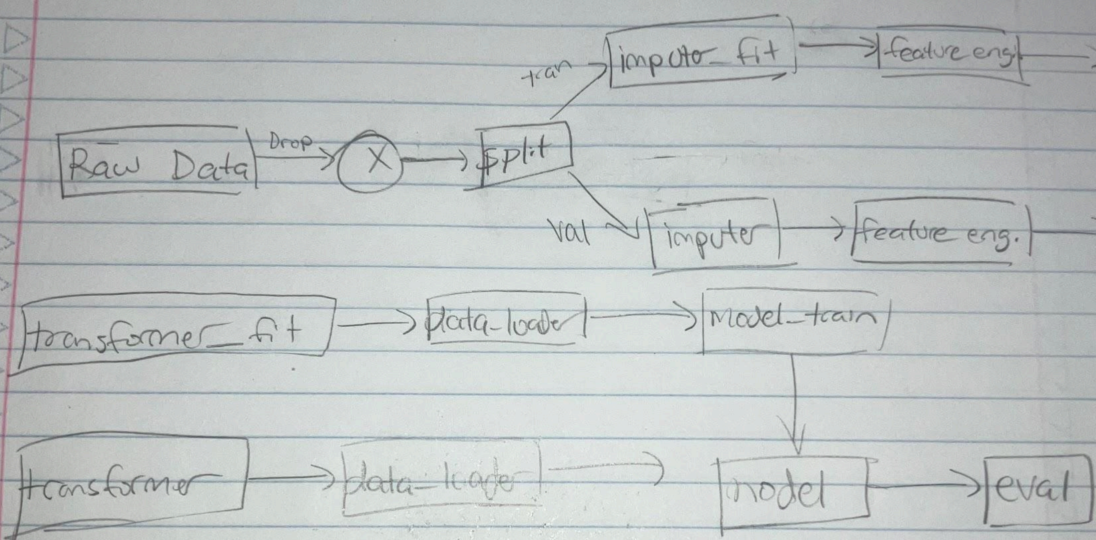
Logan Caraway
Dr. Santos-Villalobos
COSC 325
November 10, 2025

COSC 325 - Homework #3

Task #1:

Preprocessing Steps:

My preprocessing steps were fairly similar to previous implementations. However, I did some actual feature engineering this time by adding a new feature (family size.) This was based on what I had seen done in various other notebooks. While I did use SibSp and parch beforehand, this combines these features into a better representation than would be seen in the two separate variables. For example, a person might have a family size of 5: if it were themselves, two children, a spouse and a sibling, this would be equivalent to someone having three children and a spouse on board, but would cause the model to treat them as separate. I also ensured to split before preprocessing (imputing) to not cause any bias. The imputing and feature engineering occurred outside the class and then I created a ColumnTransformer to scale and one-hot-encode, which was done in the class.


Raw Data
Drop
X
split
train
imputer_fit
feature eng
val
imputer
feature eng.
transformer_fit
data_loader
model_train
transformer
data_loader
model
eval

Resources:

https://dantokeefe.medium.com/effective-data-handling-with-custom-pytorch-dataset-classes-b141bcb87b41
https://docs.pytorch.org/tutorials/beginner/basics/data_tutorial.html

Many of the resources for PyTorch seem to be about computer vision, which is why a lot of them run transformations in __getitem__ (presumably for random cropping and other things along those lines.) It took a little while to set it up how I did because of how simple it was in comparison to many other implementations. However, they did give a good starting point in setting up the Dataset class.

Task #2:

Resources Used:
https://machinelearningmastery.com/building-multilayer-perceptron-models-in-pytorch/
https://discuss.pytorch.org/t/how-to-create-mlp-model-with-arbitrary-number-of-hidden-layers/13124
https://codefinity.com/courses/v2/0033b288-2d72-4a94-be97-b5dde061eb4b/e1db76a4-3121-4f18-8269-27a02a912f6f/22912028-1034-413a-a7c6-9a78956849e3

Dr. Santos' Colab Code

Of the above two, the Colab code was the most useful. I took the training and evaluation functions directly from this, as well as the plotting code. The MLP process I used of creating the layers and then unpacking them into a nn.Sequential was a combination of the techniques discussed in the above two links. The codefinity link was for help with more feature engineering.

Hyperparameter Search:
I trained 36 models using the following hyperparameters:

```
depths = [2, 3]
hidden_starts = [input_size*2, input_size*3]
lrs = [0.01, 0.001, 0.0001]
l2_reg_weights = [0.05, 0.005, 0.0005]
```

The hidden start value was used to ensure that the depth and hidden_size matched up. Below is a snippet of the output:

 # epochs = 10:

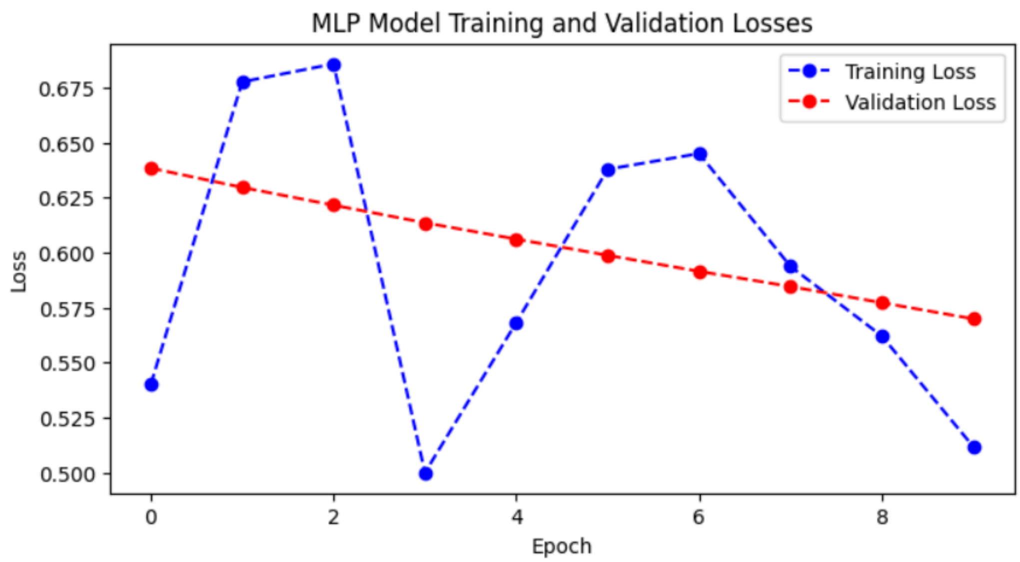
	depth	hidden_sizes	lr	weight_decay	final_val_acc	final_val_loss
0	2	[28, 14]	0.0100	0.0500	75.977654	0.552865
1	2	[28, 14]	0.0100	0.0050	73.743017	0.554302
2	2	[28, 14]	0.0100	0.0005	78.770950	0.562703
3	2	[28, 14]	0.0010	0.0500	61.452514	0.665940
4	2	[28, 14]	0.0010	0.0050	62.569832	0.690268
5	2	[28, 14]	0.0010	0.0005	35.754190	0.760273
6	2	[28, 14]	0.0001	0.0500	40.782123	0.813257
7	2	[28, 14]	0.0001	0.0050	60.893855	0.682308
8	2	[28, 14]	0.0001	0.0005	46.927374	0.711115
9	2	[42, 21]	0.0100	0.0500	76.536313	0.580114
10	2	[42, 21]	0.0100	0.0050	67.039106	0.594391
11	2	[42, 21]	0.0100	0.0005	67.039106	0.577591
12	2	[42, 21]	0.0010	0.0500	60.335196	0.684665
13	2	[42, 21]	0.0010	0.0050	54.189944	0.695355
14	2	[42, 21]	0.0010	0.0005	61.452514	0.683933
15	2	[42, 21]	0.0001	0.0500	46.368715	0.695173
16	2	[42, 21]	0.0001	0.0050	31.284916	0.779638
17	2	[42, 21]	0.0001	0.0005	67.597765	0.645400
18	3	[28, 14, 7]	0.0100	0.0500	62.011173	0.630680
19	3	[28, 14, 7]	0.0100	0.0050	81.564246	0.497495
20	3	[28, 14, 7]	0.0100	0.0005	69.273743	0.633514
21	3	[28, 14, 7]	0.0010	0.0500	62.569832	0.665124
22	3	[28, 14, 7]	0.0010	0.0050	62.011173	0.645433

The best performing model was:

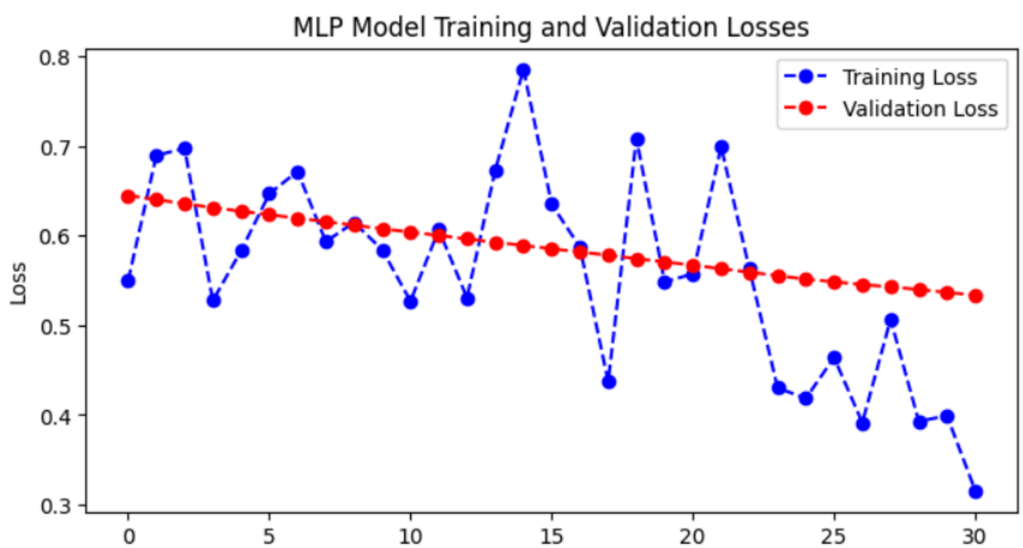
```
num_epochs = 10
depth = 3
hidden_sizes = [28, 14, 7]
learning rate = 0.01
l2 weight decay = 0.005
```

Which gave a validation accuracy of 81.56%.

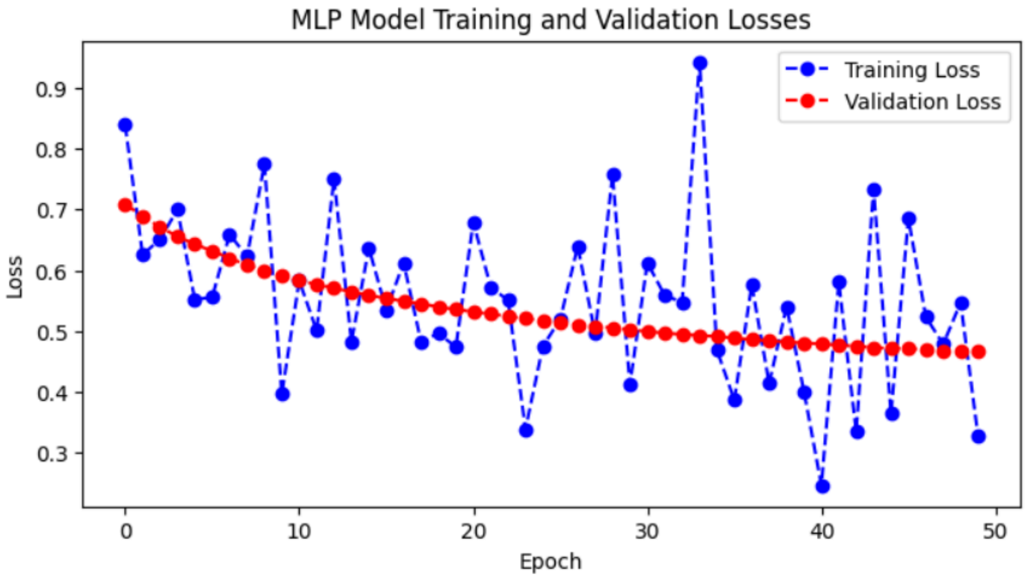
I only plotted this one:



After some more tuning and testing, I came up with this:



I then tested further and chose 50 epochs with a depth of 2 and hidden sizes of [28,14], which gave an validation accuracy of ~79% and the following graph:



Submitting this to the competition only gave a score of 74.440, which was significantly worse than my other implementations.

I did some significant feature engineering (more than previous) because the MLP is likely not able to significantly represent such a simple dataset. I viewed other Kaggle notebooks for inspiration and came up with the following:

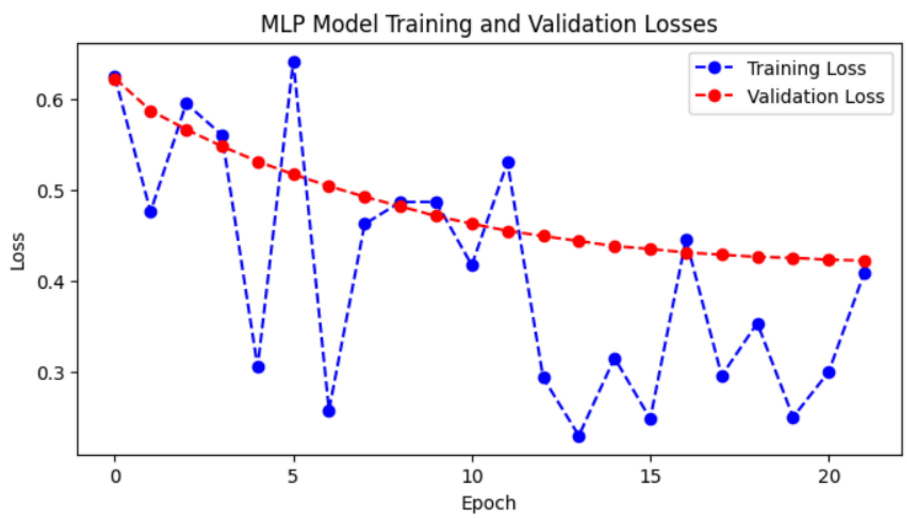
```
df['FamilySize'] = df['SibSp'] + df['Parch'] + 1
df['isAlone'] = (df['FamilySize'] == 1).astype(int)
df['isChild'] = (df['Age'] < 18).astype(int)
df['isMother'] = ((df['Sex'] == 'F') & (df['Parch'] > 0) & (df['Age']
> 18)).astype(int)
df['Age_Pclass'] = (df['Pclass'] * df['Age'])
df['Fare_Pclass'] = (df['Pclass'] * df['Fare'])
```

I then ran another hyperparameter search and got a very familiar result:

	depth	hidden_sizes	lr	weight_decay	final_val_acc	final_val_loss
0	2	[36, 18]	0.0100	0.0500	79.888268	0.501202
1	2	[36, 18]	0.0100	0.0050	75.977654	0.474659
2	2	[36, 18]	0.0100	0.0005	78.770950	0.461651
3	2	[36, 18]	0.0010	0.0500	61.452514	0.606640
4	2	[36, 18]	0.0010	0.0050	56.983240	0.684354
5	2	[36, 18]	0.0010	0.0005	59.217877	0.654430
6	2	[36, 18]	0.0001	0.0500	67.597765	0.634315
7	2	[36, 18]	0.0001	0.0050	60.335196	0.691198
8	2	[36, 18]	0.0001	0.0005	38.547486	0.863271
9	2	[54, 27]	0.0100	0.0500	81.005587	0.488097
10	2	[54, 27]	0.0100	0.0050	81.564246	0.453960

Final Submission:

After some more fine-tuning and hyperparameter searches, I used those insights to get the following:



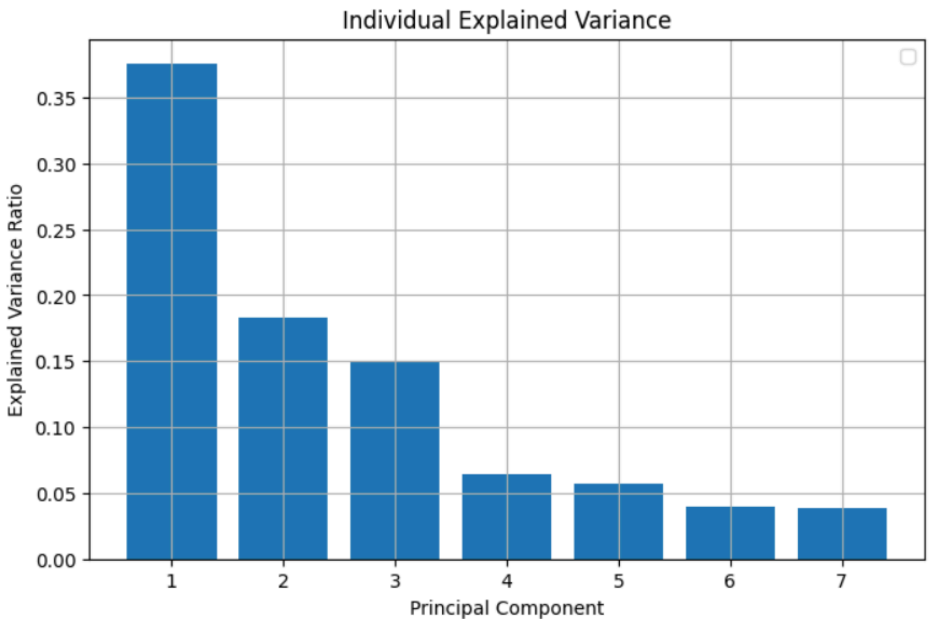

```
25 epochs
[36, 18] <- depth of 2
lr=0.01
wd=0.0005
batch_size = 16
```

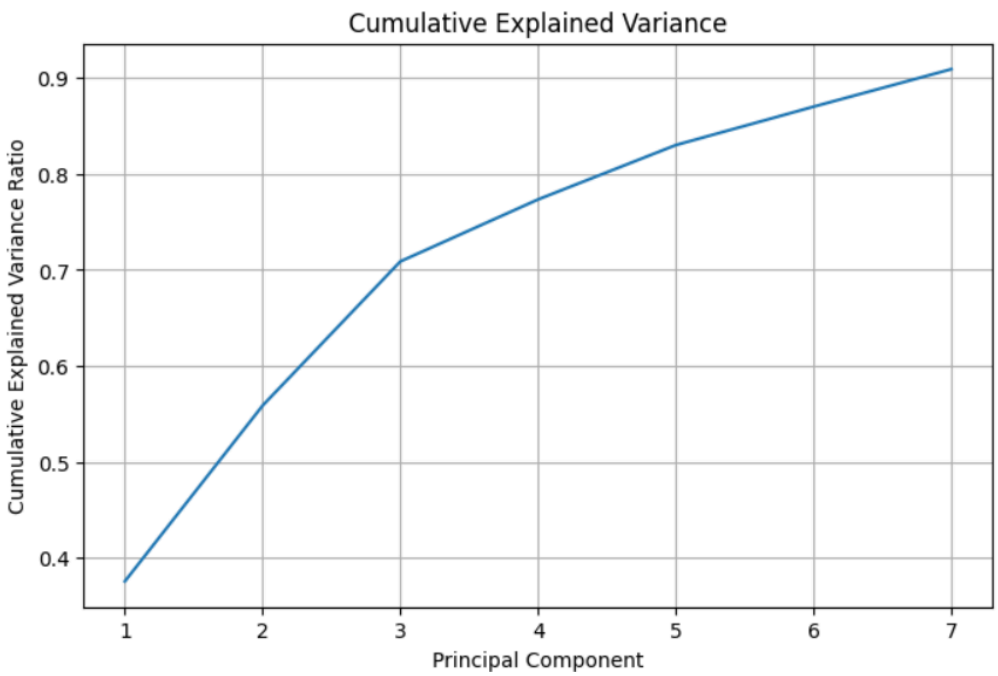
I modified my code to show the max of the validation accuracies (before I was just doing final) which showed me that while training over 50 epochs, the above configuration performed the best. I lowered the batch size to 16 as 32 and 64 seemed to be overfitting more. Submitting this to the competition gave me my highest score yet at 0.78229. That's a fairly significant improvement over my second best of 0.76315. However, were I to go back and perform the feature engineering done here, it would likely surpass my MLP implementation.

Task #3:

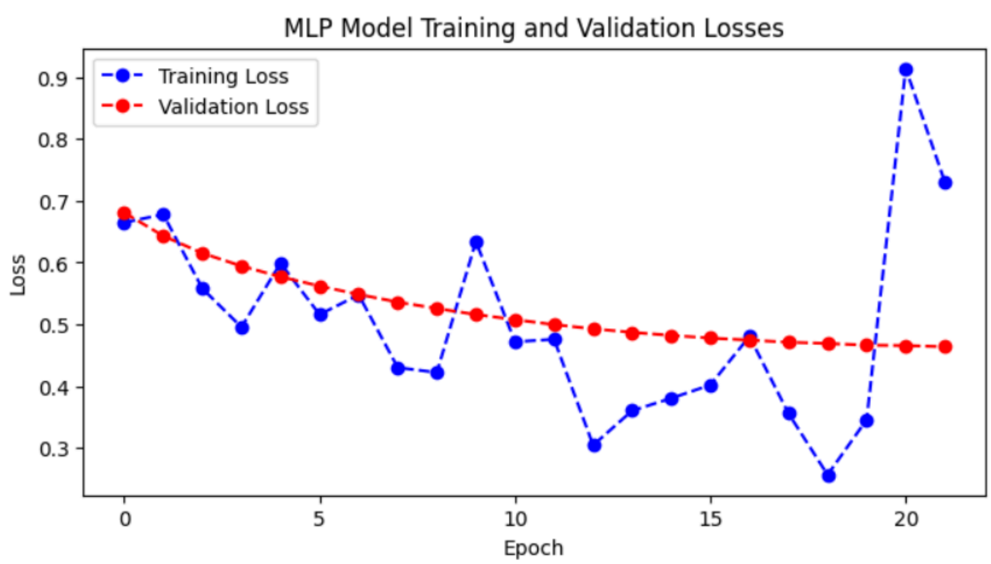
Disclaimer: For Logistic Regression and Random Forest, I performed the same feature engineering I did for MLP for a fair comparison. Thus, the feature dimensions and EVR graphs are the same for each.

Original Feature Dimension: 18
PCA Feature Dimension: 7



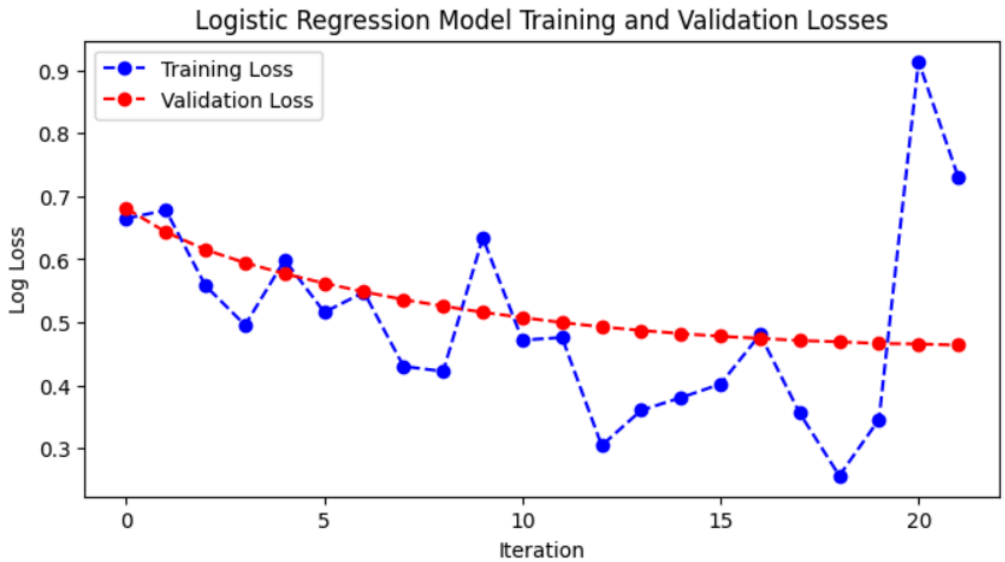


MLP:



Submission Score: **0.74162**

Logistic Regression:



Submission Score: **0.77262**

Random Forest:

Training Accuracy: **0.948**
Validation Accuracy: **0.793**

Submission Score: **0.74880**

Takeaways:

Logistic Regression improved significantly, whereas Random Forest and the MLP models degraded significantly. This makes sense because logistic regression is linear whereas the other two are non-linear. PCA reduces collinearity which helped the logistic regression model, but it also probably removed some non-linear relationships that the other two models could have picked up on. The MLP model also benefits from a larger feature space (which is why I even upped my feature engineering to begin with,) so PCA just reversed that.

Task #5:

The hardest task was task 2 by and large. The biggest thing was trying to figure out how to use PyTorch and set up an MLP model. As I tested more and more, I continually changed my dataset with feature engineering and other techniques and had to search for hyperparameters each time. All in all, it took me about 7 hours.