
Housing Prices: Advance Regression Techniques

Name	Shubham Gupta
Student No.	a1787223
Class	Ms. Data Science
Department	COMP SCI
Email	shubham.gupta@student.adelaide.edu.au
Date	August 26, 2020



THE UNIVERSITY
of ADELAIDE

Abstract

The best rank achieved on Kaggle for this assignment was 2083 with an RMSE score of almost 0.135. Extensive data pre-processing was done to achieve the results. From dropping features to reducing the skewness of the target feature. Best performing models were chosen with cross validation. Hyper-parameter tuning was done on best models to achieve even greater performance. There is still a lot of improvement that can be achieved for even better results.

Introduction

Housing Price: Advanced Regression Techniques is one of the most popular data-sets on with over 5000 participation's. It is an expanded version of the often cited Boston Housing dataset. Created by Dean De Cock, it describes almost every aspect of residential homes in Ames, Iowa, with over 79 features. In this project I will analyse what influences the sale price of a house and aim to predict the correct SalePrice of the respective house. Our aim will be to achieve the lowest possible RMSE score and best possible rank.

DataSet

The data is provided to us by two files named train.csv and test.csv where our target values "SalePrice" are only given in train. Our aim is to clean the data and predict the "SalePrice" for test file. Initial analysis showed that both train and test have missing values, so we have to fix the missing values in both dataset. Looking at the training data we see that it has a shape of (1460,81) with 'ID' and 'SalePrice' included as features. Test data has a shape of (1459,80) with all the features as training data except 'SalePrice', which we have to predict from our machine learning model. First thing I did was to drop the 'ID' feature from both dataset as it is not useful (Figure 1).

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Feedr
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	Norm
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	Norm
...
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Gilbert	Norm
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	NWAmes	Norm
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Crawfor	Norm
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	NAmes	Norm
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Edwards	Norm

1460 rows × 81 columns

Figure 1: Data Visualization

Data Cleaning

After reading the data description on kaggle, and removing some of the given outliers from the training data, we are left with (1455,80). As both training and test have missing values we will combine the two so that we don't have to clean these two datasets separately. Because we have to separated these again I gave training and test data tags of 0 or 1, with 0 meaning not from training and 1 meaning from training data. The combined data has shape of (2914, 81) see Figure 2, which makes sense as the two data sets have exactly the same features except the SalePrice. For SalePrice the test data is filled automatically with NaN values as soon as we concatenated.

Few things which can be observed from seeing the type of data filled in each column, we see that 'MSSubClass', 'YrSold', 'MoSold' are given as integer type, and based on the description of the features given, these should be categorical. Our machine learning model might wrongly assume a correlation as MSSubClass-60 is not better or worse than MSSubClass-20. Same goes for year sold and month sold as it might give wrong correlation to our machine learning model. We are not changing year built and year remodeled as these might have correlation to SalePrice as old house can be either expensive or cheaper given their condition or old remodeled houses might be much more expensive.

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1
0	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm
1	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Feedr
2	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm
3	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	Norm
4	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	Norm
...
1454	160	RM	21.0	1936	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	MeadowV	Norm
1455	160	RM	21.0	1894	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	MeadowV	Norm
1456	20	RL	160.0	20000	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Mitchel	Norm
1457	85	RL	62.0	10441	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Mitchel	Norm
1458	60	RL	74.0	9627	Pave	NaN	Reg	Lvl	AllPub	Inside	Mod	Mitchel	Norm

2914 rows x 81 columns

Figure 2: Combined Data

	Feature	Percentage_of_Missing_Values
71	PoolQC	99.656829
73	MiscFeature	96.396706
5	Alley	93.239533
72	Fence	80.507893
79	SalePrice	50.068634
56	FireplaceQu	48.661633
2	LotFrontage	16.678106

Figure 3: Percentage of Missing Values

After checking for missing data (see Figure 3), we see that some features have more than 80% of their values as missing. Filling these values in any way might introduce a bias in our results. For example if we give 'PoolQC' as NA or None (according to data description), the result will be a heavy tailed feature which is not at all useful. SalePrice is there because our test set does not have SalePrice values, so that can be ignored. According to [6] we can remove the features with this many missing values. So we will just remove the top four features which have more than 80% of their values as missing. After dropping these columns we are left with a shape of (2914, 77).

To better visualize the data, we will separate the numerical data and categorical data. For example description() function works different for numerical data and categorical data. Therefore we separated the two types of data. The shape of our numerical data is (2914, 35) and of categorical is (2914, 42), which clearly indicates that a lot of features are categorical than numerical.

FireplaceQu	1418
GarageCond	158
GarageQual	158
GarageFinish	158
GarageType	156
BsmtCond	82
BsmtExposure	82
BsmtQual	81
BsmtFinType2	80
BsmtFinType1	79
MasVnrType	24
MSZoning	4
Utilities	2
Functional	2
KitchenQual	1
Exterior2nd	1
Electrical	1
Exterior1st	1
SaleType	1

(a) Categorical Features Missing Data

SalePrice	1459
LotFrontage	486
GarageYrBlt	158
MasVnrArea	23
BsmtHalfBath	2
BsmtFullBath	2
BsmtFinSF1	1
GarageArea	1
BsmtFinSF2	1
BsmtUnfSF	1
TotalBsmtSF	1
GarageCars	1

(b) Numerical Features Missing Data

Figure 4: Features with Missing Values

Looking at the Figure 4 we can see that total of 19 features are missing values in categorical type and total of 11 features missing values in numerical type (ignoring the SalePrice). I analyzed each feature individually and filled the missing values accordingly. For example, 'FireplaceQu' has almost 48% of their data as missing, seeing Figure 5 we can see how different houses are categorised on fireplace quality with 'TA' being average and 'Gd' being good. Filling the missing data (almost 48%) with mode i.e. 'Gd' may lead to over-fitting as and to avoid that we fill the missing data with 'TA'. Similarly in figure 4a we can see that garage related features all the same number of missing values. Suggesting that there may be a correlation and there might not be any garage (based on text description). Therefore we filled these with 'None' meaning no garage. All the other categorical features were similarly filled and a clean categorical data was achieved with no missing values.

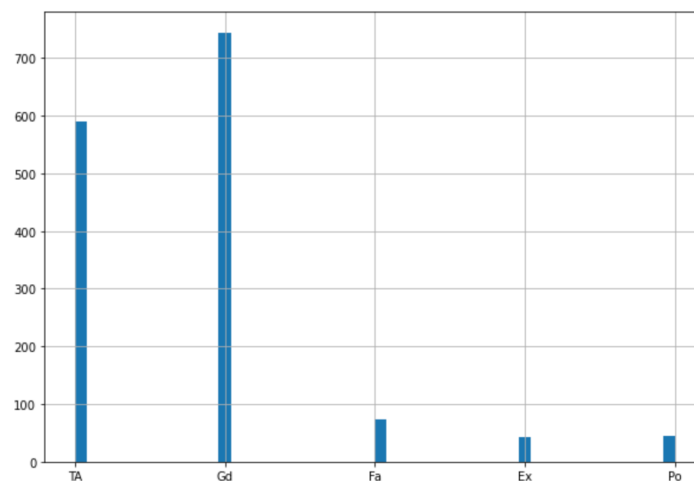
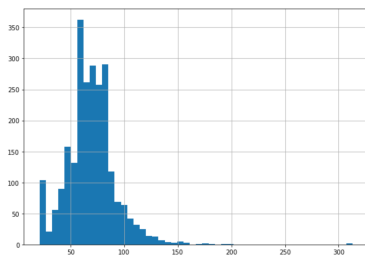


Figure 5: FireplaceQu Distribution

For missing numerical feature, we started with 'LotFrontage' which is linear feet of street connected to property. Seeing the distribution for this feature (Figure 6a) we can see that it has some outliers. Given few number of outliers we assign missing values with median of the distribution.



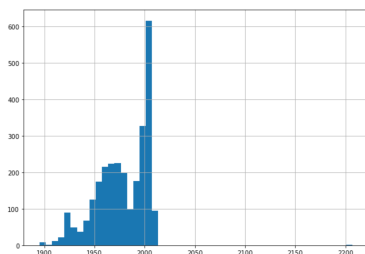
(a) LotFrontage Distribution

```
count    2428.000000
mean      69.301895
std       23.330111
min       21.000000
25%      59.000000
50%      68.000000
75%      80.000000
max      313.000000
Name: LotFrontage, dtype: float64
```

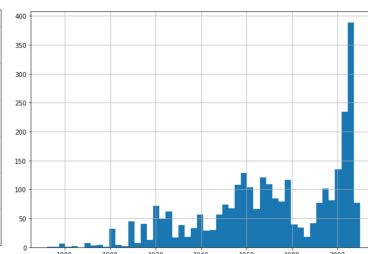
(b) Data Description for LotFrontage

Figure 6: Feature - LotFrontage

For 'GarageYrBlt', seeing the distribution (Figure 7a) we can see that it contains outliers. Normally we can just delete the outliers but this belongs to the test set and we are not allowed to remove it as it changes the number of houses in final submission. For filling the missing values in 'GarageYrBlt' we analyse the 'YearBuilt' feature which does not have any missing values or any obvious outliers. Randomly observing some of the values of 'YearBuilt' and 'GarageYrBlt' (Figure 7c), we can see for most of 'GarageYrBlt' are same as 'YearBuilt' or after 'YearBuilt' which makes logical. Therefore we will just assign the year in which the house was built to 'GarageYrBlt' as it makes logical sense. Similarly the other missing values were filled either with mean or median of the distribution. Now we have a clean numerical category data and we can combine the two.



(a) GarageYrBlt Distribution



(b) YearBuilt Distribution

36	1994	1995.0
37	1954	1954.0
38	1953	1953.0
39	1955	NaN
40	1965	1965.0
41	1959	1959.0
42	1983	1983.0
43	1975	1977.0
44	1959	1959.0
45	2005	2005.0
46	2003	2003.0

(c) Comparison - 7b Vs. 7a

Figure 7: Garage Year Built

After combining numerical and categorical features, we want to convert categorical features into numerical features. As most machine learning algorithms prefer to work with numbers [6], so we have to convert categorical feature i.e text to numbers. There are multiple options for this like 'OrdinalEncoder' and 'OneHotEncoder'. But for our data we will be using 'get_dummies' from [4]. This converts categorical variable into dummy/indicator variables. After converting categorical features to numerical using 'get_dummies', we can now separate the training and test dataset. Also our target feature 'SalePrice' was separated from training data and stored in a new variable. Now our training data has a shape of (1455, 281), which was expected because [4] creates a dummy variable for every category. Now our training data is ready to fed to machine learning models (Figure 8).

	LotFrontage	OverallQual	OverallCond	YearBuilt	YearRemodAdd	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFlrSF	GrLivArea
0	65.0	7	5	2003	2003	706.0	0.000000	150.0	856.0	856	1710
1	80.0	6	8	1976	1976	978.0	0.000000	284.0	1262.0	1262	1262
2	68.0	7	5	2001	2002	486.0	0.000000	434.0	920.0	920	1786
3	60.0	7	5	1915	1970	216.0	0.000000	540.0	756.0	961	1717
4	84.0	8	5	2000	2000	655.0	0.000000	490.0	1145.0	1145	2198
...
1450	62.0	6	5	1999	2000	0.0	0.000000	953.0	953.0	953	1647
1451	85.0	6	6	1978	1988	790.0	8.018288	589.0	1542.0	2073	2073
1452	66.0	7	9	1941	2006	275.0	0.000000	877.0	1152.0	1188	2340
1453	68.0	5	6	1950	1996	49.0	13.022941	0.0	1078.0	1078	1078
1454	75.0	5	6	1965	1965	830.0	9.419894	136.0	1256.0	1256	1256

1455 rows × 281 columns

Figure 8: Final Data

Correlation

Now as all our data is numeric, we can check the correlation of different features with SalePrice. Correlation gives us a measure of how a feature is related to other. A positive values suggest that both features move in tandem with each other i.e. both increases or both decreased. Therefore strongest positive correlation is 1 (Correlation of variable with itself). A negative correlation means both features move in opposite direction. Strongest negative correlation is -1 . A correlation of 0 means there is no correlation between the features. We want to remove features which have correlation close zero with each other. Applying the correlation, it was found that there were 17 features which showed strong correlation with the SalePrice (Either more than 0.5 or less than -0.5). Scatter plot of some of the features with SalePrice can be seen in Figure 9.

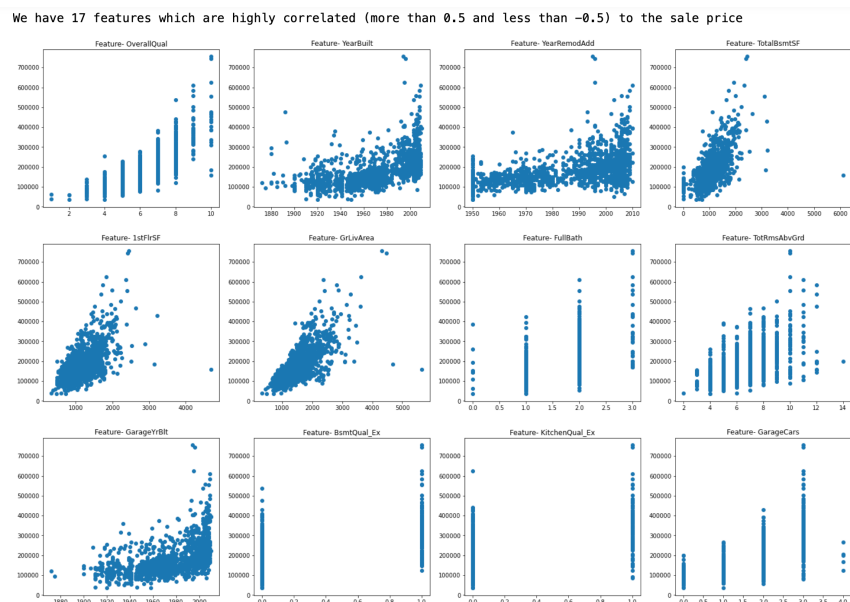


Figure 9: Features with Best Correlation

Implementation

All testing was performed in Jupyter Notebook using a Python kernel. When we ran the most basic regression i.e. Linear Regression through this data, the Root Mean Squared Error (RMSE) value was very high. Now that we have a baseline for performance, we will make changes to our data in order to achieve better results from the same model. Starting with 'SalePrice', looking at its distribution (Figure 10a) we see that it is skewed.

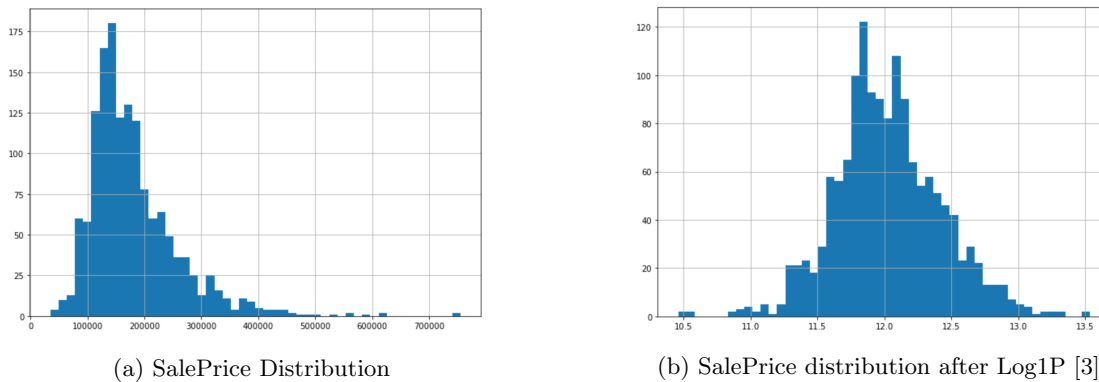


Figure 10: SalePrice

Now to fix the skewness of a variable, we can use Log1P of the feature[3]. It returns the natural logarithm of one plus the input array, element-wise. This reduces the skewness of this feature and can be seen in Figure 10b. Mathematically the skewness reduced from 1.892297 to 0.155425. Applying Linear Regression through our data, we now achieved a RMSE score of 0.146770 which is a significant improvement over the first score. Going back to our features, we checked what other features had skewed data. If the skewness was too high the feature was dropped entirely or tried to reduce using box-cox transform [1] or Log1P[3]. After removing the features which now have the shape (2914, 283) and applying Linear regression algorithm our RMSE score improved to 0.139270.

Models

Now we will apply different Regression models on our data from [6]. Few examples of the models are KNearestNeighbourRegressor, RandomForestRegressor, ElasticNetCV, LassoCV, RidgeCV. We will compare performance of all the models on our data and submit the best results for kaggle competition [2]. One way to know that our model is not over-fitting the data is using Cross-Validation [6]. It randomly splits the training set into distinct subsets called folds, then it trains and evaluated again on the model. After applying cross validation on all our models, we found that the our best performing models was Random Forest and RidgeCV (Figure 11). Picking the best performing models after cross validation ensures that the model is not overfitting the data.

```
Mean RMSE score for LinearRegression() is 0.17553244543138066
Mean RMSE score for RandomForestRegressor() is 0.1478652047803451
Mean RMSE score for KNeighborsRegressor(n_neighbors=3) is 0.2157663512984867
Mean RMSE score for ElasticNetCV() is 0.20407255669479518
Mean RMSE score for LassoCV() is 0.20406896898386145
Mean RMSE score for RidgeCV(alphas=array([ 0.1, 1. , 10. ])) is 0.15706386511674442
```

Figure 11: Cross Validation Results

Hyper-parameter Tuning

We saw the best performing model were Random Forest Regressor and RidgeCV. Now we tuned these model even further to achieve better performance. This was done using Grid Search [6].

Starting with Random Forest regressor, different hyper parameters of this algorithm were studied [5]. Extensive tuning was done with this model. It was computationally heavy and therefore it took a lot of time to train but we achieved our best combination of parameters as shown. After tuning the RMSE value decreased to 0.139218 from 0.147865 which is an almost 6% increase in performance.

Another way of reducing over-fitting for linear model is by constraining the weight of the model. This is where RidgeCV, Lasso and ElasticNet models comes in.

RidgeCV is a regularized version of linear regression [6]. After applying grid search to RidgeCV, it was found that the default parameters were at the best possible value. So it was left untouched.

Evaluation and results

We submitted the results from both of our top models in the kaggle competition [2], achieving a rank of 2083 (Figure 12), with a score of 0.13526. Our best performing model on the test data was RidgeCV. Although there is room for improvement and I will keep working on this dataset for achieving higher results.







2081	Ameya Shukla		0.13522	3
2082	Hayouni Achref		0.13523	3
2083	Shubham Gupta #5		0.13526	4
Your Best Entry  Your submission scored 0.14932, which is not an improvement of your best score. Keep trying!				
2084	heffjeff321		0.13526	8
2085	takky217K		0.13527	4

Figure 12: Final Results

References

- [1] Handling skewness-<https://towardsdatascience.com/top-3-methods-for-handling-skewed-data-1334e0debf45>.
- [2] Kaggle: Housing prices- advance regression techniques-<https://www.kaggle.com/c/house-prices-advanced-regression-techniques>.
- [3] Numpy log1p - <https://numpy.org/doc/stable/reference/generated/numpy.log1p.html>.
- [4] Pandas- get.dummies - https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html.
- [5] Random forest - <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.randomforestregressor.html>.
- [6] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.