Comparison of Supervised and Unsupervised Learning Algorithms on Housing Sale Price Prediction or Clustering

Vincent Chiu, Jorge Marcano, Manoj Karthick Selva Kumar, Sethuraman Annamalai, Ehsan Montazeri

Abstract

We compare different supervised and unsupervised learning algorithms and their performance when applied to the problem of predicting house sale prices. The objective is to show the differences not only between the algorithms themselves, but the effect that different preprocessing steps applied to the data have in the overall performance of each model.

1 Dataset

The dataset used for this project is part of the ”House prices : Advanced regression techniques” Kaggle competition, in which users are challenged to predict house sale prices of residential homes in Ames, Iowa. The dataset contains 79 features that describe aspects of each home, and it can be found at the following link: Kaggle House Prices [dat(2017)]. The competition contains 1460 rows for the training and another 1460 rows for the test set. The information on features can be found through the link in the abstract. In the following sections, we will report the results of various tests done on this dataset using different models.

2 Supervised Learning

In this section we will show the performance of different supervised learning algorithms on the dataset and compare their prediction accuracy. The models that will be studied are neural networks, random forest regression, linear regression, and support vector regression. To compare the performance of our models, we uploaded our predictions to Kaggle. Kaggle outputs a score that is based on Root Mean Squared Logarithmic Error (RMSLE). The RMSLE is calculated as:

$$\epsilon = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i+1)-\log(a_i+1))^2}$$

Where ϵ is the RMSLE value (score), n is the total number of observations in the (public/private) data set, p_i is your prediction, a_i is the actual value for i and $\log(x)$ is the natural logarithm of x. [kag(2017)]

2.1 Neural Networks

We have experimented with applying neural networks to predict the sale price of a given home. We used the mean as the prediction for all the values of the target variable to establish a baseline performance. Then we proceeded to test neural networks of increasing complexity. We tuned various hyper-parameters to achieve the model with the lowest L2 loss.

2.1.1 Preprocessing

We used one-hot encoding on all categorical variables and standardized all numerical features.

2.1.2 Baseline Performance

The mean sale price is $180{,}344.99$. If we predicted the mean price for all houses, we would have a loss of $6{,}041{,}617{,}452.45$. This means that on average, our prediction would be above or below our prediction by about $77{,}727.84$.

2.1.3 Tuning Hyper-parameters

There are several approaches to hyper-parameter tuning, including brute force grid search, randomized search, and the hyper-parameter descent method. In hyper-parameter descent, we first perform a high resolution search to find several promising neural network configurations. Then you proceed to 'fine tune' these promising configurations by making small changes iteratively to each of the hyper-parameters. [Bengio(2012)] Due to limited computational resources, we have elected to first experiment with hyper-parameter descent.
[Bengio(2012)] provides many practical tips for parameter tuning. We explore various methods to lower the loss of our model including adding more nodes per layer, adding more layers, increasing number of epoch iterations, adding dropout, adjusting dropout rate, and using various activation functions. We also try to reduce the time taken to run the training algorithm by using various optimizers and adjusting our learning rate.
Explanation of hyper-parameters:
Initial learning rate: very important, has to be constantly tuned. Typical values for standardized inputs range from 10^{-6} to 1.
Number of Layers: Having more layers allow you to have a more complicated hypothesis space that can describe more complicated relationships between your predictor variables and your target variables. More is usually better in terms of accuracy. However, there is the vanishing gradient problem where the gradients can go to zero when the network have too many hidden layers. Deeper networks also take more computational resources to train and make interpretations of the weights more difficult.
Number of Nodes per Layers: Having too many nodes per layer usually does not lower the accuracy, but makes the training more computationally expensive. It is better to err on the side of having more initially and then slowly decreasing the number of nodes per layer.
Dropout: This is a method for preventing over-fitting. The idea is that you cause some nodes to randomly fail to prevent over-fitting. [Srivastava et al.(2014)Srivastava, Hinton, Krizhevsky, Sutskever, and Salakhutdinov]

2.1.4 Best Model

We have found that our best model has a decreasing number of nodes per layer. Our best model used 3 hidden layers, first layer had 120 nodes, about 50 percent more than the number of features. Second layer had 60 and third layer had 30 nodes. The final layer had one output node. We used a rectified linear unit on all the nodes. We also used the Adam optimizer and decreased the learning rate from the default to 0.0001.

2.1.5 Loss and Hyper-parameters

We obtained a RMSLE score 2.19296 with our best neural network.
Below we compare the loss and hyper-parameters of various neural networks:

Table 1: Neural Net Loss and Hyper-parameters

Model	Mean Square Loss	Square Root of Loss	Hid. Layers	Number Epochs
Mean	6041617452	77727.8422	0	1000
Linear Model	3569206415	59742.83568	1	1000
Linear Model	2505214775	50052.12059	1	5000
3 Hidden Layers	871969714	29529.13331	3	5300

2.1.6 Programming

We used Keras [Chollet et al.(2015)] to program our neural networks.

2.2 Linear Regression

2.2.1 Introduction

Linear regression is used to establish a linear relationship between the input attributes to predict the value of the continuous output variable. Firstly, we start with two types of data preprocessing - centering and log transformation. For modeling, we start with a simple linear regression model and then proceed to its variations of ridge and lasso regression by tuning its respective learning parameters. Finally, we combine both ridge and lasso models to form an ensemble of linear models. Check the table in the section 2.2.7 to view the results.

2.2.2 Preprocessing

Each of the discrete valued attributes are converted into binary valued dummy variables. All missing values in continuous attributes are replaced by the mean of the attribute. Then, each of the continuous valued attributes are applied with two different data processing techniques, standardization and log-transformation. Each of the linear regression models is trained and scored separately to determine which method serves better for the dataset on hand.

2.2.3 Simple Linear Regression Model

A simple linear regression model, taking all input variables without any penalty parameters is trained and tested on the dataset for both the above mentioned data preprocessing.

2.2.4 Ridge Regression Model

The simple linear regression model is extended to form the ridge regression model, where L2 norm penalty is assumed to reduce the model complexity. λ - learning rate hyper parameter is tuned by doing a grid search on a set of values - [1, 5, 10, 0.1, 0.01, 0.001, 0.0001]. The best λ is found to be 0.1.

2.2.5 Lasso Regression Model

The same methodology as the Ridge Regression Model was used, except the L1 norm penalty is assumed to reduce the model complexity. The best λ is found to be 0.0001.

2.2.6 Ensemble - Combination of Ridge and Lasso

The ridge and lasso models trained on the test data with their respective best learning rate are taken. An ensemble model is created by making predictions based on the outputs of both the models as shown below.

- 0.6 of Lasso and 0.4 of Ridge - The predicted value of house prices is computed by taking a weighted average of ridge and lasso where lasso has a weight of 0.6 and ridge has a weight of 0.4
- Equal weighted average - The predicted value of house prices is computed by taking a weighted average of ridge and lasso where both have equal weights
- 0.4 of Lasso and 0.6 of Ridge - The predicted value of house prices is computed by taking a weighted average of ridge and lasso where lasso has a weight of 0.4 and ridge has a weight of 0.6

2.2.7 Linear Models comparison

Table 2 shows the performance of various linear models based on the data preprocessing done on the dataset. All the measures shown are MSLE - Mean Squared Logarithmic Error on the test data submission on Kaggle. From the table above, it can be

Table 2: Performance of linear models

Model	Centering	Log Transformation
Simple Linear Regression	0.41230	0.3464
Ridge Regression	0.14261	0.12698
Lasso Regression	0.16500	0.12373
Lasso 60 — Ridge 40	0.1678	0.12708
Lasso 50 — Ridge 50	0.1609	0.12138
Lasso 40 — Ridge 60	0.1655	0.12160

seen that the ensemble model created by taking an equal weighted average of Lasso and Ridge regression produces the least MSLE on test data, and hence it can be concluded that this is the best linear model for the given dataset.

2.3 Support Vector Regression

2.3.1 Introduction

Support vector regression is a regression variant of SVM. In this section we will be using the Kernel SVR to predict the sale price of houses. Intuitively, support vector machines are made to perform regression by using an $\mathcal{E}$-insensitive loss function which does not penalize any point that is within $\mathcal{E}$ units from the margin. This gives SVR the ability to be flexible, making it a convex optimization problem that can be solved as a quadratic programming problem.

2.3.2 Preprocessing

Initial exploratory analysis revealed that of the total 81 features, six features contained moxre than 85% nulls, and were hence removed. After developing an intuition how each variable varied according to the response variable, the correlations between them showed that three sets of features, namely (TotalRmsAbvGround, GrLivArea), (TotalBsmtSF, 1stFlrSF), and (GarageCars, GarageArea) had high correlation. This makes sense because the number of rooms above ground is proportional to total living area. Since the basement and the 1st floor have similar floor plans, the two area variables might have high correlation. The number of cars that can fit in the garage is dependent on the area. Hence one variable was used and the others were dropped. The analysis also proved overall quality and living area had highest correlation with the response, which was in line with our intuition. The missing numerical values were replaced with the mean and standardized, while categorical variables were converted to binary.

2.3.3 Parameter Tuning

The best set of hyper-parameters was found using 10-fold cross validation on the training data.
- *C*: This refers to the penalty parameter which controls how the margin is. If the value of C is large, the model will try to use a narrower margin which might cause the model to over-fit by trying to accomodate noise or outliers. A smaller C value allows the model to have a wider margin, meaning the error points aren't penalized as much.
- $\mathcal{E}$: Refers to the epsilon-tube within which any point does not contribute towards the loss.
- *kernel*: The kernel makes calculation at higher dimensions much easier,allowing for linear separation at higher dimensions.

The best hyper-parameters found were **C=100** and **kernel=linear**. Having $\mathcal{E}$ anywhere from 2 to 4 did not change the error much.

2.3.4 Evaluation

An RMSLE (Root mean squared logarithmic error) of 0.14277 was obtained on submission to Kaggle. RMSLE was used because taking logarithm on the actual and resonse values makes sure errors in predicting higher house prices have the same impact when predicting lower house prices.

2.4 Principal Component Regression and Partial Least squares

2.4.1 Analysis

In PCR (Principal component regression), we first perform PCA to find the most important features and use them to fit a linear regression model. PCA on the training data showed that just 53 features accounted for 90% of the variance. These features were then used to fit a linear regression model. In PLS, each feature in tandem with the response variable is used separately to fit linear regression. PLS goes beyond the 1st principal component and finds the actual component that is related to the response. Cross-validation using the training data revealed that using only three features produced the lowest RMSLE.

2.4.2 Results

Using 53 components, PCR produced an RMSLE of 0.16687 on Kaggle. While PLS used only three components, it resulted in an RMSLE of 0.21333 which was astounding.

2.5 Random Forests

2.5.1 Introduction

Random forests are powerful classifiers and regressors, are resistant to noise, and achieve a lower variance compared to a single decision tree [Liaw et al.(2002)Liaw, Wiener, et al.], so we decided to explore them as well in this study.
The method uses an ensemble of N regression trees that are independently trained but used together for prediction, where their decisions are averaged and presented as the output. The dataset used for training each tree is generated from the original dataset with the use of bagging, where M samples are randomly picked for each tree, while remaining in the pool to be possibly used for the other trees as well. To make these generated datasets uncorrelated, each tree uses a subset of randomly selected features, rather than using all of them.

2.5.2 Preprocessing

We used *RandomForestRegressor* from *scikit-learn* to construct and implement our model in Python and trained it using the dataset. Decision trees are well capable of handling categorical features, but *scikit-learn* can only handle numerical values, so all categorical values were converted into dummy variables using binary *pandas.getdummies()*. Similar to the other methods already discussed, missing values were replaces with mean values for each feature and the numerical values were standardized to have zero mean and unit variance.

2.5.3 Hyper-Parameter Tuning

Using 10-fold cross validation, the following hyper parameters were found to be best using *GridSearchCV* from *sciki-learn*.

- max_feature = 0.5 This parameter indicates that 50% of the features were randomly used for training each tree.
- max_depth = 20 This parameter sets the maximum depth of each tree to 20 levels.
- n_estimators = 100 This parameters assigns 100 trees to the forest.

The effect of these three hyper parameters on the validation set in shown in Figure 1, using the R^2 metric.

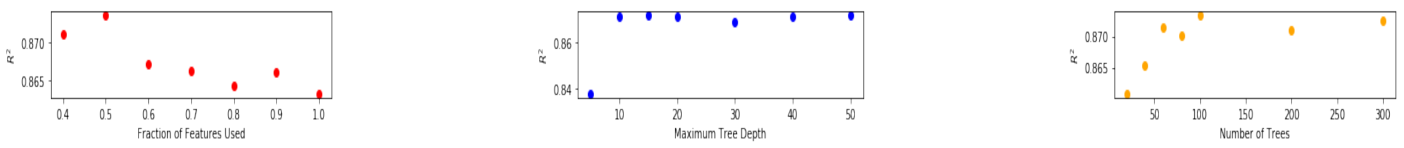

Figure 1: Effect of different hyper-parameters

2.5.4 Evaluation

The trained model was used to make predictions on the test set. The results were uploaded to Kaggle and an RMSLE score of 0.14396 was obtained. We also ran the algorithm after standardizing the data, which resulted in a slightly poorer score of 0.14538. We observe that standardization does not make a negligible difference in the performance of random forest. This is expected, since decision trees use information gain as their error metric, which is invariant to scaling
The ten most informative features are shown below in Figure 2, where the mutual information values between the features and the sale price variable have been plotted. It can be seen that many of these features are also very highly correlated with sale price .

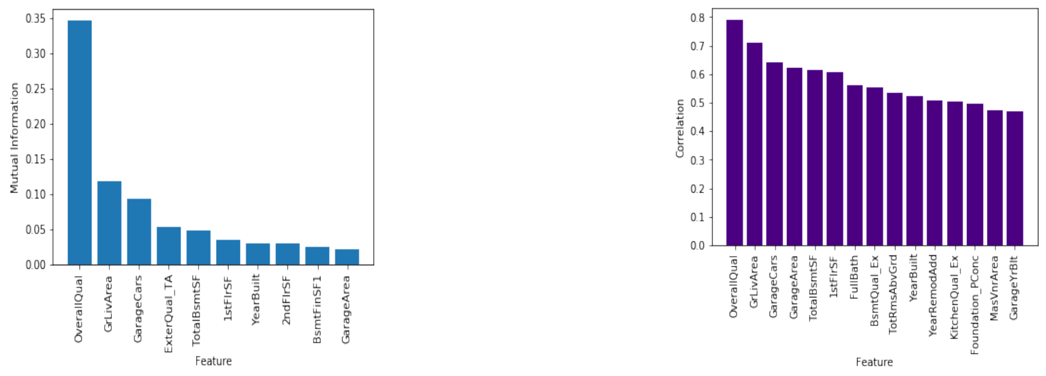

Figure 2: Most informative and most highly correlated features with respect to Sale Price

3 Unsupervised Learning

All the tests shown in this section were done by applying the algorithms using 3 clusters. It makes sense to divide the input data into "low", "medium" and "high" price ranges. Additionally each algorithm uses the same cluster initialization method named "Cao" which seems to be the most modern approach. [Bishnu and Bhattacherjee(2013)].
We have decided to use two clustering algorithms that are able to handle categorical data. The first algorithm is K-Modes which is a variation of K-means that works only with categorical data. The second algorithm is K-prototypes which applies K-Means to numerical features and K-Modes to the categorical features and then combines the results of the two into the final clustering. These two algorithms are explained in [Huang(1997)].
We used the kmodes python package for the implementation of both K-Modes and K-Prototypes.

3.1 K-modes algorithm

We show 2 test results. One for the full data (except numerical features) and another by removing the examples of the data that were considered ”medium” and ”high” prices since these created a very noticeable imbalance in the data set. The first one is to show the results of the algorithm without manipulating the data, and the second are the best results obtained from the different tests applied.
We first cluster using the full dataset without the numerical features. We then cluster again with a subset of the data that excludes ’medium’ and ’high’ prices because these data points created a class imbalance.
For the first test we have used all the data available. The results are shown in the table 3.
It can be seen that most of the ”low” prices are grouped under Cluster 2. Most ”medium” and ”high” priced houses are grouped under Cluster 1. A low number of elements is grouped under Cluster 3. We would like to check if this is due to the large imbalance in the data, so we will try removing the 158 ”medium” and ”high” examples and apply the process again. The results of this can be seen in table 4.

Table 3: K-modes with all examples

Price Range	Cluster 1	Cluster 2	Cluster 3
Low	512	697	93
Medium	140	10	0
High	8	0	0

Table 4: K-modes with only low prices

Price Range	Cluster 1	Cluster 2	Cluster 3
Low	9	124	77
Medium	229	149	415
High	237	13	59

In this case, the data is clustered in a more sensible way. Cluster 2 contains most of the elements in the ”low” range, Cluster 3 contains most elements of the ”medium” range, and Cluster 1 has most of the elements in the ”high” range. That said, Cluster 2 and 1 both have a large number of elements of the two different categories.

3.2 K-prototypes algorithm

As the results we obtained from the K-modes algorithm were unexpected, we would like to explore what happens when we use both numerical and categorical features to cluster the data. We will now use no preprocessing and use all the data for the clustering. The results of this test are shown in table 5. It can be seen that most of the data is grouped under Cluster 1, while the other clusters are left with very few elements. This could be due to the difference in scale between the different numerical features or the imbalance in the data prices.
Next we would like to show the effects of normalizing numerical features and removing the imbalances in the data. The results for this test can be seen in table 6.

Table 5: K-Prototypes with all the data

Price Range	Cluster 1	Cluster 2	Cluster 3
Low	1262	39	1
Medium	140	7	3
High	5	3	0

Table 6: K-Prototypes with only low prices and numerical features normalized

Price Range	Cluster 1	Cluster 2	Cluster 3
Low	1	123	86
Medium	216	146	431
High	241	17	41

The results of this test seem to be strikingly similar to results of K-modes in table 4. Just like in that table, most ”low” elements are grouped under Cluster 2, most ”medium” ones in Cluster 3 and most ”high” ones in Cluster 1. Both Cluster 1 and 2 have a significant amount of elements from other price categories.

3.3 Model Comparison

Both K-modes and K-prototypes failed to generate clusters that reflected different sale price ranges. When clustering on only low prices, both algorithms produced similar clusters because normalizing the numerical features lowers their importance relative to the categorical features. Cluster 1 contained medium and high prices and Cluster 2 contained low and medium prices and Cluster 3 contained most of the medium prices.

4 Comparing Supervised and Unsupervised Learning

We compare the results we obtained from unsupervised learning and from our best supervised learning model, Lasso and Ridge regression ensemble. We append the predictions from ensemble linear regression as the prices for the test data. We

merge the training and test data with predicted prices into one new dataset. We then cluster on this new merged dataset. If the results have the same behavior as the best test of K-prototypes with only the training set, this would mean that the dataset is consistent. It would also mean that the clustering process was done correctly and that the prediction was good.
These results will be shown in a table as in the last section and will be based on the same preprocessing steps as the best test of the K-prototypes algorithm. It is important to note that the imbalance in the price ranges is maintained even after adding the predicted data. After doing so, the data set is left with 2612 ”low” prices, 294 ”medium” prices and 13 ”high” prices, which justifies the process of dropping the 294 ”medium” and 13 ”high” prices in order to produce a more balanced data set.
The results of this test can be seen in the table 7.

With these results, we observed that after adding the predicted data to the original training data, the clustering process and

Table 7: K-Prototypes with only Low prices and Numerical features normalized and both train and test sets

Price Range	Cluster 1	Cluster 2	Cluster 3
Low	2	224	150
Medium	353	292	768
High	450	21	69

results are consistent with the ones seen in table 6. Aside from Cluster 3 being predominantly filled with ”medium” prices, the other 2 clusters have a balance of ”low” and ”medium” prices or ”medium” and ”high” just like in previous tests of K-prototypes. High prices have a high probability of landing in Cluster 1, medium ones in Cluster 3 and low ones in Cluster 2. This confirms that the data predicted from the supervised learning process is consistent with the original training data and that the clustering process is representing the data correctly.

5 Conclusion

We have compared 4 supervised learning methods. We have also compared our best supervised learning methods to our best unsupervised learning method. In the case of linear regression, using logarithmic transformation gave better results than

Table 8: Comparison of all methods

Model	Kaggle Score
Linear Regression Ensemble	0.12138
Support Vector Regression	0.14277
Principal Component Regression	0.16687
Partial Least Squares	0.21333
Random Forest	0.14396
Neural Network	2.19296

standardization, and the use of an ensemble of Lasso and Ridge was the best choice. For random forests, it was observed that standardization did not have a significant effect on the performance. As for the clustering techniques, it appears that standardizing the numerical features and removing imbalances in the data set produces clusters that make more sense for the price categories we described. For neural networks, standardizing the features did not have much effect on the loss.

6 Contributions

Each member of the team worked on a different model, including preprocessing steps, coding of the model, validating and testing it as well as writing the section of the report associated with that model. Everyone worked on the final editing of the report as well as the conclusion, abstract and model comparison sections.

- Vincent Chiu : Neural Networks
- Jorge Marcano : Unsupervised Learning
- Manoj Karthick Selva Kumar : Support Vector Regression, Principal Components Regression, Partial Least Squares
- Sethuraman Annamalai: Simple Linear Regression, Ridge and Lasso Regressions, Ensemble of Ridge and Lasso
- Ehsan Montazeri : Random Forests

References

[dat(2017)] House prices: Advanced regression techniques, 2017. URL `https://www.kaggle.com/c/house-prices-advanced-regression-techniques`.

[kag(2017)] The home of data science and machine learning, 2017. URL `https://www.kaggle.com/wiki/RootMeanSquaredLogarithmicError`.

[Bengio(2012)] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. *ArXiv e-prints*, June 2012.

[Bishnu and Bhattacherjee(2013)] Partha Sarathi Bishnu and Vandana Bhattacherjee. *A Modified K-Modes Clustering Algorithm*, pages 60–66. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-45062-4. doi: 10.1007/978-3-642-45062-4_7. URL `https://doi.org/10.1007/978-3-642-45062-4_7`.

[Chollet et al.(2015)] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[Huang(1997)] Zhexue Huang. Clustering large data sets with mixed numeric and categorical values. In *In The First Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 21–34, 1997.

[Liaw et al.(2002)Liaw, Wiener, et al.] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[Srivastava et al.(2014)Srivastava, Hinton, Krizhevsky, Sutskever, and Salakhutdinov] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=2627435.2670313`.