Machine Learning Engineer Nanodegree

Capstone Project-Kaggle House Prices: Advanced Regression Techniques

Süleyman Diker
July 07st, 2018

I. Definition

Estimates will be made about the monetary value of a house over the most suitable model using real house property and price data.After an exhaustive feature engineering, multiple machine learning regression algorithms will be trained and tuned, to obtain a better predictive performance.

Project Overview

The purpose of this project is to estimate the price of house in Iowa with the help of data containing 81 property definitions. The data for this project was obtained from Kaggle in the House Prices: Advanced Regression Techniques competition.

Problem Statement

Our main goal is to estimate the house price, which we have with the data we have, as accurate as possible. We can list the most basic items that we need to solve in the framework of this problem as follows:

- Exploratory data analysis: The nature of the features in the data is crucial for accurate estimation of missing values of the existing properties, number of outliers and detection of outliers.
- Feature preprocessing: It is a process of visualizing the data to detect if there is an inconsistency or incomplete data, and to process the data in this direction.
- Benchmark modeling: To solve the problem we start by creating a model using a standard technique.
- Model improvement: We try to optimize our model by setting parameters and using learning techniques to improve the model's performance

Metrics

In this project we will measure the success of our model using the RMSLE (Root Mean Squared Logarithmic Error) metrics, which calculate the difference between the actual price and the model predicted by our model.

As the difference between the actual value of the house and the estimated value increases, the value of RMSLE also increases, so we need to have the minimum RMSLE value.

II. Analysis

Data Exploration

The training set includes categorical and numerical features as well as some missing values.Since we often can not have a complete dataset in every direction, it is important that we do a good analysis of the dataset.Our data set is divided into two parts as training set and test set.The training set consists of 1460 rows with 81

features. The test set consists of 1459 rows with 80 features.Our test dataset is missing a column according to our training dataset. For this reason, we will guess the SalePrice column by modeling the data through our training dataset.

When we examine the house price histogram in Figure 1, we see that the skewed value is high. (Skew Value: 1.8828757597682129) For this reason, we need log transformations to normalize the skewed value, so we use RMSLE as our evaluation metric. After the log transformations, we see that the skewed value falls and that the Figure 2 is a more balanced graph. (Skew Value: 0.12133506220520406)

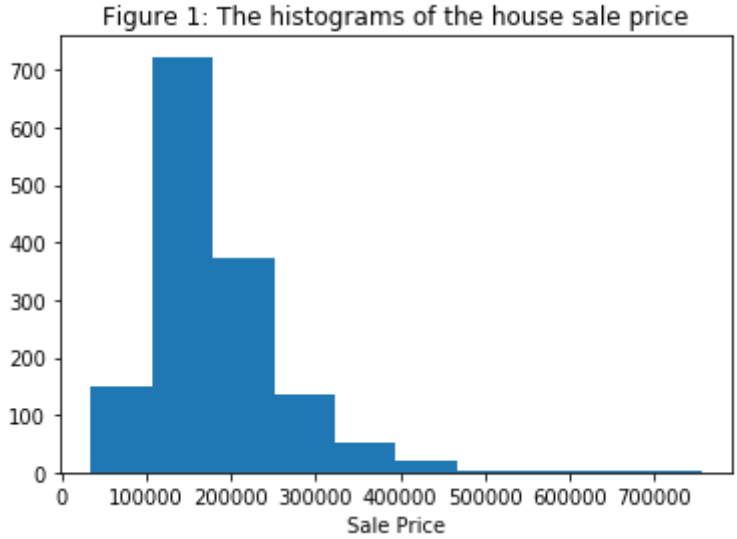
In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

print ("Train data shape:", train.shape)
print ("Test data shape:", test.shape)
```

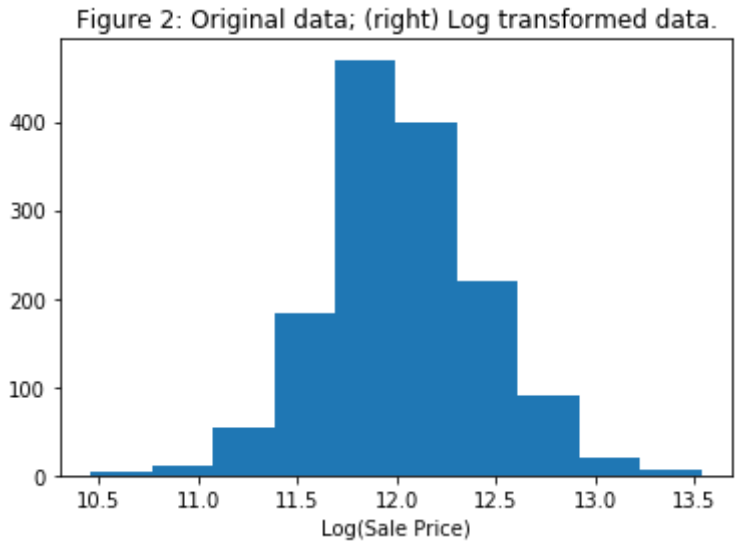
```
('Train data shape:', (1460, 81))
('Test data shape:', (1459, 80))
```

In [2]:

```
plt.xlabel("Sale Price")
plt.title("Figure 1: The histograms of the house sale price")
plt.hist(train.SalePrice)
plt.show()
print ("Sale Price skew is:", train.SalePrice.skew())

target = np.log(train.SalePrice)
plt.title("Figure 2: Original data; (right) Log transformed data.")
plt.xlabel("Log(Sale Price)")
plt.hist(target)
plt.show()
print ("Log(Sale Price) skew is:", target.skew())
```



('Sale Price skew is:', 1.8828757597682129)



('Log(Sale Price) skew is:', 0.12133506220520406)

Exploratory Visualization

The correlation values between the numerical variables and the home sales are as follows. We can not say that each of the feature has the same effect on the sales value of the house, each feature has positive, negative or neutral effects on the sales value.

In [3]:

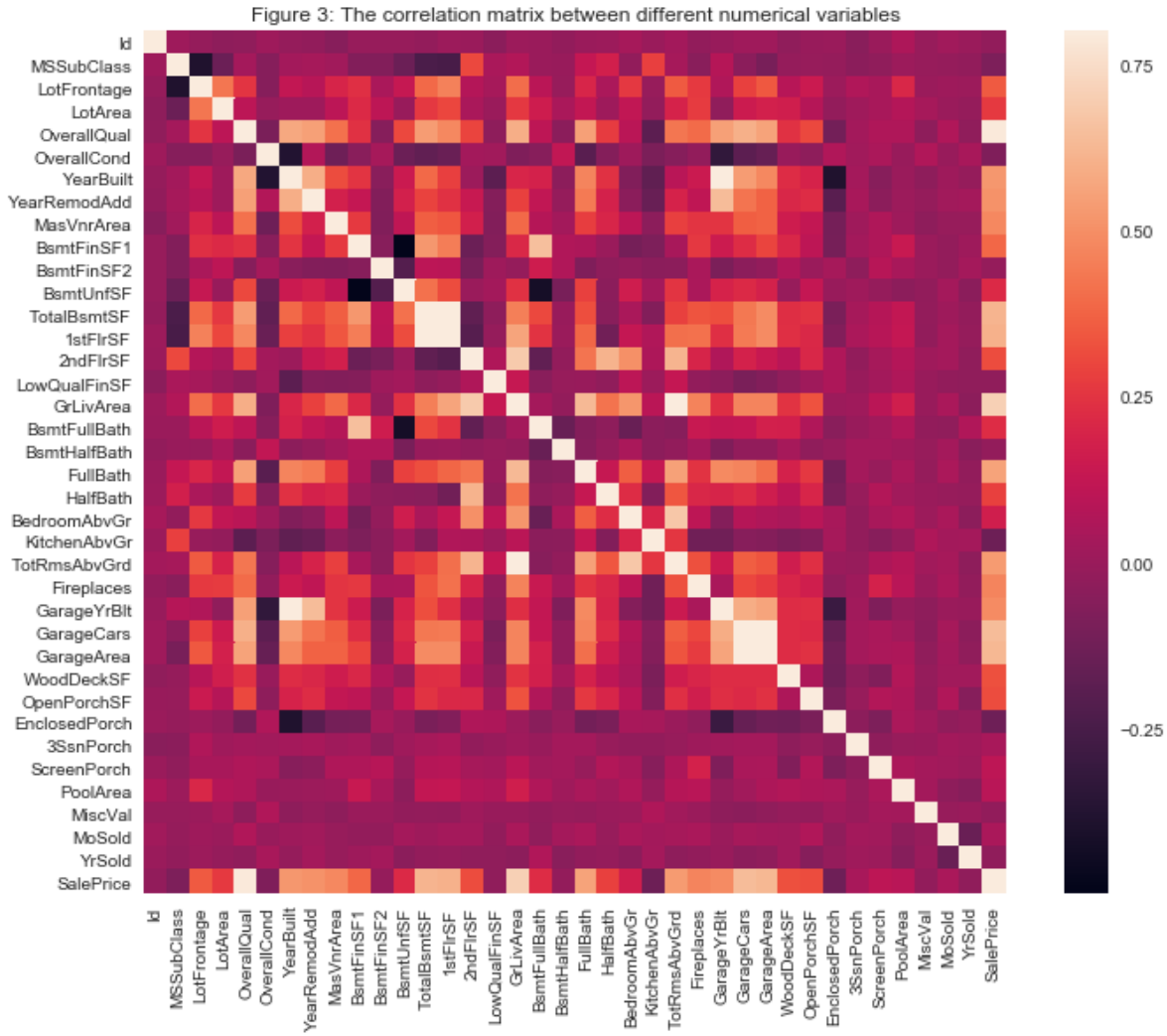
```
numeric_features = train.select_dtypes(include=[np.number])
numeric_features.dtypes
corr = numeric_features.corr()

print (corr['SalePrice'].sort_values(ascending=False)[0:], '\n')
```

```
(SalePrice        1.000000
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
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
Name: SalePrice, dtype: float64, '\n')
```

We can see the correlation matrix between the different numerical variables below:

In [4]:

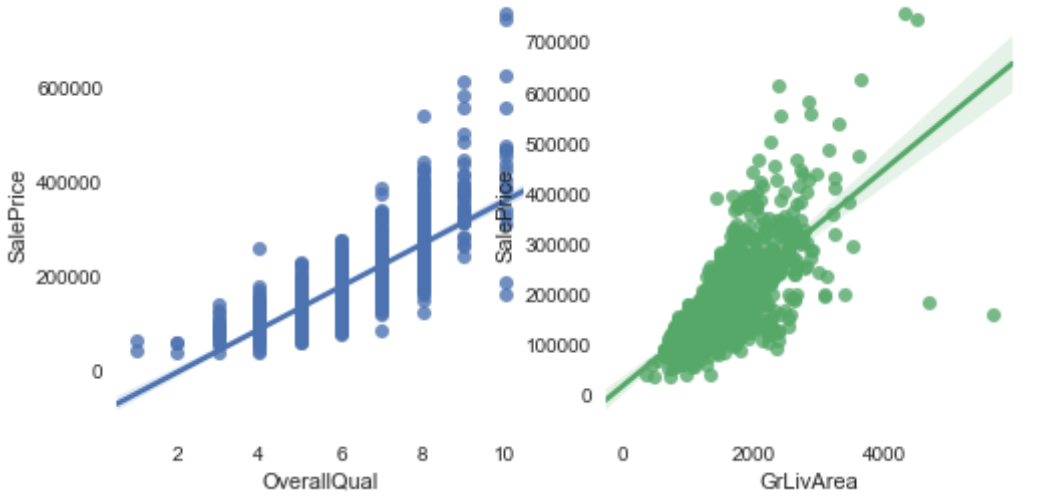
```
import matplotlib.pyplot as plt
import seaborn as sns

corr_matrix = train.corr()
plt.subplots
sns.set(rc={'axes.facecolor':'white', 'figure.facecolor':'white'})
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corr_matrix, vmax=.8, square=True)
sns.heatmap
plt.title("Figure 3: The correlation matrix between different numerical variable
plt.show()
```

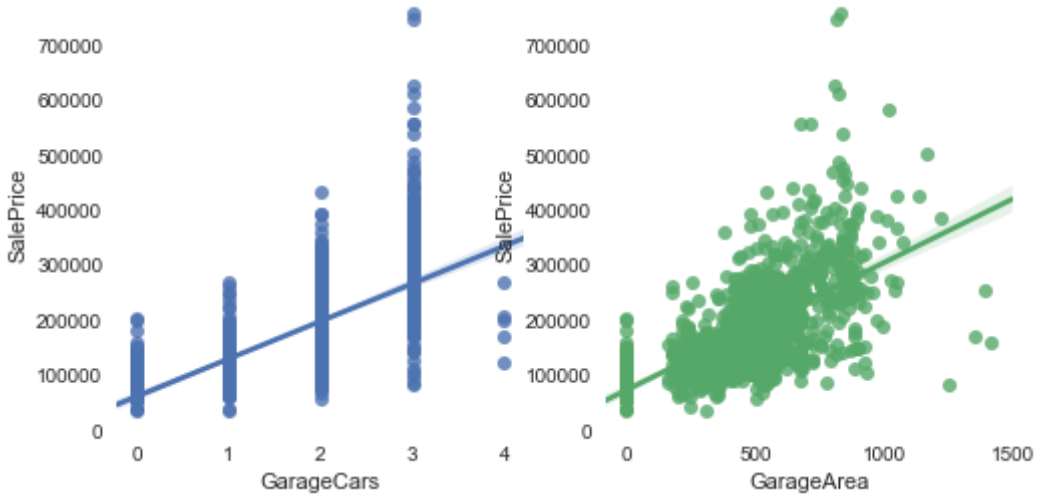


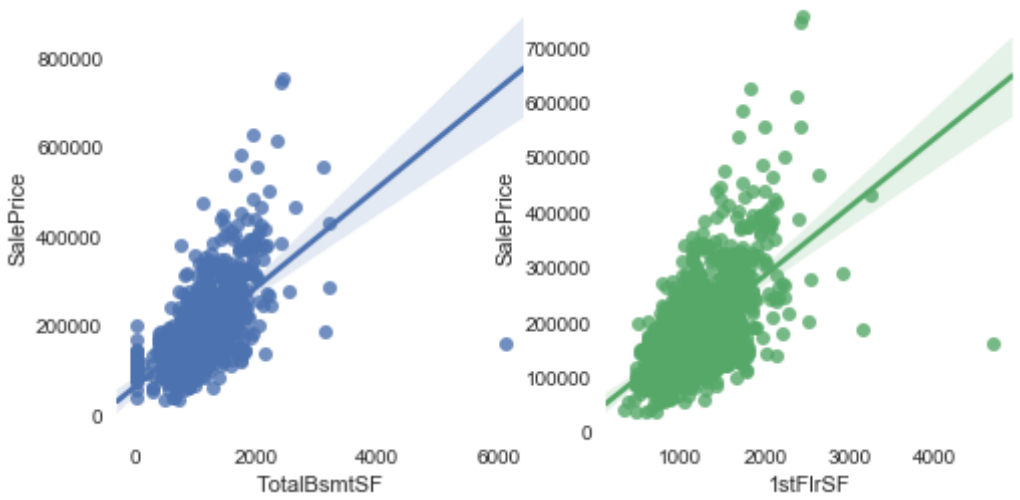
We can look at the top 6 feature distribution charts that have the strongest correlation that affects home selling prices:

In [5]:

```
fig1, (ax1, ax2) = plt.subplots(ncols=2,figsize = (8, 4))
sns.regplot(x = 'OverallQual', y = 'SalePrice', data = train,ax=ax1)
sns.regplot(x = 'GrLivArea', y = 'SalePrice', data = train,ax=ax2)
plt.show()

fig2, (ax1, ax2) = plt.subplots(ncols=2,figsize = (8, 4))
sns.regplot(x = 'GarageCars', y = 'SalePrice', data = train,ax=ax1)
sns.regplot(x = 'GarageArea', y = 'SalePrice', data = train,ax=ax2)
plt.show()

fig3, (ax1, ax2) = plt.subplots(ncols=2,figsize = (8, 4))
sns.regplot(x = 'TotalBsmtSF', y = 'SalePrice', data = train,ax=ax1)
sns.regplot(x = '1stFlrSF', y = 'SalePrice', data = train,ax=ax2)
plt.show()


```





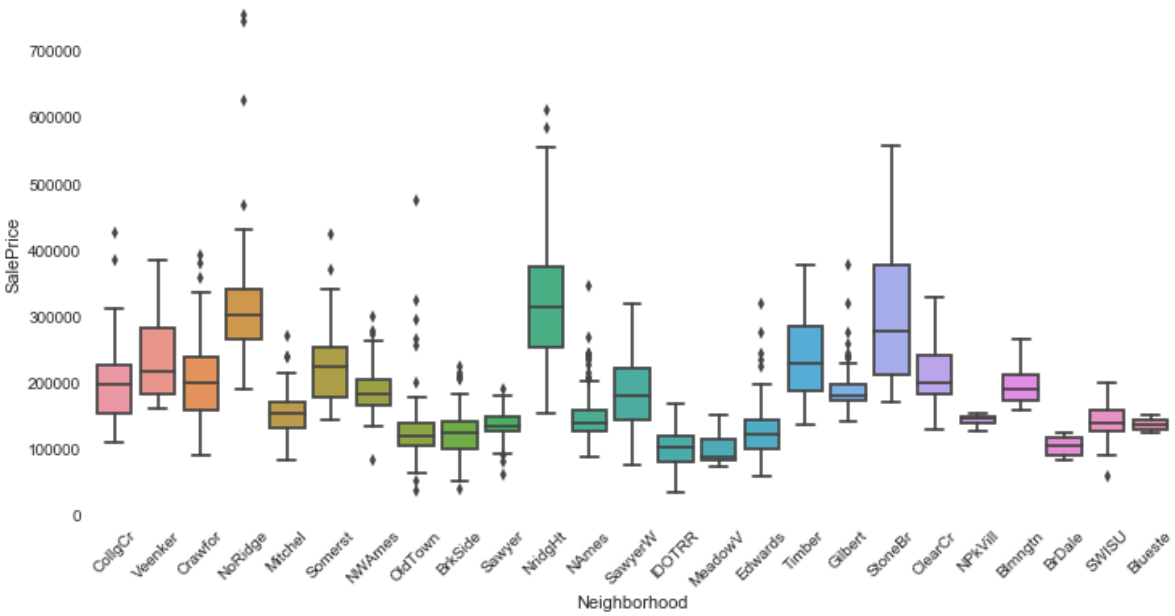


The relationship between the neighborhood and the resale price is clearly seen in Figure 5..We observe that the neighborhood feature is a feature that seriously affects the price of the house. For this reason, in categorical variables, all values should be evaluated as equal.If values such as 0 and 1 are given to convert to number, we will not reflect the effect of this variable correctly on the model.So we should pay particular attention to the transformations of categorical variables.

In [6]:

```
plt.figure(figsize = (12, 6))
sns.boxplot(x = 'Neighborhood', y = 'SalePrice',  data = train)
xt = plt.xticks(rotation=45)
plt.title("Figure 4:Boxplot of sale price in different neighborhood.")
plt.show()
```

Figure 4:Boxplot of sale price in different neighborhood.



Algorithms and Techniques

Our main goal is to make the best guess scoring by doing experiments within the scope of advanced regression techniques, so I tried the community learning method.This technique aims to learn the weighted combinations of basic models.

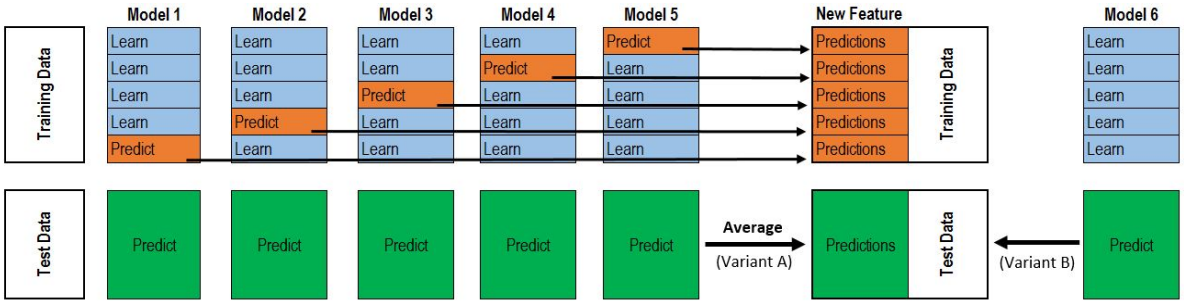
This technique aims to learn the weighted combinations of basic models.This technique is also called technical stacking.This technique combines all model predictions to minimize the generalization error and tries to capture the most accurate prediction score.

we see two levels of five-fold stacking schemes as schematics below.First, the training dataset need to be split into 5 folds. Second, we iterate over this 5 folds training dataset. In each iteration, each base model will be trained using 4 folds and predict on the hold out fold. At the same time, each base model also need to provide a prediction on the entire test dataset. After the iteration over all folds, we will have the prediction of the entire training dataset for each model and 5 copies of the prediction of the entire test dataset for each model. Finally, we train second level model, or stacker, using the prediction in the training dataset as new features and use the average of the 5 copies of the test dataset predictions as the test input for the trained model to provide the final prediction.

In [7]:

```
1 from IPython.display import Image
2 Image(filename='images/QBuDOjs.jpg')
```

Out[7]:



Benchmark

The Random Forest regression is selected as our benchmark model for this project. The default parameters in scikit-learn package will be used as the benchmark.

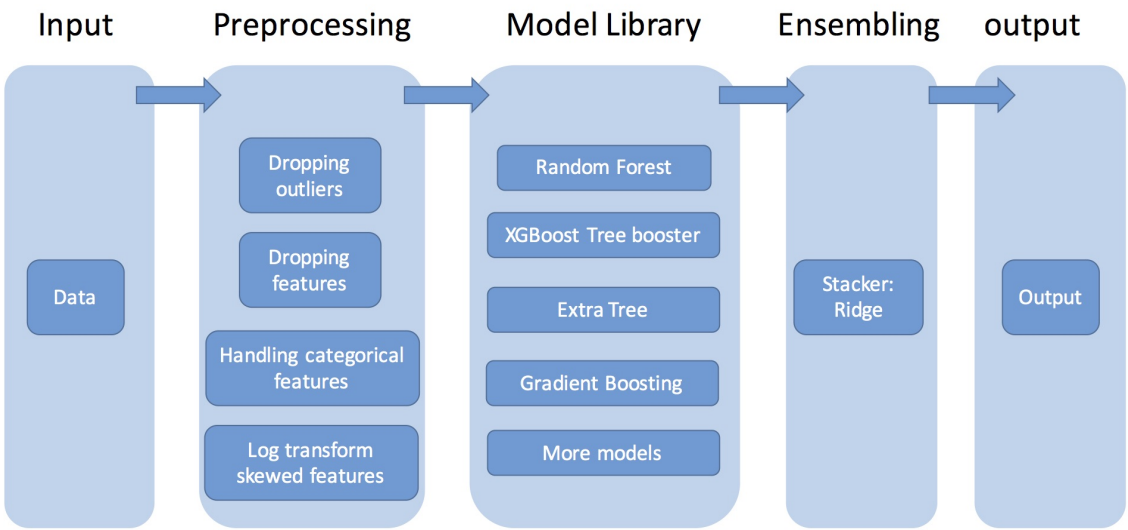
III. Methodology

In general, the methodology and modeling approach is illustrated in the flow chart below.

In [8]:

```
from IPython.display import Image
Image(filename='images/flowchart.jpg')
```

Out[8]:



Data Preprocessing

We can summarize the data pre-processing operations as follows:

- Outliers deletion: We examined the data intensities of the features with the highest correlation effect.We have found data that are very different from the data intensities of these properties and we have eliminated the data as follows:

In [21]:

```
outlier_idx = []

for i in train[train['GrLivArea'] > 4000]['GrLivArea'].index:
    outlier_idx.append(i)

for i in train[train['GarageArea'] > 1200]['GarageArea'].index:
    outlier_idx.append(i)


for i in train[train['TotalBsmtSF'] > 3000]['TotalBsmtSF'].index:
    outlier_idx.append(i)

for i in train[train['1stFlrSF'] > 3000 ]['1stFlrSF'].index:
    outlier_idx.append(i)

for i in train[train['MasVnrArea'] > 800]['MasVnrArea'].index:
    outlier_idx.append(i)

for i in train[train['BsmtFinSF1'] > 2000]['BsmtFinSF1'].index:
    outlier_idx.append(i)

for i in train[train['LotFrontage'] > 150]['LotFrontage'].index:
    outlier_idx.append(i)

for i in train[train['WoodDeckSF'] > 600]['WoodDeckSF'].index:
    outlier_idx.append(i)

for i in train[train['2ndFlrSF'] > 1650]['2ndFlrSF'].index:
    outlier_idx.append(i)

for i in train[train['OpenPorchSF'] > 400]['OpenPorchSF'].index:
    outlier_idx.append(i)

for i in train[train['LotArea'] > 50000]['LotArea'].index:
    outlier_idx.append(i)

for i in train[train['BsmtUnfSF'] > 2000]['BsmtUnfSF'].index:
    outlier_idx.append(i)

print(outlier_idx)

train.drop(train.index[outlier_idx],inplace=True)


```

```
[523, 691, 1182, 1298, 581, 1298, 440, 523, 1298, 523, 1298, 70, 224,
 349, 691, 798, 1169, 523, 898, 1182, 1298, 1182, 1298, 691, 1169, 118
2, 523, 313, 1298, 224, 477, 581]
```

- Feature selection: The categorical variables with the greatest number of missing values are Alley, FirePlaceQu, PoolQC, Fence and MiscFeature. We can see this clearly in the following table. We delete these features with very high amounts of missing data as follows:

In [10]:

```
total = train.isnull().sum().sort_values(ascending = False)
percent = (train.isnull().sum() / train.isnull().count()).sort_values(ascending
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(10)
```

Out[10]:

	Total	Percent
PoolQC	1402	0.997155
MiscFeature	1357	0.965149
Alley	1320	0.938834
Fence	1132	0.805121
FireplaceQu	684	0.486486
LotFrontage	251	0.178521
GarageCond	79	0.056188
GarageType	79	0.056188
GarageYrBlt	79	0.056188
GarageFinish	79	0.056188

In [11]:

```
all_data = pd.concat((train.loc[:,'MSSubClass':'SaleCondition'],
                      test.loc[:,'MSSubClass':'SaleCondition']))

to_delete = ['Alley','FireplaceQu','PoolQC','Fence','MiscFeature']
all_data = all_data.drop(to_delete,axis=1)
```

- Log transform skewed features: All features that have a skewness larger than 0.75 are log transformed.

In [13]:

```
from scipy.stats import skew

numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index
skewed_feats = train[numeric_feats].apply(lambda x: skew(x.dropna()))
skewed_feats = skewed_feats[skewed_feats > 0.75]
skewed_feats = skewed_feats.index
all_data[skewed_feats] = np.log1p(all_data[skewed_feats])
```

- Handle categorical features: The categorical features are handled by the pandas get_dummies function that converts categorical variables into dummy or indicator variables.We do this transformation in the following way:

In [14]:

```
all_data = pd.get_dummies(all_data)
all_data = all_data.fillna(all_data.mean())
```

Implementation

Before applying the ensemble learning method, I trained and tuned 4 base model including Random Forest regressor, Extra Trees regressor, Gradient Boosting regressor, Extreme Gradient Boosting regressor.

At the beginning of the ensembling implementation, I use only 4 base models to form my model library, including Random Forest regressor, Extra Trees regressor, Gradient Boosting regressor, Extreme Gradient Boosting regressor. And I use the Ridge Regression as my second level model. The idea of the stacking approach is to use the predictions from the first level model as the new features for the second level model and make prediction based on that.

The base models should be as unrelated as possible and this is why we tend to include more models in the ensemble even though they may not perform well. For my final, I use a much larger much library including Random Forest regressor, Extra Trees regressor, Gradient Boosting regressor, Extreme Gradient Boosting regressor, LassoCV regressor, K neighbors regressor, LassoLarsCV regressor, Elastic Net regressor, Support Vector Machine regressor. The Extreme Gradient Boosting regressor is from the XGBoost package, all other techniques are from scikit- learn package. The basic idea of the final model is to use a larger base model library in order to achieve better results. Some of the base models are used only with their scikit-learn default parameter values without fine tuning.

There are two challenging parts during the coding process. The first one is that we need to make sure the feature for the final second level prediction is based on the prediction of the first level base models. Thus, we also need to use the model prediction as the features for the test dataset. The second one is how exactly should we choose the base models. This is a tricky and not fully addressed question in this project since there are so many models to select.

Refinement

The refinement is conducted using the grid search technique. For the learning purpose of this project, I only use a few tunable parameters. Before the parameter tuning, I just use the scikit-learn default parameter values for each model. After the refinement of the base models, we using stacking technique to ensembling them together to provide the final submission.

Random forecast regression: Best Params:{} Best CV Score: 0.146323732016

eXtreme Gradient Boosting regression: Best Params:{} Best CV Score: 0.127444960531

Extra trees regression: Best Params:{} Best CV Score: 0.149811744332

Gradient boosted tree regression: Best Params:{} Best CV Score: 0.126801721411

IV. Results

Model Evaluation and Validation

One interesting observation during the model ensembling process is that increasing the size of the model library can potentially increase both the validation score and the real score. A variety of base models are included in the model library with their scikit-learn default parameter values.I need to acknowledge that the approach is somewhat brute force and a more elegant way to do this is to find the base models to be as uncorrelated as possible.

Justification

In summary, the ensemble approach in this project greatly improve the performance of the model in terms of the Kaggle Leaderboard score and rank from the benchmark random forecast regression ranked.The result is quite encouraging and opens a lot more to think, learn and explore.

V. Conclusion

Free-Form Visualization

As my ultimate goal in this project is to build a model as accurate as possible measure by the Kaggle Leaderboard score, I will use the Kaggle Leaderboard score at the different model development stage as my final project visualization.This project opens several avenues for future work. One part is for data preprocessing, such as more advanced outlier detection techniques, more advanced feature selection techniques, fine tuning of the log transform threshold of the skewed variable and so on. More creative feature engineering will be also extremely useful. Another part is for the regression techniques. My intuition for ensemble learning technique is that a better score may be achieved by increasing the size of the model library.The dilemma here is a tradeoff between the amount of time and engineering efforts you want to spend and the actual value of the tiny improvement.

Reflection

For this project, I started with an exploratory data analysis, in which I found that there are some features being skewed, some features are categorical, and some missing values. Several data preprocessing steps were conducted in order to make the data ready for the machine learning modeling part. For the modeling part, I firstly trained several base models and tuned their parameters. Then, I applied the ensemble learning method to stack all the base models together to provide the final submission. One of difficult parts is data preprocessing since it is hard to explain in a rigorous way of why I did a specific choice to improve the model. There are a lot of exploratory tests and investigations to justify the choice only based on the local cross validation scores. The other challenge is the stacking approach itself since it is more complicated than a single model approach. It is quite easy to use the wrong features for the stacker since it makes prediction based on the predictions of all the base models. The key to success is to write a pseudocode and check it carefully before writing the production code.

Improvement

This project opens several avenues for future work. One part is for data preprocessing, such as more advanced outlier detection techniques, more advanced feature selection techniques, fine tuning of the log transform threshold of the skewed variable and so on. More creative feature engineering will be also extremely useful. Another part is for the regression techniques. My intuition for ensemble learning technique is that a better score may be achieved by increasing the size of the model library.The dilemma here is a tradeoff between the amount of time and engineering efforts you want to spend and the actual value of the tiny improvement.

In [23]:

```
import numpy as np
import pandas as pd
import datetime
from sklearn.cross_validation import KFold
from sklearn.cross_validation import train_test_split
import time
from sklearn import preprocessing
from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor, Gradie
from sklearn.grid_search import GridSearchCV
from sklearn.cross_validation import ShuffleSplit
from sklearn.metrics import make_scorer, mean_squared_error
from sklearn.linear_model import Ridge, LassoCV,LassoLarsCV, ElasticNet
from sklearn.kernel_ridge import KernelRidge
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from scipy.stats import skew

def create_submission(prediction,score):
    now = datetime.datetime.now()
    sub_file = 'submission_'+str(score)+'_'+str(now.strftime("%Y-%m-%d-%H-%M"))
    print ('Creating submission: ', sub_file)
    pd.DataFrame({'Id': test['Id'].values, 'SalePrice': prediction}).to_csv(sub


def data_preprocess(train,test):

    outlier_idx = []

    for i in train[train['GrLivArea'] > 4000]['GrLivArea'].index:
        outlier_idx.append(i)

    for i in train[train['GarageArea'] > 1200]['GarageArea'].index:
        outlier_idx.append(i)


    for i in train[train['TotalBsmtSF'] > 3000]['TotalBsmtSF'].index:
        outlier_idx.append(i)

    for i in train[train['1stFlrSF'] > 3000 ]['1stFlrSF'].index:
        outlier_idx.append(i)

    for i in train[train['MasVnrArea'] > 800]['MasVnrArea'].index:
        outlier_idx.append(i)

    for i in train[train['BsmtFinSF1'] > 2000]['BsmtFinSF1'].index:
        outlier_idx.append(i)

    for i in train[train['LotFrontage'] > 150]['LotFrontage'].index:
        outlier_idx.append(i)

    for i in train[train['WoodDeckSF'] > 600]['WoodDeckSF'].index:
        outlier_idx.append(i)

    for i in train[train['2ndFlrSF'] > 1650]['2ndFlrSF'].index:
        outlier_idx.append(i)

    for i in train[train['OpenPorchSF'] > 400]['OpenPorchSF'].index:
        outlier_idx.append(i)
```

```
60
61      for i in train[train['LotArea'] > 50000]['LotArea'].index:
62          outlier_idx.append(i)
63
64      for i in train[train['BsmtUnfSF'] > 2000]['BsmtUnfSF'].index:
65          outlier_idx.append(i)
66
67      print(outlier_idx)
68
69      train.drop(train.index[outlier_idx],inplace=True)
70      all_data = pd.concat((train.loc[:,'MSSubClass':'SaleCondition'],
71                            test.loc[:,'MSSubClass':'SaleCondition']))
72
73      to_delete = ['Alley','FireplaceQu','PoolQC','Fence','MiscFeature']
74      all_data = all_data.drop(to_delete,axis=1)
75      train["SalePrice"] = np.log1p(train["SalePrice"])
76
77      numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index
78      skewed_feats = train[numeric_feats].apply(lambda x: skew(x.dropna())) #comp
79      skewed_feats = skewed_feats[skewed_feats > 0.75]
80      skewed_feats = skewed_feats.index
81      all_data[skewed_feats] = np.log1p(all_data[skewed_feats])
82      all_data = pd.get_dummies(all_data)
83      all_data = all_data.fillna(all_data.mean())
84      X_train = all_data[:train.shape[0]]
85      X_test = all_data[train.shape[0]:]
86      y = train.SalePrice
87
88      return X_train,X_test,y
89
90
91  def mean_squared_error_(ground_truth, predictions):
92      return mean_squared_error(ground_truth, predictions) ** 0.5
93  RMSE = make_scorer(mean_squared_error_, greater_is_better=False)
94
95  class ensemble(object):
96      def __init__(self, n_folds, stacker, base_models):
97          self.n_folds = n_folds
98          self.stacker = stacker
99          self.base_models = base_models
100     def fit_predict(self,train,test,ytr):
101         X = train.values
102     y = ytr.values
103         T = test.values
104         folds = list(KFold(len(y), n_folds = self.n_folds, shuffle = True, rand
105         S_train = np.zeros((X.shape[0],len(self.base_models)))
106         S_test = np.zeros((T.shape[0],len(self.base_models)))
107         for i,reg in enumerate(base_models):
108             print ("Fitting the base model...")
109             S_test_i = np.zeros((T.shape[0],len(folds)))
110             for j, (train_idx,test_idx) in enumerate(folds):
111                 X_train = X[train_idx]
112                 y_train = y[train_idx]
113                 X_holdout = X[test_idx]
114                 reg.fit(X_train,y_train)
115                 y_pred = reg.predict(X_holdout)[:]
116                 S_train[test_idx,i] = y_pred
117                 S_test_i[:,j] = reg.predict(T)[:]
118             S_test[:,i] = S_test_i.mean(1)
119
120
```

```
        print ("Stacking base models...")

        # tuning the stacker
    param_grid = {
         'alpha': [1e-3,5e-3,1e-2,5e-2,1e-1,0.2,0.3,0.4,0.5,0.8,1e0,3,5,7,1e1],
    }
    grid = GridSearchCV(estimator=self.stacker, param_grid=param_grid, n_jobs=1
        grid.fit(S_train, y)
        try:
            print('Param grid:')
            print(param_grid)
            print('Best Params:')
            print(grid.best_params_)
            print('Best CV Score:')
            print(-grid.best_score_)
            print('Best estimator:')
            print(grid.best_estimator_)
            print(message)
        except:
            pass

        y_pred = grid.predict(S_test)[:]
        return y_pred, -grid.best_score_

train = pd.read_csv("train.csv") # read train data
test = pd.read_csv("test.csv") # read test data

# build a model library (can be improved)
base_models = [
        RandomForestRegressor(
            n_jobs=1, random_state=0,
            n_estimators=500, max_features=14
        ),
        RandomForestRegressor(
            n_jobs=1, random_state=0,
            n_estimators=500, max_features=20,
        max_depth = 7
        ),
        ExtraTreesRegressor(
            n_jobs=1, random_state=0,
            n_estimators=500, max_features=15
        ),
        ExtraTreesRegressor(
            n_jobs=1, random_state=0,
          n_estimators=500, max_features=20
        ),
        GradientBoostingRegressor(
            random_state=0,
            n_estimators=500, max_features=10, max_depth=6,
            learning_rate=0.05, subsample=0.8
        ),
    GradientBoostingRegressor(
            random_state=0,
            n_estimators=500, max_features=15, max_depth=6,
            learning_rate=0.05, subsample=0.8
        ),
        XGBRegressor(
            seed=0,
            n_estimators=500, max_depth=10,
            learning_rate=0.05, subsample=0.8, colsample_bytree=0.75
        ),

```

```
        XGBRegressor(
            seed=0,
            n_estimators=500, max_depth=7,
            learning_rate=0.05, subsample=0.8, colsample_bytree=0.75
        ),
    LassoCV(alphas = [1, 0.1, 0.001, 0.0005]),
    KNeighborsRegressor(n_neighbors = 5),
        KNeighborsRegressor(n_neighbors = 10),
        KNeighborsRegressor(n_neighbors = 15),
        KNeighborsRegressor(n_neighbors = 25),
    LassoLarsCV(),
    ElasticNet(),
    SVR()
    ]

ensem = ensemble(
        n_folds=5,
    stacker=Ridge(),
        base_models=base_models
    )

X_train,X_test,y_train = data_preprocess(train,test)
y_pred, score = ensem.fit_predict(X_train,X_test,y_train)

create_submission(np.expm1(y_pred),score)
```

```
/Users/tcsdiker/anaconda2/lib/python2.7/site-packages/sklearn/linear
_model/least_angle.py:313: ConvergenceWarning: Regressors in active
 set degenerate. Dropping a regressor, after 112 iterations, i.e. al
pha=1.266e-04, with an active set of 102 regressors, and the smalles
t cholesky pivot element being 1.490e-08. Reduce max_iter or increas
e eps parameters.
  ConvergenceWarning)
/Users/tcsdiker/anaconda2/lib/python2.7/site-packages/sklearn/linear
_model/least_angle.py:313: ConvergenceWarning: Regressors in active
 set degenerate. Dropping a regressor, after 132 iterations, i.e. al
pha=8.659e-05, with an active set of 122 regressors, and the smalles
t cholesky pivot element being 1.490e-08. Reduce max_iter or increas
e eps parameters.
  ConvergenceWarning)
/Users/tcsdiker/anaconda2/lib/python2.7/site-packages/sklearn/linear
_model/least_angle.py:313: ConvergenceWarning: Regressors in active
 set degenerate. Dropping a regressor, after 132 iterations, i.e. al
pha=8.659e-05, with an active set of 122 regressors, and the smalles
t cholesky pivot element being 1.054e-08. Reduce max_iter or increas
e eps parameters.
```

In [20]:

```
import datetime
import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn import preprocessing
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor,
from sklearn.grid_search import GridSearchCV
from sklearn.cross_validation import ShuffleSplit
from sklearn.metrics import make_scorer, mean_squared_error
from sklearn.kernel_ridge import KernelRidge
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from scipy.stats import skew

def mean_squared_error_(ground_truth, predictions):
    return mean_squared_error(ground_truth, predictions) ** 0.5
RMSE = make_scorer(mean_squared_error_, greater_is_better=False)

def create_submission(prediction,score):
    now = datetime.datetime.now()
    sub_file = 'submission_'+str(score)+'_'+str(now.strftime("%Y-%m-%d-%H-%M"))
    print ('Creating submission: ', sub_file)
    pd.DataFrame({'Id': test['Id'].values, 'SalePrice': prediction}).to_csv(sub

def data_preprocess(train,test):
    train.drop(train.index[outlier_idx],inplace=True)
    all_data = pd.concat((train.loc[:,'MSSubClass':'SaleCondition'],
                          test.loc[:,'MSSubClass':'SaleCondition']))

    to_delete = ['Alley','FireplaceQu','PoolQC','Fence','MiscFeature']
    all_data = all_data.drop(to_delete,axis=1)

    train["SalePrice"] = np.log1p(train["SalePrice"])
    #log transform skewed numeric features
    numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index
    skewed_feats = train[numeric_feats].apply(lambda x: skew(x.dropna())) #comp
    skewed_feats = skewed_feats[skewed_feats > 0.75]
    skewed_feats = skewed_feats.index
    all_data[skewed_feats] = np.log1p(all_data[skewed_feats])
    all_data = pd.get_dummies(all_data)
    all_data = all_data.fillna(all_data.mean())
    X_train = all_data[:train.shape[0]]
    X_test = all_data[train.shape[0]:]
    y = train.SalePrice

    return X_train,X_test,y

def model_random_forecast(Xtrain,Xtest,ytrain):

    X_train = Xtrain
    y_train = ytrain
    rfr = RandomForestRegressor(n_jobs=1, random_state=0)
    param_grid = {}
    model = GridSearchCV(estimator=rfr, param_grid=param_grid, n_jobs=1, cv=10,
    model.fit(X_train, y_train)
    print('Random forecast regression...')
    print('Best Params:')
    print(model.best_params_)
    print('Best CV Score:')
```

```
    print(-model.best_score_)

    y_pred = model.predict(Xtest)
    return y_pred, -model.best_score_

def model_gradient_boosting_tree(Xtrain,Xtest,ytrain):

    X_train = Xtrain
    y_train = ytrain
    gbr = GradientBoostingRegressor(random_state=0)
    param_grid = {}
    model = GridSearchCV(estimator=gbr, param_grid=param_grid, n_jobs=1, cv=10,
    model.fit(X_train, y_train)
    print('Gradient boosted tree regression...')
    print('Best Params:')
    print(model.best_params_)
    print('Best CV Score:')
    print(-model.best_score_)

    y_pred = model.predict(Xtest)
    return y_pred, -model.best_score_

def model_xgb_regression(Xtrain,Xtest,ytrain):

    X_train = Xtrain
    y_train = ytrain

    xgbreg = xgb.XGBRegressor(seed=0)
    param_grid = {}
    model = GridSearchCV(estimator=xgbreg, param_grid=param_grid, n_jobs=1, cv=
    model.fit(X_train, y_train)
    print('eXtreme Gradient Boosting regression...')
    print('Best Params:')
    print(model.best_params_)
    print('Best CV Score:')
    print(-model.best_score_)

    y_pred = model.predict(Xtest)
    return y_pred, -model.best_score_

def model_extra_trees_regression(Xtrain,Xtest,ytrain):

    X_train = Xtrain
    y_train = ytrain

    etr = ExtraTreesRegressor(n_jobs=1, random_state=0)
    param_grid = {}
    model = GridSearchCV(estimator=etr, param_grid=param_grid, n_jobs=1, cv=10,
    model.fit(X_train, y_train)
    print('Extra trees regression...')
    print('Best Params:')
    print(model.best_params_)
    print('Best CV Score:')
    print(-model.best_score_)

    y_pred = model.predict(Xtest)
    return y_pred, -model.best_score_


# read data, build model and do prediction
train = pd.read_csv("train.csv") # read train data
```