CSE 151B Project Final Report

Myo Zaw Win
mwin@ucsd.edu

Pratyush Sahu
psahu@ucsd.edu

George Chong
gchong@ucsd.edu

Niha Malhotra
n1malhot@ucsd.edu

Our models and data processing can be found at: https://github.com/mwin02/kaggle_competition_sp2023

1 Task Description and Background

1.1 Problem A

The task of this project is to be able to predict the travel time of a taxi in Porto, Portugal given information about the taxi call including the time of the call, the type of call, the taxi stand if applicable and other such data. This is an important task because knowing the travel time of a taxi trip would allow the taxi dispatch business to run more smoothly and more efficiently, and it would allow taxi drivers to better understand how long a trip could potentially take.

In terms of customers, being able to have an estimate of trip times would also be an excellent feature. Apps like Uber and Lyft offer an estimated arrival time that allows users to understand how long it will take the rider to get to you and the estimated time that you will arrive to the destination by which is very useful for planning trips.

1.2 Problem B

In mathematical terms, we have a regression problem where we want to predict a real numerical time in seconds given an input. The data regarding taxi we have available are a unique id of the taxi ride, the type of the taxi call (a dispatch, a call from a taxi stand, or a street call), an origin_call identification given for dispatch called taxis, an origin_stand identification given for taxi stand taxis, the id of the taxi, and the timestamp. We want a model that given these inputs can predict the number of seconds the trip is likely to take. Using a one-hot encoding description, the model looked something like
For input
$$\begin{bmatrix} 0 \\ 1 \\ \vdots \\ 1 \\ 0 \\ 0 \end{bmatrix}$$
For output
x, where x represents the predicted time for the particular trip.

In terms of an optimization problem, we want to minimize RMSE which is the root mean square error between the travel time that our model predicts and the actual travel time across all of the trips and we do this by changing the weights in the model. The model is run against all the samples in the training data set and the predicted output is compared to the actual output to calculate

Preprint. Under review.

the loss. The model then "learns" by taking the gradient of the loss with respect to the weights and changing the weights in order to try to find the weights that produce the minimum loss.

It might be hard to expand this model beyond the Taxis in Porto Portugal because alot of the input parameters are closely tied to the taxi infrastructure in Porto but our model can give us other problems related to Porto's Taxi Business. Given more location data about the trips, we could potentially do more than just estimate the trip time. This model which takes in as input the parameters for a trip could be used to calculate other outputs like the potential destination of a trip which can help taxi drivers choose their trips accordingly and helps dispatch plan trips efficiently or it could be used to estimate the potential cost of a trip based on the time and distance travelled which would help in the economics side of the taxi business, given that those data are available.

2 Exploratory Data Analysis

2.1 Problem A

The training data set consists of 1710670 entries each with 9 columns. Out of the 1710670 entries, 10 of the entries are missing their 'POLYLINE' column which is used to derive the ground truth for the dataset. The 'POLYLINE' column is the geographical coordinates of the trip taken in 15 second intervals and they describe the trip from start to end. Each trip has a unique trip-id value. In terms of call types, 48% of trips are type B, 31% are of type C, and 21% are of type A. The 'ORIGIN_CALL' column only has values for call type A with the value missing for some entries and the 'ORIGIN_STAND' column only has values for call type B with some values missing as well. Each trip also has the id of one of the 448 taxis operating in Porto. The timestamp is given in epoch time and when interpreted, we see that the training data starts from July/1/2013 and ends at June/30/2014.

The test data size consists of only 320 entries and each entry has the same columns as the training data size except for the 'POLYLINE' column. In terms of the time of the trips, all the trips in the test dataset are on one of five possible days, Aug/14/2014, Sep/30/2014, Oct/6/2014, Nov/1/2014, Dec/21/2014.

One data sample looks like :

```
In [9]: df_tr.iloc[21312]
Out[9]: TRIP_ID                                  1373024965620000435
        CALL_TYPE                                                  C
        ORIGIN_CALL                                              NaN
        ORIGIN_STAND                                             NaN
        TAXI_ID                                             20000435
        TIMESTAMP                                         1373024965
        DAY_TYPE                                                   A
        MISSING_DATA                                           False
        POLYLINE        [[-8.58573,41.148558],[-8.585748,41.148558],[-...
        Name: 21312, dtype: object

In [11]: df_tr['POLYLINE'][21312]
Out[11]: '[[-8.58573,41.148558],[-8.585748,41.148558],[-8.585712,41.148801],[-8.58591,41.148918],[-8.586468,41.148891],[-8.586
         252,41.148234],[-8.587125,41.147361],[-8.585028,41.146263],[-8.583597,41.145183],[-8.581833,41.144463],[-8.57925,41.1
         45471],[-8.580096,41.146218],[-8.579709,41.147757],[-8.578935,41.149701],[-8.580348,41.152032],[-8.582688,41.154588],
         [-8.582796,41.157954],[-8.581212,41.162013],[-8.58231,41.165235],[-8.585667,41.166792],[-8.593695,41.17122],[-8.59811
         4,41.17158],[-8.602137,41.171589],[-8.604207,41.172246],[-8.605071,41.172867],[-8.60526,41.17293],[-8.605593,41.17361
         4],[-8.603667,41.173686],[-8.601003,41.173866],[-8.600391,41.17419],[-8.599833,41.174811],[-8.599806,41.176818],[-8.5
         99446,41.177628],[-8.59941,41.178429],[-8.601354,41.179068],[-8.603001,41.179572],[-8.603487,41.179851],[-8.603424,4
         1.180004],[-8.603163,41.180472],[-8.603586,41.180463],[-8.603712,41.180256],[-8.604207,41.180301],[-8.604189,41.18033
         7],[-8.604162,41.180373],[-8.604162,41.180382],[-8.604198,41.18031],[-8.604198,41.18031],[-8.604198,41.180328],[-8.60
         4207,41.180346],[-8.604198,41.180355],[-8.60427,41.180382],[-8.604531,41.180427]]'
```

2.2 Problem B

Before we did any processing, we first cleaned the dataset and reorganized it a little so that it would be useful for our purposes. We removed the 10 entries that were missing the 'POLYLINE' attribute. We parse the epoch timestamp into Year, Month, Day, Hour, Day of Week columns that would be more useful for learning. We also mapped the 448 Taxi ID values into indexes 0-448 just so we have smaller values to work with.

With the large dataset, we split the data into a training set and a validation set at a 80:20 ratio with random sampling from the original data set. The training data set ended up with 1368528 data points and the testing data set ended up with 342132 data points. We chose to do it this way because the

dataset is very large so the training set and evaluation set will still have enough samples in it that we can meaningful train and evaluate.

We explored different features that we could use for our model. During our data exploration we noticed that the Month, Day and Hour had significant impact on the travel time of a trip so we decided to start off with these features. Since these features were categorical, we experimented with one-hot-encoding as well as sin-cos encoding to represent them and we found that one-hot-encoding worked better so we chose that for our model.

In some of our models, we also attempted to use the taxi stand location data that was provided in metaData_taxistandsID_name_GPSlocation.csv by getting the corresponding longtitude and latitude of Taxi Stands. However, Taxi Stand information was only available for Trips with call type B and even for these trips, taxi stand information could be missing so we had to be careful with this and we chose to normalize the longitude and latitude values.

Another feature that we decided to include was the Taxi ID because there were only 448 taxis, it is very reasonable that different drivers might have different habits and our data showed some patterns. Because the IDs were categorical, we also chose to use one hot encoding so we had a one-hot encoded vector of size 448.

Since there were only three call types, we also included it as a feature. We included the one hot encoded vector of the call type as part of our feature vectors. Another approach that we considered and experimented was to build three separate models based on the call types, since each call type had significant differences in what information was available but we found that while this approach worked better for call type B which had origin stand data available, the model didn't perform well for call type C and A and on average a general model performed better than this ensemble model.

3 Deep Learning Model

3.1 Problem A

We processed the features that we recognized as useful into a flattened vector in order to be compatible with PyTorch's fully connected layer. The input for our latest model is a flattened feature vector of size 570. The feature vector includes the one-hot encoding of month, day of the month, hour, day of the week, call type, taxi id. The output is a singular integer that is the estimated time that the input trip would take.

The final model that we ended up with was a multi-layer linear model. To come up with this model, we just experimented with different model architectures, different amount of layers and different amount of parameters in each layer. We found that just a regression layer was limited in the output that it could produce, especially since our feature vector was very sparse so we started experimenting with different layers. The final model that we ended up with after experimentation and looking at some real world practices for regression models was 3 linear / fully connected layers where the first layer takes the initial input and outputs a vector of size 700, the second layer decreases the vector to 300 and the final layer outputs a singular number which is the predicted time.

The loss function that we used throughout all of our models was Pytorch's MSELoss. Since we are working with a regression model, MSE as the loss function made the most sense and there wasn't any other loss functions that would have worked better.

3.2 Problem B

For this project, we decided to stick with linear regression models because our focus was more on feature exploration and engineering. Therefore we only tried variations of the linear regression model with different layer architecture. The models we tried and experimented with include:

- Simple One Layer Linear Regression Model: This model was just one PyTorch fully connected linear layer that went from input feature vector to output. This model was quite limited in what it could do and we just used it as a base reference while we tried to narrow down and choose the features that we thought would be useful.

- Multi-Layer Perceptron Without Regularization: After settling on the features that we thought would be useful, we started expanding on our model architecture and complexity. We tried different layers

with ReLU activation functions at each hidden layer. We found that this activation function was necessary, otherwise, we would end up with exploding gradients and weights.

- Multi-Layer Perceptron With Dropout: While running the MLP without regularization, we noticed over fitting occurring when trained for long epochs where the training loss would continue to decrease but the validation loss increased. Therefore, we decided to employ some regularization techniques. In our final model, we include a drop out layer with a rate of 0.7. We chose to use dropout because it's a mechanism that we have heavily discussed in class. Furthermore, using BatchNorm, would always lead to the Loss value going to a NaN value, and the reason for that could be attributed to us one-hot encoding the features into a feature vectors, and hence, the number of 0s in the feature vector would lead to Normalization failing.

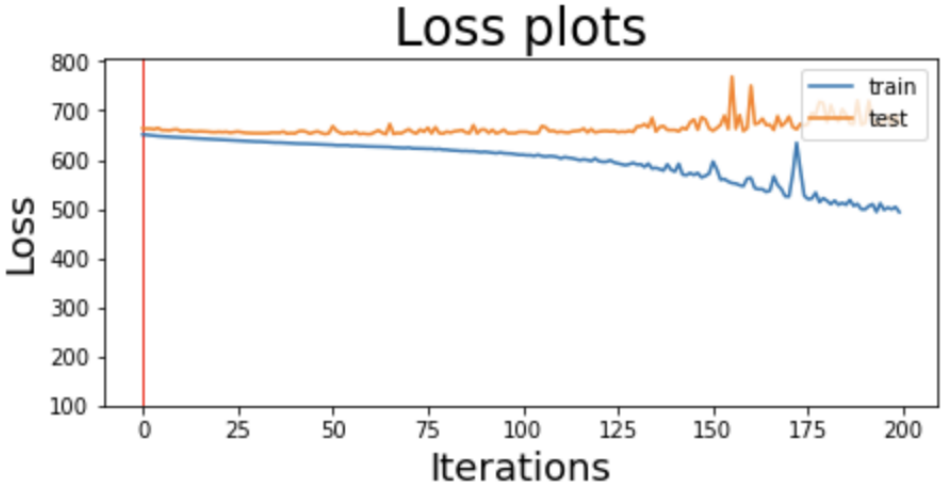
4 Experiment Design and Results

4.1 Problem A

For training/testing, we used the GPU provided on datahub.ucsd.edu. The environment provided has 8 CPU, 16G RAM, 1 GPU and is sufficiently fast enough that it fits the needs of our projects. With the GPU, we found that there was enough computing power to train over the entire 1.6 million dataset.

In terms of optimizer, we tried PyTorch's SGD optimizer as well as several other optimizers such as PyTorch's ADAM optimizer. We found that for our model, the SGD optimizer gave out better results, with lower training loss and lower test loss as well. As for learning rate, learning rate decay, momentum and other hyperparameters, we experimented with different rates, taking note of the validation loss achieved. We isolated each of the components and tried increasing them and decreasing at small rates, running them for 100 epochs due to a limit in time. We chose the best results from this, making decisions based on the acheived training loss, validation loss as well as test loss. After these experiments, our latest model ended up with a learning rate of 0.0001 and no momentum.

In our design, we defined an epoch as one iteration across all the data points in the training set and we processed all the data points in batch sizes batches, updating the gradient after processing every batch size. We found that with trying multiple hyperparameters, training for around 100 epochs would lead to the test loss converging and we found that a batch size of around 5000 was large enough that the weight updates were significant and small enough that the datapoints can be loaded into the GPU for training. One epoch of training took around 12s using the GPU on datahub. For our final model, we trained it for around 50 epochs which took around 10 minutes. We decided to stop around here because when we ran it for more than 100 epoch, we saw a lot of overfitting occuring where the training loss would decrease but the validation loss increased instead. This can be seen in this figure where test loss means validation loss.



4.2 Problem B

Model Description	Training Time	Test Loss	Number of Parameters (input vector)
Simple One-layer Linear Regression Model	10 min (100 epochs)	790.3	532
Multi-Layer Perceptron without Regularization	4 min (100 epochs)	776.4	84, hidden size 20
Multi-Layer Perceptron with One dropout layer	7 min (80 epochs)	700.6	570, hidden size 700 and 300

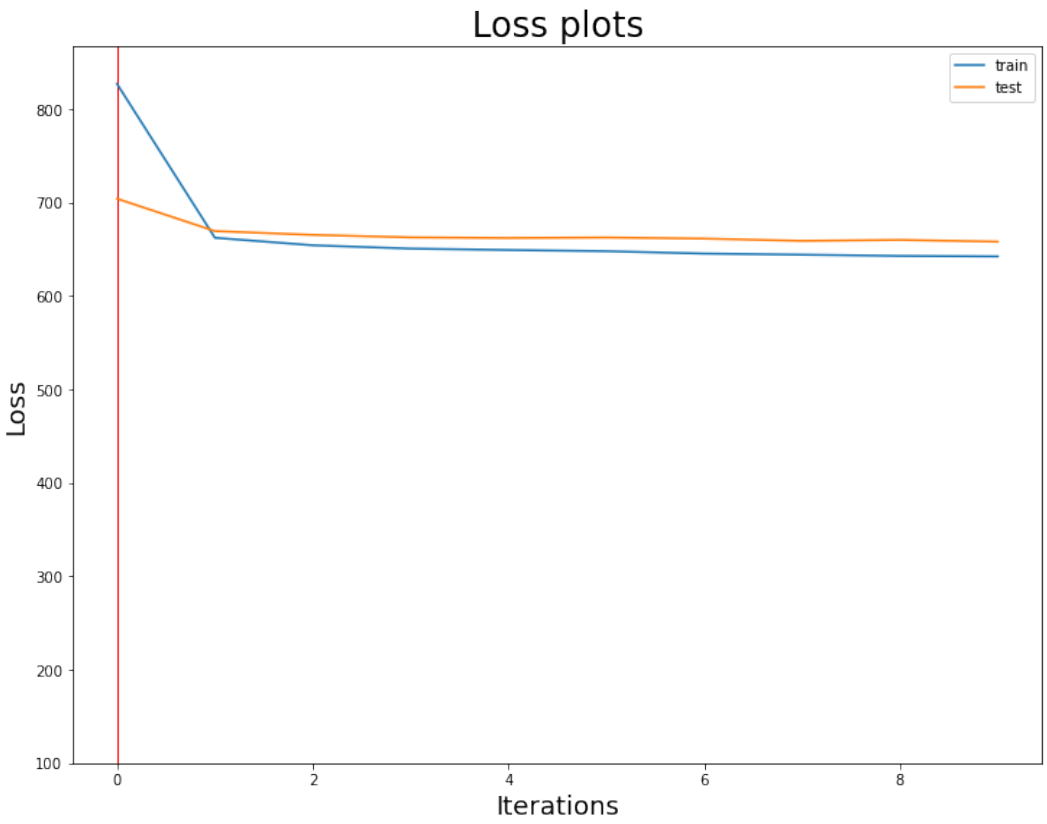
While we stuck to using Multi-layered perceptron for training our model, we did focus on feature engineering, and getting the best possible result from the given features available to us. From the above table, it is clear that

1) Regularization helped in allowing the model to not overfit on the given training data, as seen in the big difference in the test loss from the given best model where our MSE was 700 and test loss MSE of the MLP model without regularization, which had a much larger value of loss.

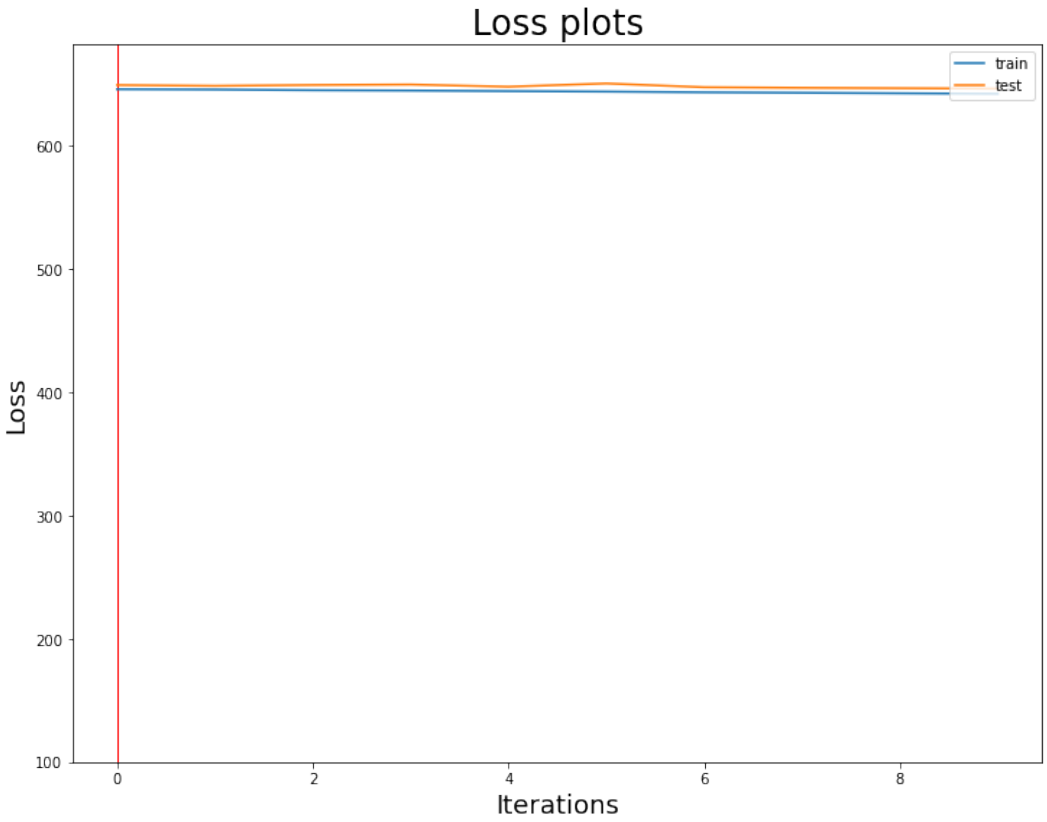
2) Adding more layers through the use of MLP, also allowed for more hidden layers, and through these more hidden layers, the model could learn better, again shown by the difference between our single linear layer model and our MLP model.

Something interesting to note was the time it took to train, and that reflected more of the parameters passed into the model than the actual structure of the model itself. We tried to decrease the learning time, by having larger batch size, of 5000, which seemed to give the best result. Moreover, while we did change the learning rate as well, and tried to increase the learning rate for decreasing the learning time, the model would perform poorly, as the the learning rate, would make the gradient overshoot, the minima, making the loss go up after a certain number of epochs.

4.3 Problem C

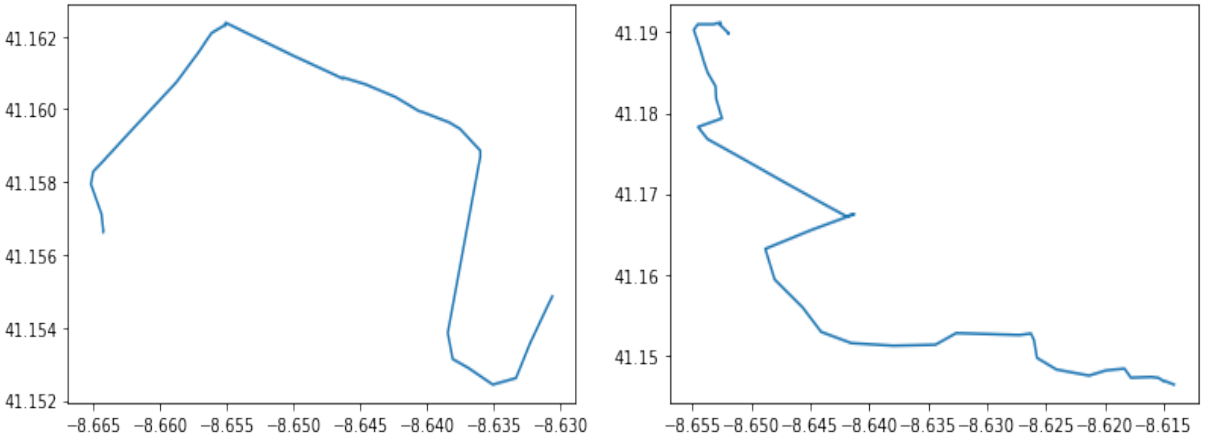


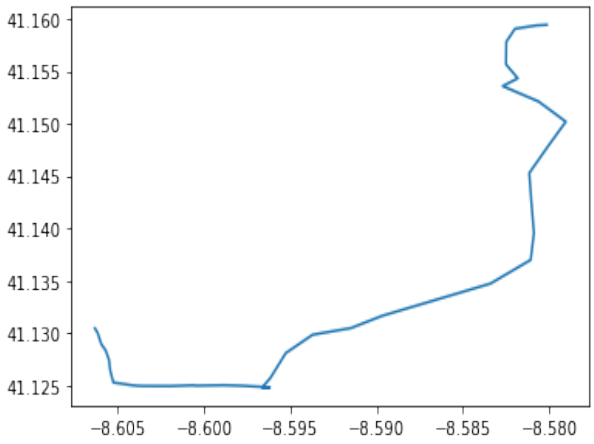
Graph1: Final Loss for the first 10 epochs of running (out of 40)

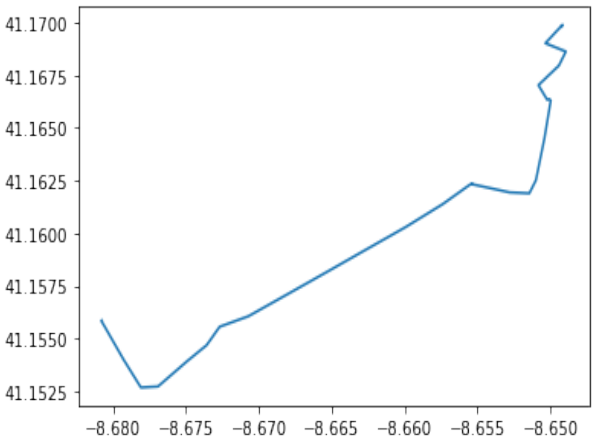


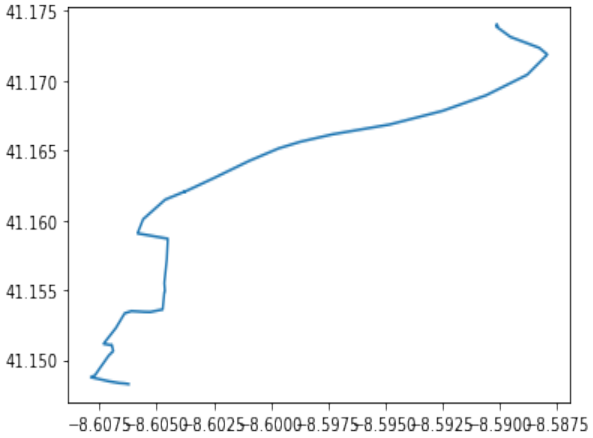
Graph2: Final Loss for the last 10 epochs of running (out of 40)

What we see above is the loss over 10 epochs (iterations). Our model was run over 10 epochs multiple times (4 times), and the first graph shown above is for the first 10 epochs. We see that there is more or a resemblance to an exponential decay, more than the 30-40 epoch losses, due to the larger change in weights initially, in accordance to Stochastic Gradient Descent. Furthermore, as we used batch sizes, this is not a perfect exponential decay over epochs. Considering our training samples after training 40 epochs, these are some of the visualizations of the samples with the highest training loss.









Something else we noticed was that the ground truth of the data points with the highest loss included trips that were smaller than usual, with values like [735, 750, 90, 465, 465, 555, 225, 390].
Currently, our MSE loss for 30% of the test data given (public) is 700.8, which places us in Rank 21.

5 Discussion & Future Work

5.1 Problem A

From this project, we learned a lot about the steps in training a model for real world data. Data exploration was a crucial step that allowed us to choose the features that would be used in our models and from this we saw how messy and useless real world data could be. A lot of fields are missing for some data sets, the values of data can be large ranges, a lot of data are categorical requiring processing and embedding to use in our neural network models. However, choosing the right features to represent and taking steps such as embedding, normalizing are crucial before even choosing what model architecture to use.

Once we settled on the features to use from data visualization, a lot of improvement in our ranking came from different model designs. Since our features consisted of one hot encoded categorical information, our feature vectors were very sparse and we tried different model architectures to deal with this problem. With this we were able to decrease our RMSE loss which was stuck in the 780 range to 700.

The biggest bottleneck was the knowledge of more complicated model architectures combined with the time required to experiment with different hyper parameters and different models. With more complicated models with more layers, the run time of training the model would increase meaning we could only do a little hyper parameter tuning. Combined this with a lack of understanding about good decisions on how to design model architectures, we had to spend a lot of time just trying out different models and seeing the results.

If we had more time and resources, we would explore more complicated models beyond a regression model. Some of the more advanced techniques we came across while doing research on what model architecture to use were techniques like decision tree and random forest regression but due to a lack of understanding on how to properly apply them, we stuck with just linear regression.

References

[1] Rose Yu. (2023). UCSD CSE 151B Class Competition. Kaggle. https://kaggle.com/competitions/ucsd-cse-151b-class-competition

[2] Rose Yu (2023). CSE 151B - Deep Learning. UCSD. https://sites.google.com/view/cse151b

[3] PYTORCH Documentation. PyTorch documentation - PyTorch 2.0 documentation. (n.d.). https://pytorch.org/docs/stable/index.html