Introduction

A house value is more than location and square footage. Like the features that make up a person, many aspects make up value of a house. This project is going to be focused on solving the problem of predicting house prices for both house buyers and house sellers.

In the project, I will evaluate the performance and predictive power of neural network model that has been trained and tested on data collected from houses in suburbs of Boston, Massachusetts. The model trained on this data that is seen as a good fit could then be used to make certain predictions about a house – in particular, its monetary value. The model would prove to be invaluable for people like real estate agents who could make use of such information on a daily work, or people who would like to find their dream home with a reasonable price tag.

The dataset used here, Boston housing dataset, is a very ubiquitous one. This dataset contains information about 506 census tracts of Boston from the 1970 census. And each of the 506 entries represent aggregated data about 14 features for homes from various suburbs in Boston, Massachusetts, including average number of rooms per dwelling, pupil-teacher ratio, and so on.

Like 'hello world', the Boston Housing Dataset has become part of a common vocabulary. And it will remain so, not only because thoroughly labeled datasets for machine learning are still not that easy to find, but because using the same dataset for decades to test different algorithms has allowed scientists to control for that variable and highlight the differences in algorithm performance. However, the Boston housing dataset is small, especially in age of big data. If the amount of data is too small, the model is likely to appear overfitting, resulting in poor model performance. In order to reduce the impact of this problem on the training model, I intend to optimize the data preprocessing and model selection by, for example, reducing the number of features, cross-validation, and reducing the complexity of the model.

Neural Network Model

Algorithm

Neural Networks are a set of algorithms, modeled loosely on the human brain, a neural net consists of thousands or even millions of simple processing nodes that are densely interconnected. Most of today's neural nets are organized into layers of nodes, and they're "feed-forward", meaning that data moves through them in only one direction. An individual node might be connected to several nodes in the layer beneath it, from which it receives data, and several nodes in the layer above it, to which it sends data.

To each of its incoming connections, a node will assign a number known as a "weight." When the network is active, the node receives a different data item — a different number — over each of its connections and multiplies it by the associated weight. It then adds the resulting products together, yielding a single number, which is called net input, as Equation 1 shows.

$$n = \mathbf{W}\mathbf{p} + b \qquad \text{Equation 1}$$

Then, the neuron output can be calculated by Equation 2, depending on the particular transfer function.

$$a = f(\mathbf{W}\mathbf{p} + b) \qquad \text{Equation 2}$$

When a neural net is being trained, all of its weights and biases are initially set to random values. Training data is fed to the bottom layer — the input layer — and it passes through the hidden layers, getting multiplied and added together in complex ways, until it finally arrives, radically transformed, at the output layer. During training, the weights and biases are continually adjusted until training data with the same labels consistently yield similar outputs.

Experimental Setup

After preprocessing data, I selected variable ‘MEDV’ as target, and rest as features. Then, ‘train_test_split()’ function was used to split data into train part and test part.

Here, ‘LMPRegressor()’ was chosen to be used as Neural Network model in PyCharm, containing many parameters like ‘activation’, ‘solver’, ‘max_iter’, and so on.

To get the best parameters, another python file was created to adjust different parameters mentioned above. Both ‘GridSearchCV’ and ‘RandomizedSearchCV’ methods were applied to find the best parameters.

‘GridSearchCV’ is grid search and cross-validation. Grid search, a search for parameters, adjusts the parameters in order, according to the step size within the specified parameter range, uses the adjusted parameters to train the learner, and finds the parameters with the highest accuracy on the verification set from all the parameters, which actually is a cycle and comparison process. Although it can guarantee to find the most accurate parameters within the range, this is also the defect of grid search. Because it requires traversing all possible parameter combinations, which is very time-consuming in the face of large data sets and multiple parameters.

‘RandomizedSearchCV’ is used in the similar way as ‘GridSearchCV’, but it replaces ‘GridSearchCV’'s grid search for parameters by randomly sampling in the parameter space. For parameters with continuous variables, ‘RandomizedSearchCV’ will sample them as distributions. This is what grid search cannot do. Thus, ‘RandomizedSearchCV’ is chosen finally. Its search ability depends on the set ‘n_iter’ parameter.

Then, I trained the model using the best parameters by the customized function ‘train_model()’ and got MSE and R^2 score to judge the performance of model. Finally, the return value ‘loss_curve_’ and prediction were used to plot the results.

Results

In adjusting parameter part, I focus on ‘activation’, activation function for the hidden layer; ‘solver’, the solver for weight optimization; ‘batch_size’, size of minibatches for stochastic optimizers, which is useful when the model is not converge; ‘max_iter’, the maximum number of iterations, which is also an important parameter for convergence; ‘early_stopping’, whether to use early stopping to terminate training when validation score is not improving. Table 1 lists the best parameters.

Table 1
Best Parameters

Parameters	activation	solver	batch_size	max_iter	early_stopping
Value	relu	sgd	70	11000	True

Note: The number of sampling without replacement ‘n_iter’ in ‘RandomizedSearchCV’ is 350. Best score in ‘RandomizedSearchCV’ is 0.622.

Parameter ‘relu’ is the rectified linear unit function as Equation 3 expresses

$$f(x) = \max(0, x) \qquad \text{Equation 3}$$

The advantage of this activation is that the convergence rate of SGD obtained by using ReLU will be much faster than sigmoid/tanh. It can get the activation value without calculating a lot of complicated operations, compared to sigmoid/tanh.

Parameter ‘sgd’ refers to stochastic gradient descent. Since it is not a loss function on all training data, but a loss function on a certain training data in each iteration, so that the update speed of each round of parameters is greatly accelerated.

MSE and R^2 score measure whether the model is good.

MSE is a risk function, which can be calculated by Equation 4, corresponding to the expected value of the squared error loss. MSE is almost always strictly positive. It is a measure of the quality of an estimator. The closer to zero the value is, the better the model is.

$$MSE = \frac{1}{n}\sum_i (y_i - y_{\text{pred}})^2 \qquad \text{Equation 4}$$

MSE of neural network model is 10.352, close to zero, indicating the model performs well.

R squared score is a statistical measure of how close the data pairs in a set are to their fitted regression line. This measure ranges from 0 to 1, suggesting the extent to which the dependent variable in a data set is predictable. A R squared of 0 means that the dependent variable cannot be predicted by the independent variable, while 1 means that it can be predicted without error. Thus, the larger R squared score is, the more predictable the dependent variables are. The result of neural network model is 0.896, which also shows that the model performs well.

What’s more, two images are also improved to judge the performance.

Firstly, the image of error and epochs is listed as Figure 1. We can see that as the number of iterations increases, the mean square error of the model gradually decreases. When the number of iterations reaches about 150, the mean square error of the model has been reduced to 0. This indicates that the model has converged at this time, which is roughly the same as ‘n_iter’.

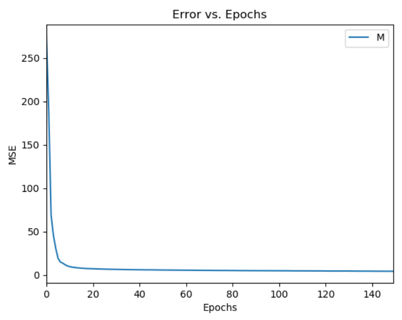

Figure 1. Error vs. Epochs

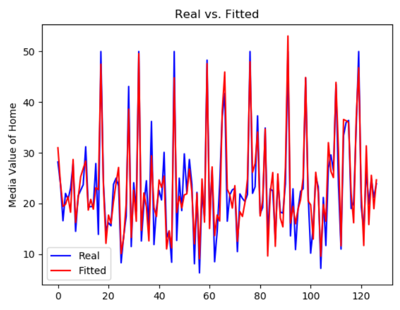

Figure 2. Real Value Curve vs. Fitted Curve

Secondly, the fitted curve and true value curve is shown in Figure 2. The blue line represents the real value of house price, while red line represents the fitted value predicted by neural network model. We can see that the red line and the blue line are roughly coincident, which directs that the performance of model is good and it can predict most house price correctly.