Data Science – Mini Project 3

Introduction

Buying a home is one of the most important milestones during a person's lifetime. However, housing prices have steadily been rising at a pace faster than income, making it more and more difficult to acquire enough wealth to buy a home. The value of a home is determined by several different factors. The purpose of this project is to create a model that predicts the price of a home based on some of these factors. This can help shift the market to be more favourable for those who want to buy their first home, increasing the net worth of the total population.

For this project, the choice of model will be a deep neural network (DNN). Machine learning models have proved to be useful for predicting tasks, and DNNs especially excel in these tasks.

Exploring the data

To train the DNN data is needed. As training data, the classic Boston Housing data set is used. The benefit of the chosen data set is that it has no missing values, which reduces the need of pre-processing slightly.

As can be seen in **Part 1** of the code, the price of a home is strongly correlated to some features like the number of rooms, and lower crime rate in the town where the house is located is correlated to a higher price. However, between some features and the house value, and between each other of the features, no obvious correlation can be seen.

Also as seen in **Part 1**, the data comes in various ranges, with means and maximums ranging from less than 1 to several hundreds. This will play a part in designing the model's architecture.

Model architecture

For the DNN, an architecture needs to be chosen. The task of predicting housing prices falls under the category of regression problems, so intuitively a normal multilayer perceptron makes sense as the choice of DNN type (as opposed to e.g. a convolutional neural network). Several configurations of numbers of hidden layers and neurons were tested, starting from an architecture with no hidden layers. The configuration that ended up performing the best was as follows: Input layer with 13 neurons (1 for each input column in the data set), hidden layer with 128 neurons, normalization layer, hidden layer with 256 neurons, normalization layer, hidden layer with 256 neurons, normalization layer, hidden layer with 256 neurons, normalization layer, output layer with 1 neuron. The purpose of the normalization layers is to make the data more uniform, since the unnormalized datapoints affect the activation functions differently otherwise.

In addition to testing different layer-node configurations, a Grid Search cross-validation was executed to test out some hyperparameters of the DNN. These hyperparameters included

activation function (RELU vs linear to name a few), kernel initializer (normal vs lecun uniform and more), amount of training epochs between 100 and 1500, and dropout layers with dropout rates in the range 0.0 to 0.9 with 0.1 increments. The configuration that ended up performing the best was a DNN with RELU activation function, normal kernel initializer, 1500 epochs, and instead of dropout layers some kernel regularizers were used (**Part 3** in the code). The addition of kernel regularizers helps to prevent the model from overfitting the training data. The Grid Search was removed from the final version of the code since running it may take several hours.

When training the DNN on the training data set, the performance starts in the first epoch with a mean absolute percentage error of over 69% and a loss of 62. However, as training progresses the network starts to converge and the final mean absolute percentage error is around 13% (in one of the runs), final loss around 3 and final validation loss around 4. Even though the Grid Search gave 1500 epochs as the best performing parameter, it can be seen from the graph in **Part 4** in the code that the performance of the DNN rapidly improves in the beginning but after 200 epochs the improvement is minimal. The DNN also does not seem to overfit the training data that much, as it did without the kernel regularizers.

Conclusion

The purpose was to create a model that predicts the value of a house, given some information about the house such as size and age of the house, crime rate where it is located and more. The success of this project was decent. The DNN manages to predict the price of a house with an absolute error percentage of around 13-18%, and a loss of around 3,5. However, considering how notoriously accurate DNNs are, a lower error percentage was expected.

There were several challenges encountered during the project. First and foremost, tuning the architecture of a DNN is more like an art than an exact science; it is a skill that comes with experience, which the author had very little of. Researching how each aspect of the DNN impacts its performance took an enormous amount of time.

As the charts from **Part 1** in the code showed, some features did not really seem to properly correlate with the median house value, and processing the data set more and removing some data might have improved the model's accuracy. However this could not be done within the given timeframe.