Machine Learning Engineer Nanodegree

Capstone Project

Yibo Gong

July 25th, 2019

I. Definition

Project Overview

I am very interested in the real estate market, which has been very hot in Toronto during the past few years, so for this capstone project I want to focus on something related to the realty market.

When a home buyer describes their dream houses, they usually will not begin with the height of the basement ceiling or the proximity to an east-west railroad. But, besides the number of bedrooms or a white-picket fence, there are a lot more factors that may impact a home price. So for this capstone project, I picked a project from Kaggle: House Prices: Advanced Regression Techniques (https://www.kaggle.com/c/house-prices-advanced-regression-techniques/overview/description). I will apply the Machine Learning techniques I have learnt from the Udacity Nano Degree, to predict the home prices in Ames, Iowa, from 79 features described in the dataset.

The dataset I use (the Ames Housing dataset) was compiled by Dean De Cock for use in data science education. It's an incredible alternative for data scientists looking for a modernized and expanded version of the often-cited Boston Housing dataset.

There are also some research has been done in this area. The below two papers
https://nycdatascience.com/blog/student-works/housing-price-prediction-using-advanced-regression-analysis/
https://arxiv.org/pdf/1809.04933.pdf
have shown some methodologies regarding how to process the data, and how to explore and analyze the data, and also the models can be applied on the data.

Problem Statement

The goal is to predict the sales price for each house. For each Id in the test set, the value of the SalePrice variable will be predicted. This will be a supervised regression learning. We can use Random Forest or Gradient Boosting to analyze the data. We can also apply some pre-processing on the data like normalization, or apply PCA to extract higher level features.

Below are the steps I plan to take to solve the problem:

1. Data exploration and feature engineering
    a. Visualize the data and decide if there is any skewed data or not, and see if data normalization is needed or not
    b. Determine outliers if there is any
    c. Apply one-hot encoding if needed
    d. Use PCA to generate high level features which will include most of the data variance
2. Model selection with grid search and k-fold validation
    a. Decision Tree
    b. Gradient Boost
    c. Random Forest

Metrics

The evaluation metrics will be root mean square error:

RMSE = $\sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}$

RMSE is the standard deviation of prediction errors. It tells you how concentrated the data is around the line of best fit. In this project, we will use RMSE to determine how good the model's performance is.

The reason I choose RMSE over other metrics like MAE (mean absolute error) or R2 is that:

1. Compared with MAE, RMSE gives relatively higher emphasis to errors with outliers
2. Compared with R2, RMSE has a range of 0 to infinite; while R2 score has a range of 0 to 1. In this particular case, we are predicting prices, so RMSE will give us a better idea how the predictions differ from the actual prices.

II. Analysis

Data Exploration

After exploring the data, we find there are in total 1460 rows of data, with 81 columns. We dropped the 'Id' column, which is useless for predicting the sales price, and result in 80 columns, with 79 features and 1 target value, as shown below:

'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street', 'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType', 'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF',

'TotalBsmtSF', 'Heating', 'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual', 'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType', 'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual', 'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC', 'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType', 'SaleCondition', 'SalePrice'.

A sample of the data looks like below:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal
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

Some of the important features include continues feature – ‘LotArea’, discrete feature – ‘FullBath’, nominal feature - ‘Heating’, ordinal feature – ‘OverallQual’.

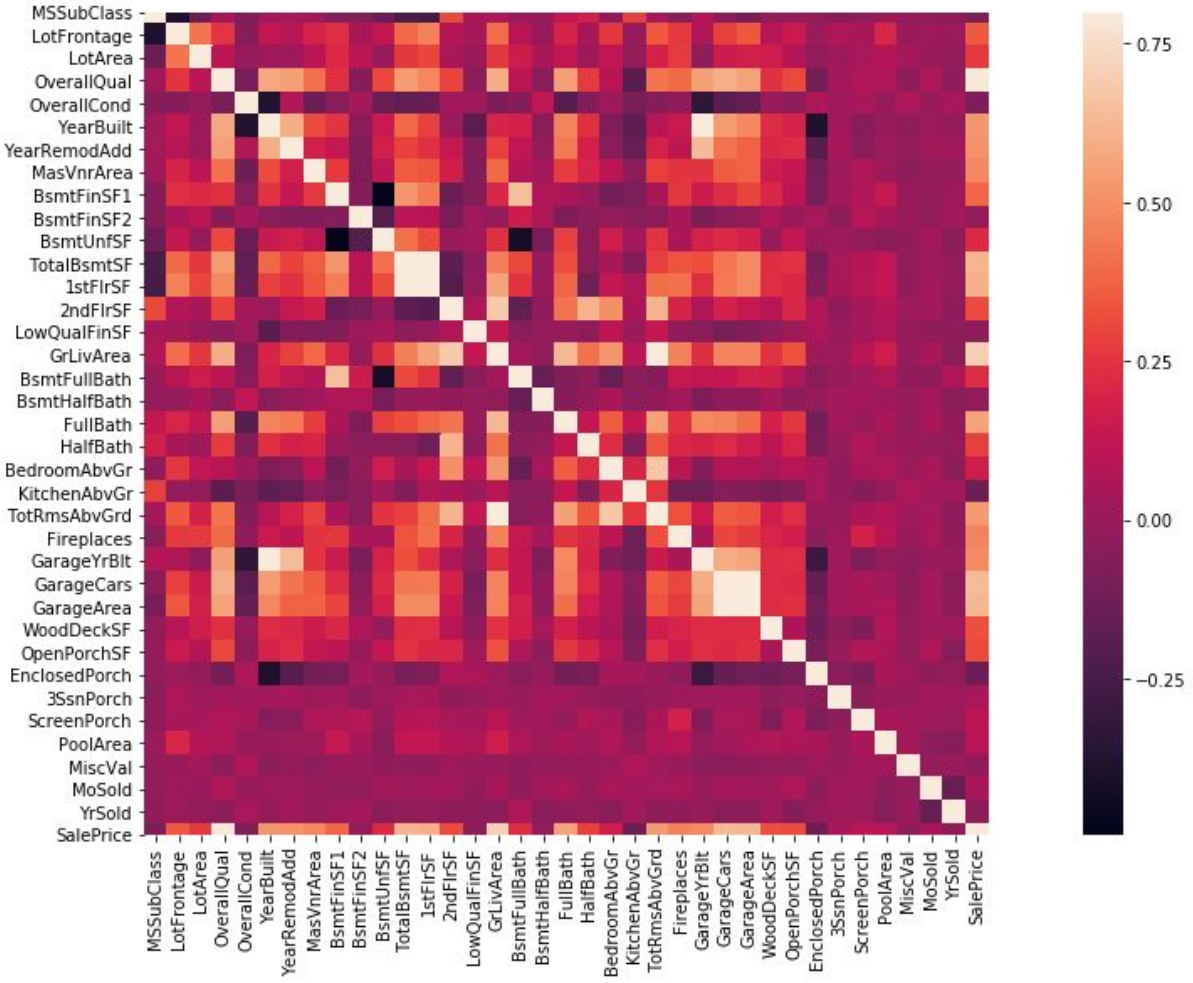
For some of the features, there are also missing data in them, which we need to fill them up as well.

	Missing Ratio
PoolQC	99.588477
MiscFeature	96.296296
Alley	93.758573
Fence	80.727023
FireplaceQu	47.325103
LotFrontage	17.764060
GarageYrBlt	5.555556
GarageType	5.555556
GarageFinish	5.555556
GarageQual	5.555556
GarageCond	5.555556
BsmtFinType2	2.606310
BsmtExposure	2.606310
BsmtFinType1	2.537723
BsmtCond	2.537723
BsmtQual	2.537723
MasVnrArea	0.548697
MasVnrType	0.548697
Electrical	0.068587

Exploratory Visualization

Correlation Heat Map

Below is the correlation heatmap for the features before any data manipulation. Here we can see the correlations between the features and target sales price.



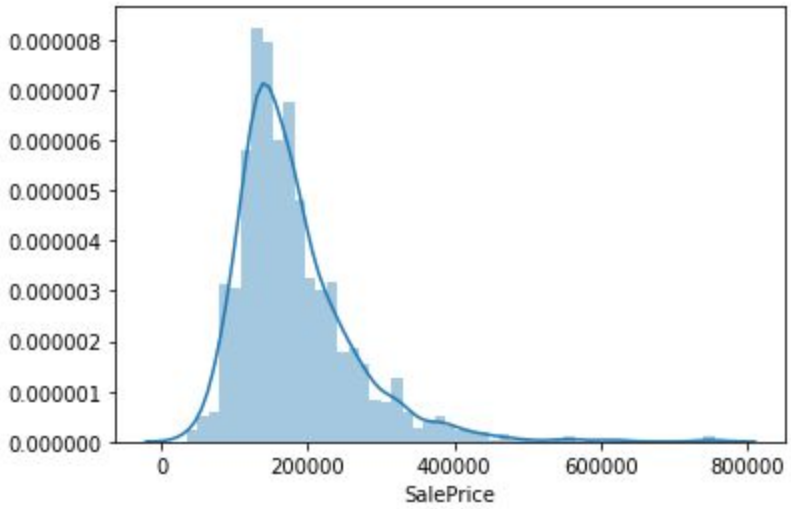
Target Value - Sales Price

Below is the general description of the sales price:

count: 1460, mean: 180921.195890, std: 79442.502883, min: 34900, max: 755000
25%: 129975, 50%: 163000, 75%: 214000

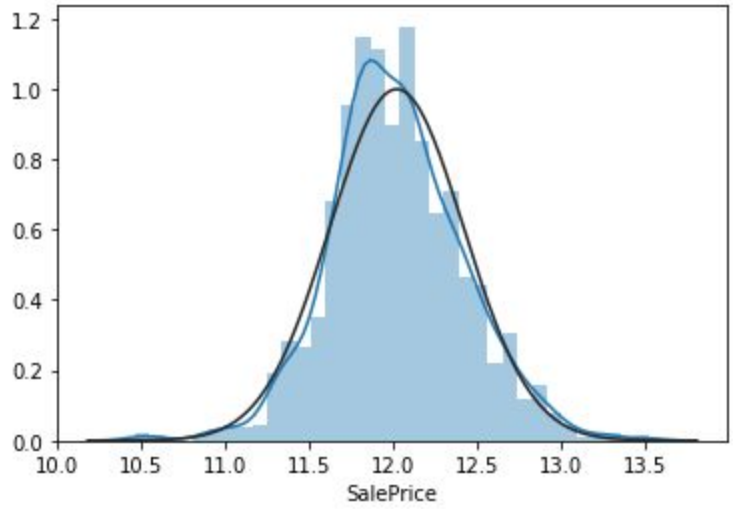
Below are the skewness and kurtosis of the data before any preprocessing:

```
Skewness: 1.882876
Kurtosis: 6.536282
```



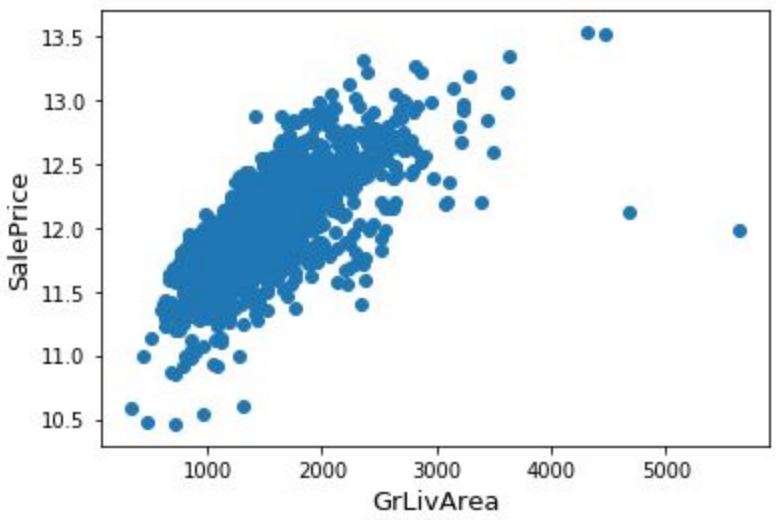
After log normal transformation, we can get a much better distribution:

```
Skewness: 0.121347
Kurtosis: 0.809519
```

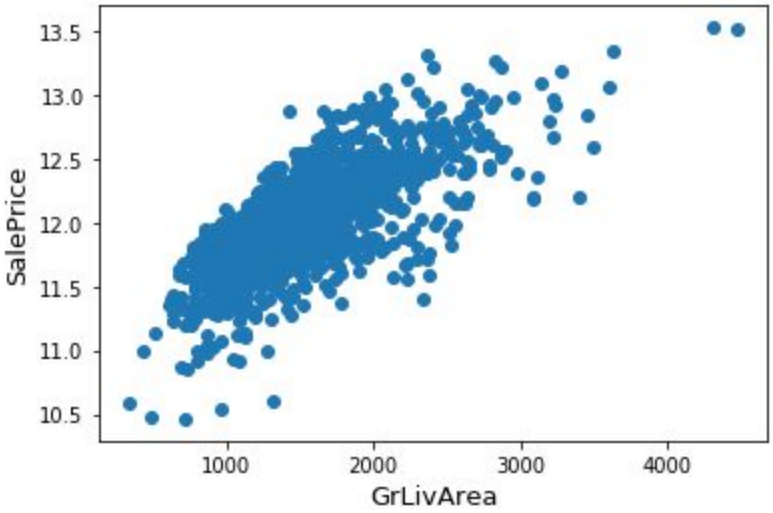


Outliers

We can also find that, there are outliers when we plot the trend between GrLivArea vs SalePrice:



I choose to remove these two data points because generally, larger living area will result in higher sale price. These two outliers have a sale price unreasonably lower than the others, which makes no sense to me. Below is the data distribution after I removed the outliers:



Missing Data

As discussed above, we have a lot of missing data for some of the features in the data set. We need to fill up the missing data as well.

For features like 'PoolQC', 'MiscFeature', 'Alley', 'Fence', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1','BsmtFinType2', 'MasVnrType', 'MSSubClass', we fill the missing data with ‘None’, as these are categorical features.

For features like 'GarageYrBlt', 'GarageArea', 'GarageCars', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'BsmtFullBath', 'BsmtHalfBath', 'MasVnrArea', we fill the missing data with 0, as these are numerical features.

For features like 'MSZoning', 'Electrical', 'KitchenQual'], 'Exterior1st', 'Exterior2nd', 'SaleType', we fill the missing data with the most common values.

For feature like 'Functional', we fill the missing data with 'Typ', which is typical value.

For feature like 'LotFrontage', we fill the missing data with the median of the column.

Also we dropped 'Utilities', which has the same value for all rows.

Label Encoding

For features like 'FireplaceQu', 'BsmtQual', 'BsmtCond', 'GarageQual', 'GarageCond', 'ExterQual', 'ExterCond','HeatingQC', 'PoolQC', 'KitchenQual', 'BsmtFinType1', 'BsmtFinType2', 'Functional', 'Fence', 'BsmtExposure', 'GarageFinish', 'LandSlope','LotShape', 'PavedDrive', 'Street', 'Alley', 'CentralAir', 'MSSubClass', 'OverallCond', 'YrSold', 'MoSold', we label encoded them.

Note that for features like 'MSSubClass', 'OverallCond', 'YrSold', 'MoSold', they were numerical features, we take them as categorical features as well since it makes more sense to treat them as categories.

Add New Feature

We also added a new feature: 'TotalSF', which is the sum of 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF'. We create this new feature because all 3 features measures the living area. So it makes sense to create a feature which measures the total square footage.

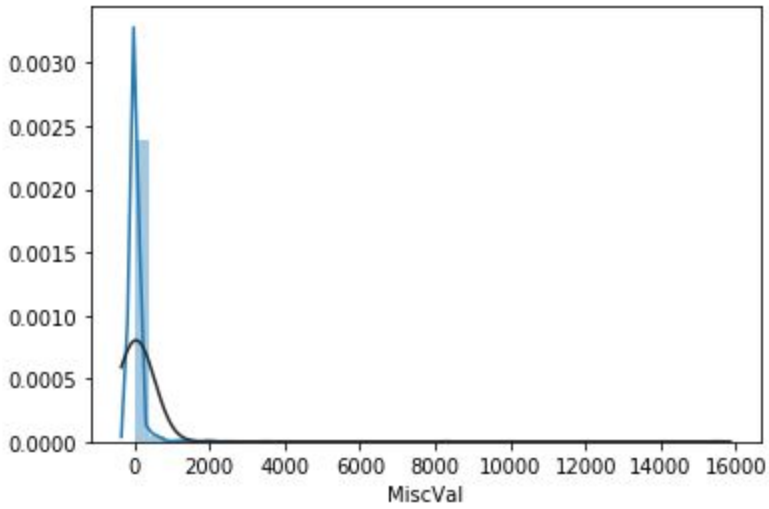
Feature Transformation

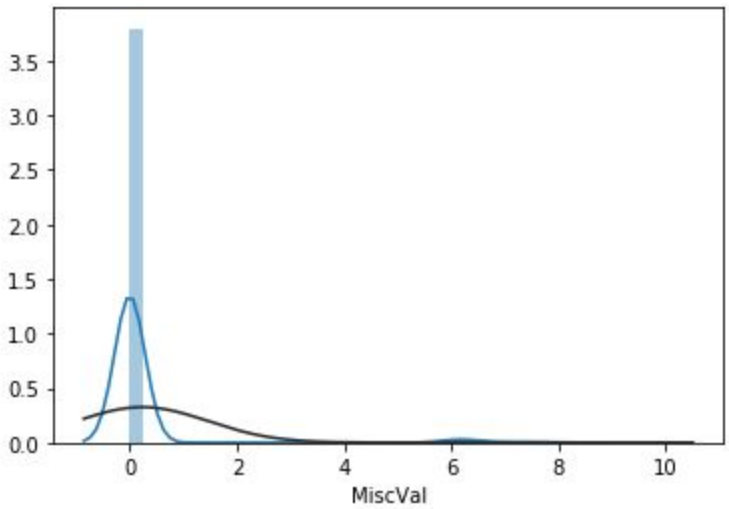
Below is the list of features that have high skewness:

	Skew
MiscVal	24.434913
PoolArea	15.932532
LotArea	12.560986
3SsnPorch	10.286510
LowQualFinSF	8.995688
LandSlope	4.805032
KitchenAbvGr	4.480268
BsmtFinSF2	4.247550
ScreenPorch	4.114690
BsmtHalfBath	4.095895

I applied log normalization on these features to remove the skewness.

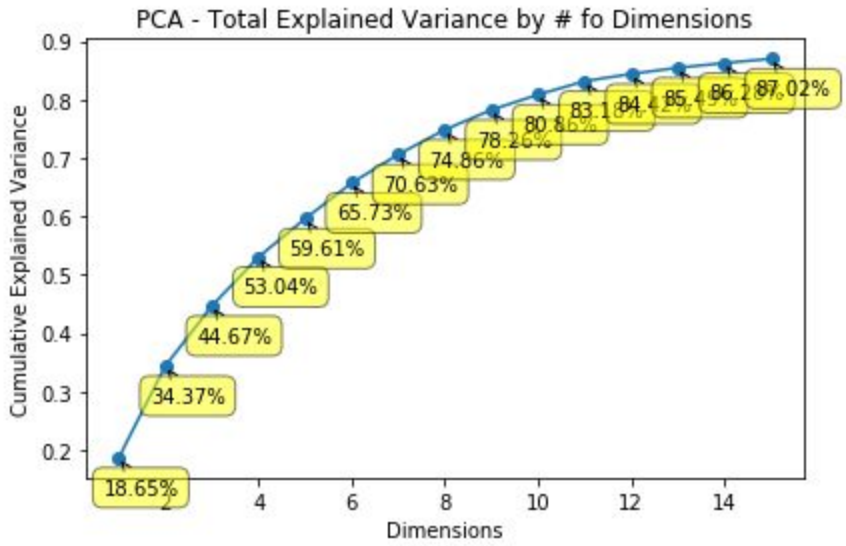
Below is an example for feature 'MiscVal' that before and after transformation:





PCA

After all the data processing, we ended up with 220 features. So I think it should be a good idea to perform PCA to reduce the number of dimensions. I find that I can get 87% of the total variance by using 15 components:



Algorithms and Techniques

Below is a summary of the algorithms and techniques I used for this exercise:

1. Log transforms: By using this transformation of data, we can improve the distribution of features and target.

2. One hot encoding: convert categorical features into numerical data to feed into models

3. PCA: Transform input features into principal components, and use PCs as new features; PCs are directions in data that maximize the variance and are independent with each other; By applyng PCA, we reduce the dimension from 220 to 15 by retaining 87% of the variance

4. Decision Tree Regressor: This is a decision tree based regressor. Should be fast to optimize.

5. Gradient Boost Regressor: It is an ensemble method with boosting and with more hyperparameter to adjust. It performs well with many weak learners. Boosted trees try to learn a more complex decision boundary by successively fitting trees on the error with a low learning rate over hundreds of trees. Below is the theory behind gradient boost:

    a. Get a simple model on the data

    b. Punish the classified data more and re-run to get a new model

    c. Repeat step 2 for n times

    d. Combine all the n models together by model weights

6. Random Forest: I choose to run on random forest because random forest is an ensemble method based on decision tree algorithm. So since I used decision tree as the benchmark model

in this project, I should see that random forest should outperform decision tree in the results. Random forests reduce overfit by building lots of weak or shallow decision trees and using a sort of majority vote to result in complex decision boundaries.

Benchmark

In this exercise, Decision Tree was used as the benchmark, below is the result by using DT on the data with GridSearch:

Time algo takes: 2.702 seconds, Train score: 0.0145, Test error: 0.0150

DecisionTreeRegressor(criterion='mse', max_depth=5, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best')

III. Methodology

Data Preprocessing

All my data preprocessing steps are listed below:

1. Target SalePrice: Since the target value distribution is positively skewed, I applied log transformation. And after that, the data distribution is improved a lot.

2. Outliers: We found there are two outliers when comparing GrLivArea vs SalePrice. And we removed the outliers as they are very likely will negatively impact the model results.

3. Missing data: There are many features have missing data. For features like 'PoolQC', 'MiscFeature', 'Alley', 'Fence', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1','BsmtFinType2', 'MasVnrType', 'MSSubClass', we fill the missing data with ‘None’, as these are categorical features; For features like 'GarageYrBlt', 'GarageArea', 'GarageCars', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'BsmtFullBath', 'BsmtHalfBath', 'MasVnrArea', we fill the missing data with 0, as these are numerical features; For features like 'MSZoning', 'Electrical', 'KitchenQual'], 'Exterior1st', 'Exterior2nd', ‘SaleType’, we fill the missing data with the most common values; For feature like ‘Functional’, we fill the missing data with ‘Typ’, which is typical value; For feature like ‘LotFrontage’, we fill the missing data with the median of the column.

4. Delete feature: We dropped ‘Utilities’, which has the same value for all rows.

5. Feature type transformation: We convert some numerical features into categorical features as they make more sense to be categorical features ('MSSubClass', 'OverallCond', 'YrSold', 'MoSold').

6. Label encoding: For features like 'FireplaceQu', 'BsmtQual', 'BsmtCond', 'GarageQual', 'GarageCond', 'ExterQual', 'ExterCond','HeatingQC', 'PoolQC', 'KitchenQual', 'BsmtFinType1', 'BsmtFinType2', 'Functional', 'Fence', 'BsmtExposure', 'GarageFinish', 'LandSlope', 'LotShape', 'PavedDrive', 'Street', 'Alley', 'CentralAir', 'MSSubClass', 'OverallCond', 'YrSold', 'MoSold', we use label encode to encode them into numerical values.

7. Add new feature: We added a new feature 'TotalSF', which is the sum of 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF'. It makes more sense as all the 3 features measure the living area, so the sum of them will have a higher correlation with the target.

8. Normalize skewed data: I applied log transformation on the high skewed data.

9. One hot encoding: I applied one hot encoding on some data set to convert all features into numerical values to feed into the model.

10. PCA: After all the data process, I use PCA to reduce the dimensions to 15 which can still maintain 87% of the variance.

Implementation

I firstly split the data into training and testing set:

```
from sklearn.model_selection import train_test_split
features = full_data.drop(['SalePrice'], axis=1)
y = full_data['SalePrice']
X_train,X_test, y_train, y_test = train_test_split(features, y, test_size=0.2, random_state=42)
```

Then, I also try to apply PCA on the data so that I can get a dataset with reduced dimensions:

```
from sklearn.decomposition import PCA
import visuals as vs

pca = PCA(n_components=15)
trainPCA = pd.DataFrame(pca.fit_transform(X_train))
testPCA = pd.DataFrame(pca.transform(X_test))
```

Now I have two data sets (one with PCA one without PCA). And I used 3 models to compare their performance. In total there are 6 sets of results to compare.

I use grid search to search for the hyperparameter combinations to optimize the model algorithm. All the code will be in the jupyter notebook file attached. For example, below is the code I used to run the benchmark results:

```
from sklearn.tree import DecisionTreeRegressor
from time import gmtime, strftime, time
from sklearn.model_selection import ShuffleSplit, GridSearchCV
from sklearn.metrics import mean_squared_error

def rmse(prediction, yval):
    return np.sqrt(mean_squared_error(prediction, yval))

def DTR(Xtrain, Xtest, ytrain, ytest):
    params = {'max_depth': list(range(1, 30))}

    cv_sets = ShuffleSplit(n_splits = 5, test_size = 0.2, random_state = 0)
    regressor = DecisionTreeRegressor()
    t0 = time()
    grid = GridSearchCV(estimator = regressor, param_grid = params,
                        scoring = 'neg_mean_squared_error', cv = cv_sets)

    grid = grid.fit(Xtrain, ytrain)
    test_score = rmse(grid.predict(Xtest), ytest)
    print('Time algo takes: {:.3f} seconds'.format(time() - t0))
    print('Train score: {:.4f}'.format(np.sqrt(-grid.best_score_)))
    print('Test error: {:.4f}'.format(test_score))

    print(grid.best_estimator_)
    pass
```

I use grid search to find the optimal parameters for this model, and the parameter range I set is (1, 29) for the maximum tree depth. Then I use PCA features and non-PCA features to run on the model to compare the performance on these two data sets and I get the benchmark results as below:

```
DTR(X_train,X_test, y_train, y_test)
```

```
Time algo takes: 2.687 seconds
Train score: 0.0145
Test error: 0.0152
DecisionTreeRegressor(criterion='mse', max_depth=5, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=None, splitter='best')
```

```
DTR(trainPCA, testPCA, y_train, y_test)
```

```
Time algo takes: 1.333 seconds
Train score: 0.0195
Test error: 0.0210
DecisionTreeRegressor(criterion='mse', max_depth=5, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=None, splitter='best')
```

Refinement

In the exercise, I use grid search to find the optimal hyperparameters. I also use cross validation during the grid search to avoid over-fitting.

At first, when my parameter range is not large enough, I was not able to get the optimal hyperparameters and the model results are not optimized. So the main challenge for me in this task is to get the proper parameter range to tune the model. After setting the range large enough, I was able to get better and more reasonable results. The tuning process is mainly in gradient boost model run.

In the beginning, I set the parameters for grid search as in the below range:

'learning_rate': np.arange(0.05, 0.15, 0.01),
'max_depth': list(range(1, 20)),
'n_estimators': list(range(1, 20))

Then I find that the Gradient Boost results are very close to the Random Forest results, and the optimal parameters the Gradient Boost found are very close to the edge of the upper bound I set for the Grid Search:

'learning_rate' optimal parameter 0.14 and 'n_estimators' optimal parameter 19 are on the upper bound for the parameter range np.arange(0.05, 0.15, 0.01), and list(range(1, 20)).

So I think probably the parameter ranges I set for Grid Search need to be updated. Then I ran with new parameter bounds for Gradient Boost:

'learning_rate': np.arange(0.1, 0.2, 0.01),
'max_depth': list(range(1, 10, 2)),
'n_estimators': list(range(100, 152, 10))

And then after run the model on the new grid search parameters, I can get better results from gradient boost models.

IV. Results

Model Evaluation and Validation

Both Random Forest and Gradient Boost give much better results than Decision Tree. Which makes sense as both models are ensemble models and they usually have better performance.

While I am using grid search to tune the hyperparameters, Decision tree takes very fast to run, while the ensemble models take much longer, especially gradient boost, it takes hours to run. I think the reason is that it has more parameters to tune.

Also, all the PCA data produce relatively worse results. They do run faster while tuning the parameters (which makes sense as the dimension is reduced from 220 to 15), but the performance is much worse. It is reasonable as the 15 PCA features only contains 87% of the total variance. Maybe using more PCA features will produce better results.

I choose to use gradient boost without PCA as the final model, and then did some further tests on it.

Firstly, I tried to run the model multiple times with different random states, and I get the below test scores. It turns out the model is very robust as all the test scores is very close to 0.01.

```
0.009944972390274608
0.00996925080706154
0.01004156224074419
0.009989985802067855
0.009958992912307698
0.00995790759286781
0.009938821580428851
0.009987389356793629
0.01004557859526823
0.010069522645681336
```

I am also trying to run the prediction on different k-fold settings. I update cv_sets from

cv_sets = ShuffleSplit(n_splits = 5, test_size = 0.2, random_state = 0) to

cv_sets = ShuffleSplit(n_splits = 10, test_size = 0.25, random_state = 10)

And I am getting the similar optimal model parameters and test scores for gradient boost. So in summary, I think the model is very robust.

```
GBR_different_CV(X_train,X_test, y_train, y_test)
```

```
Time algo takes: 2122.320 seconds
Train score: 0.0097
Test error: 0.0097
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                          learning_rate=0.11, loss='ls', max_depth=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=150,
                          n_iter_no_change=None, presort='auto',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)
```

Justification

Below are the results I generated for the 3 models with/ without PCA:

	Without PCA	**PCA**
Decision Tree	Time algo takes: 2.687 seconds Train score: 0.0145 Test error: 0.0152 *DecisionTreeRegressor(criterion='mse', max_depth=5, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False,*	Time algo takes: 1.333 seconds Train score: 0.0195 Test error: 0.0210 *DecisionTreeRegressor(criterion='mse', max_depth=5, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False,*

	random_state=None, splitter='best')	*random_state=None, splitter='best')*
Random Forest	Time algo takes: 688.445 seconds Train score: 0.0107 Test error: 0.0116 *RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=28, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=28, n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)*	Time algo takes: 329.238 seconds Train score: 0.0155 Test error: 0.0168 *RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=12, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=28, n_jobs=None, oob_score=False, random_state=None, verbose=0, warm_start=False)*
Gradient Boost	Time algo takes: 5110.769 seconds Train score: 0.0109 Test error: 0.0116 *GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None, learning_rate=0.14, loss='ls', max_depth=6, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=19, n_iter_no_change=None, presort='auto', random_state=None, subsample=1.0, tol=0.0001, validation_fraction=0.1, verbose=0, warm_start=False)*	Time algo takes: 1002.493 seconds Train score: 0.0158 Test error: 0.0175 *GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None, learning_rate=0.14, loss='ls', max_depth=5, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=19, n_iter_no_change=None, presort='auto', random_state=None, subsample=1.0, tol=0.0001, validation_fraction=0.1, verbose=0, warm_start=False)*

Then I find that the Gradient Boost results are very close to the Random Forest results, and the optimal parameters the Gradient Boost found are very close to the edge of the upper bound I set for the Grid Search:

'learning_rate' optimal parameter 0.14 and ‘n_estimators’ optimal parameter 19 are on the upper bound for the parameter range np.arange(0.05, 0.15, 0.01), and list(range(1, 20)).

So I think probably the parameter ranges I set for Grid Search need to be updated. Then I ran with new parameter bounds for Gradient Boost and get the below results:

	Without PCA	**PCA**
Gradient Boost	Time algo takes: 1139.105 seconds Train score: 0.0098 Test error: 0.0100 *GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None, learning_rate=0.15,*	Time algo takes: 2022.305 seconds Train score: 0.0151 Test error: 0.0165 *GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,*

	loss='ls', max_depth=3, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=150, n_iter_no_change=None, presort='auto', random_state=None, subsample=1.0, tol=0.0001, validation_fraction=0.1, verbose=0, warm_start=False)	*learning_rate=0.13999999999999999, loss='ls', max_depth=5, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=131, n_iter_no_change=None, presort='auto', random_state=None, subsample=1.0, tol=0.0001, validation_fraction=0.1, verbose=0, warm_start=False)*

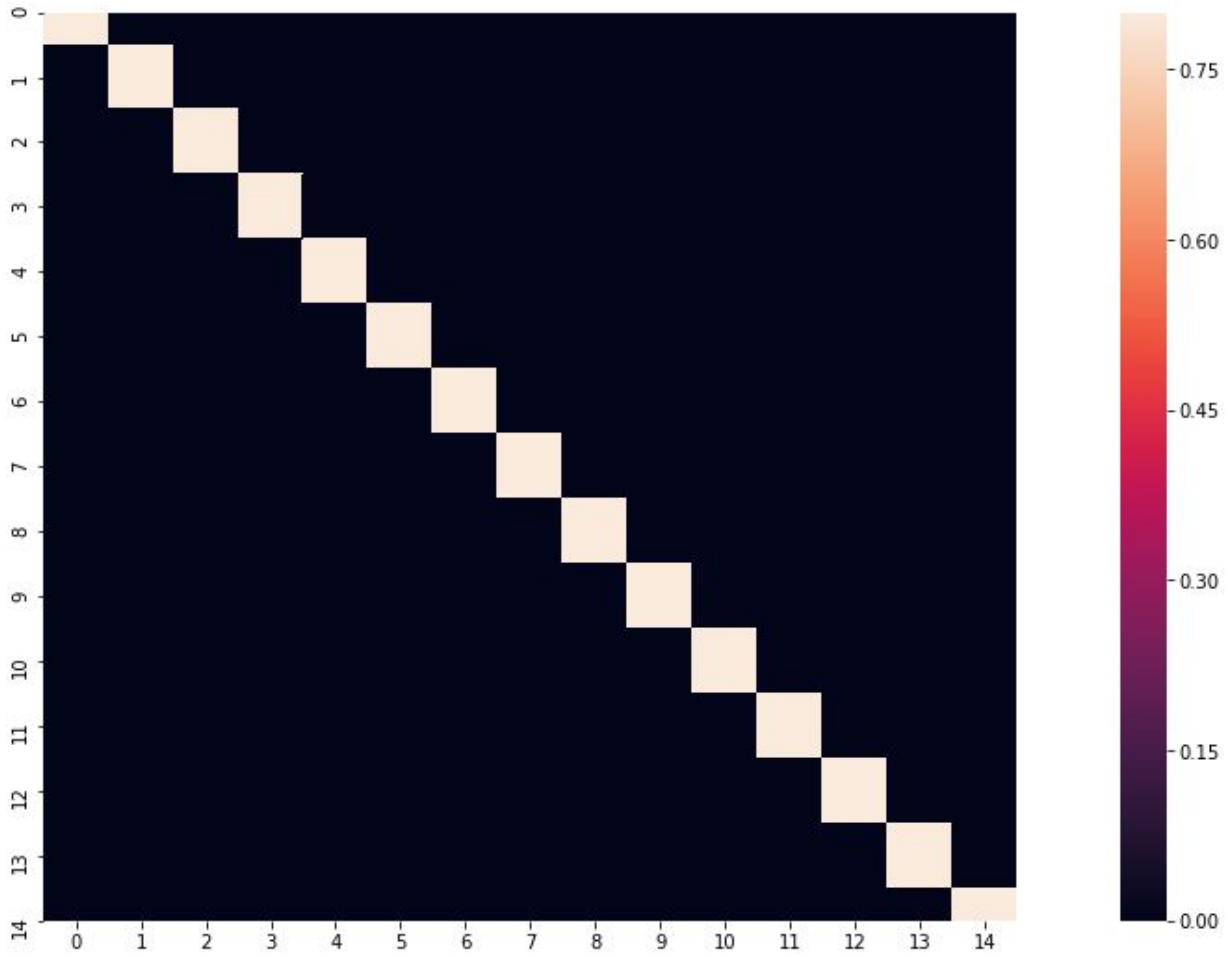
After updating the range of the parameters ('learning_rate': np.arange(0.1, 0.2, 0.01), 'max_depth': list(range(1, 10, 2)), 'n_estimators': list(range(100, 152, 10))), I can further improve the model results.

V. Conclusion

Free-Form Visualization

Below is the correlation heat map for PCA components:

As we can see, the correlation between all the components are close to 0, which means all the 15 PCA features are independent from each other.



Reflection

Below is my entire end-to-end problem solving process:

1. Explore Data
    a. It is important to understand the features and the meaning of them
    b. It is important to understand the feature values and their correlations and distributions
    c. Visualize data is always helpful for us to have a better understanding of data
2. Clean Data
    a. It is important to find out outliers and know whether the outliers need to be deleted or not

    b. It is also important to know how to fill in the missing data, and what values to fill in

3. Feature Treatments
    a. Learns to use one hot encoding to convert categorical data into numerical data
    b. PCA reduces dimensions, but it is not always helpful to generate a better model
4. Model Selection
    a. It is very expensive to optimize the models
    b. It is fun to see how different models have better results

Improvement

I think the improvement can be made on the grid search process. I could try out on more hyperparameter combinations to find a better solution. I did not do that because it is too computationally expensive.

I could also have visualize the relationship between the hyperparameter and the model result. for example, I could plot out the relationship of max_depth vs model result, and see how it could impact the model.

Finally, I could also try out different algorithms. For example, MLP or XGBoost can also be tested to see if they can produce better results.

Citation and Sources

https://www.kaggle.com/serigne/stacked-regressions-top-4-on-leaderboard#Modelling

https://www.kaggle.com/pmarcelino/comprehensive-data-exploration-with-python#5.-Getting-hard-core