Assignment 2

Kavin Ravi

Introduction

For this assignment, I will be using two types of machine learning models to see if I can predict house valuation from the California Housing dataset, which includes 20,640 rows of data. The target column is MedHouseVal, and all predictor columns are continuous values. The first model used is a typical fully connected Neural Network, and the second will be a more basic machine learning model (in this case, a Gradient Boosting Forest model) in order to compare relative performances and see if a Neural Network was necessary for this problem.

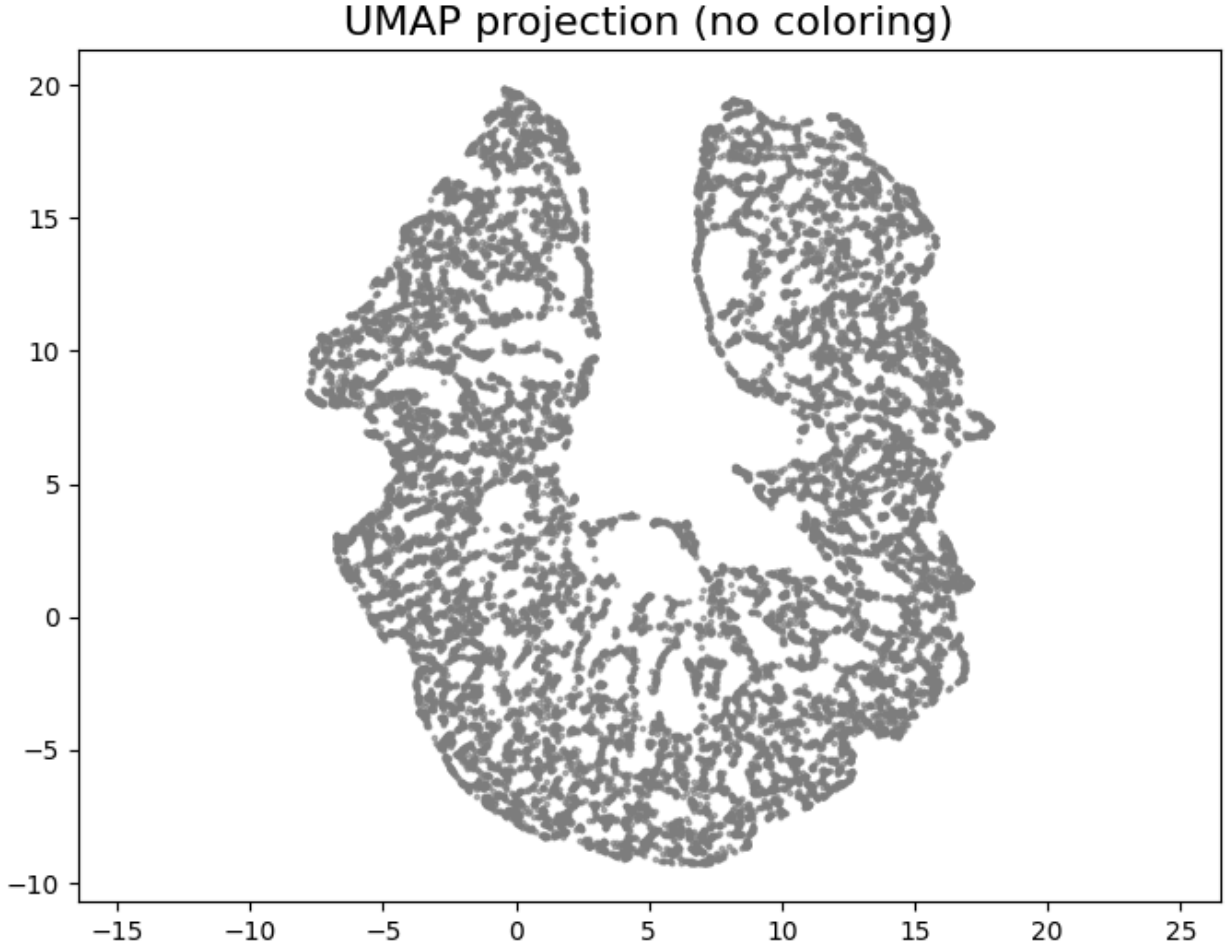
Analysis

The first order of business was to get a feel for the dataset, in order to determine any necessary data preprocessing before proceeding with model construction, was to view the correlation matrix. Highly correlated independent features, also known as multicollinearity, could add implicit bias to the final model and could indicate a need for either removing some kind of regularization (L1 or L2), or a complete data transformation technique, such as Principal Component Analysis (PCA). Correlation matrix shown below:

	index	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
index	1.000000	0.071673	-0.181132	0.042471	-0.011169	0.024880	0.005545	0.081062	-0.113254	0.072086
MedInc	0.071673	1.000000	-0.119034	0.326895	-0.062040	0.004834	0.018766	-0.079809	-0.015176	0.688075
HouseAge	-0.181132	-0.119034	1.000000	-0.153277	-0.077747	-0.296244	0.013191	0.011173	-0.108197	0.105623
AveRooms	0.042471	0.326895	-0.153277	1.000000	0.847621	-0.072213	-0.004852	0.106389	-0.027540	0.151948
AveBedrms	-0.011169	-0.062040	-0.077747	0.847621	1.000000	-0.066197	-0.006181	0.069721	0.013344	-0.046701
Population	0.024880	0.004834	-0.296244	-0.072213	-0.066197	1.000000	0.069863	-0.108785	0.099773	-0.024650
AveOccup	0.005545	0.018766	0.013191	-0.004852	-0.006181	0.069863	1.000000	0.002366	0.002476	-0.023737
Latitude	0.081062	-0.079809	0.011173	0.106389	0.069721	-0.108785	0.002366	1.000000	-0.924664	-0.144160
Longitude	-0.113254	-0.015176	-0.108197	-0.027540	0.013344	0.099773	0.002476	-0.924664	1.000000	-0.045967
MedHouseVal	0.072086	0.688075	0.105623	0.151948	-0.046701	-0.024650	-0.023737	-0.144160	-0.045967	1.000000

Immediately, a few values stuck out to me. First was the correlation of 0.847621 between AveRooms and AveBedrms, next was the correlation of -0.924664 between Latitude and Longitude. To examine them one by one, they both intuitively make sense. A house with more rooms is probably likely to have more bedrooms, and vice versa. The relationship between Latitude and Longitude is a bit more complex. Despite the immediate relationship between the two variables (they're both coordinates), it actually doesn't necessarily stand to reason that they should have such a *strong* inverse correlation. The only possible explanation I can think of is some kind of spatial sampling bias, for example a Northwest-Southeast sample, which would roughly approximate a y = -x path, which would result in a strong negative correlation between the two. Nonetheless, these two both indicate some correlation, meaning that some form of regularization could be beneficial, especially when building the Neural Network.

Next, I attempted to visualize (a projection of) the data, using a technique known as Uniform Manifold Approximation and Projection, also known as UMAP. Of course, it's impossible to visualize data in any dimension greater than 3, or 2 for a static screenshot, which is what makes methods such as PCA, t-SNE, and UMAP so useful in the EDA phase. Since this isn't a classification problem, perhaps this wasn't as necessary a step, but I felt like exploring whether I could glean any insight at all from this dataset upon seeing its visualization.



Instead, I was met with something that looked remarkably like a brain scan drawn on an Etch-A-Sketch. If anything, the one insight I could draw is that this data would likely be difficult with any linear regression algorithm, such as linear regression without polynomial features, a network with no nonlinear activation function, etc.

Methodology

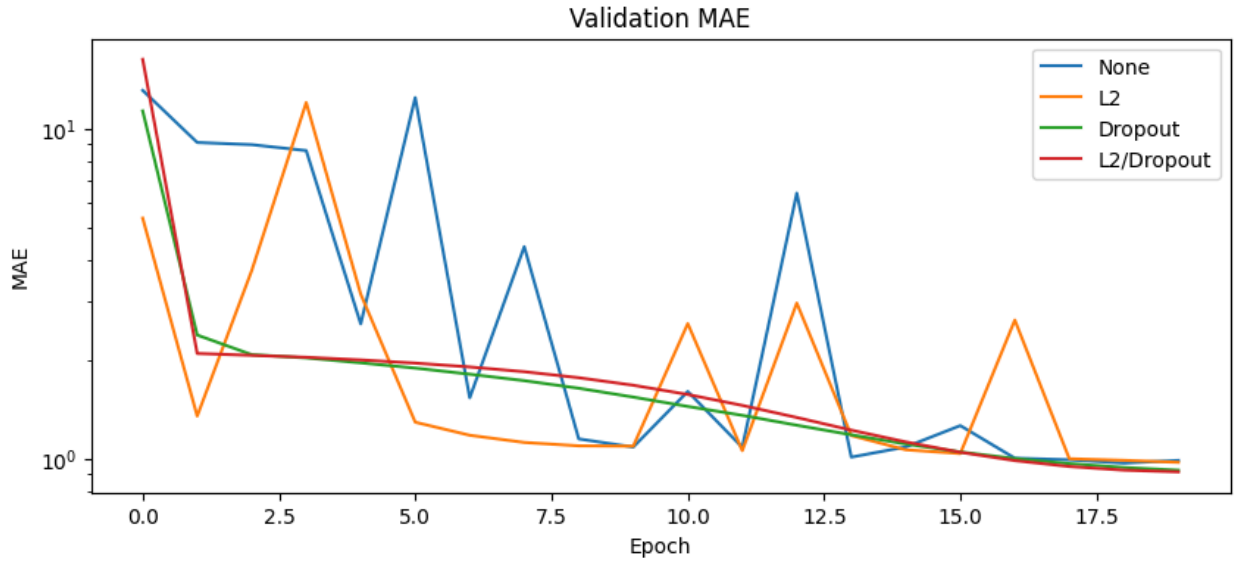
Before any models were implemented, the data was imported and splitted 80/20 with a seed of 3619 (arbitrary). I also defined the MakeColumnTransformer object early, since I intended to use it for both models in each case. I intended to pass all the features through StandardScaler, since all independent variables were continuous, which made it quite easy.

As previously mentioned, the first model was a simple fully connected Neural Network. Actually, in reality, I tested 5 different models: the first with no regularization, the second with L2 regularization only, the third with Dropout only, the fourth with both L2 and Dropout, and the fifth with the best performing of the previous 4 along with Early Stopping (patience = 2). All models used the same base architecture and were trained in similar ways. They all had an input layer of size 9, and then 3 hidden layers of sizes 512, 256, and 512. Activation function was always set to 'relu', optimizer was always 'adam' (with default lr), and loss and accuracy were measured with mse and mae, respectively. They were all trained on 20 epochs with a batch size of 256.

The simpler ML model I chose, as previously mentioned, was a gradient boosting model, more specifically a Gradient Boosting Regressor (since the target column was a continuous value). I used K-Fold Cross Validation (5 folds) to ensure stable train/test results, taking the mean of the performance metrics as the outputs. Notably, the model was initialized without any hyperparameter tuning (max depth, number of trees, learning rate, subsample ratio, etc.), since I wanted to compare a very basic Gradient Boosting model versus my Neural Network.

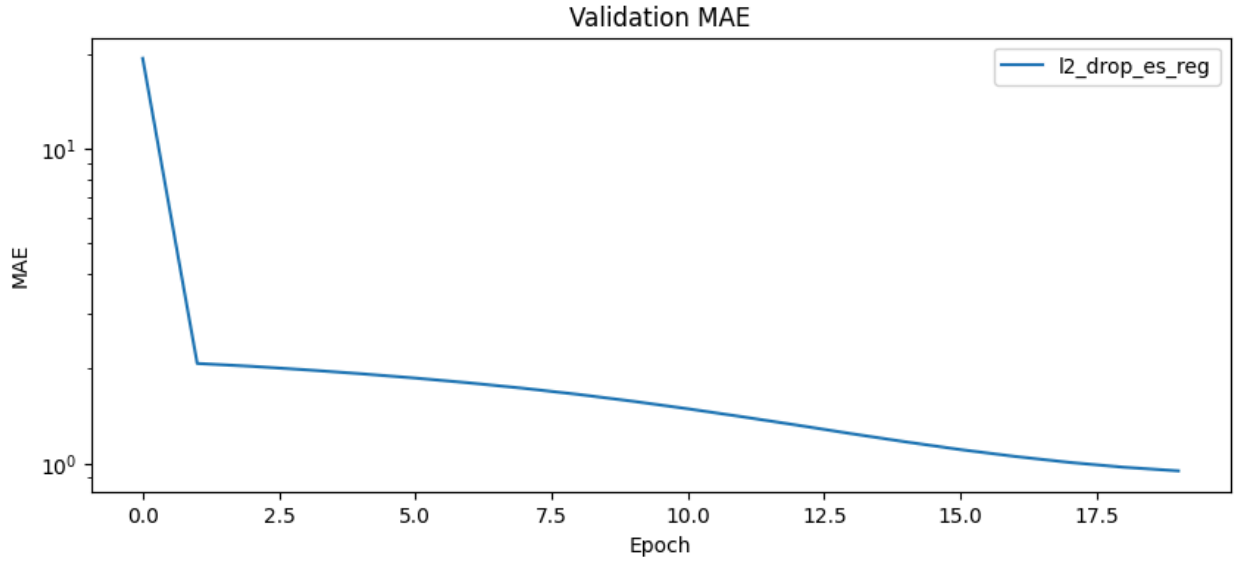
Results

The results were certainly interesting. To firstly address the Neural Network results, the following is a chart of the validation MAE of all 4 initial "test" networks. Note that the y-axis is log scaled, since if not log-scaled all 4 models' val_mae levels would all converge and be almost impossible to distinguish.



My conclusion from this chart is that, although 4 networks still seem to converge somewhat, it's a contest between Dropout and L2/Dropout. I ended up choosing L2/Dropout for a couple reasons. The first is that it technically achieved a slightly lower MAE than only using Dropout (0.9145 vs 0.9253), and also as previously mentioned in the Analysis section, the model would probably benefit from some form of regularization due to the concerning correlations seen

between a couple of the independent variables. Then, I implemented early stopping with a patience of 2, tracking validation loss, meaning that if validation loss didn't improve for 2 consecutive epochs, it would stop training and return to the best weights. However, this early stopping never actually triggered, which is a good sign, since it indicates that the training wasn't overfitting to the data. This can be further visualized in the chart, which implements early stopping in the L2/Dropout network from the previous chart:



The next step was to implement and train the Gradient Boosting Forest model, which as mentioned previously was initialized with all default parameters, with a K-Fold Cross Validation of 5 folds. The results are summarized below, which are both impressive and concerning:

```
Train MSEs: [0.2591771868331233, 0.23815281293291213, 0.22579218496607492, 0.24445580706083866, 0.22636955002119324]
Test MSEs : [1.6385181922075052, 0.3938046063794335, 0.46593126187326284, 0.4586498529183736, 0.46384932669664924]
Train MSE : 0.23878950836282847
Test MSE  : 0.6841506480150448
Train MAEs: [0.35575411598938533, 0.3356433238317597, 0.323974416915259, 0.34410937648098133, 0.3253791648863723]
Test MAEs : [1.1780296369166465, 0.4753804069797488, 0.4875353088802947, 0.5108532722071403, 0.5086770919982166]
Train MAE : 0.3369720796207515
Test MAE  : 0.6320951433964094
```

Recall that the best performance achieved by the Neural Networks was an MAE of approximately 0.91, meaning that this Gradient Boosting model absolutely eclipsed the Neural Network in terms of MAE, which was the accuracy metric of choice. However, I still don't think that necessarily means that the Gradient Boosting model is *better*, since the difference of almost double between training and testing MAE is extremely concerning. It's possible that, similar to the Neural Network, the Gradient Boosting model could've also benefitted from some type of regularization to prevent overfitting. I tested it with a K-Fold value of 10 instead of 5, and it changed almost nothing other than the time it took to train.

Overall, I think that for simple datasets such as this, a Neural Network is not necessary. However, as datasets scale, feature space increases, or patterns of nonlinearity dominate, Neural Networks tend to be more efficient for a variety of reasons. The first is that, outside of less well-known libraries such as CuML, it's hard to utilize the speedups from GPUs on traditional

models (notably, SKLearn doesn't have CUDA support). Even with 5 folds, for example, the Gradient Boosting model took many times longer than the 20-epoch neural network to train, meaning that it may be practical to use Neural Networks purely for their flexibility and scalability. Additionally, I find the ease of changing the type of task (regression/classification) with Neural Networks to be far easier than with traditional ML models. Finally, we live in a world governed by data, and this is something that a Neural Network would benefit from, whereas it's something that typical ML models would actually struggle with.

Reflection

This process was an interesting one. From changing the dataset I used to struggling to figure out how to do my own train-test-split with Tensorflow, it was a process no doubt riddled with small, annoying issues, but it was one from which I was able to learn a lot. One thing I learned on the conceptual level was how powerful of a tool Dropout was. Upon initially learning about it, I had my doubts, since to me it seemed random and haphazard compared to the mathematical precision of a technique such as L2 regularization. However, I was swiftly proven wrong and shown how effective it could be both in the main assignment and extra credit. In the future, I'd definitely keep all the methods of regularization and dealing with overfitting in mind. Another useful tool I now have in my arsenal is UMAP, which will be very helpful for classification and unsupervised learning problems in particular. Though not terribly useful for this task, the learning experience was certainly memorable.

Extra Credit

Best Performance:

```
Epoch 24/24
938/938 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8694 - loss:
0.3672 - val_accuracy: 0.8714 - val_loss: 0.3745
```

Model Architecture:

```
model = keras.Sequential([
   keras.layers.Dense(128,activation='relu', input_shape=(784,)),
   keras.layers.Dropout(0.30),
   keras.layers.Dense(128,activation='relu'),
   keras.layers.Dropout(0.30),
   keras.layers.Dense(64, activation='linear'),
   keras.layers.Dense(10,activation='softmax') ])
```

Hyperparameters:
optimizer: Adam (lr=0.003)
loss: categorical_crossentropy
metric: accuracy
epochs: 24 (this was a typo, was meant to be 25, but it actually had better performance than 25)

batch_size: 64

Model Summary:

Model: "sequential_24"

Layer (type)	Output Shape	Param #
dense_100 (Dense)	(None, 128)	100,480
dropout_45 (Dropout)	(None, 128)	0
dense_101 (Dense)	(None, 128)	16,512
dropout_46 (Dropout)	(None, 128)	0
dense_102 (Dense)	(None, 64)	8,256
dense_103 (Dense)	(None, 10)	650

Total params: 377,696 (1.44 MB)

Trainable params: 125,898 (491.79 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 251,798 (983.59 KB)