FML Homework 4 - Erin Choi

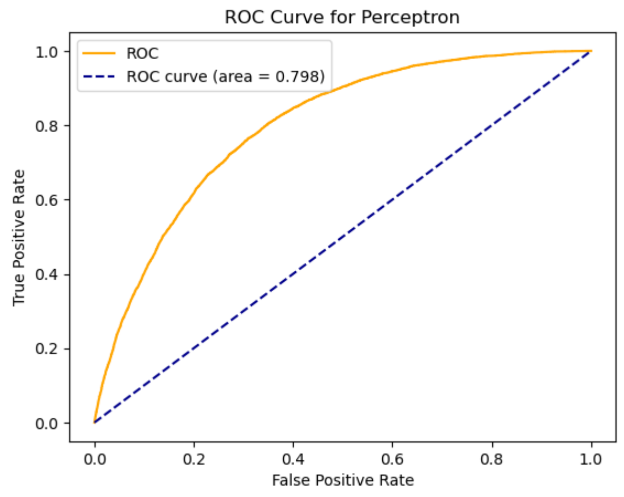
Data Cleaning and Preprocessing

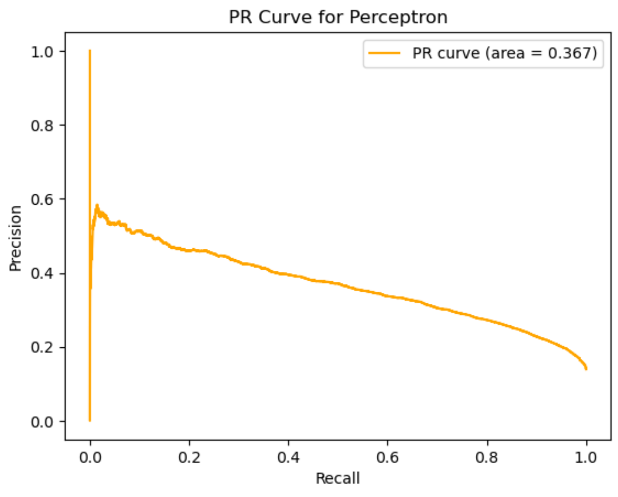
I set a seed of 42 in random, numpy, and PyTorch to help make my results reproducible. I loaded the data and changed the binary values of BiologicalSex from 1 and 2 to 0 and 1 to make it consistent with the other binary variables. I dropped the Zodiac feature as I thought it would have a negligible effect on classification or regression. I then separated the features from the target and split the data using StratifiedShuffleSplit to maintain the same class imbalance from the overall dataset in both the training and test set. I scaled some predictors after this split to prevent data leakage between the training and test sets. I fit and transformed the continuous and ordinal training features using MinMaxScaler to get their ranges similar to that of the binary variables (0 to 1), then transformed the same features in the test set with the same scaler. Finally, to make the data suitable for PyTorch, I turned each part of the split data into a tensor.

Question 1

I built a Perceptron using scikit-learn, with the class_weight parameter set to balanced because of the diabetes class imbalance. I started with 1000 as the maximum iteration value and tested other hyperparameters; changing the penalty and alpha value did not improve the AUC, so I returned to default settings. I decreased the max_iter value to 100, and the results did not change. Additionally, I tried changing my seed to different values, but most seeds yielded worse results for all metrics, so I continued to use the seed of 42 for all other questions after observing these results. After training the final Perceptron on the training set, I made predictions on the test set to compute the AUROC, AUPRC (average precision score), accuracy, F1 score, and confusion matrix - the same performance metrics I calculated for all classical methods in Homework 3.

The Perceptron resulted in an **AUROC of 0.798,** AUPRC of 0.367, accuracy of 0.761, and F1 score of 0.433. The AUC scores are improved from the original results for the default Perceptron model: AUROC of 0.775 and AUPRC of 0.341. Accuracy decreased (originally 0.861) as a result of optimizing AUC for this problem, but the F1 score increased greatly from 0.035. The plots for the ROC curve and PR curve are below.



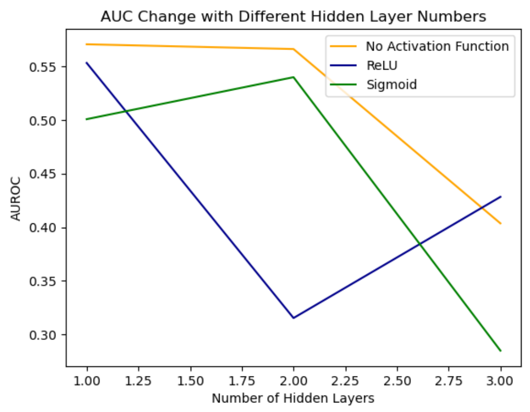


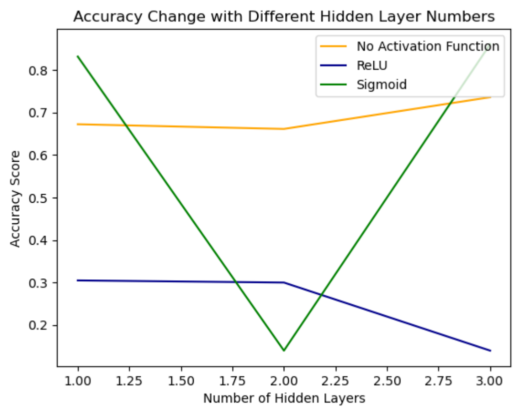
The Perceptron's overall AUC of 0.798 could be better, but it is fair and indicates that the model does a decent job at separating the classes. The accuracy of 0.761 is also an acceptable value. The precision and recall are not the best, however, as indicated by the lower AUPRC and F1 score. This comes from the high false positive rate of 0.2212, derived from the confusion matrix, which shows that the model classifies many units without diabetes as having diabetes. Additionally, the false negative rate is even higher (0.3453), and it is a problem for a diabetes classifier to have a higher false negative rate than false positive rate, as the model is not catching those who actually have diabetes at a higher rate than misclassifying those who don't actually have diabetes. Thus, while this model results in decent AUC and accuracy values, it may not be very good for tackling the classification problem at hand due to the high false classification rates.

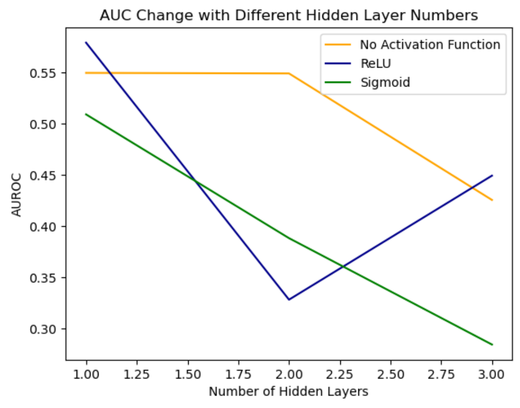
```
Confusion matrix:
 [[34005  9662]
 [ 2441  4628]]
```

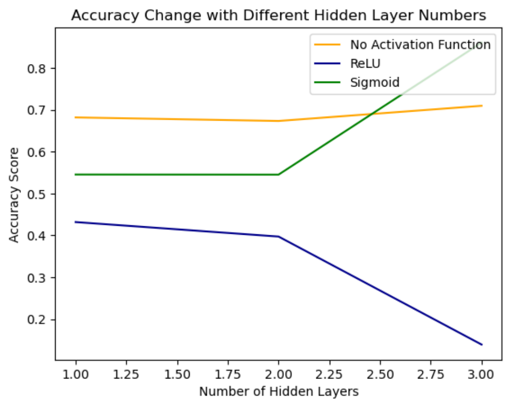
Question 2

I first defined some parameters: a learning rate of 0.001, an input size of 20 (for 20 predictors), 100 units for each hidden layer, and an output size of 2 (for 2 possible classes). I computed class weights to pass to the loss function later on. I then defined 9 different fully-connected neural networks, with all combinations of 1, 2, or 3 hidden layers and one of 3 activation functions: no activation function, ReLU, and Sigmoid. In a training function, I set the criterion (loss function) as cross-entropy loss for the classification task and used stochastic gradient descent as the optimizer. I trained each model for 100 epochs to start, calculating accuracy and AUC for each epoch, then used a test function to save the final test AUROC, AUPRC, accuracy, and F1 score. I varied the number of epochs between 500, 200, and 100 to search for the best results among all the models. I also tried changing the number of hidden units to 50.









I ended up with two sets of hyperparameter settings I wanted to compare: neural networks trained over 200 epochs with either 100 or 50 hidden units. Plots for changes in AUC and accuracy for each activation function over different numbers of hidden layers (1 to 3) are shown above. The top row corresponds to 100 units per hidden layer, while the bottom row corresponds to 50 hidden units.

The graphs show that for networks with no activation function, AUC decreases and accuracy slightly increases with more hidden layers. When increasing the number of hidden layers and using ReLU, AUC appears to dip then increase again, and accuracy decreases. When using the Sigmoid activation function, AUC generally decreases; with more hidden units (100 versus 50), the AUC increases at 2 hidden layers before decreasing. Accuracy generally increases with the Sigmoid nonlinearity, but with more hidden units, the value decreases greatly at 2 hidden layers before increasing again. **The variance in AUC as the number of hidden layers increases does appear to depend on the activation function used**, as the lines for each activation function in the AUC graphs follow different patterns and do not change in the same ways.

From the 18 neural networks trained and tested for the above graphs, the 50-unit 2-layer network with no activation function yielded the highest AUC of 0.579 and had an accuracy of 0.432. However, I would choose a different network as the best one overall; the 100-unit 1-layer network with no activation function had a similar **AUC of 0.571** and a higher **accuracy of 0.672**. It also yielded an AURPC of 0.215 and an F1 score of 0.401. **The performance of this best overall network was worse than the Perceptron in all metrics** (AUROC of 0.798, AUPRC of 0.367, accuracy of 0.761, F1 score of 0.433), which is not what I expected. Considering the superior performance of the (linear) Perceptron and the observation that neural networks with no activation functions generally outperformed networks with ReLU and Sigmoid nonlinearities, I suspect that the relationship between the predictors and diabetes classes is better modeled by classical methods than neural networks.

Question 3

The work I did for the previous question contributed greatly to my decisions for this question. I deduced from the results of Question 2 that creating a deep network with no activation function would be best for classifying diabetes for this dataset, so I opted not to try more networks using ReLU or Sigmoid activation functions. Instead, I tried to improve my best deep network from Question 2, which had 2 hidden layers with 100 units each and no activation function, trained for 200 epochs. I also trained and tested a network with identical settings but one additional hidden layer, just in case changing other hyperparameters would make it better than the 2-hidden-layer network. Since I already tried several different epochs in the previous question, I varied the hidden size and tried values of 75, 50, and 25 for each network.

The network that yielded the highest AUC among the ones trained and tested for this question was the one with 2 hidden layers, 50 hidden units, and 200 epochs. **It resulted in an AUROC of 0.670, AUPRC of 0.249, accuracy of 0.395, and F1 score of 0.125.** The AUROC improved a lot compared to the 100-unit version tested in Question 2 (with an AUC of 0.579), but this optimization occurred at the cost of a further drop in accuracy (compared to 0.432). Even this improved network has worse performance than the Perceptron, and its metric values are poor overall. The AUC indicates that it does fairly better than random at separating classes, but its low AUPRC, accuracy, and F1 score show that it is misclassifying more units than correctly classifying them. The poor results of hyperparameter tuning for this deeper linear network were surprising, but this supports my idea that the relationship between the predictors and diabetes classes may be best modeled by simpler linear methods.

Additionally, **there is no benefit of using a more complex CNN or RNN** for this classification because there is no spatial or temporal/sequential dimension to this dataset. There is no need to complicate the models beyond feedforward neural networks, as they make it easy to fulfill the main requirement of this task: learn a mapping from the input features to the output.

Question 4

For this new regression problem, I had to preprocess the data again. I separated the new target, BMI, from the other features and split the data into training and test sets. I could use train_test_split this time since there was no longer a class imbalance to address. Again, I scaled the continuous and ordinal predictors after the split to prevent data leakage. I did not scale BMI so that I could compare the predicted values directly to the true target values. Finally, I turned each part of the data into a tensor to use with PyTorch.

I defined 3 feedforward networks, each with one hidden layer and 100 hidden units, that used one of 3 activation functions: no activation function, ReLU, and Sigmoid. The output number was changed to 1 because the problem changed from binary classification to prediction. Similar to the process in Question 2, I used training and test functions. I set the loss function to MSELoss for the regression task and again used stochastic gradient descent as the optimizer. I initially trained each model for 200 epochs and computed the loss (training MSE) for each epoch. My test function was applied after training each network to calculate and save the test RMSE. I increased the number of epochs to 300 to slightly improve the performance of the models.

The results of training over 200 epochs were RMSEs of 7.148 for no activation function, 7.028 for ReLU, and 6.521 for Sigmoid. After training over 300 epochs, these values were reduced, yielding RMSEs of **6.947 for no activation function, 6.859 for ReLU, and 6.474 for Sigmoid.** There is a pattern in the results, with the lowest RMSE coming from the Sigmoid network and the greatest coming from the network with no activation function. Thus **RMSE appears to depend on the activation function, and the network that uses the Sigmoid activation function fits the dataset to predict BMI most accurately.**

Question 5

Since I tested some networks with one hidden layer in the previous question, I focused on testing networks with two hidden layers here. I defined 3 models, each with two layers, one of the three activation functions I have been using, a hidden size of 100, and a learning rate of 0.001, trained for 300 epochs.

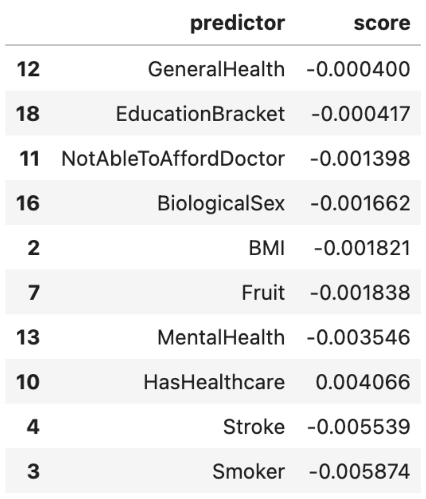
I trained the networks with 3 hidden layers as is, which lowered all RMSEs but, strangely, made the Sigmoid network yield the highest RMSE. I tried training all 3 networks with 200 epochs and hidden sizes 100, 75, 50, and 25; hidden size 50 resulted in the lowest RMSEs. To save time, I chose to explore the ReLU network with a hidden size of 50. I varied the number of epochs between 200, 250, and 300 and changed the learning rate to 0.002 before adding another hidden layer with a ReLU activation, continuously lowering RMSE. Trying some of these settings with the Sigmoid function did not result in any lower RMSEs, so I kept my ReLU models.

My final model had an **RMSE of 6.136**. The network was trained over 300 epochs and had 3 hidden layers, ReLU activation functions, a hidden size of 50, and a learning rate of 0.002. Using 2 or 3 **hidden layers** decreased the RMSE compared to the one-hidden-layer networks tested in Question 4, and as seen both in this question and the previous one, using **different activation functions** also affects the RMSE. While retraining the network with different hyperparameters, I observed that there is a need to balance the **number of epochs**, as the RMSE decreased and then increased again during training when the number of epochs was too high. **Learning rate** also matters; if it is increased too much, training begins with a very unstable loss, with RMSE jumping up and down by hundreds between individual epochs and taking a long time to stabilize before beginning a constant descent. I did not use batches to train my networks, but I believe that would also have an impact. There are many hyperparameters you can vary to affect the final RMSE, so it is a complex process to find the best settings to minimize RMSE for any one network.

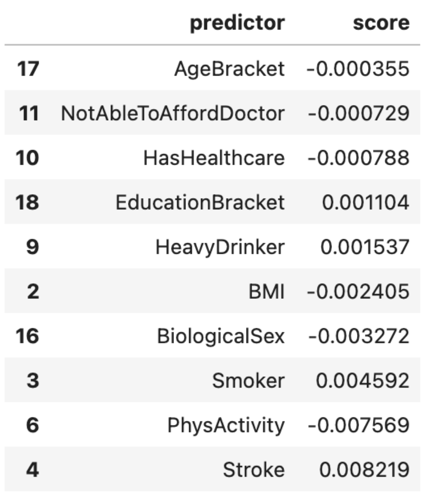
Extra Credit Part A

For my best models from Questions 3 and 5, I performed permutation feature importance. I shuffled the values of each predictor in the test set and saved their scores, or the difference in the test metric and the network's best metric for each shuffled feature, respectively. I then sorted the results by the score.

For the best diabetes classifier network, the scores were sorted from least to greatest absolute value, as the features with scores closest to 0 are the least impactful predictors according to this method. The top 10 features with the least impact on AUC (left) and accuracy (right) are shown below. **GeneralHealth** is the least impactful on AUC, and **AgeBracket** is the least impactful on accuracy. Many features also appear on both lists of the 10 least impactful features: **EducationBracket, NotAbleToAffordDoctor, BiologicalSex, BMI, HasHealthcare, Stroke, and Smoker.**

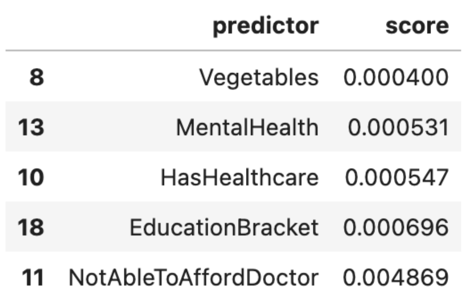


	predictor	score
12	GeneralHealth	-0.000400
18	EducationBracket	-0.000417
11	NotAbleToAffordDoctor	-0.001398
16	BiologicalSex	-0.001662
2	BMI	-0.001821
7	Fruit	-0.001838
13	MentalHealth	-0.003546
10	HasHealthcare	0.004066
4	Stroke	-0.005539
3	Smoker	-0.005874



	predictor	score
17	AgeBracket	-0.000355
11	NotAbleToAffordDoctor	-0.000729
10	HasHealthcare	-0.000788
18	EducationBracket	0.001104
9	HeavyDrinker	0.001537
2	BMI	-0.002405
16	BiologicalSex	-0.003272
3	Smoker	0.004592
6	PhysActivity	-0.007569
4	Stroke	0.008219

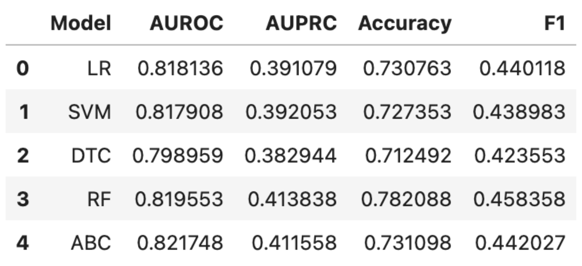
It does not make sense that some of these features do not have an impact on a model that is predicting diabetes. My analysis of classical methods in Homework 3 showed that GeneralHealth and BMI are among the best predictors for classifying diabetes, and these features have connections to diabetes, e.g. being overweight is known to increase your likelihood of having diabetes. Being over a certain age also contributes to the likelihood of developing diabetes, and having diabetes is related to a higher likelihood of having strokes. As mentioned in previous questions, I believe a neural network may not be the best way to model a diabetes classifier, thus features that are intuitively connected to having diabetes are not impacting the model as one would expect.



	predictor	score
8	Vegetables	0.000400
13	MentalHealth	0.000531
10	HasHealthcare	0.000547
18	EducationBracket	0.000696
11	NotAbleToAffordDoctor	0.004869

For the best BMI prediction network, the scores were sorted from least to greatest, as features that don't cause a lot of change in the RMSE when shuffled are the least impactful on the model. The 5 features with the smallest scores are shown on the left. I would expect that the Vegetable feature (eating vegetables daily or not) contributes directly to predicting BMI, but it makes more sense that the other features don't have as big of an impact, as least not directly. Neural networks may be a decent way to model the relationship between the features and BMI, or at least better than using them to predict diabetes.

Extra Credit Part B



	Model	AUROC	AUPRC	Accuracy	F1
0	LR	0.818136	0.391079	0.730763	0.440118
1	SVM	0.817908	0.392053	0.727353	0.438983
2	DTC	0.798959	0.382944	0.712492	0.423553
3	RF	0.819553	0.413838	0.782088	0.458358
4	ABC	0.821748	0.411558	0.731098	0.442027

Neural networks are great for learning and modeling complex relationships, but there are also times when simpler methods are a better choice. As discussed several times throughout this report, the performance of the neural networks that predict diabetes is consistently worse than the (linear) Perceptron, even after hyperparameter tuning, as well as the classical methods explored in the prior homework (results shown in the table). It also seemed that the networks with no activation functions performed better at the classification task than when using activation functions, so the simpler models could be better for classifying diabetes.

This dataset is somewhat large, but not very large, as different seeds and network initializations result in large performance differences. Classical methods are better suited for smaller datasets since it takes a lot of data to train all of the parameters of a neural network well. Neural networks may not be the best choice if there is not enough data to train them with.

Neural networks also involve many hyperparameters that may be tuned to optimize their performance. These networks already take so much time to train individually, so retraining or validating them with different parameters takes much more time and effort over simpler models. It took significantly more time to complete this assignment than the previous one, which used the same data but simpler models that yielded better performance.

While I cannot compare the performance for the regression task (predicting BMI) with any simpler method, using neural networks to perform regression using this dataset worked fairly, with the networks resulting in decent RMSEs. This contrasts with the classification networks and shows that, depending on the problem you are trying to solve, the same data can be used in very different models for the best results. More complex algorithms are capable of representing complicated relations but are not always the best choice.