Machine Learning Engineer Nanodegree

Capstone Project

Huang Cheng Lin

April 3rd, 2019

I. Definition

Project Overview

The prediction of house prices is clearly a much needed and important part in the real estate world and there is a lot of money involved. An accurate prediction is of high importance for the seller and the buyer as well. With a lot of features and a supposedly easy problem, the task offers a lot of room for data analysis, feature engineering and exploration of algorithms.

Problem Statement

The goal of the project is applying basic machine learning concepts on data collected for housing prices to predict the selling price of a new home. Since this is a typical regression problem, we will use a couple of regression models to predict housing prices and find the best one.

Metrics

In this project, we will use the coefficient of determination, R2, to quantify the model's performance. The coefficient of determination for a model is a useful statistic in regression analysis, as it often describes how "good" that model is at making predictions. The formula of R2 is given by

$$R^2 = 1 - \frac{SSE}{SST}$$

Where SSE is the sum of squared errors of our regression model

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

And SST is the sum of squared errors of our baseline model.

$$SST = \sum_{i=1}^{n} (y_i - \bar{y}_i)^2.$$

The values for R2 range from 0 to 1, which captures the percentage of squared correlation between the predicted and actual values of the target variable. A model with an R2 of 0 is no better than a model that always predicts the mean of the target variable, whereas a model with an R2 of 1 perfectly predicts the target variable. Any value between 0 and 1 indicates what percentage of the target variable, using this model, can be explained by the features.

II. Analysis

Data Exploration

The provided data has 1460 rows and 79 features of which 38 are of numerical nature. A complete description of all the features can be found in data_description.txt.

The statistics of the target variable:

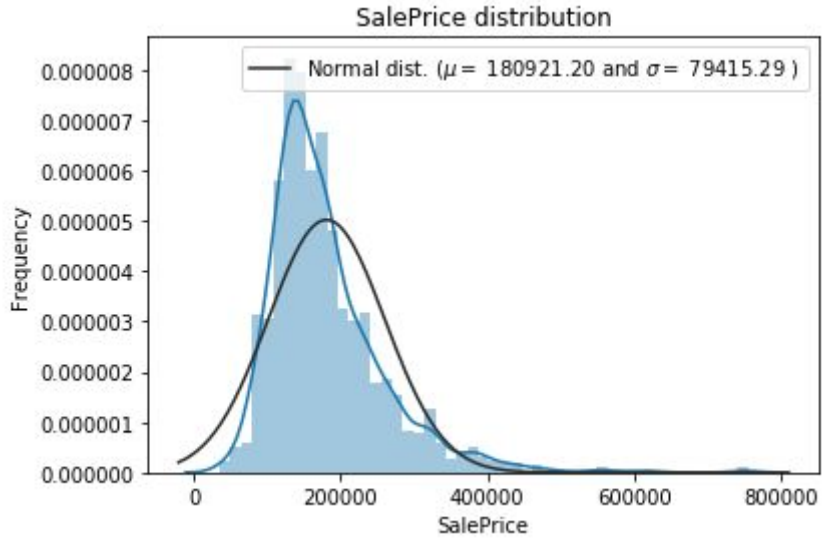
```
price = data['SalePrice']

#descriptive statistics summary
price.describe()
```

```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

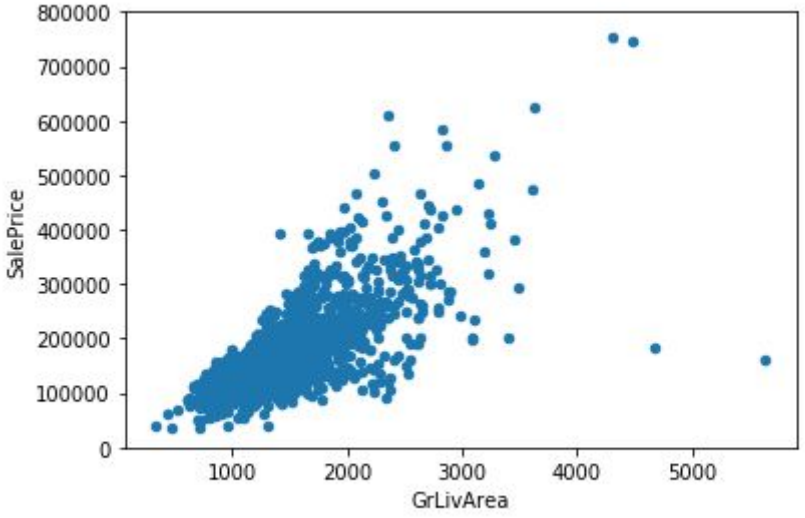
Exploratory Visualization

The distribution of the target variable ‘SalePrice’:

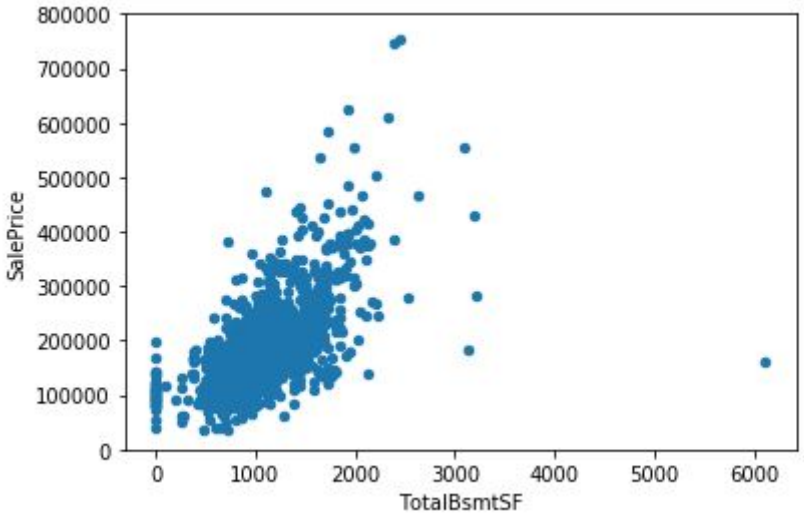


SalePrice is not normal. It shows peakedness, positive skewness. We will come back to this later.

The scatter plot of SalePrice and GrLivArea:



The scatter plot of SalePrice and TotalBsmtSF



Algorithms and Techniques

Regression

Regression analysis is a statistical technique for estimating the relationships among variables. In this project we will be using different regression methods to predict the house price.

Cross-validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

Benchmark

We use LASSO (least absolute shrinkage and selection operator) as our benchmark in this project.

```
from sklearn.linear_model import Lasso
lasso = Lasso()
score = r2_cv(lasso)
print ("LASSO R2 score:")
print("Training: {:.4f}".format(score.mean()))

predictions = (lasso.fit(X_train, y_train)).predict(X_test)
print("Testing: {:.4f}".format(r2_score(y_test, predictions)))
```

```
LASSO R2 score:
Training: 0.1183
Testing: 0.0924
```

The R2 scores are 0.1183 and 0.0924 on training and testing respectively, which means the model is bad at predicting. We will use more regression models to do this job and see which is the best.

III. Methodology

Data Preprocessing

Missing Data

First, we need to handle the missing values in the dataset. Let’s take a look at them:

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

We'll consider that when more than 500 of the data is missing, we should delete the corresponding variable and pretend it never existed. This means that we will not try any trick to fill the missing data in these cases. According to this, there is a set of variables (e.g. 'PoolQC', 'MiscFeature', 'Alley', etc.) that we should delete.

```
# Remove columns with more than 500 missing values
data.drop(['PoolQC', 'MiscFeature', 'Alley', 'Fence', 'FireplaceQu'], axis = 1, inplace = True)
```

In the remaining cases:

LotFrontage: Since the area of each street connected to the house property most likely have a similar area to other houses in its neighborhood , we can fill in missing values by the median LotFrontage of the neighborhood.

GarageYrBlt: No Garage, replacing missing data with None.

GarageType: No Garage, replacing missing data with None.

GarageFinish: No Garage, replacing missing data with None.

GarageQual: No Garage, replacing missing data with None.

GarageCond: No Garage, replacing missing data with None.

BsmtFinType2: No basement, replacing missing data with None.

BsmtExposure: No basement, replacing missing data with None.

BsmtFinType1: No basement, replacing missing data with None.

BsmtCond: No basement, replacing missing data with None.

BsmtQual: No basement, replacing missing data with None.

MasVnrArea: No masonry veneer, replacing missing data with 0.

MasVnrType: No masonry veneer, replacing missing data with None.

Electrical: Replace missing data with the most common value.

```
cols_fill_with_none = ['GarageYrBlt','GarageType',
                       'GarageFinish','GarageQual','GarageCond','BsmtFinType2','BsmtExposure','BsmtFinType1',
                       'BsmtCond','BsmtQual','MasVnrType']
for col in cols_fill_with_none:
    data[col] = data[col].fillna('None')


# LotFrontage
data['LotFrontage'] = data.groupby("Neighborhood")["LotFrontage"].transform(lambda x: x.fillna(x.median()))

# MasVnrArea
data['MasVnrArea'] = data['MasVnrArea'].fillna(0)

# Electrical
data['Electrical'] = data['Electrical'].fillna(data['Electrical'].mode()[0])
```

```
data.isnull().sum().max()
```

```
0
```

Outliers

The primary concern here is to establish a threshold that defines an observation as an outlier. To do so, we'll standardize the data. In this context, data standardization means converting data values to have mean of 0 and a standard deviation of 1.

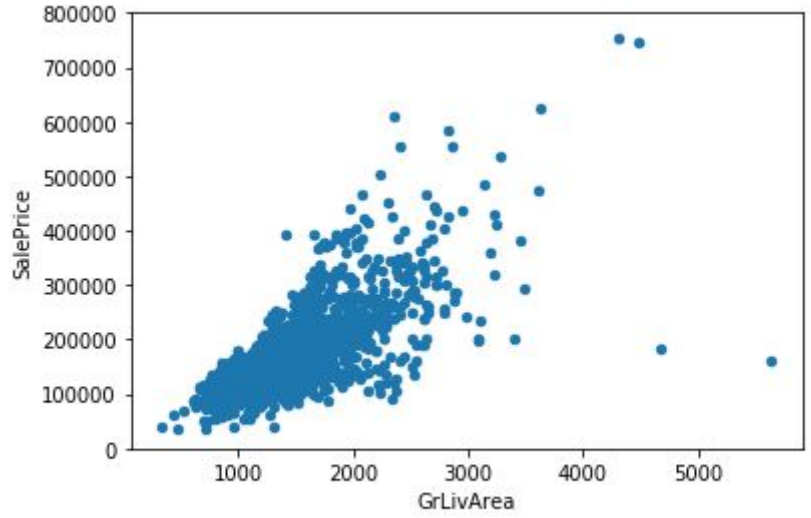
```
from sklearn.preprocessing import StandardScaler

#standardizing data
saleprice_scaled = StandardScaler().fit_transform(price[:,np.newaxis]);
low_range = saleprice_scaled[saleprice_scaled[:,0].argsort()][:10]
high_range= saleprice_scaled[saleprice_scaled[:,0].argsort()][-10:]
print('outer range (low) of the distribution:')
print(low_range)
print('\nouter range (high) of the distribution:')
print(high_range)
```

```
outer range (low) of the distribution:
[[-1.83870376]
 [-1.83352844]
 [-1.80092766]
 [-1.78329881]
 [-1.77448439]
 [-1.62337999]
 [-1.61708398]
 [-1.58560389]
 [-1.58560389]
 [-1.5731    ]]

outer range (high) of the distribution:
[[3.82897043]
 [4.04098249]
 [4.49634819]
 [4.71041276]
 [4.73032076]
 [5.06214602]
 [5.42383959]
 [5.59185509]
 [7.10289909]
 [7.22881942]]
```

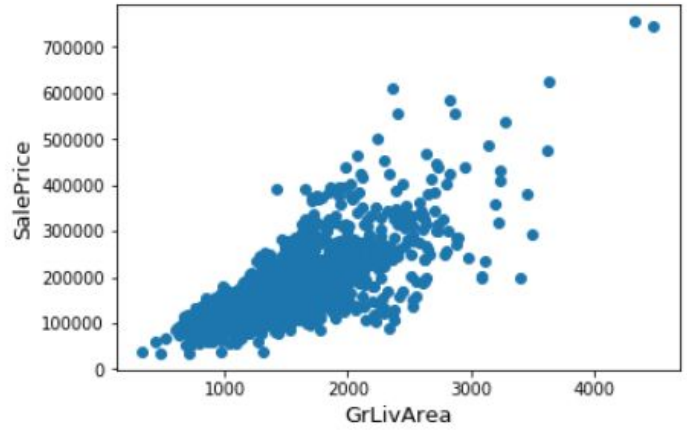
For now, we'll not consider any of these values as an outlier but we should be careful with those two 7.x values.



We can see at the bottom right two with extremely large GrLivArea that are of a low price. These values are huge outliers. Therefore, we can safely delete them.

```
#Deleting outliers
data = data.drop(data[(data['GrLivArea']>4000) & (data['SalePrice']<300000)].index)

#Check the graphic again
fig, ax = plt.subplots()
ax.scatter(data['GrLivArea'], data['SalePrice'])
plt.ylabel('SalePrice', fontsize=13)
plt.xlabel('GrLivArea', fontsize=13)
plt.show()
```



Abnormalities

First we are dealing with the target variable ‘SalePrice’. We use log transformation to normalize it.

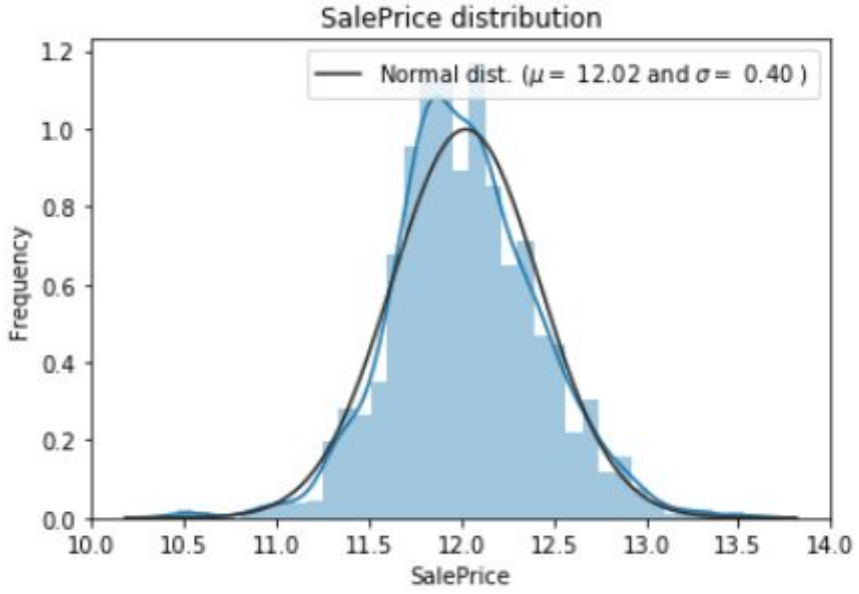
```
#We use the numpy fuction log1p which  applies log(1+x) to all elements of the column
data['SalePrice'] = np.log1p(data['SalePrice'])

#Check the new distribution
sns.distplot(data['SalePrice'] , fit=norm);

# Get the fitted parameters used by the function
(mu, sigma) = norm.fit(data['SalePrice'])
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))

#Now plot the distribution
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)],
            loc='best')
plt.ylabel('Frequency')
plt.title('SalePrice distribution')
```

```
 mu = 12.02 and sigma = 0.40
```



Then we check the skew of all numerical features.

```
numeric_feats = data.dtypes[data.dtypes != "object"].index

# Check the skew of all numerical features
skewed_feats = data[numeric_feats].apply(lambda x: skew(x.dropna())).sort_values(ascending=False)
print("\nSkew in numerical features: \n")
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness.head(10)
```

```
Skew in numerical features:
```

	Skew
MiscVal	24.426546
PoolArea	15.927003
LotArea	12.556596
3SsnPorch	10.282886
LowQualFinSF	8.992490
KitchenAbvGr	4.478508
BsmtFinSF2	4.245879
ScreenPorch	4.113045
BsmtHalfBath	4.094248
EnclosedPorch	3.082631

We use boxcox function to fix them.

```
from scipy.special import boxcox1p
skewed_features = skewness.index
lm = 0.5
for feature in skewed_features:
    data[feature] = boxcox1p(data[feature], lm)
```

Dummy Encoding

Finally we do dummy encoding on our data.

```
price = data['SalePrice']
data.drop(['SalePrice'], axis = 1, inplace = True)
final_data = pd.get_dummies(data)
print(final_data.shape)
```

```
(1458, 400)
```

Implementation

Before training any data, we need to split out our data into a training set and a test set. Scikit-learn provides a helpful function ‘train_test_split’ for partitioning data.

We also need to define a cross validation strategy. We use the cross_val_score function in Scikit-learn and choice r2 score as our performance measurement.

```
X_train, X_test, y_train, y_test = train_test_split(final_data, price, test_size=0.25, random_state=42)

#Validation function
n_folds = 4

def r2_cv(model):
    kf = KFold(n_folds, random_state=42).get_n_splits(X_train)
    r2_scores = cross_val_score(model, X_train, y_train, scoring="r2", cv = kf)
    return(r2_scores)
```

Here are the regression models used to predict the prices:

1. LASSO (benchmark)
2. Decision Tree Regression
3. Support Vector Regression
4. AdaBoost Regression
5. Kernel Ridge Regression
6. GradientBoosting Regression

Refinement

After trying all the regression models listed above, we will choose from them the best model. We will then perform a grid search optimization for the model over the entire training set (X_train and y_train) by tuning at least one parameter to improve upon the untuned model's R2 score.

IV. Results

Model Evaluation and Validation

Here are the R2 scores for each model:

Model	R2 score (training)	R2 score (testing)
LASSO	0.1183	0.0924
Decision Tree	0.6921	0.7481
Support Vector	0.1991	0.2121
AdaBoost	0.8122	0.8119
Kernel Ridge	0.8969	0.9129
GradientBoosting	0.8904	0.9151

According to the result, we think that GradientBoosting is the best maodel for predicting prices in this dataset. Although Kernel Ridge performed quite good as well, GradientBoosting got little higher R2 score in testing data. It's an important factor for a good model to predict well on the unseen data, plus Gradient boosting is fairly robust to overfitting. Therefore we choose it to do model tuning.

The parameters for the grid search are:

1. **n_estimators:** [100,200,300]
2. **alpha**: [0.5,0.7,0.9]
3. **max_depth**: [2,3,4]
4. **learning_rate**: [0.05,0.075,0.1,0.2,0.3]

After optimization, the R2 score:

Model	R2 score
Unoptimized Model	0.9148
Optimized Model	0.9155

the best parameters for our model are:

n_estimators	0.7
alpha	0.075
max_depth	3
learning_rate	300

Justification

Model	R2 Score
Benchmark model(LASSO)	0.0923
Final model (GradientBoosting)	0.9155

Compared to the R2 score generated from the benchmark(LASSO), the final model performs much better. The final model gets good R2 scores on both training set and testing set. Therefore, we believe that our final model is suitable for solving the problem.

V. Conclusion

Reflection

In the end we have achieved a model with a decent precision. The predictive power of the model might be comparable to a subject matter expert though this would have been examined further.

One thing I'd like to talk about is the linear regression model - Kernel Ridge. Surprisingly it do really well with little feature engineering. The key point is to to log_transform the numeric variables since most of them are skewed.

Improvement

Several additional aspects could be explored:

- Inflation and regression over the observed time span could be addressed.
- More feature engineering like dimensionality reduction could be done.
- Different metrics could be used to judge the performance.
- More regression methods like stacking could be used.