ECS 171 Final Project Report

Abhay Manu Sawhney, Anastasia Levchenko, Aric Luo,
Arthur Hlaing, Matthew Reese, Nir Voloshin, Tyler Gordon

November 28, 2018

Abstract

The ability to accurately predict real estate prices is an important tool for potential buyers to make informed decisions about what could be the largest and most expensive purchase in their lifetime. Zillow, a real estate database company, first released its free home valuations to the public eleven years ago and has found widespread success, with a median absolute error rate of 14 percent (Zillow, 2017). By 2017, that median error rate had dropped drastically, to just 4.3 percent and Zillow still strives to improve its prediction accuracy every year. This project applies two popular algorithms, Artificial Neural Network and Extreme Gradient Boosting, to predict the log error between Zillow's prediction (Zestimate) and actual sales price. This was inspired by Zillow's Home Value Prediction competition on Kaggle that asks data scientists to build their own models that aim to improve Zestimate's accuracy. Using an Artificial Neural Network with a 27 - 9 - 1 architecture and an Extreme Gradient Boost model with a max depth of 6,500 estimators, and a learning rate of 0.01, our project achieved a mean squared prediction error of 0.0292 and 0.0288 respectively. These results do not give us any insight into which method is best suited to this task, because our models performed very similarly on this task. Further investigation must be conducted to make progress on this front.

1 Introduction

The increase in computing power in recent decades has allowed complex machine learning algorithms to be utilized to make predictions about future economic trends. These trends include prices for publicly traded stocks, real estate, and cryptocurrencies. Artificial Neural Network (ANN) and Extreme Gradient Boosting (XGBoost) have been the most popular algorithms used for these purposes.

Forecasted real estate values allow homeowners and customers to make informed decisions about when to buy and sell. Zillow's Zestimate home valuation was first released 11 years ago, estimating based on hundreds of data points for each property and synthesizing that data into an accurate estimated market value. This marked the first time that information of this kind was made available to

non-professional customers free of charge. Their estimates were quite accurate from the start, and because of this Zillow has become the most trusted home value estimator on the market.

Having established itself as one of the largest and most trusted marketplaces for real estate in the United States, Zillow works hard to improve their Zestimate algorithm every year. Last year, Zillow published a competition offering a one-million dollar grand prize on Kaggle to challenge the data science community to help push the accuracy of the Zestimate even further. Participants in the competition employed different methods including Linear Regression, XGBoost, LightGBM, GBM, OLS, and ANN to predict the future prices of properties. In the following months after the competition closed, all of the submitted models were evaluated based on how accurately they predicted the log error of the Zestimate and the actual sale price of properties.

$$logerror = \log(Zestimate) - \log(SalePrice)$$

Because at the time we started this project, the contest had already been completed and the results published, we decided to use this competition to conduct a meta-analysis of the approaches that were considered by the participants and perform an experiment comparing two of the most effective methods. Each participating group only relied on one machine learning algorithm to compute their results. However, in order to improve the performance of machine learning models in general, we believe that it is important to make direct comparisons between different algorithms to be able to better understand to what contexts and problems each algorithm is best suited. Chakraborty et al. (2018) is an excellent example of a meta-analysis of techniques which seeks to bring light to the advantages of analyzing different approaches. They use the methodology to analyze the models that predict energy consumption and make a claim that machine learning algorithms are particularly adept at modeling time series data.

Although much work has been done to craft individual algorithms suitable for the Zillow dataset, there remains a need for deeper analysis to compare the different machine learning techniques. Rathee (2017) demonstrates that analysis of this kind using data and results from Kaggle competitions is both feasible and necessary. Based on the analysis of the competition's results, we have chosen two algorithms, which have been demonstrated to be effective yet quite dissimilar. We decided to do an in-depth comparison of the predicting performance of Extreme Gradient Boosting (XGB) and Artificial Neural Networks (ANN) by using these two algorithms to predict the log error between the Zestimate and actual sale price.

2 Methods

In an attempt to draw comparisons between popular machine learning algorithms, we first performed a linear regression on the dataset. From there, we selected two of the more widely used models: neural networks and gradient boosting. For neural networks, we decided to use a standard ANN, i.e, the multilayer

perceptron, because these types of models are well suited for complex data sets with many interrelated variables. We built the ANN using Keras, which is a high-level neural network API running on top of the Tensorflow framework.

The advantage of using XGBoost over regular Gradient Boosting is that XGBoost provides useful tools like regularization which helps the model by discouraging it from overfitting. Overfitting is when an algorithm models the training set very well almost as if the model memorized the training data set and as a result it does not generalize to data it has not seen before. In addition, XGBoost handles missing values and allows us to set a maximum depth, thereby making the pruning process more effective. It also has an built-in functionality to implement cross-validation to allow us to further train our model (Jain, 2001), and its built-in parallel processing allows it to run significantly faster.

2.1 Data Preprocessing

The first step of data preprocessing was to determine which features would be the most informative and most useful in predicting the values of the dependent variable. Some of the features in the dataset have a large percentage of missing values, which can hinder our prediction capabilities. Although such features may have a significant correlation with the dependent variable, they remain difficult to use with such a large number of missing values. Filling in the missing values would be an effective approach for some of the features, but we decided that for variables with over 60% missing values, it is not worth trying to replace them because there is not enough data to get a satisfactory representation of these missing values. It is also important to note that some of the variables have missing values not because the data was not collected, but rather because there was no data to collect. An example of this would be a variable like "decktypeid", which would only be filled in with a value if the house actually had a deck. Filling in variables of this type with "predicted values" would not make sense, which is why we decided that many variables of this type were acceptable to remove.

Looking at Figure 1 below, in which the horizontal axis represents the proportion of missing values, we can see that there are multiple variables that have over 60% missing values, and those variables were the ones we eliminated from the dataset.

The next step after removing the appropriate features was to normalize the remainder of the data. Different features have different scales; therefore, by using mean normalization, we ensure that all the features are equally represented in the model, and that we can fill the missing values without changing the mean of the variables.

$$\text{Mean normalization } (\bar{x}) \text{ can be calculated as: } \bar{x} = \frac{x - \mu(x)}{\max(x) - \min(x)}$$

In the formula given above, $\bar{x}$ is the normalized value, $\mu(x)$ is the mean of the feature, and max(x) and min(x) are the maximum and minimum of feature x, respectively. This allows us to achieve a new range between -1 and 1 for each

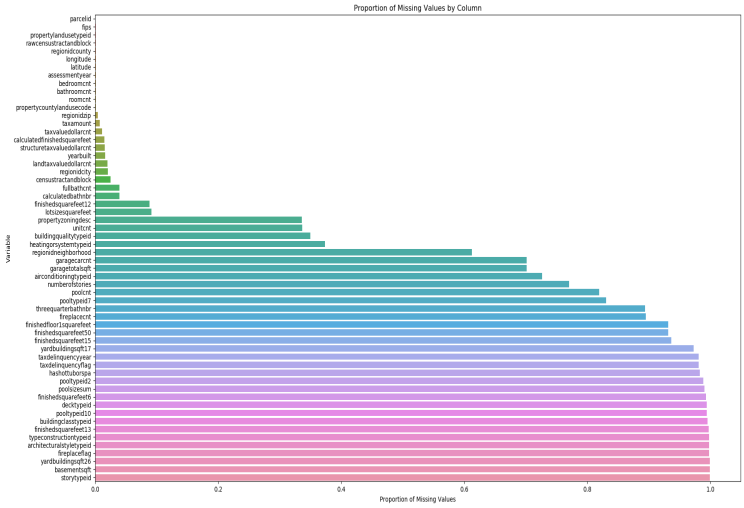

Figure 1: Proportion of NaN Values for Each Feature

of the features. The remaining missing values for the features that we kept have been filled with 0, because 0 is the mean of each feature after we performed mean normalization. We also considered using other methods such as linear and logistic regression to predict the missing values; however, to use logistic regression to predict the categorical and discrete variables, we would need to know all of the values for the other features. Our data set does not contain at least one row where all the features are present even after removing feature columns with majority missing values. In fact, the only column with no missing values is the "parcelid" column, which is the unique identifier for each observation and not an independent variable. Therefore, we opted to fill the missing values with 0, which is the normalized average value of all of the features, to replace the NaN values.

We also considered replacing NaNs with zeros before normalization; but by doing so, the zeros inserted before normalization would change the mean of that variable, which would introduce bias into the predictions. Instead, we performed normalization first to ensure the mean of the variable would already be 0, thus allowing the missing values to be replaced with zeros without altering the mean of that feature. Replacing the missing values only applies to the ANN; in XGBoost, the model handles the missing values internally, which means that there is no need to replace the missing values ourselves.

We wanted to make sure that the transaction dates could also be used in the model. The time of the year can affect home prices significantly, because of the increase of home purchases during the home buying season. The home

buying season begins around the start of spring and ends towards the end of summer because it is much easier for people to move at this time, and because more new listings come on the market during the spring than any other time of year. Anderson (2017) even mentions that the best selection and deals are during late summer - meaning that the time of year that a house is purchased definitely has an effect on its cost.

We converted the transaction dates into timestamps so that they can be processed by the ANN, allowing the neural network to use that data for prediction. After conversion, samples that are close in date will have timestamp floating point numbers that are close to each other numerically, which should allow the model to account for any relationship between the date of the sale and the price.

2.2 Linear Regression (Baseline Model)

Before running more complex algorithms on the data, we used linear regression to justify the use of more complex methods and provide a baseline for our results. Linear regression is a much simpler method than ANN and XGBoost for predicting output given a set of input features, thus it is well suited for a baseline metric. Even before using linear regression, we can see from Table 1 (see appendix for tables) that the correlation between logerror and each individual feature is very small.

This already indicates that using a linear regression model is not likely to be the best way of modeling this data. However, even though there is not a strong correlation between the predictor variable and any of the features independently, it is still useful to build a linear regression model to provide a baseline for what magnitude of prediction error we should expect.

Initially, we built a model using all 27 features to assess its performance. Surprisingly, the R^2 for the model is very low with a value of 0.005083189347150241. A consequence that arises when using R^2 is that it increases as you add more variables to the model. Although R^2 is usually expected to grow significantly, we can make the conclusion from this R^2 that a linear model does not accurately represent the data. As a result of this, other methods which have the ability to encode nonlinearities might work better for this type of data. Testing out a linear regression model gave us an idea of a baseline value which we hope to improve upon using the more complex methods of ANN and XGBoost.

2.3 Extreme Gradient Boost (XGBoost)

Extreme Gradient Boosting works by making splits in the dataset. Boosting methods have inherent advantages such as reducing variance and bias, and XGBoost takes all of those advantages and builds on them. Gradient Boosting Machines (GBMs) convert weak learners (typically decision trees) into strong learners: that means that they take multiple classifiers that are slightly correlated to the true classification and uses them to make a classifier that is strongly correlated to the true classification. Weak learners with just one split in the dataset, also known as decision stumps, are combined to create strong learners.

Trees are added one at a time, but once they are in the model they don't change. The model improves by sequentially correcting the errors of the previous trees.

The goal is to construct a model, F, that predicts values by minimizing the mean squared error:

$$\frac{1}{n}\sum_i(\hat{y} - y_i^2)$$

At each stage of construction, there is an "imperfect" model, Fm. The gradient boosting algorithm then improves this model by constructing another model that is adding an estimator to create a better model, as such:

$$F_{m+1} = F_m(x) + h(x)$$

where h will be fitted to the residual. Each F_{m+1} is supposed to be an improvement of its predecessor, F_m.

Extreme Gradient Boosting is effective due to its large scalability and efficient memory usage of datasets. XGBoost has become increasingly popular over the past few years and it has won several Kaggle competitions due to its versatility and quick deployment time.

We implemented the Extreme Gradient Boosting algorithm using XGBRegressor (Chen, 2016). We found the optimal parameters by performing a parameter sweep using the Grid Search Cross Validation function from the SKlearn library (Pedregosa, 2011). Using that function, we compared the parameters of 'max_depth' values of [4, 6, 8], 'n_estimators' (number of weak learners) values of [500,1000], and 'learning_rate' of [0.1, 0.01, 0.2, 0.3]. The variable 'max_depth' tells us the maximum depth a tree should go before pruning back and 'n_estimators' tells us the number of sequential trees to be modeled.

The parameter sweep tested all 24 combinations, cross-validating with 3 folds at every iteration. The model with the best loss was saved to "best_params" and those params were used to create our final XGBRegressor object. We found that the function returned the best values of 'max_depth' of 4, 'n_estimators' of 500, and 'learning_rate' of 0.01.

Using these optimal configurations, we trained the XGBoost model with the ten different stratified folds. After evaluating the model, we found the mean squared errors for each fold, which can be found in Table 4. We then averaged the mean squared errors computed and found an average mean squared error of 0.029230.

2.4 Artificial Neural Network (ANN)

To find the optimal configuration for our Artificial Neural Network, we performed a parameter sweep (grid search) on the number of hidden layers (1, 2, 3) and the number of nodes in each layer (3, 4, 6, 9, 12). Table 2 shows the testing loss for each combination of layers/nodes.

From Table 2, it can be deduced that 9 nodes per hidden layer with 1 hidden layer, 2 hidden layers, and 3 hidden layers and 12 nodes per hidden layer with 2 hidden layers, and 3 hidden layers are the five ANN configurations with the least

mean squared error (MSE). Among the top five configurations, the testing MSE values do not vary by a significant amount. Since simpler networks are able to generalize better than more complex ones with similar MSE values (Sietsma et al, 1991), we decided to use the simplest model with 9 nodes per hidden layer with 1 hidden layer as our ANN configuration. Sietsma et al. (1991) also found that additional nodes/layers beyond the minimum required to perform the task can be removed with no ill effects.

With the configuration chosen above, we built the ANN model using Keras, which is a high-level neural network API running on top of the Tensorflow framework. As shown in the preprocessing subsection, we have reduced the number of features to 27; therefore, our ANN has an input layer with 27 nodes, a single hidden layer with 9 nodes, and an output layer with 1 node since the output is a single continuous value.

For layer activations, we used Rectified Linear Units (ReLu) for the hidden layer and a linear combination of inputs and weights for the output layer. ReLu has been shown to perform better than sigmoid and tanh activations for Deep Neural Networks (Maas et al, 2013). We have also trained the ANN with different activations such as sigmoid, tanh, and ReLu; ReLu achieved the lowest MSE on the testing set. Therefore, we chose ReLu activation for the hidden layer, resulting in 27 - 9 - 1 ANN architecture. We want this network to be able to generalize to unseen data well.

Stochastic Gradient Descent (SGD) is found to be better at generalizing than adaptive optimization methods such as Adam, Adagrad, and RMSprop (Keskar et al, 2017), but we have also tested each optimizer with our network to substantiate the previously presented claim that SGD works the best particularly with our housing prices dataset. Many of our training instances converge after 500 epochs; therefore, we have set the 500th iteration as our final epoch. Moreover, setting epochs at 500 allows us to train and test different hyperparameters without sacrificing the quality of the neural network. Batch size is set as 32 which is small compared to the total training set of 60k samples. This allows us to retain the quality of the model while saving on computation power expended on training. We have found that the learning rate of 0.1 allows us to converge faster and at a lower loss than 0.01.

2.5 Stratified 10-Folds Cross Validation

Different prediction algorithms perform differently depending on how the training/test set is split; for a given testing set, some algorithms will perform better than others. Therefore, we would like the performance of the model to be representative of the whole dataset made available to us. We have chosen the MSE as a common metric to compare among the models. When determining the MSE, we would also like the method to have low bias and low variance. As we have a dataset with a large number of observations, leave-one-out cross-validation (LOOCV) is not a viable option because it is computationally expensive. For real-world datasets, stratified ten-fold cross-validation performs better than other estimation methods such as holdout and bootstrap (Kohavi, 2001). This

is relatively computationally inexpensive compared to LOOCV because each model only needs to be trained ten times and tested ten times.

To create ten folds that are stratified, we shuffled the data until the mean values of all the folds were similar. Kohavi (2001) recommends that even if computation allows for more folds, stratified ten folds cross-validation is the best method to use as it can produce low bias and variance. Tables 4, 5, and 6 (see appendix) shows the MSE for each testing fold.

2.6 Comparison Method

Our comparison of the performance between XGBoost and ANN was conducted by analyzing the MSEs obtained from stratified ten-fold cross-validation. Therefore, we have taken the average of the MSEs produced by the ten folds for each prediction algorithm, namely linear regression, XGBoost, and ANN. The folds are created using the same random state for all models so the ten training and testing sets are consistent across all the models.

3 Results

From Table 3, it can be deduced that the 27 - 9 - 1 ANN model and the XGBoost model show no significant difference in terms of the mean squared error. The XGBoost model is only 1.4% better than the ANN model. The values in Table 3 above are the mean of the MSE values for each of the 10 folds on the data. Though the Extreme Gradient Boosting model achieved the lowest MSE, neither it nor the Artificial Neural Network model achieved significantly better results than Linear Regression.

4 Discussion

Since all three of the MSE's are very close to each other, it is useful to choose the method that is the least computationally intensive. Because of the nature of the linear regression model, it is not as computationally intensive as ANN and XGBoost. Alternatively, it may be possible to improve the accuracy of the ANN and XGBoost models through more extensive hyperparameter tweaking.

We tuned the parameters by performing a grid sweep to ensure that the models are configured in the best way possible. We tested 24 configurations for the XGBoost model and 18 configurations for the ANN. This helped us select the optimal set of hyperparameters for our final models. So, we believe that any possible improvement may be more likely to come from the data preprocessing and feature engineering rather than the hyperparameters of the algorithm. For examples, features such as 'pool typeid', 'airconditionertypeid', and 'buildings class typeid' might not be as significant as features such as 'regionid city', and 'regional dzip'. One way we suggest to overcome this potential problem would be to use dimension reduction methods such as Principal Components Analysis (PCA).

As previously mentioned, the transaction date is an important feature when predicting home prices. In our project, we opted to convert the transaction date into timestamps, which might not have been the most suitable method. Alternatively, since housing price valuations change according to seasons, we could model the transaction date as a sine or a cosine function or a combination of both (Stolwijk, 1999). By doing this, the months would no longer be represented as twelve different values; instead, months that are similar in terms of home prices will have similar values. This approach should be considered for future work since it can potentially help improve the machine learning algorithms, especially neural networks.

We compared the results of our models using mean squared error (MSE). This allows comparison of performance between models but is only useful for relative comparisons. Future approaches could make use of another metric, such as the mean absolute percentage error (MAPE), to provide an assessment of the performance of our models in an absolute sense, which would facilitate interpretation of the results.

5 Conclusion

This project aims to compare various machine learning algorithms to predict the log error of predicted house values from a dataset produced by Zillow, a popular online real estate valuation company. It stems from a Kaggle competition posted by the company challenging participants to improve their house valuation metric called the Zestimate. By comparing artificial neural networks and extreme gradient boosting, we discovered that each model performs similarly on the dataset. Thus, we conclude that performing linear regression, our baseline model, is most effective in this circumstance, as less complex models are more scalable. Future work might improve prediction accuracy by combining models, perhaps using the weights received from linear regression as training features for a neural network or gradient boost model. In this way, we might extract some important meta-features of the data, and use more complex models to discover how these features correlate with the dependent variable. In addition, although our models showed no significant difference in effectiveness, it remains important to compare models in order for us to understand the strengths and weaknesses of each. Although both ANN and XGBoost perform similarly on predicting the log error, this does not mean that either will not outperform the other on different datasets and tasks.

In summary, performing comparisons of machine learning methods is an important line of inquiry for researchers and engineers alike. Modern machine learning algorithms are multitudinous and varied. In order to make progress in the field, we must understand the benefits and drawbacks of each method so that we may be able to choose the most appropriate method in real-world application to problems.

References

[1] Anderson, J. (2017, May 4). What's the Best Time of Year for Home Buyers? Retrieved from https://www.zillow.com/research/strategy-best-time-to-buy-15066/

[2] Chakraborty, D., & Elzarka, H. (2018). Advanced machine learning techniques for building performance simulation: A comparative analysis. Journal of Building Performance Simulation, 1-15. doi:10.1080/19401493.2018.1498538

[3] Chen, T., & Guestrin, C. (2016). XGBoost. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD 16, 785-794. doi:10.1145/2939672.2939785

[4] Jain, A. (2016, March 01). Complete Guide to Parameter Tuning in XGBoost (with codes in Python).
Retrieved from https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

[5] Keskar, N. S., & Socher, R. (2017). Improving generalization performance by switching from adam to sgd. arXiv preprint arXiv:1712.07628.

[6] Kohavi, Ron. (2001). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. 14.

[7] Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013, June). Rectifier nonlinearities improve neural network acoustic models. In Proc. icml (Vol. 30, No. 1, p. 3)

[8] Pedregosa, F., Varoquaux, G., Gramforst, A., Michael, V., & Thirion, B. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12. Retrieved from
http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf

[9] Rathee, A. (2017, May 18). Comparing Random Forest, XGBoost and Deep Neural Network.
Retrieved from https://www.kaggle.com/arathee2/random-forest-vs-xgboost-vs-deep-neural-network

[10] Sietsma, J., & Dow, R. J. (1991). Creating artificial neural networks that generalize. Neural Networks, 4(1), 67-79. doi:10.1016/0893-6080(91)90033-2

[11] Stolwijk, A. M., Straatman, H., & Zielhuis, G. A. (1999). Studying seasonality by using sine and cosine functions in regression analysis. Journal of Epidemiology Community Health, 53(4), 235-238. doi:10.1136/jech.53.4.235

[12] Zillow. (2017, September 15). Thanks to More Data and Great Minds, Zestimate Accuracy Is Improving. Retrieved from https://www.zillow.com/blog/zestimate-220693/

Author Contributions

- Abhay - XGBoost algorithm, XGBoostGrid Search, XGBoost writing, Methods writing, Results Discussion
- Anastasia - Data Analysis and Preprocessing writing, XGBoost Grid Search, XGBoost K-fold, XGBoost writing, Proofreading, Literature Review
- Aric - Data Preprocessing, XGBoost writing, ANN Grid Search
- Arthur - Data Preprocessing, ANN algorithm, ANN Grid Search, ANN K-Folds, Linear Regression K-Folds, ANN writing, Results Discussion
- Matthew - ANN K-Folds, ANN Grid Search, Introduction, Abstract, Conclusion, Literature Review, Works Cited & Citations, LATEX Processing, Figures and Tables, Proofreading
- Nir - Data Preprocessing, ANN K-Folds, ANN Grid Search, Linear Regression algorithm, Proofreading, Data Preprocessing writing, Linear Regression writing
- Tyler - Data Preprocessing Algorithm, Data Preprocessing Writing, ANN Grid Search, Proofreading, Final Editing

Source Code

https://github.com/tjgordon/171-Project

Appendix

Table 1: Correlation of Variables with logerror

Variable	Correlation
parcelid	0.015407
bathroomcnt	0.205817
bedroomcnt	0.031638
buildingqualitytypeid	- 0.012987
calculatedbathnbr	0.029330
calculatedfinishedsquarefeet	0.040516
finishedsquarefeet12	0.045921
fips	0.006413
fullbathcnt	0.027133
heatingorsystemtypeid	- 0.006119
latitude	- 0.011726
longitude	0.015876
lotsizesquarefeet	0.011012
propertylandusetypeid	- 0.005679
rawcensustractandblock	0.006333
regionidcity	0.001070
regionidcounty	- 0.016493
regionidzip	- 0.001295
roomcnt	0.014567
unitcnt	0.001999
yearbuilt	0.004861
structuretaxvaluedollarcnt	0.008433
taxvaluedollarcnt	0.003419
landtaxvaluedollarcnt	0.000501
taxamount	0.002199
censustractandblock	0.004613

Table 2: ANN Grid Search

nodes & layers	1	2	3
3	0.0295689444	0.0295258094	0.0296392067
4	0.0324220507	0.0325212127	0.0322290991
6	0.0298774057	0.0306068743	0.0292449751
9	0.0268356852	0.0265301411	0.0265790537
12	0.0283865527	0.0265736871	0.0265266516
15	0.0413647819	0.0299963441	0.0271200290

Table 3: Mean Squared Error Results

Model	Mean Squared Error
27 - 9 - 1 Artificial Neural Network	0.02911
Extreme Gradient Boosting	0.02882
Linear Regression	0.02914

Table 4: XGBoost 10-Folds MSE

Fold	MSE
0	0.0263445385
1	0.0370198205
2	0.0289095392
3	0.0275985979
4	0.0330329622
5	0.0238251491
6	0.0261321983
7	0.0256810375
8	0.0329820879
9	0.0266646879

Table 5: ANN 10-Folds MSE

Fold	MSE
0	0.0264224205
1	0.0375637472
2	0.0287977408
3	0.0276646458
4	0.0331822740
5	0.0250079759
6	0.0265744849
7	0.0258427596
8	0.0331310783
9	0.0269932221

Table 6: Linear Regression 10-Folds MSE

Fold	MSE
0	0.0263873262
1	0.0375021098
2	0.0287370141
3	0.0275500497
4	0.0329660276
5	0.0250509828
6	0.0270090173
7	0.0256722108
8	0.0337827884
9	0.0267456458