Kaggle Competition: Predicting House Price

Minh Vu - Duc Nguyen

***Abstract*—This project objective is to predict house price for Kaggle Advance Housing Competition. We used the data provided by Kaggle and apply different machine learning model to build an algorithm that predict house prices for Ames, Iowa. We later use this algorithm and apply on a test set Kaggle provide and compete with other people. We obtain a results of 0.13265 and ranked us at 1690 out of 5361, top 30%.**

I. Introduction

Our goal for this project is to be able create an algorithm that predict the house price as accurate as possible. We were given a training set and a test set with 79 attributes. Our job is to use the training set and that 79 attributes to find a pattern to the Sale Price of each house. To do that, we divided the work into 5 steps:

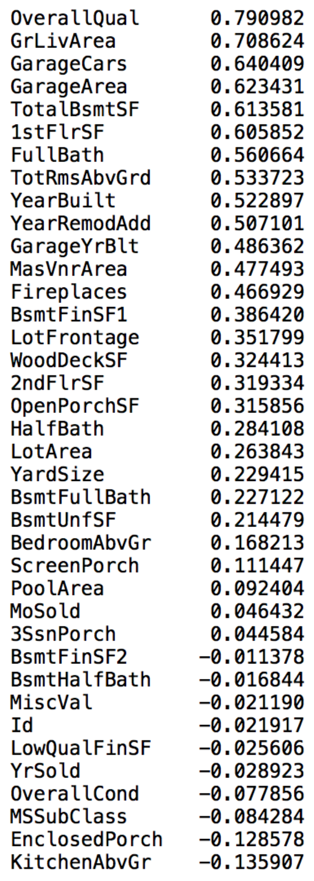
1) Preprocessing Data
2) Apply different model
3) Compare result of each model
4) Stack model
5) Results

We know that the train set comprised of 1460 rows and 81 columns (with 1 is ID and 1 is Sale Price) and the test set comprised of 1459 rows and 80 columns (the last column is SalePrice). We have to use our algorithm to predict the last SalePrice column for the test set.

II. Procedure

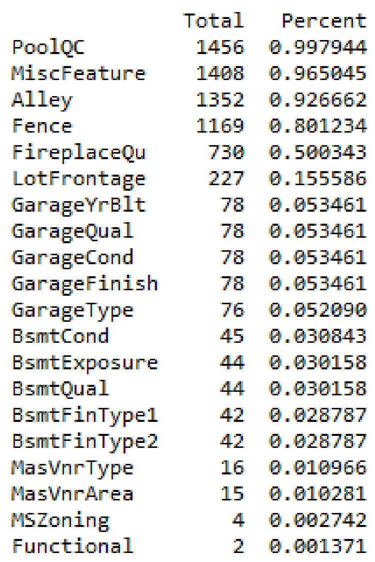
1) Data Preprocessing
   a) Understanding the data
      We use .corr() method to get the correlation between each attribute of the train set with the SalePrice and print out the correlation values (Figure 1.)
      We use this to find the column with the lowest and highest correlation to the SalePrice.

   b) Missing Data
      We find out the number of missing value for each columns and drop the on with more than 75% of the data missing (Figure 2.)
      We then drop the column with more than 80% missing value.
   c) Outliers
      At this step, we try to find any outliers that could potentially worsen the data. From the correlation matrix, we pick out the top 3 significant attribute and look for outliers within it (Figure 3.)
      We can see that GrLivArea, GarageArea and TotalBsmtSF are the top 3 attribute with the most correlation. We then use concat() function to concat



```
OverallQual      0.790982
GrLivArea        0.708624
GarageCars       0.640409
GarageArea       0.623431
TotalBsmtSF      0.613581
1stFlrSF         0.605852
FullBath         0.560664
TotRmsAbvGrd     0.533723
YearBuilt        0.522897
YearRemodAdd     0.507101
GarageYrBlt      0.486362
MasVnrArea       0.477493
Fireplaces       0.466929
BsmtFinSF1       0.386420
LotFrontage      0.351799
WoodDeckSF       0.324413
2ndFlrSF         0.319334
OpenPorchSF      0.315856
HalfBath         0.284108
LotArea          0.263843
YardSize         0.229415
BsmtFullBath     0.227122
BsmtUnfSF        0.214479
BedroomAbvGr     0.168213
ScreenPorch      0.111447
PoolArea         0.092404
MoSold           0.046432
3SsnPorch        0.044584
BsmtFinSF2      -0.011378
BsmtHalfBath    -0.016844
MiscVal         -0.021190
Id              -0.021917
LowQualFinSF    -0.025606
YrSold          -0.028923
OverallCond     -0.077856
MSSubClass      -0.084284
EnclosedPorch   -0.128578
KitchenAbvGr    -0.135907
```

Fig. 1. correlation values of all training set attribute.



```
              Total   Percent
PoolQC         1456  0.997944
MiscFeature    1408  0.965045
Alley          1352  0.926662
Fence          1169  0.801234
FireplaceQu     730  0.500343
LotFrontage     227  0.155586
GarageYrBlt      78  0.053461
GarageQual       78  0.053461
GarageCond       78  0.053461
GarageFinish     78  0.053461
GarageType       76  0.052090
BsmtCond         45  0.030843
BsmtExposure     44  0.030158
BsmtQual         44  0.030158
BsmtFinType1     42  0.028787
BsmtFinType2     42  0.028787
MasVnrType       16  0.010966
MasVnrArea       15  0.010281
MSZoning          4  0.002742
Functional        2  0.001371
```

Fig. 2. percentage of missing value for each column

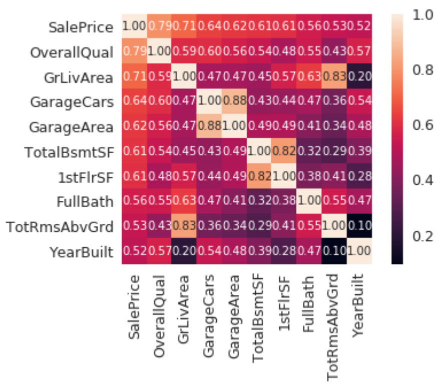


Fig. 3. correlation matrix of all training set attribute.

between the 3 attribute and SalePrice to draw a scatter plot as follow (Figure 4, 5, 6.)

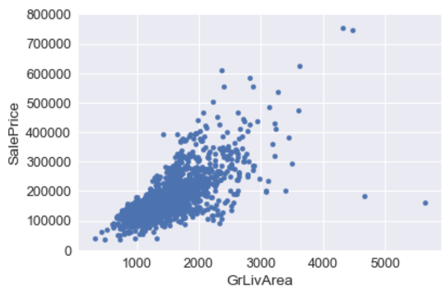


Fig. 4. GrLivArea Scatter Plot with SalePrice

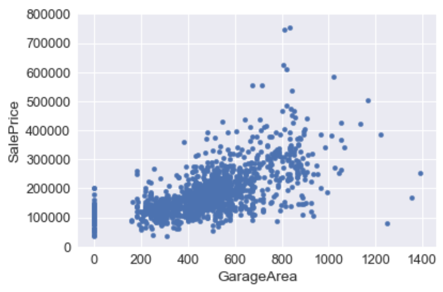


Fig. 5. GrLivArea Scatter Plot with SalePrice

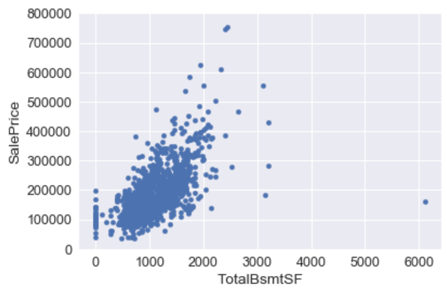


Fig. 6. GrLivArea Scatter Plot with SalePrice

From the 3 graph, we can spot the outliers, then we determine its index by using nlargest() function. For instant, TotalBsmtSF have an outliers outside of the 6000, so we use the nlargest(1) to drop that outliers.

Note: we also have another function to drop the outliers, which compares the new predictions generated from the training data set and the actual original values of SalePrice column, calculating the difference and drop any row that has a difference more than 30000. That is, we dropped any rows that generated poor predictions.

d) Dummies Variable:
There were a lot of columns is not represented by integer, so we use the built-in get-dummies function to change categorical data to numerical.

2) *Applying Models*
At this step, we use different built in model from Python library such as RandomForest, xgBoost, GradientBoostingRegressor to predict the sale price and test the accuracy.

a) Multiple Linear Regression

$$J(\vec{\theta}) = \frac{1}{2m}\sum_{i=1}^{m}(h_{\vec{\theta}}(\vec{x}^{(i)}) - y^{(i)})^2$$

$$\theta_j \leftarrow \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_{\vec{\theta}}(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

Fig. 7. Multiple Linear Regression

We built our own multiple linear regression function based on the equation above and use it for our prediction. First, we created an array call theta. This array length is equal to the number of rows of the the training set. Then we initialized it with random number from -0.5 to 0.5. Next, we set alpha(learning rate value) to 0.01. This value was chosen arbitrary. Then we created a new prediction column(h(x)). Each row in this column is the sum of its corresponding theta times its column. So now we have an original h(x) comprise of a prediction from the attributes. Next, we applied the formula(figure 7.) to the dataset to obtain new theta value. We take the predicted value h(x) and subtract the actual price from it, then we times that value with the column corresponded with theta. After that, we divide the value with the number of rows in the training set and times it with alpha. Lastly, we subtract that value with the original theta to obtain a new theta value. We do this for all the row in the set.
After having a new theta value, we repeat the process again to get a new predictor h(x) and apply the formula again on the data set to get a different

theta result. We do this for 500 times, and each time we expected to get a new and better theta value.
After 500 iterations, we used the final theta value and applied them on the test set to get the prediction.

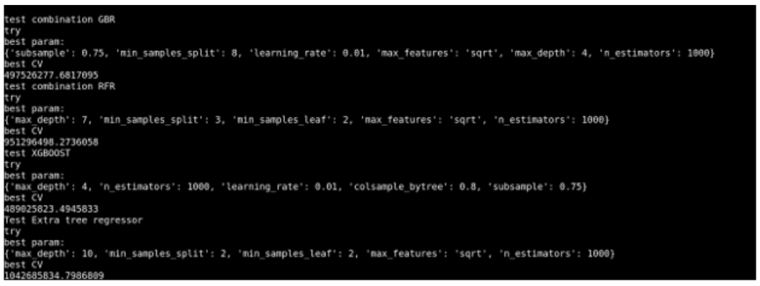
3) Built-in model
We separate the three classification model from the Multiple Linear Regression because we want to test if built in class is better.

a) Models Parameter Tuning
Firstly, we got to learn more about the important variables/ parameters used for defining a tree:
1.Min_samples_split: Defines the minimum number of samples (or observations) which are required in a node to be considered for splitting. Used to control overfitting.
2.Min_samples_leaf: Defines the minimum samples (or observations) required in a terminal node or leaf. Also used to control overfitting
3. Max_depth: The max depth of a tree, used to control overfitting
4. Max_features: The number of features to consider while searching for a best split, I always use sqrt for this since I believe it is good enough
5. Learning_rate: his determines the impact of each tree on the final outcome (step 2.4). GBM works by starting with an initial estimate which is updated using the output of each tree. The learning parameter controls the magnitude of this change in the estimates, so lower values are preferred, but they come with an expensive cost.
6. n_estimators: the number of sequential trees to be modeled
7. subsample: Fraction of observations to be selected for each tree. Typical values 0.8 generally work fine but can be fine-tuned further ¿ Most of the parameters should be tuned using cross validation

Secondly, I got to the parameter tuning process. Initially I tried out Gradient Boosting Regressor, tried tuning only n_estimators parameter using model_selection.GridSearchCV, which I got the result to be the bigger the range n_estimators we set, the bigger the result to be, so the bigger n_estimators is the better.

After trying out tuning the first parameter of GBR, We began trying tuning multiple parameters of a model at the same time, which would be expensive and take a lot of time; however, it will return the best combination of parameters for the model that Im considering.

So We tried tuning 4 different models (GBR, Extreme Gradient Regressor, Random Forest Regressor, and Extra Tree Regressor) using GridSearchCV with model.best_params_ and model.best_score_. And we got the result as below:

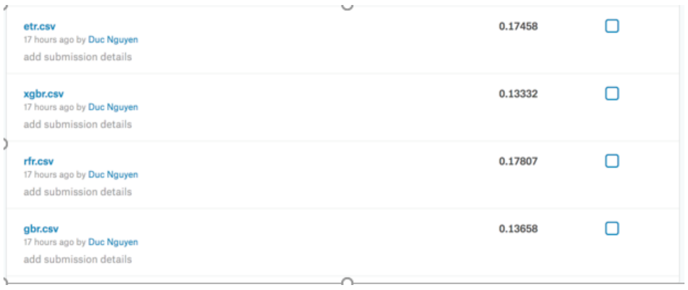


```
test combination GBR
try
best param:
{'subsample': 0.75, 'min_samples_split': 8, 'learning_rate': 0.01, 'max_features': 'sqrt', 'max_depth': 4, 'n_estimators': 1000}
best CV
497526277.6817095
test combination RFR
try
best param:
{'max_depth': 7, 'min_samples_split': 3, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'n_estimators': 1000}
best CV
951296498.2736058
test XGBOOST
try
best param:
{'max_depth': 4, 'n_estimators': 1000, 'learning_rate': 0.01, 'colsample_bytree': 0.8, 'subsample': 0.75}
best CV
489025823.4945833
Test Extra tree regressor
try
best param:
{'max_depth': 10, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'n_estimators': 1000}
best CV
1042685834.7986809
```

Fig. 8. Parameter Tuning Results

Using the result of the tuning process above, we used those combinations of parameters to generate 4 models defined above (GBR, xgBoost, RFR, and ETR) and fit the models with training set and outputSeries(which is the SalePrice column in traning data set), then generating predictions using .predict function on test data set. Then we print out the predictions to check. Finally we use the writefile() function to write prediction files for the models.

Submitting the predictions files onto Kaggle,
we got the results below:



etr.csv 17 hours ago by Duc Nguyen add submission details	0.17458	☐
xgbr.csv 17 hours ago by Duc Nguyen add submission details	0.13332	☐
rfr.csv 17 hours ago by Duc Nguyen add submission details	0.17807	☐
gbr.csv 17 hours ago by Duc Nguyen add submission details	0.13658	☐

Fig. 9.

We found this idea online from Shitao (https://github.com/Shitao/Kaggle-House-Prices-Advanced-Regression-Techniques), his idea of stacking is to use another model or stacker (which is Ridge here), to combine all previous model predictions in order to reduce generalization errors. Second, he looped through the 5 folds training data set, training each model using 4 folds and predict on the remaining fold. Each base model returns a prediction. Then he will have the prediction of entire training data set for each model and 5 copies of the prediction of data set for each model. Finally, he trained the second model, (stacker), using the predictions as new features and use the average of the 5 copies of the test data set predictions as the test input for the trained models to provide the final prediction.

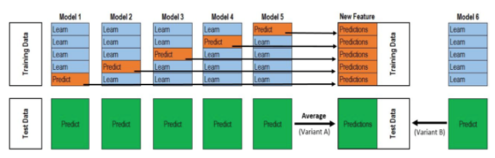
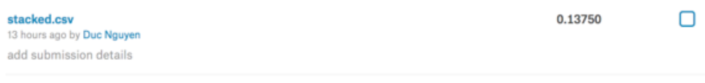


Figure 6: An illustration of stacking approach. (Credit: Kaggle forum)

Fig. 10.

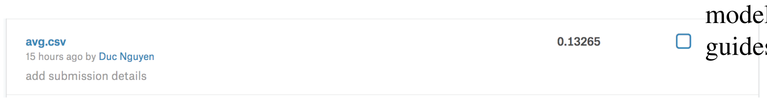
So we copied his codes into my code (which he build a new ensemble class and use it to generate predictions by stacking a set of base models), using my base models obtained above and tried it out, even though it isnt as good a result as his, which maybe caused due to different procedure of data preprocessing or tuning process, we learned a lot more about stacking models and will make use of it in the future. The stacking result is obtained as below.



stacked.csv
13 hours ago by Duc Nguyen
add submission details
0.13750

Fig. 11.

Out of curiosity, we compute the mean add the two three models, which is Gradient Boosting Regressor, Extreme Gradient Regressor and Stacked, and compute the mean of them. We use that values as our prediction, and surprisingly the results improved to 0.132 (slightly better)



avg.csv
15 hours ago by Duc Nguyen
add submission details
0.13265

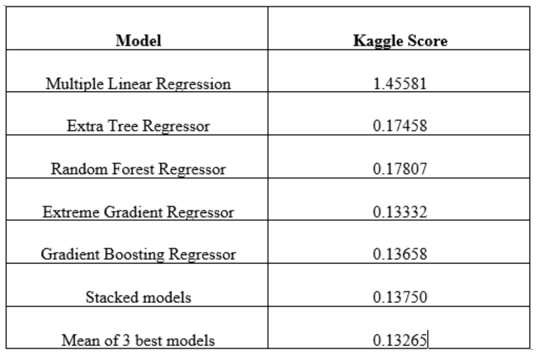
Fig. 12.

III. Result

After we ran Multiple Linear Regression, tuned 4 different models: Gradient Boosting Regressor, Random Forest Regressor, Extreme Boosting Regressor, Extra Tree Regressor, we obtained the results as follow:

Unfortunately, our Multiple Linear Regression gave the worst score, compare to other tuned built-in class. Also,the stacked model did not work well as expected, giving a slightly lower score than the built-in. Instead the best model is the built-in tuned Extreme Gradient Regressor.

.



Model	Kaggle Score
Multiple Linear Regression	1.45581
Extra Tree Regressor	0.17458
Random Forest Regressor	0.17807
Extreme Gradient Regressor	0.13332
Gradient Boosting Regressor	0.13658
Stacked models	0.13750
Mean of 3 best models	0.13265

Fig. 13.

IV. Conclusion

Even though these were not the results as we hoped for, I think we learned a lot through this experience of data mining. There are still room for major improvement to be made. First, We believe the reasons why Multiple Linear Regression gave such a horrible score is because we only have time to run 500 iterations. If we could run more times, the theta values will be tuned better and thus gave a stronger results.

Second, We expected the stack model would given us a better results than other built-in class alone, nevertheless that was not the case. From this, we will be looking deeper into Shitao's code and explore why his stacking method did not work well with our data.

Third, there are much to improve in the preprocessing data steps. We only did basic analysis and preprocessing of the data, and spent too much time focusing on upgrading our model.

At the end, we still have significant improvement with our model. Our first result was at around 0.18, we brought the result up to 0.13 by applying different models. Moving forward, we will try to have a better understanding of each model and how stacking works and learn from other available guides on Kaggle's Kernel.

References

[1] https://github.com/Shitao/Kaggle-House-Prices-Advanced-Regression-Techniques
[2] https://www.kaggle.com/pmarcelino/comprehensive-data-exploration-with-python
[3] https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/