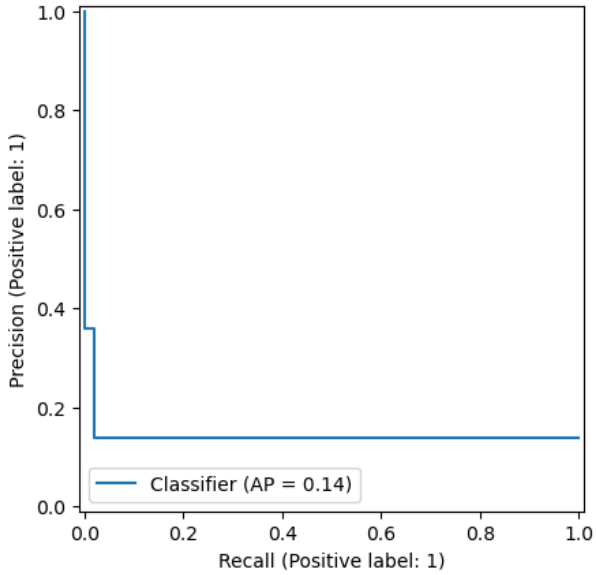
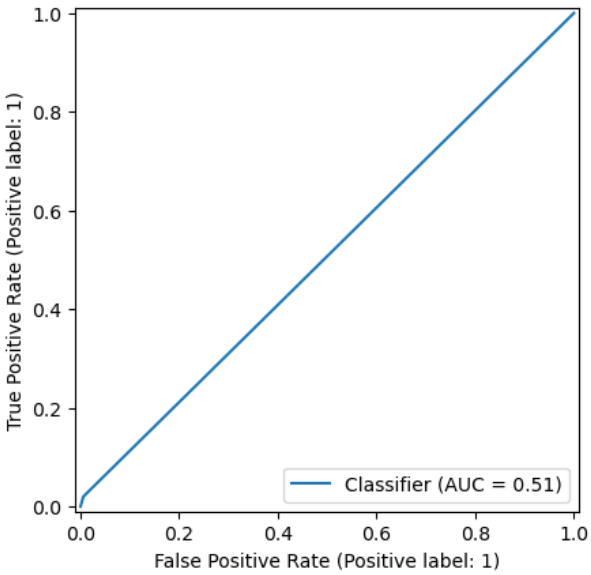
1. Build and train a Perceptron (one input layer, one output layer, no hidden layers and no activation functions) to classify diabetes from the rest of the dataset. What is the AUC of this model?

First, I removed all rows containing missing data from the dataset. As stated in the spec sheet, there is not too much missing data, and the size of the dataset remains very large, so it is okay to remove rows. I then one-hot encoded the zodiac data, because the values make sense to interpret categorically and removed the original column. I then split the data 80/20 into training/test sets. This is more training data than I previously did with the other classification methods, as neural networks require a lot of data. I then fit a neural network with 31 input nodes, 2 output nodes, and no activation function. This gave the following PR and ROC curves.

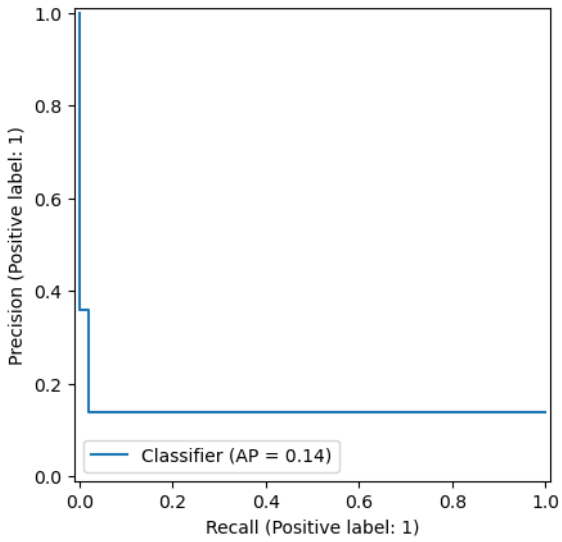


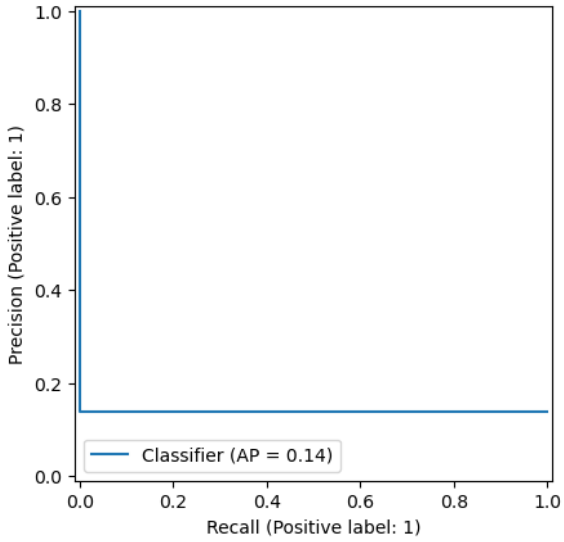


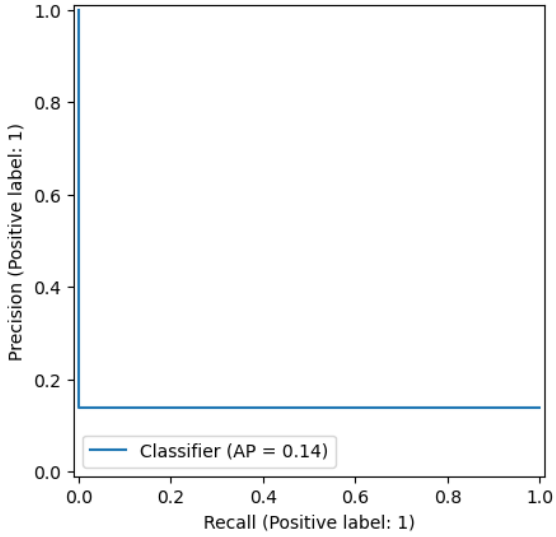
The reason that they are so different is that the classes are not balanced. This means that the model can pretty much always predict that the subject does not have diabetes and still maintain a decent AUROC. The AUPRC is more telling than the AUROC in this case because the classes are unbalanced, with only around 13.93% of the entries having diabetes. So, the AUPRC, 0.14, is a better measure of model performance in this situation. This AUPRC is not very good, given that the baseline is 0.1393.

2. Build and train a feedforward neural network with at least one hidden layer to classify diabetes from the rest of the dataset. Make sure to try different numbers of hidden layers and different activation functions (at a minimum reLU and sigmoid). Doing so: How does AUC vary as a function of the number of hidden layers and is it dependent on the kind of activation function used (make sure to include "no activation function" in your comparison). How does this network perform relative to the Perceptron?

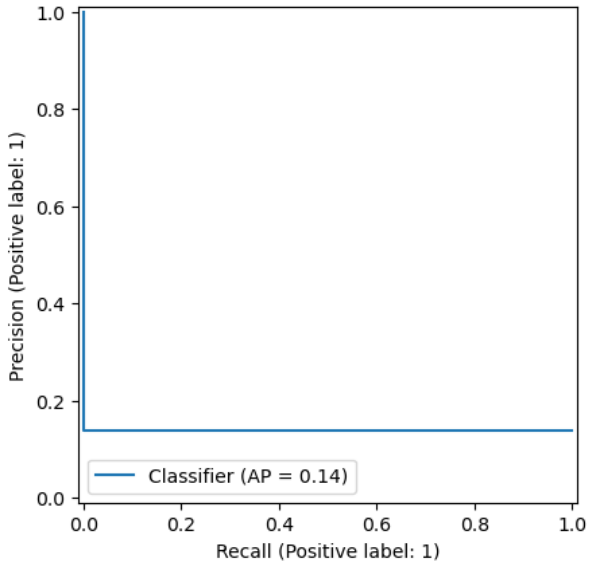
I built neural networks using 1 hidden layer with 17 nodes, which is the mean of the number of nodes in the input and output layers rounded up. I used no activation function, reLU, and sigmoid as activation functions for these. This gave me the following PR curves respectively:

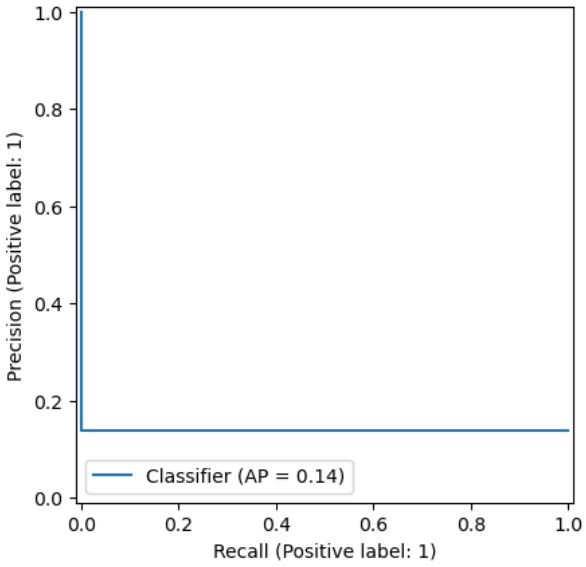


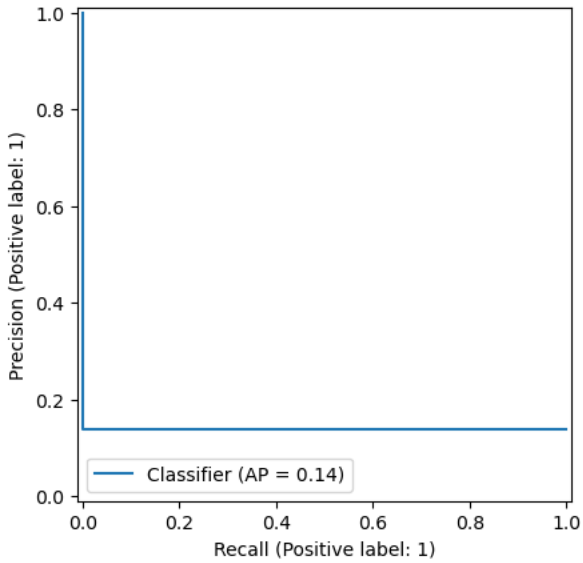




The AUPRC is the same for all of these graphs, showing that all of them had very similar performance. In order, they had accuracy 85.95%, 86.19%, and 86.19%, which shows that the reLU and sigmoid activation function using models were slightly better. I then built 3 neural networks using no activation function, reLU, and sigmoid with 2 hidden layers with 17 nodes each in order to determine the effect of increased hidden layers on these models. This gave me the following PR curves respectively:



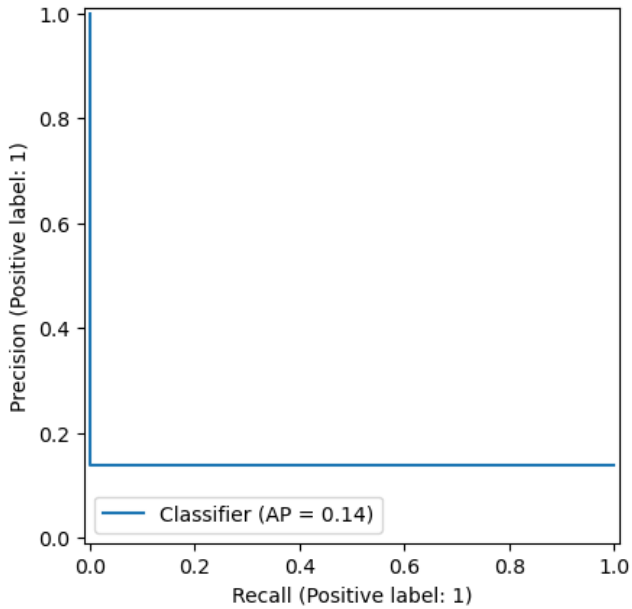




The AUPRC is the same for all of these graphs, showing that all of them had very similar performance. In order, they had accuracy 86.17%, 86.19%, and 86.19%, which shows that the reLU and sigmoid activation function using models were slightly better. The AUPRC is the same as the models with 1 hidden layer for each activation function, but the accuracy improved for the model with no activation function. These models perform very similarly to the perceptron, although the accuracy has increased, especially for the models using an activation function.

3. Build and train a "deep" network (at least 2 hidden layers) to classify diabetes from the rest of the dataset. Given the nature of this dataset, is there a benefit of using a CNN for the classification?

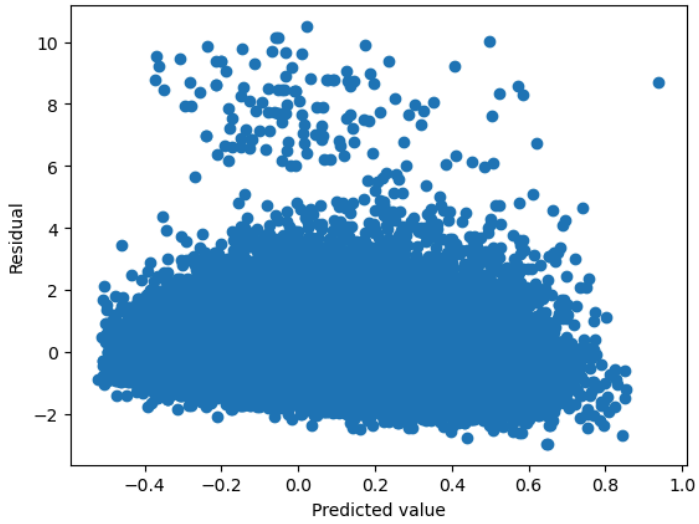
Building a deep neural network using reLU as an activation function, dropout, and two hidden layers gave the following graph:



This is the same as all of the previous models, showing that the performance is very similar. The model has an accuracy of 86.18%. Because adjacent entries have nothing to do with each other, using a CNN would not have a benefit.

4. Build and train a feedforward neural network with one hidden layer to predict BMI from the rest of the dataset. Use RMSE to assess the accuracy of your model. Does the RMSE depend on the activation function used?

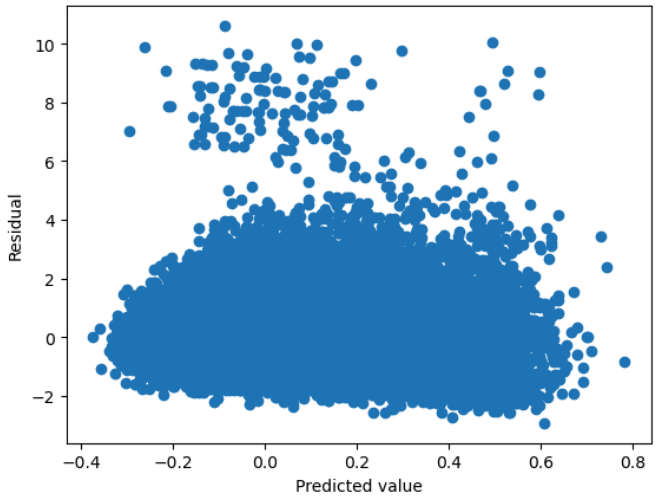
First, I normalized the BMI data by transforming every value to its z-score in order to get a more interpretable RMSE. I trained a feedforward neural network with one hidden layer with 16 nodes, the average of the number of nodes in the input and output layers, and no activation function. This gave the following graph of residuals and an RMSE of 0.9723, which suggests the model is pretty good at predicting BMI using the data.



Building the same model using reLU as the activation function gave an RMSE of 0.9890, which is slightly worse than no activation function. Doing the same using sigmoid as the activation function gave an RMSE of 0.9921, which is slightly worse than both of the previous models.

5. Build and train a neural network of your choice to predict BMI from the rest of your dataset. How low can you get RMSE and what design choices does RMSE seem to depend on?

I tried neural networks using no activation function, reLU, and sigmoid, and up to 3 hidden layers. The neural network with the lowest RMSE had no activation function and 2 hidden layers with 16 nodes in each. The RMSE was 0.9617.



The RMSE seems to depend on the activation function and the number of hidden layers. The RMSE varied between trainings on the same model, so it was hard to see whether other design choices affected the RMSE, but learning rate, weight decay, and number of nodes in hidden layers should also affect the RMSE.

Extra Credit:
b) Write a summary statement on the overall pros and cons of using neural networks to learn from the same dataset as in the prior homework, relative to using classical methods (logistic regression, SVM, trees, forests, boosting methods). Any overall lessons?

Neural networks seem to require significantly more data than other methods. With the same dataset, the neural networks were a lot more reliant on the specific test/train data split than any of the other methods. There is a lot more hyperparameters with the neural networks, and it required more trial and error in order to find the best model.