CSC240
Team Name: LTK
Team Member: Zixiang Liu, Yukun Chen, Shengbo Ge

CSC 240 FINAL PROJECT
HOUSE PRICES PREDICTION

Abstract:

In this competition, we are going to predict the prices of houses by the data given. First, we did data processing to change categorical data to numerical data, and simplified them to fit the models. Second, we used multiple models to train the data and then ensemble them to give out prediction value. In last step, we just played with the data to improve performance, which would be explained later.

Data can be found at:
https://www.kaggle.com/c/house-prices-advanced-regression-techniques

I. Data Processing

1. Deal with missing value
   a) Fill NaN with 0

      Ex: LotArea, MasVnrArea, BsmtFinSF1 and etc.

   b) Fill NaN with median

      Ex: LotFrontage (Since their shall not be property without street)

2. Change ordinal to numerical

   Match categorical data with ordinal meaning {None: 0, "Po": 1, "Fa": 2, "TA": 3, "Gd": 4, "Ex": 5}

   Ex: BsmtExposure, OverallQual, KitchenQual and etc.

3. Simplify attributes

   Using numerical data directly may have overfitting problem when applying model, so we simplify them to lower the dimension.

   a) Apply mapping {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2}

      Ex: OverallCond, GarageCond, FireplaceQu and etc.

   b) Divide year into multiple bins, with each bin represents a 20 year range

      Ex: GarageYrBltBin, YearBuiltBin and etc.

4. Find underlying relationship

   a) We find many of the data highly correlated, so we think of adding variable to show this relationship

      Ex: We assume house sold in yearBuild has higher price

      RecentRemodel = 1 if "YearRemodAdd" =="YearBuilt"

   b) Some numerical data have no meaning itself, so we change it to data correlated with sale price.

      Ex: we plot the avg sale price for month, and define some of them as HighSeason if its sale price higher than others

      Similarly, we transfer: MSSubClass, Neighbourhood and etc. to numerical value according to mean of each group.

   c) For some categorical feature, we find huge difference between one and another, so we replace it with new feature simply including 1 and 0

      Ex: "LandContour"] == "Lvl", "Electrical"] == "SBrkr"

5. Deal with skewness

   For features which are far from normal distribution (skewness>0.75), we take the log(x+1) of them to reduce skewness.

6. Convert categorical to numerical

   For data without significant numerical meaning which is covered in part 4&5 above, we take the dummy value of them, and leave them studied by our models.
   Ex: HeatingQC, PoolQC, BsmtExposure and etc.

II. Models Applying

The model used includes the following:

- XGBoost
- Lasso
- SVM
- Random Forest
- MLP
- Gradient Boosting
- Extra Trees
- Ridge

Due to time limitation, we only trained several of them using cross validation of 10 folds. The final parameter we got is attached in Appendix A. The results in Kaggle are listed as follow:

• XGBoost	0.11604
• Lasso	0.11529
• SVM	0.11843
• Random Forest	0.12090

III. Ensemble & Blending

After multiple rounds of testing, we find out that the optimal result of predicted sale price we can achieve in Kaggle is getting the mean of Lasso and XGBoost . The result of the mean is 0.11417.

Then, it took us a long time thinking about how to deal with the rest model. The prediction price from other models are valuable. However, they are not that good to be the "main" data.

So, we decided to use them to tune the sale price in small step. The method is introduced bellow:

1. We got prediction value from a new model.

2. We first blended it as a +10% influence on the prediction price, upload the result to Kaggle to get an error rate. Then, we blended it as a -10% influence on the prediction price, and got our second error rate.

3. By comparing the three error rates when: blending as +10%, not blending, blending as -10%, we got a small regression model and could calculate the peak performance out of that.

Using this method, we can ensure improvement within only 3 times of submitting. By applying this method multiple times, we utilized our models thoroughly.

IV. Final Adjustments

After trying all the ensemble method mentioned above, we reached a bottleneck: no matter how we adjust our model, we can only achieve trivial improvement.

We searched for solution on Kaggle forum, and found out the reason may be that the test and train data were not in the same kind of distribution. So we decided to trim our data to fit the test set distribution.

We first shifted the mean by adding a fixed value to all the houses. After multiple times of trying, we found the best value as +1100.

Then, we increased the distribution into a wider range. Since most of the model we used are linear regression, they shall tend to be conservative when predicting extreme values. So we added some weight to make them more detached from the mean value.

V. Final Result

The final result we got on Kaggle is 0.11285, which rank No.27 on the leader board currently.
*Note that we used each of our laptop to run the same code. However, all three outputs have some tiny difference. The 0.11285 is the best performance, but lowest performance can also assure output less than 0.113.

Appendix A

extra tree
Best is n_estimators:400, max_features:0.30, max_depth:10, Random State:None
Score:0.881376

gradient boost
Best is n_estimators:100, max_features:0.20, max_depth:7, Random State:None
Score:0.890817

neural network
hidden_layer_sizes: (100, 50) Random_state10: 0.796215

SVM
Best is:
0.920856: C: 1.000000 Epsilon: 0.010000

lasso
Best is:
0.0003709: 0.923271

Xgb (output in some range, not the best of all)
Best is Learning_rate:0.10,min_child_weight:2.3,gamma:0.000,subsample:0.4
Score:0.914523

Rf
Best is n_estimators:400, max_features:0.30, max_depth:10, Random State:None
Score:0.891661