Homework #4 - Neural Network Implementation

**Logistics**

Dataset Selection

For assignment 4, I decided to do the dataset from my assignment 3. It is an insurance dataset. Given an information about a person’s age, bmi, children, sex, smoking information, and region of residence, an insurance charge is outputted. I decided to do this one over my assignment 2 as I wanted to do linear regression over classification.

Data Preprocessing

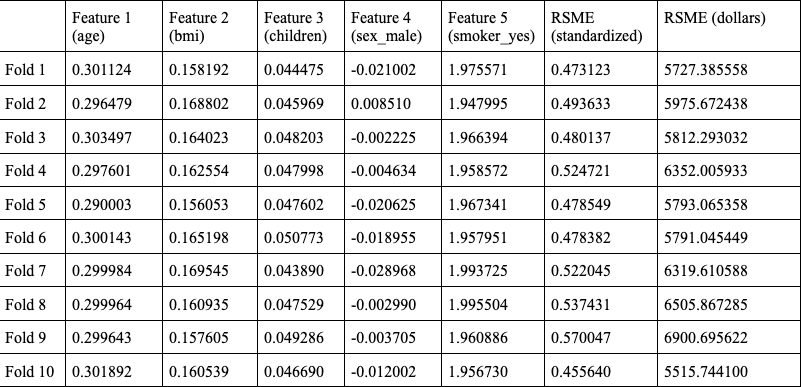
For consistency and comparison purposes, I preprocessed the data the same way I preprocessed it in assignment 3.

For feature selection, I used the person’s age, bmi, number of children, sex, and status as a smoker, but dropped region of residence to find the insurance charge amount.

For normalization, I standardized featrues such as the person’s age, bmi, number of children, and total charge using StandardScaler. I used OneHotEncoder for sex and whether or not a person was a smoker or not.

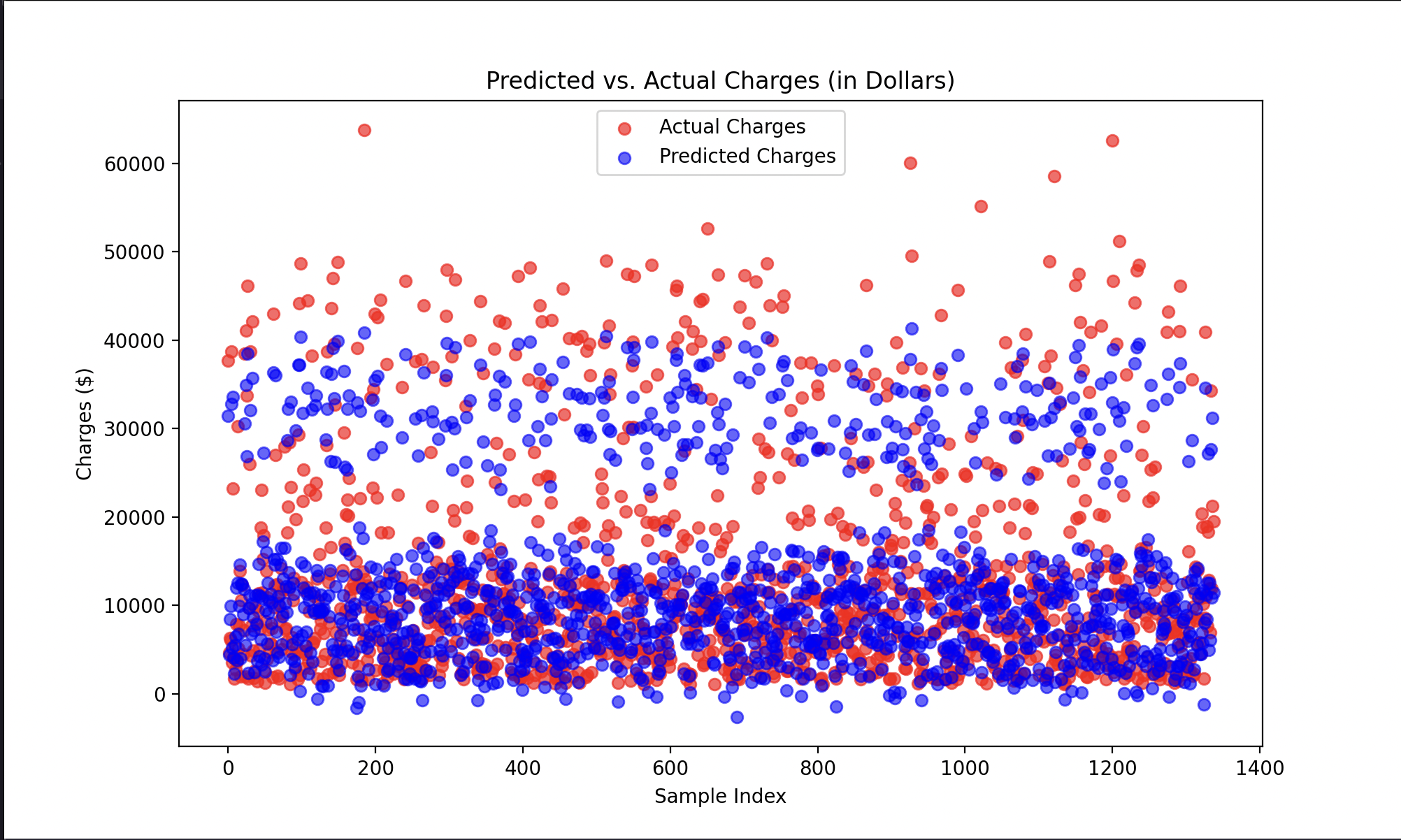
There were no missing values as previously tested, so I did not have to deal with them.

Results of the Previous Model



Average RMSE (Standardized) over 10 folds: 0.5014

Average RMSE (Dollars) over 10 folds: $6069.34



**Architecture Details**

The neural network model used for this assignment is using Scikit-Learn’s neural network: MLPRegressor for linear regression tasks.

In total, it has 4 layers: the input layer, a first and second hidden layers, and an output layer. In the input layer, it has 5 neurons for the 5 features used in the task. In the first hidden layer, it has 100 neurons with an activation function of Rectified Linear Unit or ReLU for short. This is used to capture complex, non-linear relationships between input features. In the second hidden layer, there are 50 neurons also with a ReLU activation function and it’s purpose is to refine interactions learned in the first hidden layer. In its output layer, there is 1 neuron with an activation function of linear which is default for regression tasks.

It’s loss function is mean squared error and it’s optimizer is Adam or adaptive moment estimation with a learning rate of 0.001 which is used to update the network weights based on computed gradients.

For regularization, I used a L2 Penalty of 0.001 to prevent overfitting by discouraging large weights. I used a batch size of 64 to balance training speed and convergence. For training, I set it at a max iterations of 500, had early stopping turned on, and had a tolerance of 0.0001

Libraries Used

In this assignment, a lot of previously used libraries were reused and I used some new ones. Standard libraries that I used were pandas for manipulating data/csv files as well as to create dataframes, numpy for mathematical operations, and matplotlib for plotting.

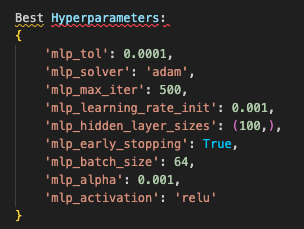
Previous sklearn imports I’ve used before were KFold for 10-fold cross validation in evaluating the results, StandardScaler, OneHotEncoder, and ColumnTransformer for data preprocessing, and mean\_squared\_error for computing the performance of the model.

New imports I used in this assignment were RandomizedSearchCV for finding the optimal hyperparameters, cross\_val\_predict for generating cross-validated estimates for each input data point, and MLPRegressor as the neural network model to perform the linear regression task.

Model Training

To find the optimal model performance I experimented with different hyperparameters including learning rate, batch size, number of epochs, and layer structure.

To expedite the process, I used Scikit-Learn’s RandomizedSearchCV during implementation to automate testing which hyperparameter values would yield to an optimal result. This is the results that I got for best hyperparameters.

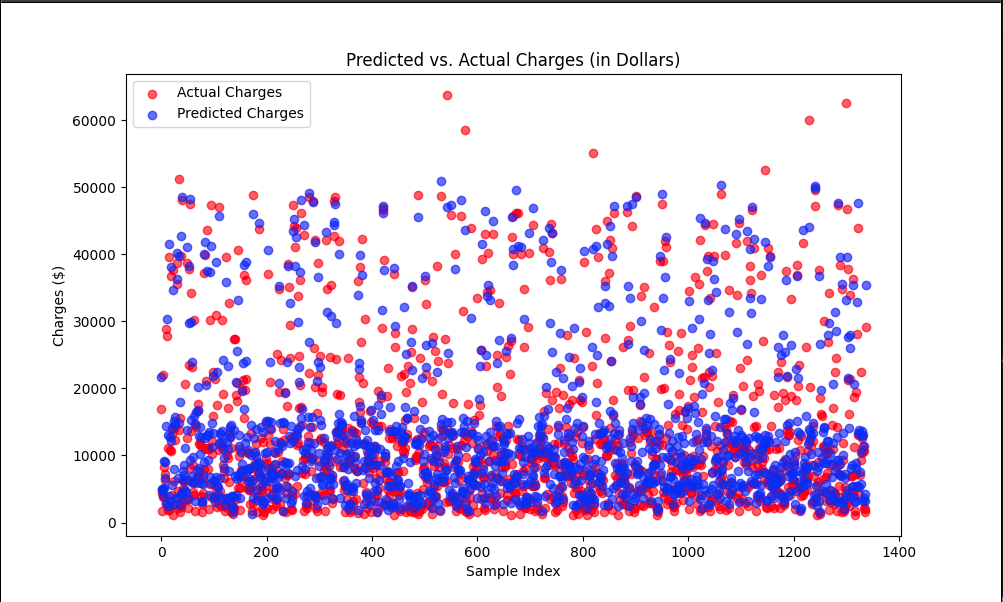


**Results and Visualizations**

| Fold | Feature 1 (age) | Feature 2 (bmi) | Feature 3 (children) | Feature 4 (sex\_male) | Feature 5 (smoker\_yes) | RMSE (Standardized) | RMSE (Dollars) |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0.112186 | 0.142435 | 0.106322 | 0.109295 | 0.190626 | 0.353997 | 4285.310234 |
| 2 | 0.114322 | 0.14941 | 0.106645 | 0.110547 | 0.189814 | 0.380736 | 4608.994909 |
| 3 | 0.116064 | 0.151662 | 0.110429 | 0.112163 | 0.192638 | 0.349759 | 4233.998069 |
| 4 | 0.110194 | 0.145723 | 0.109716 | 0.111526 | 0.192299 | 0.415065 | 5024.560213 |
| 5 | 0.115904 | 0.158083 | 0.112008 | 0.113109 | 0.188584 | 0.313419 | 3794.091661 |
| 6 | 0.116658 | 0.150771 | 0.108006 | 0.109138 | 0.189502 | 0.408601 | 4946.313379 |
| 7 | 0.113652 | 0.150055 | 0.110426 | 0.112137 | 0.190654 | 0.387348 | 4689.034316 |
| 8 | 0.10959 | 0.150996 | 0.107769 | 0.112433 | 0.191355 | 0.424621 | 5140.240373 |
| 9 | 0.11679 | 0.155715 | 0.111641 | 0.111006 | 0.188396 | 0.469691 | 5685.836279 |
| 10 | 0.111778 | 0.147301 | 0.108541 | 0.11221 | 0.19128 | 0.354772 | 4294.688425 |

Average RMSE (Standardized) over 10 folds: 0.3881

Average RMSE (Dollars) over 10 folds: $4698.66



Overall the model performed well. Here is a table of it’s performance over 10 folds and a graph of it’s predicted vs actual charges. It’s average RMSE score over 10 folds is 0.3881 in standardized units and $4698.66 in dollar units.

**Comparative Analysis**

A previous model that I will be using for comparison is my linear regression model from hw 3. Given the same dataset and the same preprocessing methods, the performance of each model was not the same.

In terms of performance, my neural network performed better. As shown previously, over 10 folds, the RMSE of the linear regression in standardized units was 0.5014, and in dollar units was $6069.34. For the neural network, it’s RMSE in standardized units was 0.3881 and $4698.66. The difference between these two models is 0.1133 in standardized units and 1370.68 in dollar units. In terms of percentage this a 22.6% difference. For everyday people, an insurance charge of over $1370 is a lot, so this difference is big.

Looking at both models’ graphs, it can be clearly seen that the neural network was also a lot better at predicting outliers, although it is not perfect and there are extreme outliers.

In terms of efficiency, the linear regression model wins as the model has less complexity and lowered computational requirements. My machine ran the linear regression model a lot faster than it did the neural network.

Linear regression had 5 parameters and no activation function as it was a linear model. The neural network had significantly higher parameters due to multiple layers and neurons and used a ReLU activation function for non-linearity.

**Insights and Learning**

After doing this assignment, I’ve learned that there is a significant tradeoff between performance and efficiency. The better the performance of a model is, the less efficient it will be. The neural network with its complexity performed better than the linear regression model, but took longer to execute. However, it is not always necessary to pick the better performing model.

I think depending on how critical the task is, maybe it’d be better to use a more simple model like linear regression. In the real world, performance is not everything, we need to consider budget and equipment. Choosing the right model for the right task is still a skill that is useful.

If I could continue on this assignment, I would try out different neural network models such as PyTorch and see if there’s a difference. I would try different hyperparameters that would be standardized to both models to limit any differences that could arrive due to different hyperparameters.

Overall, I think this assignment was a good introduction to neural networks.