would be between –1.0 and 1.0, this didn't do much to help performance, so I implemented Standard Scaler from sklearn which improved performance by roughly 5% in my earlier models.

The model itself consists of several linear layers, each of which uses ReLU activation and has Batch Normalization applied to it. Initially I had two linear layers that scaled up the input features from 12 to 100 which is then taken back down by an output layer. I eventually added another of these layers which took the input features as high as 200 which brought my overall accuracy above 90% but obviously at the cost of training time. I also introduced dropout to reduce overfitting when I began to train on higher numbers of epochs.

I tested several versions of the Adam optimization algorithm that are included in PyTorch including Adam, AdamW, SparseAdam and Adamax. I found very little improvement in the non-standard versions, so I continued using the generic Adam. For my loss function I opted to use Binary Cross Entropy with Logits Loss over the regular Binary Cross Entropy as it includes a Sigmoid activation in the same layer and according to PyTorch documentation it is more numerically stable to apply both in one layer than separately.

While tuning parameters the first thing I did was increase the number of epochs from 50 to 100 and then to 300. In the jump from 100 to 300 I found that reducing my learning rate to 0.0001 was much more effective than leaving it at 0.01. Upping my batch size also boosted my performance by a couple of percent but I found that anything after 100 made very little difference. With a batch size of 200 gaining less than a percent.



My final model without modifications to the threshold had a true positive rate of 0.93 with a false positive rate of 0.05. To adjust the threshold, I simply added a value to each prediction before they were rounded to 1 or 0, adding 0.25 to all values effectively reduces the threshold by 25%. I wrote a utility

script to iterate over a range of adjustments to the thresholds and print a confusion matrix and classification report for each evaluation. I used these metrics to plot an ROC curve. Many of the thresholds evaluated were simply to help Excel generate the curve but some of the relevant thresholds can be seen in the table below.

Threshold	Adjustment Value	False Positive Rate	True Positive Rate
50% (base)	0.0	0.051595286	0.931515792
45%	0.5	0.931515792	0.941743347
40%	0.10	0.073440644	0.951628948
35%	0.15	0.087632941	0.960830639
30%	0.20	0.10383731	0.96829147
25%	0.25	0.125287439	0.976218602
20%	0.35	0.153025295	0.983523999

I settled on lowering the threshold to 26% (setting adjustment value to 0.24). If correctly classified 1s are worth 3 points and correctly classified 0s are worth one point, this is the point in my validation set where sacrificing correct 0s for 1s was no longer worth the trade. Confusion matrices are listed below for the unmodified threshold, chosen threshold as well as the thresholds 1% either side of the chosen threshold. As well as the estimated points score if the validation set was the real test set (Note that the validation set is six times the size of the real test set, hence the large scores).

50% Base	0	1
0	26396	1436
1	2203	29965

50%: (29965*3) +26396 = 116,291

27%	0	1
0	24556	3276
1	843	31325

27%: (31325*3) + 24556= 118,531

26%	0	1
0	24418	3414
1	791	31377

26%: (31377*3) + 24418= 118,549

25%	0	1
0	24266	3566
1	741	31427

25%: (31427*3) + 24266= 118,547