Zavier Andrianarivo
Professor Wallisch
Homework # 4
NetID: zoa215

CSCI-UA 473: FML

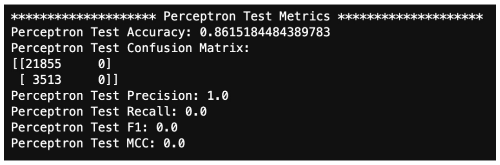
April 11 2025

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?

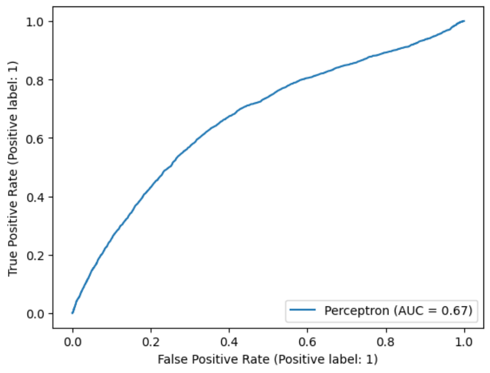
Answer: To answer this question I used the Perceptron class from Sci-kit learn. I remember in lab (as well as by reading over the lab4.ipynb file) a perceptron is essentially the same as running the SGD classifier from the SVC class in scikit-learn. I might have messed up a little bit because I didn't preprocess any of the data. I mixed up the notes between decision trees not being sensitive to data and SVMs being sensitive to data. However, I fixed this later on in my other models.

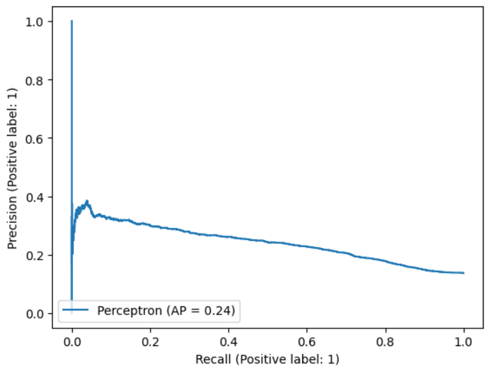
To state again, pre-processing wasn't done because of the mix-up between model sensitivity to processed data (how data is processed, etc.). However, one thing I would have done is compute the correlation coefficients between all of the vectors (inputs) of X to determine any collinearity. Since we're working with 22 features, the feature list isn't too big, but for sake of time, it could be beneficial to reduce the size of the matrix (if possible).

Since there weren't many options for design choices other than calling the library, I found that with a single perceptron, I acheived accuracy levels of **0.86**. However, I also achieved inaccurate precision, recall, and F1 scores, **1**, **0**, **0**, and **0**.

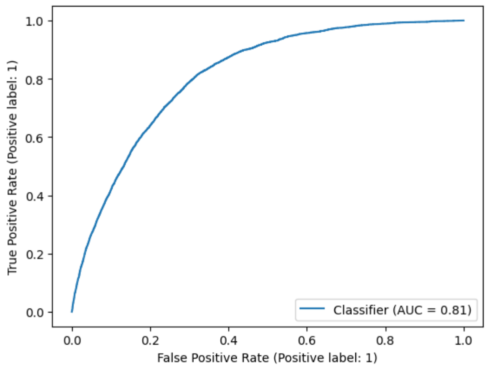

```
******************** Perceptron Test Metrics ********************
Perceptron Test Accuracy: 0.8615184484389783
Perceptron Test Confusion Matrix:
[[21855     0]
 [ 3513     0]]
Perceptron Test Precision: 1.0
Perceptron Test Recall: 0.0
Perceptron Test F1: 0.0
Perceptron Test MCC: 0.0
```

The AUROC and AUPRC were much more telling, giving an AUC of **0.67** of the RoC an AUC of **0.29** under the PRC.





Looking at the AUC of the perceptron and evaluating it's performance, we can actually note that it performs worse compared to our SVM classifier from the last homework.



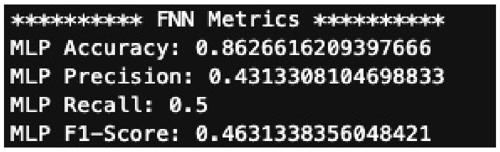
The AUROC of **0.67** of the perceptron indicates that the model performs somewhat decent, but the nature of the class imbalance within the dataset indicates that the perceptron is also prone to sensitivity among the majority class. For a lot of these linear models, we see there is a struggle to capture any patterns in the data - indicating there is some nonlinear patterns that the models were not able to uncover.

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?

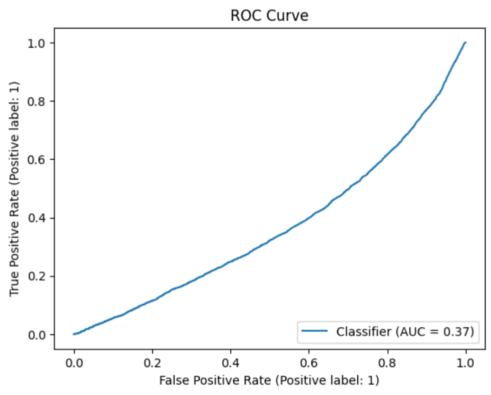
Answer: For a simple feedforward neural network, the model performed really nicely compared to all of the other networks we've had to implement on the diabetes dataset. Again, I did something really similar to question 1 and did not scale the features. The mix-up happened as I was working on question 1 and two at the same time. However, in terms of designing the models, I did a few things different than compared to the single perceptron. Design-wise, I built one neural network with one hidden layer, and then a MLP model where I can pass in the input of the activation function I would like to use.

This was done to analyze how the model performs based on different activation fucntions. This would come useful later on in the assignment (question 5). From the last assignemtn and question 1, we could see that linear models had trouble with sensitivity to the majority class of the dataset (being **0** in this case). However, choosing the activation function (and having activation functions, in general) helped a lot with the model's ability to recognize patterns in the data.

Deciding the types of activation functions and number of dense layers was a nice hump for the model to get over. Upon training and testing, we can see that the model performed better than the single perceptron, with an accuracy of **0.86**, a precision score of **0.43**, a recall score of **0.5**, and an F1-Score of **0.46**.


```
********** FNN Metrics **********
MLP Accuracy: 0.8626616209397666
MLP Precision: 0.4313308104698833
MLP Recall: 0.5
MLP F1-Score: 0.4631338356048421
```

Our AUROC and AUPRC feels a little misleading, providing an AUROC of **0.37**.



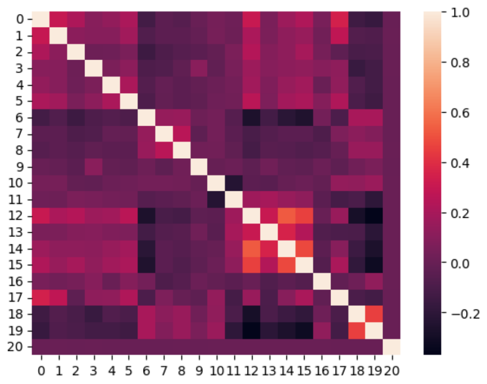
I think the usage of activation functions, for sure, helped with the mdoel being able to recognize nonlinear patterns/structures in the data. The usage of the *Sigmoid* activation function gave similar results to our logistic regression model, combining both linear and non-linear methods of classifying our output labels (given working in with a binary classification problem).

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?

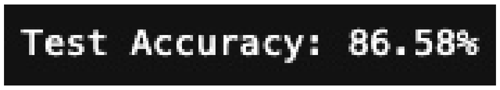
Answer: For this problem, I might have misinterpreted the question. For this problem, I did a lot. Essentially, what I did was I wanted to implement a convolutional neural network to see if we could recognize any spatial patterns in the data (I know that CNNs work nice for image classification as per lab, but spatial patterns can emerge in medical data as well). I also used PCA, given I did nothing to the data in the first two questions, to experiment more.

I chose to use PCA for this question (in addition to the CNN, which might have been overkill/too much) because for starters, I didn't do anything on the previous two questions. I also chose PCA because in the previous assignment, we could see there was some linearly dependent columns (*General Health*, *Mental Health*, *Physical Health*, etc.) so I wanted to hopefully reduce the amount of linearly dependent vectors in the feature space by using PCA. My CNN only has 2 convolutoinal layers, 2 pooling layers, and then it is fed into a forward-feeding network consisting of 3 hidden layers.

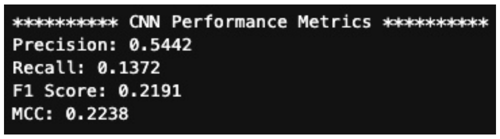
We can see a heatmap of the covariance matrix here:



We can see clusters near the end of our matrix, and clusters in the middle (health features). These clusters made the dataset ideal for PCA. After reduction, implementing, training, and testing, the model performed exceptional on both the training and test data, getting accuracy scores of **87%**, a precision score of **0.54**, recall score of **0.14**, an F1-score of **0.22**, and MCC score **0.22**.

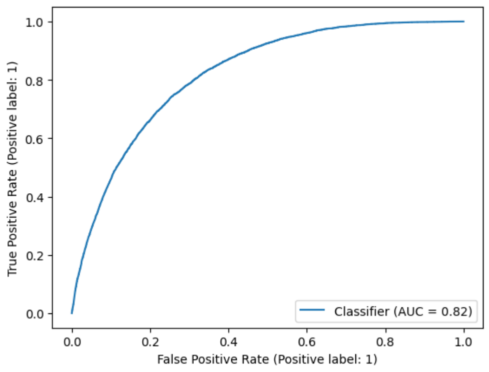


```
Test Accuracy: 86.58%
```



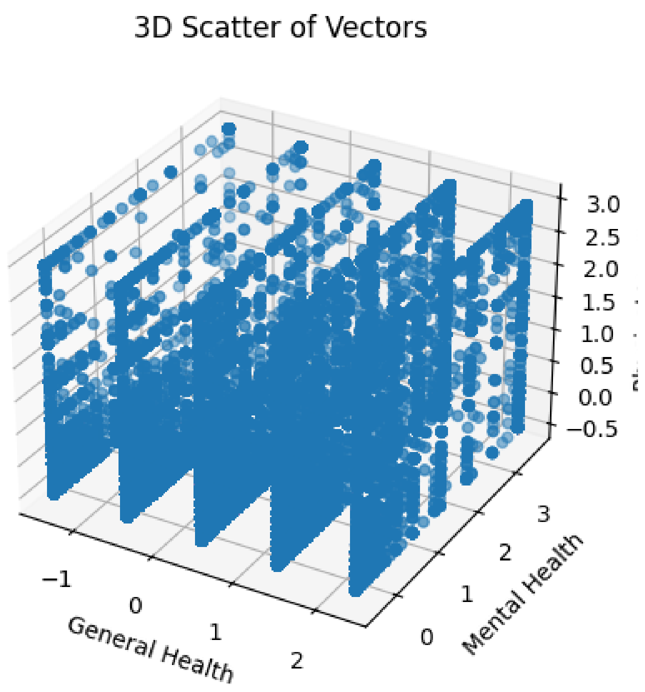
```
********** CNN Performance Metrics **********
Precision: 0.5442
Recall: 0.1372
F1 Score: 0.2191
MCC: 0.2238
```

What the metrics tell about the model is that the model was still sensitive to data, indicating linear-dimensionality reduction might have been redundant (which should have been a gimme based on my observations from the prior questions). However, the AUROC of **0.82** gives a good sign that the model performs nicely - indicating a strong ability to distinguish between positive and negative classes in our binary classification problem (diabetes classification).



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?

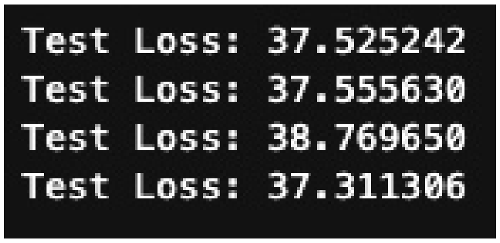
Answer: For this question - again, I computed PCA, but instead of on the entire matrix, I took a subset of vectors within the feature space, and computed PCA on those vectors (again, being the health features). The feed forward network I designed for this question, instead of choosing the activation function, I chose to be able to define the number of hidden layers in the network.



We can see in this scatter how the data is essentially skewed over each other, so though the vectors aren't exactly collinear, the relationship exists in which the vectors are parallel *almost* indicate a linear relationship - which could be enough to perform PCA on these components.

Since we were given the number of hidden layers, there was not much real design here. The model implemented consists of 1 hidden layer, with num_features ·2 dense nodes, and an output layer to one neuron, consisting of a number - given this was a regression task. However, I used a few more models just to experiment with how training the same model, but varying certain hyperparameters would affect model performance. I used cosine similarity to help deduce that the three features are indeed somewhat linearly dependent, instead of using the correlation coefficient between them. I wanted to try an out of the box solution for this part of the assignment and see how the model performed after that.

Upon reduction among a few labels, training, and testing on the dataset, one thing I did have to modify was not using diabetes as an input feature. Looking at the RMSE loss of the model, we can see the loss



```
Test Loss: 37.525242
Test Loss: 37.555630
Test Loss: 38.769650
Test Loss: 37.311306
```

is relatively high, with an average loss around **37.79**.

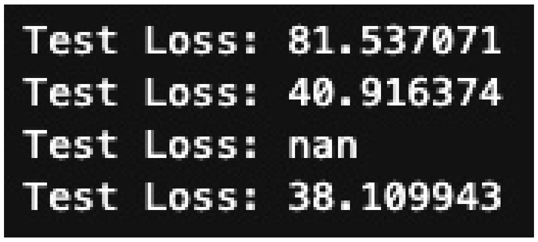
Though the RMSE seems high, given this was a regression task as opposed to a classification, the RMSE is sufficient in predicting BMI. While an RMSE of **0** is nearly impossible, the offset of error acts as a soft-margin in generalizing toward an even more diverse dataset, or even a real-world scenario. Based on the model's performance, the most telling features of regression tasks for neural networks and their effects on RMSE score are most activation functions, dense layers, and the number of neurons in a hidden layer.

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?

Answer: A neural network I *wanted* to build for this, but didn't see fit, had no idea how to, and wasn't taught in lecture yet was a transformer. I've read a paper on Resnet34 and I think using that architecture would have been cool to predict diabetes based on medical images, but that would require more domain knowledge than I have. For this model, I used a random activation function set for each hidden layer at instantiation and a regular forward feeding neural network with 2 and 3 hidden layers to predict BMI.

I randomized the activation functions because in question 2, we can see how with binary classification, a sigmoidal activation function tends to work slightly better in class prediction. Another thing I wanted to see try was to see how "deep" a network could be before adding more hidden layers for predicting binary labels was considered enough (or too much). In terms of network design, I figured that using ReLU instead of Sigmoid would cater better toward generalizing - considering the number of hidden layers would be changed. For my dense layer neurons, I, again, decided to make it num_features $\cdot 2$. There was no real thought behind why, just considering a graphic of a neural network. The networks that were trained and tested on this data set include a randomized activation function network with 2 and 3 hidden layers, and a forward feeding neural network with 2 and 3 hidden layers.

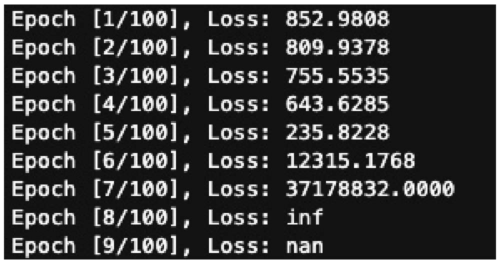
Looking at the RMSE loss,



```
Test Loss: 81.537071
Test Loss: 40.916374
Test Loss: nan
Test Loss: 38.109943
```

the networks with 2 hidden layers are the first 2 RMSEs reported and the networks with 3 hidden layers are the latter half of the metrics screenshot. (Feed Forward with 2 hidden layers, Randomized Activation with 2 hidden layers, Feed Forward with 3 hidden layers, Randomized Activation with 3 hidden layers.) To start, we can see that the FNN with 3 hidden layers failed to even converge during testing.

Again, we can see that the FNN with 3 hidden layers failed to converge during testing. Looking at the training session for the model, we can see the loss catapults to ∞ after just 7 epochs.



```
Epoch [1/100], Loss: 852.9808
Epoch [2/100], Loss: 809.9378
Epoch [3/100], Loss: 755.5535
Epoch [4/100], Loss: 643.6285
Epoch [5/100], Loss: 235.8228
Epoch [6/100], Loss: 12315.1768
Epoch [7/100], Loss: 37178832.0000
Epoch [8/100], Loss: inf
Epoch [9/100], Loss: nan
```

There are a few things this could be, but one thing that comes to mind is that the learning rate wasn't optimal. The learning rates did not change throughout the model implementation, but with the relatively high RMSE from question 4 as well, it puts into question what the optimal learning rate should be for regression tasks, considering how well the CNN performed on classification task with the same learning rate.

EC1. Are there any predictors/features that have effectively no impact on the accuracy of these models? If so, please list them and comment briefly on your findings.

Answer:

EC2. 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?

Answer: Pros, there are many. For starters, if we have a really intense dataset, we would be able to capture many nonlinear patterns and structures in the dataset. The CNN from question 3 did that really nicely, though it was overkill. One thing that does scare me is if I overfit the model to the training data with the CNN. Alas - beside the point. Another pro of using neural networks is that you can essentially validate what the margin-classifiers + logistic regression were able to predict. This is good if you want to just double-check or evaluate other models performance on the same dataset.

One con of using neural networks on the same dataset as our margin classifiers was that the neural networks do take much longer to train than trees or SVM. For datasets not too intense, neural networks can feel like overkill, training on this dataset. However, for much much more complex tasks (reinforcement learning, image classification), neural networks can definitely capture any spatial data within the a much more complex dataset (thinking about how tensors in PyTorch are made to be tuples with 3 indices, representing a tensor from linear algebra, but we essentially just projected the dataset into a higher dimension from this assignment - specifically for question 3).

In essence, not necessary, but neural nets are really good at finding complex data patterns that otherwise a linear model would not find. However, for this specific dataset, using Boosted Trees seemed to work just as well, if not better, than any of the neural networks implemented in this assignment, while training and evaluating at a fraction of the time the neural networks did,