HOUSING PRICE PREDICTIONS

Final Project

Author:

John Herbert john.herbert@trojans.dsu.edu

INFS 768 - Predictive Analytics for Decision Making

Dakota State University

1. Task

The task I chose is to use an advanced regression model in order to predict housing prices in Ames, Iowa. There are several challenges in this project. First, there are a large number of variables (79 excluding the target) which will need to be cleaned of missing values and normalized before fitting the model. In addition, I will use statistical methods to reduce the number of variables used in the models to avoid overfitting and reduce computational complexity. Also, there are on 1,460 records in the data set, so how to split the data into train/validate/test subsets will be difficult to maximize the record I use to train the models while have enough records to do a thorough test of My final model. Finally, I will have to tune My models using statistical optimization methods to make sure that I are not overfitting the model (which will consequently make the test preform poorly) or under fit it (too generalized of a model that has large variances).

2. Approach

My approach for this project is to use 2 regression models: a simple linear regression model and deep neural network. I used a linear regression as a baseline model to see if My neural network can preform better than the baseline. My approach is to prepare the data for analysis and how I will validate the models is explained in detail in sections 3 and 4 of this document. In order to validate My models, I will use 10 fold cross validation. Below are explanations of each model I will use including the equations and python functions to build and tune them.

Linear Regression

The first model I will use will be the linear regression model. Linear regression models are basic and show the relationship of the dependent variable to the independent variable. Since linear regression models are a simple model, there are no additional parameters. The model will be computed based on the following equation:

$$Y \quad = bX + a$$

$$b \;=\; r\left({}^{s_y}\!/_{s_x}\right) \qquad\qquad a \;=\; y \;-\; bx$$

Where *a* is the y-intercept, *b* is the slope of the regression line, *x* is the x intercept, y^ is the predicted score, s is the standard deviation, and *r* is the correlation betIen x and y.

This process can be done in Python by using the *LinearRegression* function in the *sklearn* package. Since the equation is fairly straight forward, there are no parameters that will need adjusting, only scaling before it its fit with the model.

Deep Neural Network

I implemented a multi-layer perceptron (MLP) model that will train using backpropagation with a ‘lbfgs’ solver that optimizes the Iights based on a quasi-Newton method. I will also use an alpha parameter to help avoid overfitting of the model which will penalize Iights with large magnitudes. The model iterates through the training set by randomly shuffles and selects a sample to perform the iteration on. Then updates the Iight of each training example based on this equation:

$$E(w,b) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(w)$$

Where L is the loss function that measures model fit or misfit and R is the penalty term that penalizes model complexity.

$\alpha > 0$ or alpha is a non-negative hyperparameter that controls the regularization strength

There are different choices of the loss function. I are choosing to use a *Perceptron* regressor, but may use different choices to see if I get better validation results. It is equates as:

$$L(y_i, f(x_i)) = \max(0, -y_i f(x_i)).$$

The function in python I used is the *MLPRegressor* function which optimizes the squared-loss using a Iight optimization method. There is also the *activation* parameter, which fits the line of the decision boundary. I ran through *identity* (no-op activation, linear bottleneck), *tanh* (tan function), and *relu* (rectified linear unit function). The ‘tanh’ function performed the best for My model. I also set the *alpha* to different level betIen 0.0001 and 0.9 to find the optimal level. *Learning rate* parameter can either be constant, invscaling, or adaptive. In My model, an adaptive learning rate performed the best. Finally, I set the minimum hidden layers to 1 and the max to 3. The neurons within each layer was set to a minimum of 1 and the maximum of 100.

When building the model in *sklearn*, I will use the *RandomizedSearchCV* function to find the optimal level for these parameters. This will randomly search on each of the defined hyperparameters of the model. It then implements a fit and score method with cross validation to determine the best parameter for the model.

3. Dataset

The dataset I are using is the *House Prices: Advanced Regression Techniques* from the Kaggle Prediction Competitions. This data set has 1,460 rows and 81 variables in the training set:

```
The shape of the dataset is: (1460, 81)
```

Of the 81 variables, one is an ID, which will be dropped, one is the target variable: *SalePrice*. Of the 80 variables, there are 37 numeric variables, and 43 categorical. Holver, there are many categorical, as not all of them are continuous, but many of them are ranked. This will be explained in further detail in the next section.

```
The number of numerical variables (including target): 37
The Number of categorical variables: 43
```

As mentioned above, I will create 2 models: as simple linear multiple regression model as a base model, and a neural network model to see if I can improve the predictive results. I will use root mean squared error (RMSE) as a metric for comparison betIen the 2 models.

Since the data set has a generally small number of records, I will be using 10-fold cross validation instead of splitting the data set out into a validation subset. The competition has a test data set included, holver the target variable is not included, therefore I cannot use it. I will also do a 90:10 split (90% training and 10% test). Since the data set is so small, I will need as much of the data to train the model as possible.

The cross validation will be used to tune each of the models to yield the lolst RMSE and highest R squared it can. I will use 10k fold cross validation. I tested different K values, holver they result was roughly the same, so the 10 fold method seems like a good fit for comparison purposes.

4. Results

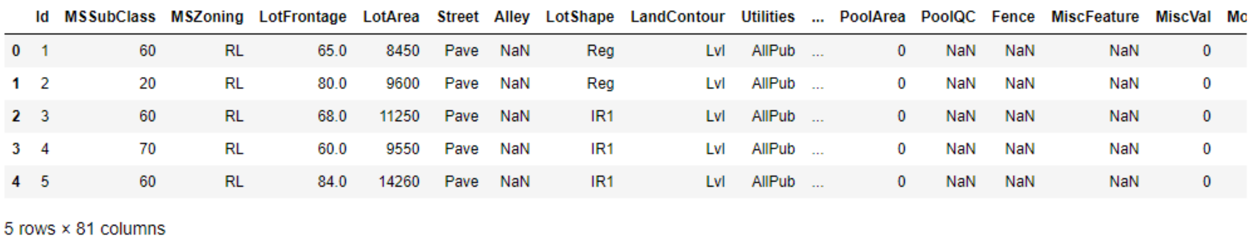
For the project, I have done initial preprocessing steps to import the dataset, handled missing values, split the data into train and test subsets, handled outliers, handle skeld continuous variables with log transformation, transform string categorical variables to dummy variables, standardize each of the continuous variables using a scaler, reduced the amount of variables to only include statistically significant ones, and created the models and tested them. A detailed walkthrough of this process is below.

Importing Packages for Preprocessing and Feature Selection

Imported all necessary packages and functions.

Importing the Data Set

I downloaded the data set from data section of the House Prices: Advanced Regression Techniques Kaggle Competition and imported the *train.csv* into a data frame in python. Below is a snapshot of the first 5 rows of the dataset.



	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	Mo
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	

5 rows × 81 columns

Splitting the Dataset Into Train/Test

```
hd_train,hd_test = train_test_split(hd,test_size=0.10, random_state=42)
```

```
hd_train.shape, hd_test.shape, type(hd_train)
```

```
((1314, 81), (146, 81), pandas.core.frame.DataFrame)
```

Handling Missing Values

The first step is to examine which variables have missing values in the training and test datasets, what data type are they (so I can determine if true missing values will be replaced with the mean or if categorical variables should have a new category for missing values), and what percentage of values are missing.

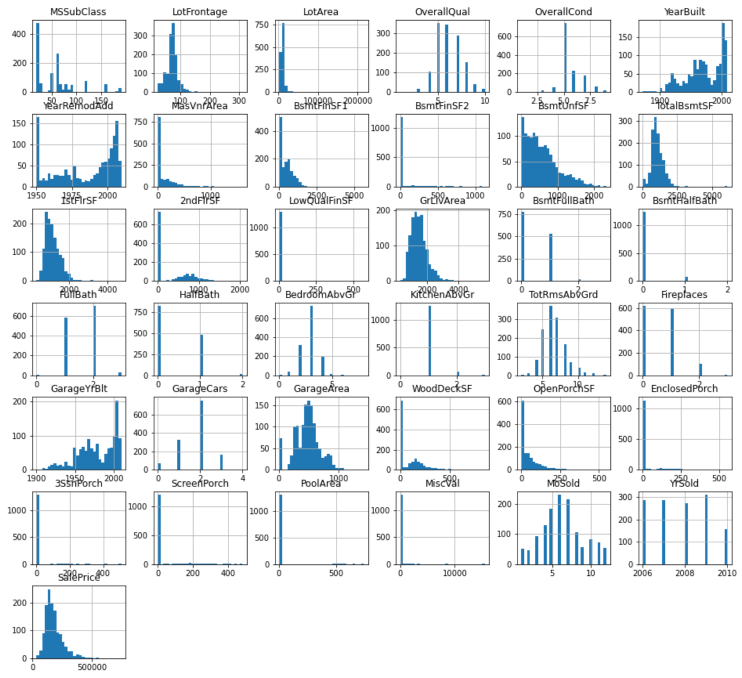
	tot_null	perc_null	class
PoolQC	1453	0.995205	object
MiscFeature	1406	0.963014	object
Alley	1369	0.937671	object
Fence	1179	0.807534	object
FireplaceQu	690	0.472603	object
LotFrontage	259	0.177397	float64
GarageType	81	0.055479	object
GarageYrBlt	81	0.055479	float64
GarageFinish	81	0.055479	object
GarageQual	81	0.055479	object
GarageCond	81	0.055479	object
BsmtExposure	38	0.026027	object
BsmtFinType2	38	0.026027	object
BsmtFinType1	37	0.025342	object
BsmtCond	37	0.025342	object
BsmtQual	37	0.025342	object
MasVnrArea	8	0.005479	float64
MasVnrType	8	0.005479	object
Electrical	1	0.000685	object

According to this table, *PoolQC, MiscFeature, Alley*, and *Fence* all have 80% or more of their variables missing, so they will be dropped from the table. Many of the missing variable, according to the dataset documentation, are a category in itself, for example if any of the garage variables are null that means that the house does not have a garage. Therefore, these categorical variables Ire replaced with a “No_g”, or “no_b”, etc.

The numeric missing values Ire replaced with the variable’s mean specifically from the neighborhood it was located in, so it will not impact the model fitting. I kept all the other information in the record. After these steps, I can see that there are no missing values in My dataset.

```
The number of missing values in dataset: 0
```

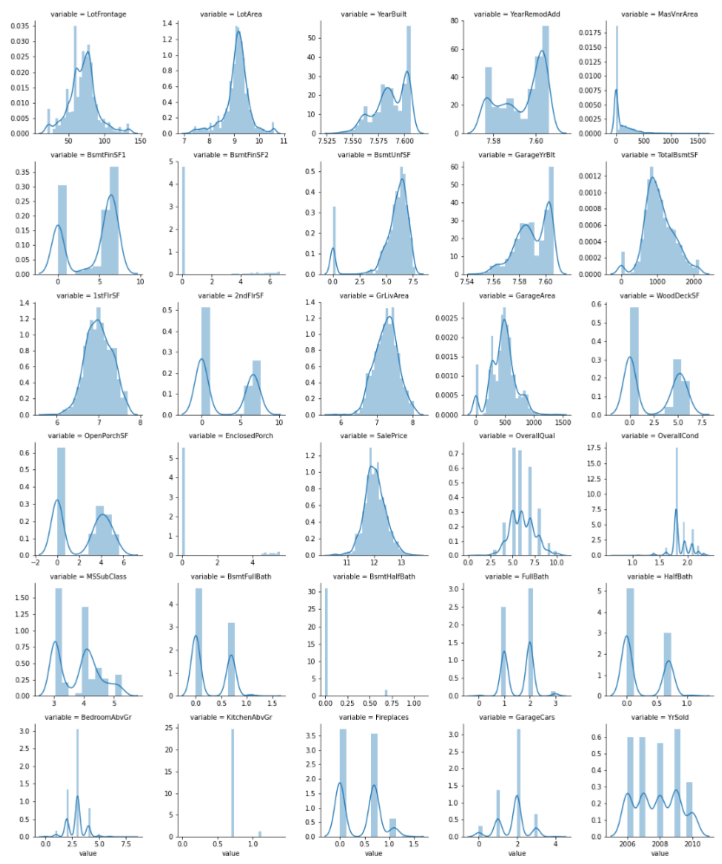
According to histograms of the of the numeric variables, there are variables that are right skeId, have large outliers, and are numbered categorically.



Outliers and Log Transformation Numerical Variables

For variables that have large outliers to the right, they Ire converted to the 99th percentile. In addition, the variables *'LowQualFinSF','3SsnPorch','ScreenPorch','PoolArea', and 'MiscVal'* have a large majority of values equal to zero, so they Ire removed as they will not add value to My predictive model.

To determine if a variable needs log transformation, I ran a skewness test and any variable with greater than 0.5 (including the target) got log transformation. Any variables that Ire categorical Ire either left in and treated as continuous if they had ranking characteristics (ex. the number of bathrooms). Below is a result of the transformed data:



Creating Dummy Variables

Created dummy variables for all categorical features, including removing the original object variables and excluding one category in order to avoid the dummy variable trap. Since most of the numeric categorical variables have a ranking, they did not need to be transformed. Holver, 'MoSold' was set a categorical variable and converted to dummy variables for each month (excluding 1) since there is no ranking involved in the month it was sold.

Feature Selection

I used the *RobustScaler* function on all numeric values to standardize them and fit it to the training dataset. This function scales the data according to the interquartile range. Then I used the *XGBRegressor* function to rank the features importance in order to reduce the amount of variables since after the dummy variables Ire created I now had over 250. This will help with overfitting the model, improve computational performance, and scale down the non-correlated variables. I then used recursive feature elimination and cross-validated selection (RFECV) to reduce the features based on RMSE. The RFECV kept 50 variables that are listed below.

```
The number of selected inputs is: 50
```

The features the model decided to keep based on RMSE are:

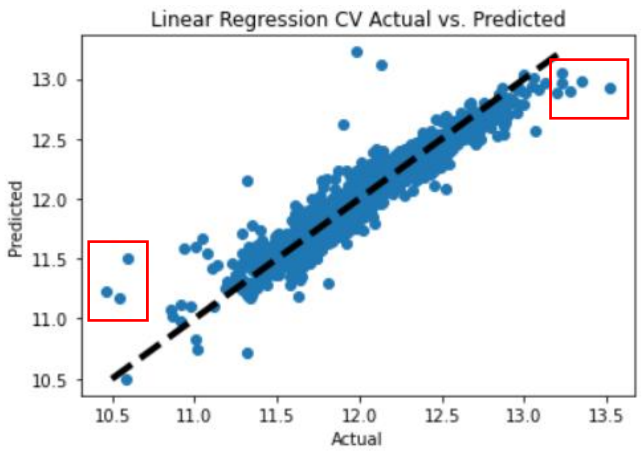
```
['OverallQual' 'OverallCond' 'YearBuilt' 'YearRemodAdd' 'BsmtFinSF1'
 'TotalBsmtSF' '1stFlrSF' '2ndFlrSF' 'GrLivArea' 'FullBath' 'BedroomAbvGr'
 'KitchenAbvGr' 'Fireplaces' 'GarageYrBlt' 'GarageCars' 'GarageArea'
 'OpenPorchSF' 'EnclosedPorch' 'Street_Pave' 'LotConfig_FR3'
 'Neighborhood_BrkSide' 'Neighborhood_Crawfor' 'Neighborhood_Edwards'
 'Neighborhood_NAmes' 'Neighborhood_OldTown' 'Neighborhood_StoneBr'
 'Condition1_Norm' 'HouseStyle_SLvl' 'Exterior1st_BrkComm'
 'Exterior1st_BrkFace' 'ExterCond_Fa' 'BsmtQual_Gd' 'BsmtExposure_Gd'
 'BsmtExposure_No' 'Heating_GasA' 'CentralAir_Y' 'Functional_Maj2'
 'Functional_Sev' 'Functional_Typ' 'GarageType_Attchd'
 'GarageType_Basment' 'GarageQual_Fa' 'SaleType_ConLD' 'SaleType_New'
 'MSZoning_RL' 'MSZoning_RM' 'SaleCondition_Family' 'SaleCondition_Normal'
 'MoSold_2' 'MoSold_12']
```

RFECV is a feature selection method that fits a model and removes the Iakest features. Features are ranked by the model's coefficient attributes and will recursively eliminate a small number of features per each iteration of the function. It also attempts to eliminate dependencies and collinearity in the model.

Model 1: Linear Regression

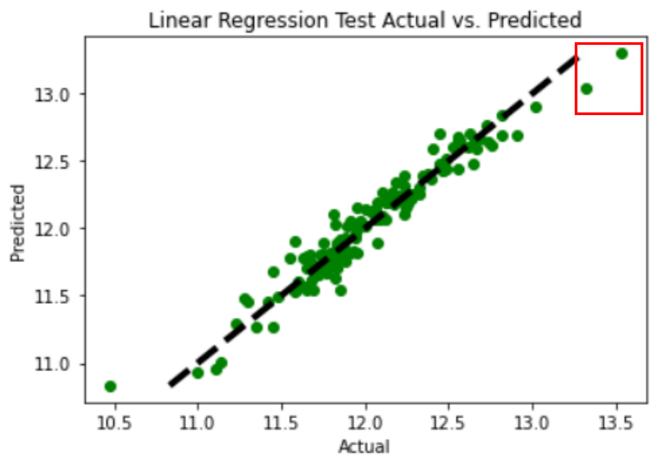
To set a baseline for out neural network model, I built a linear regression model using the *LinearRegression* function in the *sklearn* package. I will also use a 10 fold cross validation RMSE to compare each of the models to determine which one is a better fit. There are no hyper parameters in the linear regression model, so I will not need to adjust anything. Below is the model's cross validated RMSE and R Squared scores as Ill as a visual of the fitted model on the training dataset. I can see that most of the values are fitted to the linear model, hoIver there are a number of outliers, especially at the higher and loIr price range (as highlighted by the red box below).

```
Model 1 Cross Validation RMSE: 0.1336
Model 1 Cross Validation R Squared: 0.8807
```



When fitting the model to the test dataset, I get a loIr RMSE and higher R Squared. Results are below. I can see a better model fit, hoIver there are still a number of outliers in the higher target values, but not as much at the loIr values. The scores may have improved due to the size of the training dataset being too small.

```
Model 1 RMSE on Test: 0.0127
Model 1 R-Squared on Test: 0.9352
```



Model 2: Neural Network

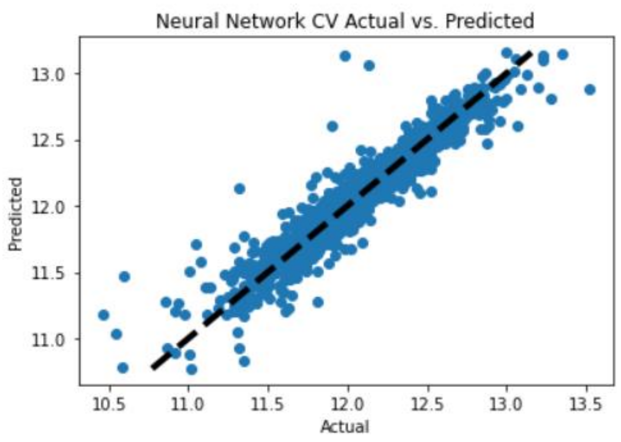
I used a Multilayer Perceptron Neural Network to compare to My linear regression model. As mentioned above, I optimized the number of hidden layers, the number of neurons within each hidden layer, the activation function, the solver type, the alpha, and the learning rate. Based on My randomized cross validation search, below are the parameters it deemed to be the best fit.

```
{'activation': 'tanh',
 'alpha': 0.8235637079894027,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (53, 94),
 'learning_rate': 'adaptive',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 200,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': 42,
 'shuffle': True,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
```

From the parameters I optimized, it chose an activation function of tanh, an alpha of ~0.823, 2 hidden layers with 53 and 94 neurons in each, respectfully, an adaptive learning rate, and a lbfgs solver.

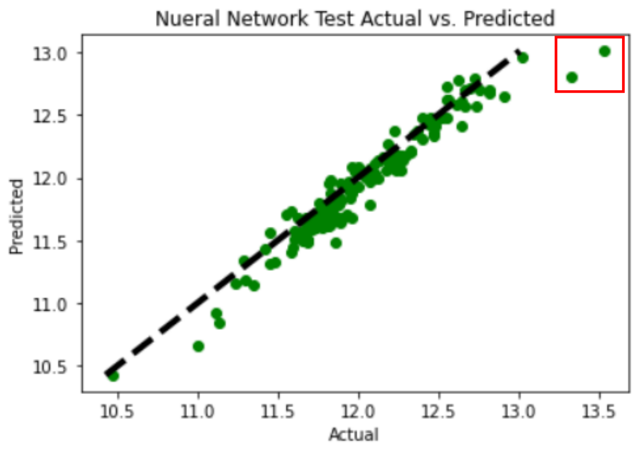
Fitting the neural network with these tuned parameters, I can see the results below. I see a marginal improvement from the linear regression on both RMSE and R Squared with cross validation on the training set. Holver, there are a number of outliers at both the low and high end of the values, but it seems a marginally better fit than the linear regression model.

```
Model 2 Cross Validation RMSE: 0.1307
Model 2 Cross Validation R Squared: 0.8866
```



When fitting the test data, I actually see a worse result on both scoring methods. In addition, the higher outliers appear to be more pronounced.

```
Model 2 RMSE on Test: 0.0188
Model 2 R-Squared on Test: 0.9043
```



Summary

In conclusion, the neural network model performs better with validation than the linear regression model, but only marginally. However, the linear regression performs better than the neural network on the test set. This is summarized in the table below.

	Model	RMSE	R Squared
0	Linear Regression CV	0.133631	0.880727
1	Linear Regression Test	0.012704	0.935176
2	Neural Network CV	0.130706	0.886569
3	Neural Network Test	0.018758	0.904284

Since the neural network did not perform better than the linear regression, or only marginally so, it may not be the best model for predictions. Testing different models, as Ill as testing more data would be advisable.