capstone

March 27, 2019

1 Machine Learning Engineer Nanodegree

1.1 Capstone Project

Anton Mironenko
March 26th, 2019

1.2 Definition

1.2.1 Project Overview

There are a lot of real estate deals being made nowdays over the world, selling and buying houses. Each deal has a continuous label - the house price, and the features - house quality, square, neighborhood, number of rooms, etc.

One may find a strong dependency between the features and the house price, and the prediction of the price based on house/deal parameters is often the challenge for people, involved in real estate business.

Kaggle ongoing competition provides a nice dataset "With 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa". The challenge of this competition is to predict the final price for a given set of houses' features.

1.2.2 Problem Statement

The problem to be solved in this project is to predict the house price based on a given dataset with as small error as possible.

In order to do that, we need to:

- Visualize and analyze the dataset, find correlations and other dependencies,
- Clean and normalize the dataset:
  - remove outliers,
  - remove useless features,
  - fill missing data,
  - transform numeric features if needed,
  - perform one-hot-encoding for categorical features,
- Try different regression methods,
- Build custom ensemble stacking model on top of regression models from above,
- Repeatedly submit the optimized kernel to Kaggle competition until we get a satisfactory result (will be described below).

1.2.3 Metrics

The main metric to be used is Root-Mean-Squared-Error between the logarithm of the predicted value and the logarithm of the observed sales price (RMSLE), as it is defined by the Kaggle competition:

Apart from that, the Mean-Absolute-Error and R2 score will be used in order to better understand the results:

- Root mean squared logarithm error: $RMSLE = \sqrt{\frac{1}{n}\sum_i (log(y_i) - log(\hat{y}_i))^2}$
- Mean absolute error: $MAE = \frac{1}{n}\sum_i \mid y_i - \hat{y}_i \mid$
- R^2 score: $R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$ where y_i - observed values, $\hat{y}_i$ - predicted values, n is the number of observations, Mean of the observed data: $\bar{y} = \frac{1}{n}\sum_i y_i$

RMSLE and MAE have range from zero to infinity, the closer to zero - the better. MAE gives a filling, what an average price error is, in dollars. R2 has range from 0 to 1, the closer to 1 - the better.

```
<IPython.core.display.Javascript object>
```

1.3 Analysis

1.3.1 Data Exploration

Let's read the training and final testing data and show the main statistical characteristics of train data like standard deviation, percentiles, min/max/average. We will agree about the following terminology:

- "Testing" set (without the word "final") will be the minor part of training data "train.csv" extracted via sklearn.model_selection.train_test_split function.
- "Training" set will be the major part of training data "train.csv" extracted via sklearn.model_selection.train_test_split function. This dataset is provided with the "SalePrice" label - the one we are going to predict and submit to Kaggle competition.
- "Final testing" set will be the dataset "test.csv", which is provided without the "SalePrice" label.

```
Training data dimensions: 1460 data points X 81 features
```

```
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0   1          60       RL         65.0     8450   Pave   NaN      Reg
1   2          20       RL         80.0     9600   Pave   NaN      Reg
2   3          60       RL         68.0    11250   Pave   NaN      IR1
3   4          70       RL         60.0     9550   Pave   NaN      IR1
4   5          60       RL         84.0    14260   Pave   NaN      IR1

  LandContour Utilities    ...     PoolArea PoolQC Fence MiscFeature MiscVal  \
0         Lvl    AllPub    ...            0    NaN   NaN         NaN       0
1         Lvl    AllPub    ...            0    NaN   NaN         NaN       0
```

```
2         Lvl    AllPub    ...            0    NaN   NaN         NaN       0
3         Lvl    AllPub    ...            0    NaN   NaN         NaN       0
4         Lvl    AllPub    ...            0    NaN   NaN         NaN       0

  MoSold YrSold  SaleType  SaleCondition  SalePrice
0      2   2008        WD         Normal     208500
1      5   2007        WD         Normal     181500
2      9   2008        WD         Normal     223500
3      2   2006        WD        Abnorml     140000
4     12   2008        WD         Normal     250000

[5 rows x 81 columns]


                Id   MSSubClass  LotFrontage        LotArea  OverallQual  \
count  1460.000000  1460.000000  1201.000000    1460.000000  1460.000000
mean    730.500000    56.897260    70.049958   10516.828082     6.099315
std     421.610009    42.300571    24.284752    9981.264932     1.382997
min       1.000000    20.000000    21.000000    1300.000000     1.000000
25%     365.750000    20.000000    59.000000    7553.500000     5.000000
50%     730.500000    50.000000    69.000000    9478.500000     6.000000
75%    1095.250000    70.000000    80.000000   11601.500000     7.000000
max    1460.000000   190.000000   313.000000  215245.000000    10.000000

       OverallCond    YearBuilt  YearRemodAdd   MasVnrArea   BsmtFinSF1  \
count  1460.000000  1460.000000   1460.000000  1452.000000  1460.000000
mean      5.575342  1971.267808   1984.865753   103.685262   443.639726
std       1.112799    30.202904     20.645407   181.066207   456.098091
min       1.000000  1872.000000   1950.000000     0.000000     0.000000
25%       5.000000  1954.000000   1967.000000     0.000000     0.000000
50%       5.000000  1973.000000   1994.000000     0.000000   383.500000
75%       6.000000  2000.000000   2004.000000   166.000000   712.250000
max       9.000000  2010.000000   2010.000000  1600.000000  5644.000000

           ...         WoodDeckSF  OpenPorchSF  EnclosedPorch    3SsnPorch  \
count      ...        1460.000000  1460.000000    1460.000000  1460.000000
mean       ...          94.244521    46.660274      21.954110     3.409589
std        ...         125.338794    66.256028      61.119149    29.317331
min        ...           0.000000     0.000000       0.000000     0.000000
25%        ...           0.000000     0.000000       0.000000     0.000000
50%        ...           0.000000    25.000000       0.000000     0.000000
75%        ...         168.000000    68.000000       0.000000     0.000000
max        ...         857.000000   547.000000     552.000000   508.000000

       ScreenPorch     PoolArea       MiscVal       MoSold       YrSold  \
count  1460.000000  1460.000000   1460.000000  1460.000000  1460.000000
mean     15.060959     2.758904     43.489041     6.321918  2007.815753
std      55.757415    40.177307    496.123024     2.703626     1.328095
```

```
min       0.000000     0.000000       0.000000     1.000000  2006.000000
25%       0.000000     0.000000       0.000000     5.000000  2007.000000
50%       0.000000     0.000000       0.000000     6.000000  2008.000000
75%       0.000000     0.000000       0.000000     8.000000  2009.000000
max     480.000000   738.000000   15500.000000    12.000000  2010.000000

           SalePrice
count    1460.000000
mean   180921.195890
std     79442.502883
min     34900.000000
25%    129975.000000
50%    163000.000000
75%    214000.000000
max    755000.000000

[8 rows x 38 columns]
```

Let's get the most important numeric features - whose correlation with SalePrice is more than 0.4 - and draw a heatmap for them. Actually, some of these numeric features will be categorical, with different numbers for each category. We will emphasize them later.

```
Most important features=['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars',
'GarageArea', 'TotalBsmtSF', '1stFlrSF', 'FullBath', 'TotRmsAbvGrd', 'YearBuilt',
'YearRemodAdd', 'GarageYrBlt', 'MasVnrArea', 'Fireplaces']
```

Let's describe these most important columns, from data_description.txt: - OverallQual: Rates the overall material and finish of the house, categorical numeric - GrLivArea: Above grade (ground) living area square feet, numeric - GarageCars: Size of garage in car capacity, numeric - GarageArea: Size of garage in square feet, numeric - TotalBsmtSF: Total square feet of basement area, numeric - 1stFlrSF: First Floor square feet, numeric - FullBath: Full bathrooms above grade, numeric - TotRmsAbvGrd: Total rooms above grade (does not include bathrooms), numeric - YearBuilt: Original construction date, numeric/year - YearRemodAdd: Remodel date (same as construction date if no remodeling or additions), numeric/year - GarageYrBlt: Year garage was built, numeric/year - MasVnrArea: Masonry veneer area in square feet, numeric - Fireplaces: Number of fireplaces, numeric

Here are a few examples of other categorical features with string values:

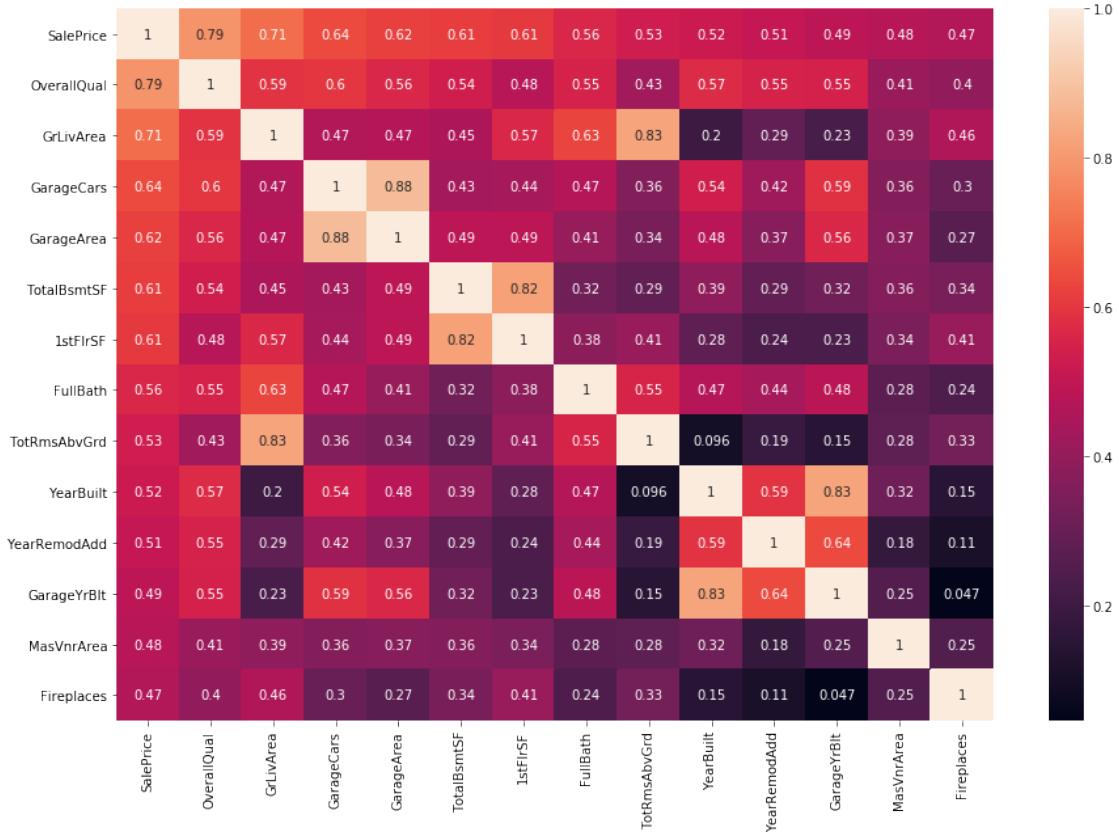
```
Street: Type of road access to property

       Grvl Gravel
       Pave Paved

Alley: Type of alley access to property

       Grvl Gravel
       Pave Paved
```

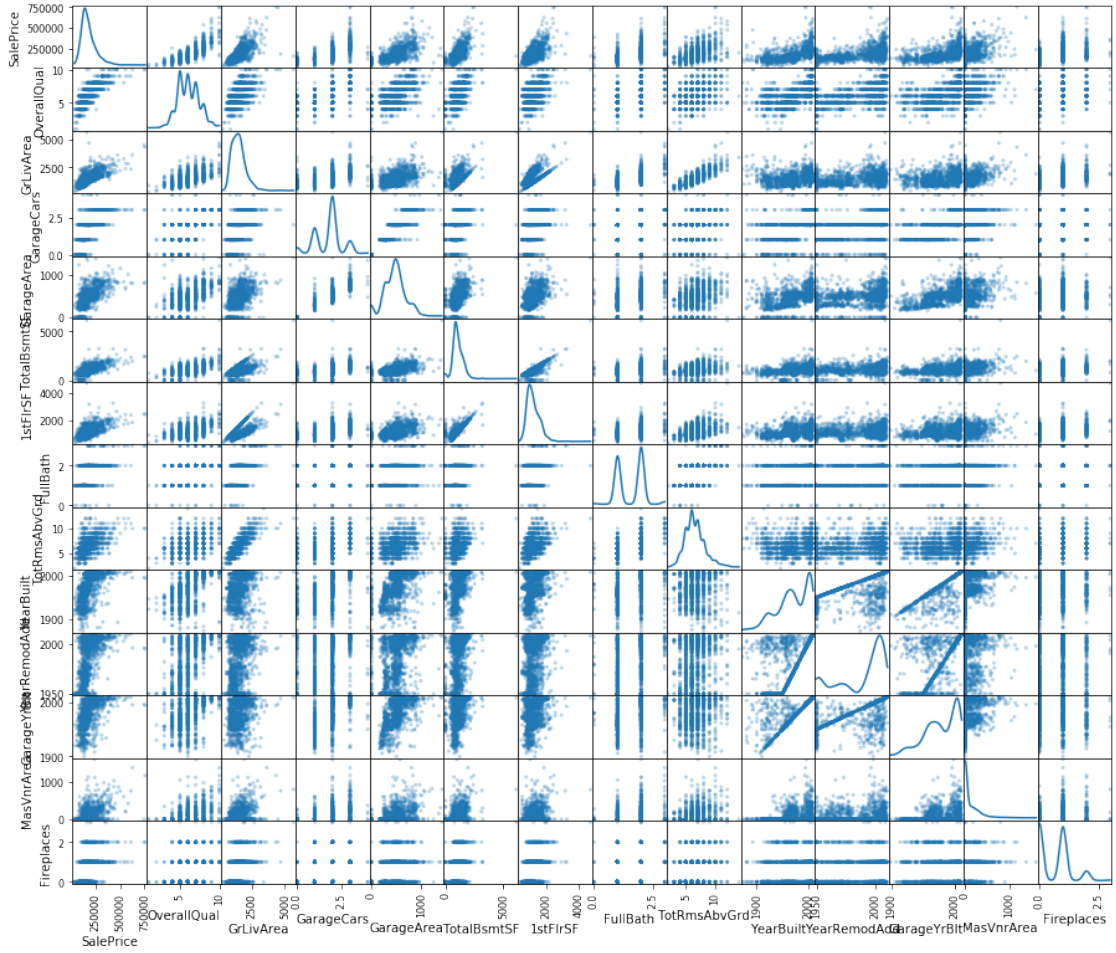
```
       NA   No alley access

LotShape: General shape of property

       Reg  Regular
       IR1  Slightly irregular
       IR2  Moderately Irregular
       IR3  Irregular

LandContour: Flatness of the property

       Lvl  Near Flat/Level
       Bnk  Banked - Quick and significant rise from street grade to building
       HLS  Hillside - Significant slope from side to side
       Low  Depression
```

Please note, that we won't describe all 79 features.

1.3.2 Exploratory Visualization

Let's draw a heatmap and scatter matrix for the most important features.





We can see that there is a dependency close to linear between the SalePrice and many of the most important features, even if these features are categorical with numeric values, like OverallQual.

1.3.3 Algorithms and Techniques

The following regression models will be used:

- LinearRegression
- Ridge
- Lasso
- DecisionTreeRegressor as a base estimator for ensemble methods

And the ensemble methods to be used are:

- AdaBoostRegressor
- RandomForestRegressor
- GradientBoostingRegressor
- XGBoost

Then we are going to build a custom stacking ensemble model on top of chosen regression models from above.

Here is a brief description of all these methods:

- **LinearRegression**: is a linear model, e.g. a model that assumes a linear relationship between the input variables (x) and the single output variable (y). More specifically, that y can be calculated from a linear combination of the input variables (x). Understanding Linear Regression in Machine Learning
- **Ridge**: is a technique for analyzing multiple regression data that suffer from multicollinearity. When multicollinearity occurs, least squares estimates are unbiased, but their variances are large so they may be far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors. Ridge Regression
- **Lasso**: is a type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models (i.e. models with fewer parameters). This particular type of regression is well-suited for models showing high levels of muticollinearity or when you want to automate certain parts of model selection, like variable selection/parameter elimination. Lasso Regression: Simple Definition
- **DecisionTreeRegressor**: builds regression model in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. Decision Tree Regression
- **RandomForestRegressor**: Random Forest (strong learner) is built as an ensemble of Decision Trees (weak learners) to perform regression. Random Forests are trained via the bagging method. Bagging is a simple ensembling technique in which we build many independent predictors/models/learners and combine them using some model averaging techniques. Bagging or Bootstrap Aggregating, consists of randomly sampling subsets of the training data, fitting a model to these smaller data sets, and aggregating the predictions. This method allows several instances to be used repeatedly for the training stage given that we are sampling with replacement. Tree bagging consists of sampling subsets of the training set, fitting a Decision Tree to each, and aggregating their result. Random Forests, Explained
- **AdaBoostRegressor**: short for Adaptive Boosting, is a machine learning meta-algorithm. It can be used in conjunction with many other types of learning algorithms to improve performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier (or regressor). Boosting is an ensemble technique in which the predictors are not made independently like bagging from above, but sequentially. AdaBoost
- **GradientBoostingRegressor**: Gradient Boosting is an example of boosting algorithm. We want our predictions, such that our loss function (MSE) is minimum. By using gradient descent and updating our predictions based on a learning rate, we can find the values where MSE is minimum. Gradient Boosting from scratch
- **XGBoost**: stands for eXtreme Gradient Boosting. It does this by tackling one of the major inefficiencies of gradient boosted trees: considering the potential loss for all possible splits to create a new branch (especially if you consider the case where there are thousands of features, and therefore thousands of possible splits). XGBoost tackles this inefficiency by looking at the distribution of features across all data points in a leaf and using this information to reduce the search space of possible feature splits. Although XGBoost implements a few regularization tricks, this speed up is by far the most useful feature of the library, al-

lowing many hyperparameter settings to be investigated quickly. Gradient Boosting and XGBoost

1.3.4 Benchmark

The dataset is taken from Kaggle ongoing competition, so that the competition's results can be taken as a benchmark model. The goal stated in the Capstone proposal is to provide a model with a metric fit in top 50%.

1.4 Methodology

1.4.1 Data Preprocessing

Let's preprocess the data in a simplest way (prep 1):

- Fill NA with averages,
- One hot encoding of categorical features,
- Reduce to the same set of features in training and final testing datasets, because one hot encoding may give different results for training and final testing datasets.

The first idea was to try the simplest model on raw input dataset, but we have to do the steps above, otherwise the LinearRegression model would give us an error.

```
--- Preprocessing training dataset .. ---
Filling missing data with mean value ..
Performing one hot encoding for categorical features ..

--- Preprocessing final testing dataset .. ---
Filling missing data with mean value ..
Performing one hot encoding for categorical features ..
```

Dividing the data into training/test datasets ..
Building a simplest model - LinearRegression - and evaluating it.

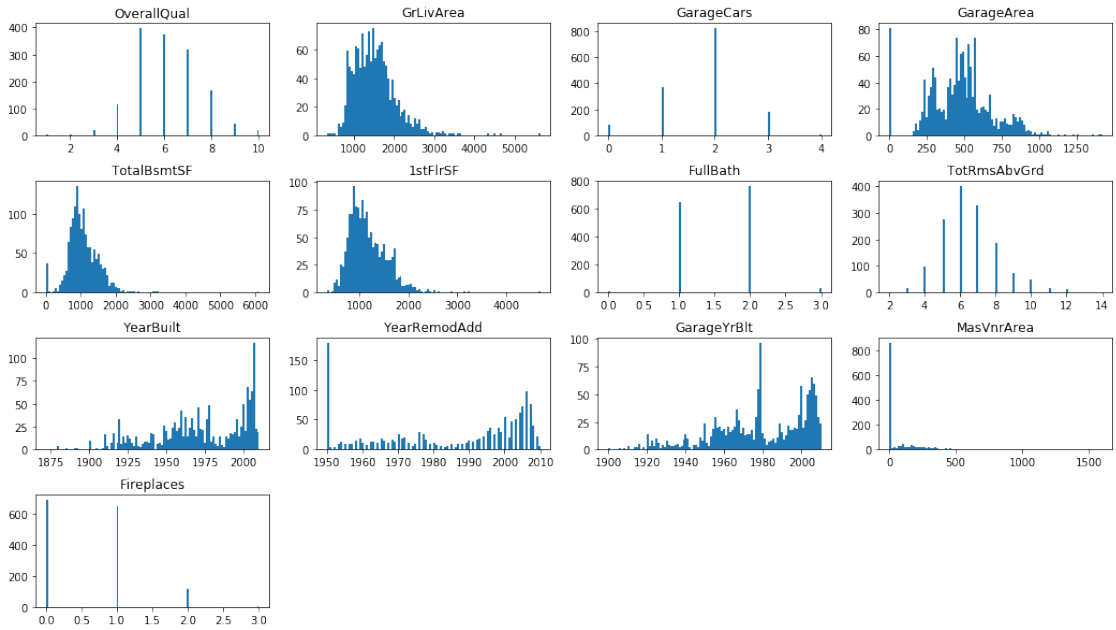
```
-- LinearRegression, original Price, prep 1, y_test --
RMSLE=0.18675, MAE=22468, R2=0.82407

-- LinearRegression, original Price, prep 1, y_train --
RMSLE=0.38258, MAE=21379, R2=0.80507

Transforming price to log and train a model on it ..

Transforming price back to normal (logarithmed value to exp) for calculating metrics
..

-- LinearRegression, log Price, prep 1, y_test --
RMSLE=0.14792, MAE=19004, R2=0.88087

-- LinearRegression, log Price, prep 1, y_train --
RMSLE=0.14555, MAE=18082, R2=0.78275
```

As we can see, transforming the price to logarithm improved the result a lot: RMSLE decreased from 0.38258 to 0.14555 on traing dataset.

Let's preprocess the data in a more complex way (prep 2): - Remove features with NA more than 10% - Fill the rest NA with averages, - Remove outliers, - One hot encoding of categorical features, - Reduce to the same set of features in training and final testing datasets.

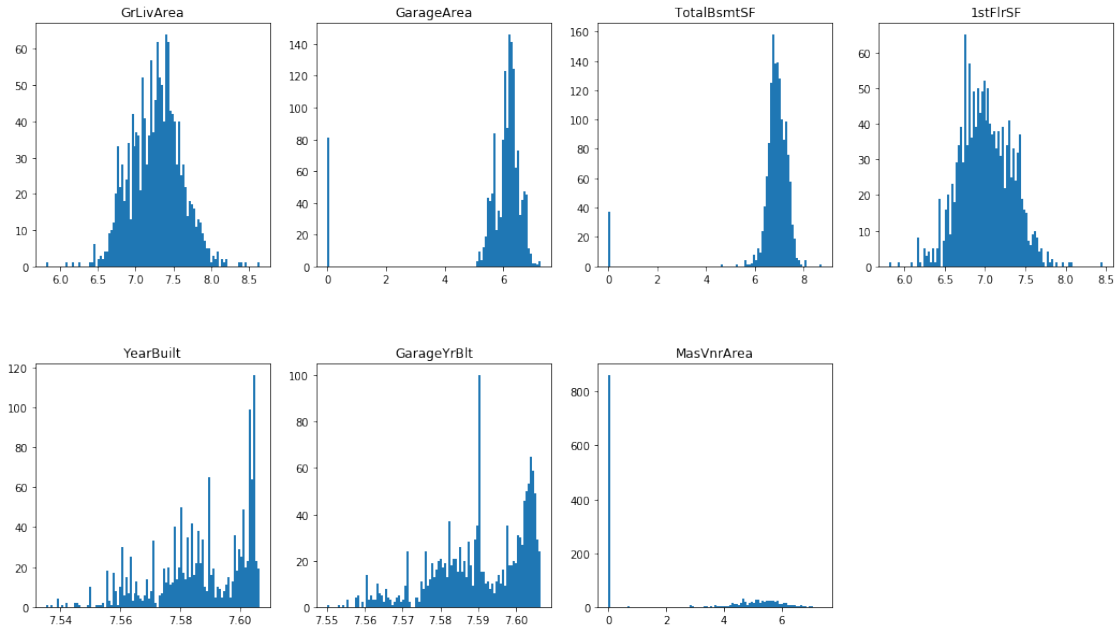
Outliers removal was made based on a difference between 75th and 25th percentiles multiplied by some factor. I tried different factors and got 2.1 as the one showing the best prediction results. The rows with more than one outlier were removed.

```
Fill NA more carefully ..

--- Preprocessing training dataset .. ---
Features with more than 10% NA, to be removed=['PoolQC', 'MiscFeature', 'Alley',
'Fence', 'FireplaceQu', 'LotFrontage']
Filling missing data with mean value ..

--- Preprocessing final testing dataset .. ---
Features with more than 10% NA, to be removed=['PoolQC', 'MiscFeature', 'Alley',
'Fence', 'FireplaceQu', 'LotFrontage']
Filling missing data with mean value ..

Histograms for most important features=['OverallQual', 'GrLivArea', 'GarageCars',
'GarageArea', 'TotalBsmtSF', '1stFlrSF', 'FullBath', 'TotRmsAbvGrd', 'YearBuilt',
'YearRemodAdd', 'GarageYrBlt', 'MasVnrArea', 'Fireplaces']
```




```
Manually selected features that requre transformation to logarithm=['GrLivArea',
'GarageArea', 'TotalBsmtSF', '1stFlrSF', 'YearBuilt', 'GarageYrBlt', 'MasVnrArea']
Transforming manually selected features to log ..
```

Histograms for these features:




```
Outliers number=713, percentage of all dataset=48.84%
Outliers for more than one feature, to be removed: outliers number=237, percentage of
all dataset: 16.23

--- Preprocessing training dataset .. ---
Performing one hot encoding for categorical features ..

--- Preprocessing final testing dataset .. ---
Performing one hot encoding for categorical features ..
```

Let's now try the set of models and compare their performance.

```
-- AdaBoostRegressor, log Price, prep 2, y_test --
RMSLE=0.12317, MAE=15175, R2=0.87715

-- AdaBoostRegressor, log Price, prep 2, y_train --
RMSLE=0.03506, MAE=4434, R2=0.99263

-- RandomForestRegressor, log Price, prep 2, y_test --
RMSLE=0.12983, MAE=16847, R2=0.84995

-- RandomForestRegressor, log Price, prep 2, y_train --
RMSLE=0.06324, MAE=7217, R2=0.97721

-- GradientBoostingRegressor, log Price, prep 2, y_test --
RMSLE=0.11218, MAE=13768, R2=0.91167
```

```
-- GradientBoostingRegressor, log Price, prep 2, y_train --
RMSLE=0.05067, MAE=6666, R2=0.98269

-- LinearRegression, log Price, prep 2, y_test --
RMSLE=0.12629, MAE=13774, R2=0.90658

-- LinearRegression, log Price, prep 2, y_train --
RMSLE=0.07714, MAE=9797, R2=0.95805

-- Lasso, log Price, prep 2, y_test --
RMSLE=0.19313, MAE=24632, R2=0.45742

-- Lasso, log Price, prep 2, y_train --
RMSLE=0.18558, MAE=22651, R2=0.79779

-- Ridge, log Price, prep 2, y_test --
RMSLE=0.11103, MAE=13282, R2=0.89591

-- Ridge, log Price, prep 2, y_train --
RMSLE=0.08189, MAE=10264, R2=0.95554
```

GradientBoostingRegressor showed the best performance on testing dataset.

1.4.2 Refinement

Now let's see how we can refine the model. I tried to use KFold cross validation on several models, but what I saw was that the improved model started overfitting: it showed better results for training data, but worse results for testing data. So the parameters were tuned manually to control overfitting. Here are the tuned parameters of GradientBoostingRegressor as the most promising one:

```
(n_estimators=200, min_samples_leaf=2, max_features=226, random_state=42)
```

Now let's build the custom ensemble stacking model on top of submodels: Ridge, GradientBoostingRegressor and LinearRegression. Meta ensemble is chosen to be LinearRegression. The previous version of the custom model had XGBoost instead of Ridge, but replacing it with Ridge gave me another 6% in the Kaggle top. Other combinations were also evaluated, but this one showed the most promising results.

The idea of custom ensemble model is taken from another Kaggle kernel. At this step we forget about our division of "train.csv" to training and testing datasets, and start working with "train.csv" as the whole dataset via KFold, number of folds = 4. For each combination of 3 and 1 folders we will train each submodel on 3 folders, and will predict the label based on the remaining 1 folder, plus we will predict the label based on the "test.csv" as the final test dataset. Then the final testing dataset predictions are averaged between 4 folders. This will be the final result we are going to submit in the Kaggle kernel.

```
-- Ridge in stacking log, y_train --
RMSLE=0.11214, MAE=13363, R2=0.91772

-- GradientBoostingRegressor in stacking log, y_train --
RMSLE=0.11524, MAE=13880, R2=0.90870
```

```
-- LinearRegression in stacking log, y_train --
RMSLE=0.12033, MAE=13879, R2=0.91438

Merge models into one ensemble model ..

-- LinearRegression merge ensemble log, y_train --
RMSLE=0.10593, MAE=12530, R2=0.92749
```

1.5 Results

1.5.1 Model Evaluation and Validation

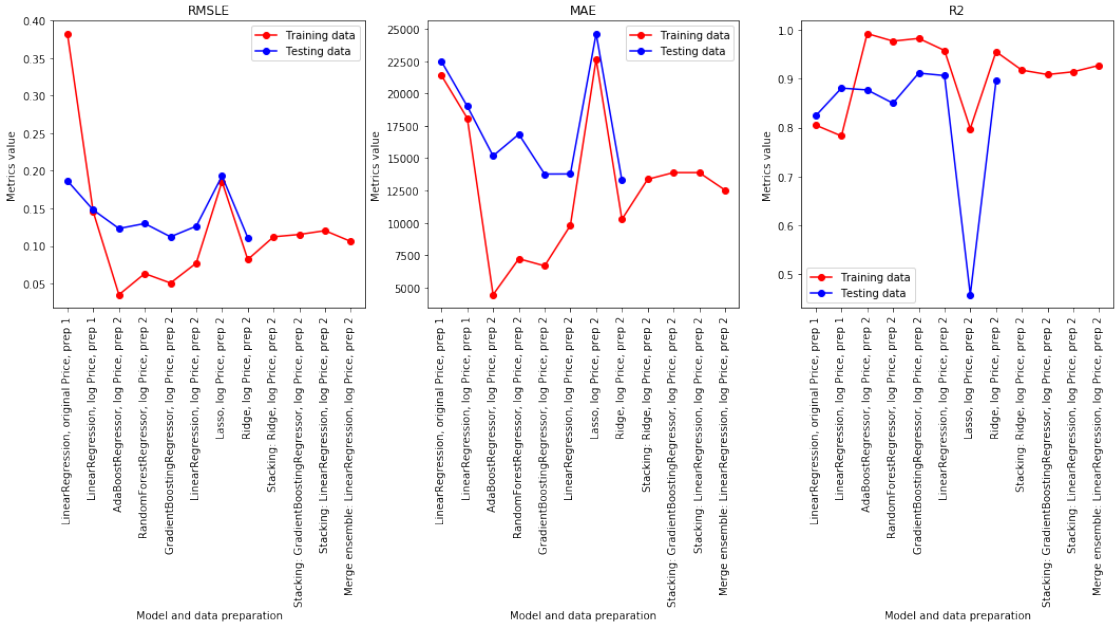
The custom ensemble stacking model showed very good results! RMSLE = 0.10593 on training data set. Here is the history of evaluating different models and improving. Please note that "Testing data" line is shorter than "Training data" line, because starting from Custom ensemble stacking model, we gave up the "testing data" which was a 20% subset of training data, and switched to training data as the whole dataset.

df_metrics_train:

	RMSLE	MAE	R2
LinearRegression, original Price, prep 1	0.382585	21379.218907	0.805069
LinearRegression, log Price, prep 1	0.145551	18082.278693	0.782755
AdaBoostRegressor, log Price, prep 2	0.035057	4433.665905	0.992627
RandomForestRegressor, log Price, prep 2	0.063238	7217.485923	0.977207
GradientBoostingRegressor, log Price, prep 2	0.050670	6666.171728	0.982692
LinearRegression, log Price, prep 2	0.077138	9797.170990	0.958051
Lasso, log Price, prep 2	0.185580	22651.018476	0.797792
Ridge, log Price, prep 2	0.081893	10264.310182	0.955536
Stacking: Ridge, log Price, prep 2	0.112144	13363.300463	0.917717
Stacking: GradientBoostingRegressor, log Price,...	0.115243	13880.276004	0.908698
Stacking: LinearRegression, log Price, prep 2	0.120330	13879.207747	0.914379
Merge ensemble: LinearRegression, log Price, pr...	0.105933	12530.166086	0.927495

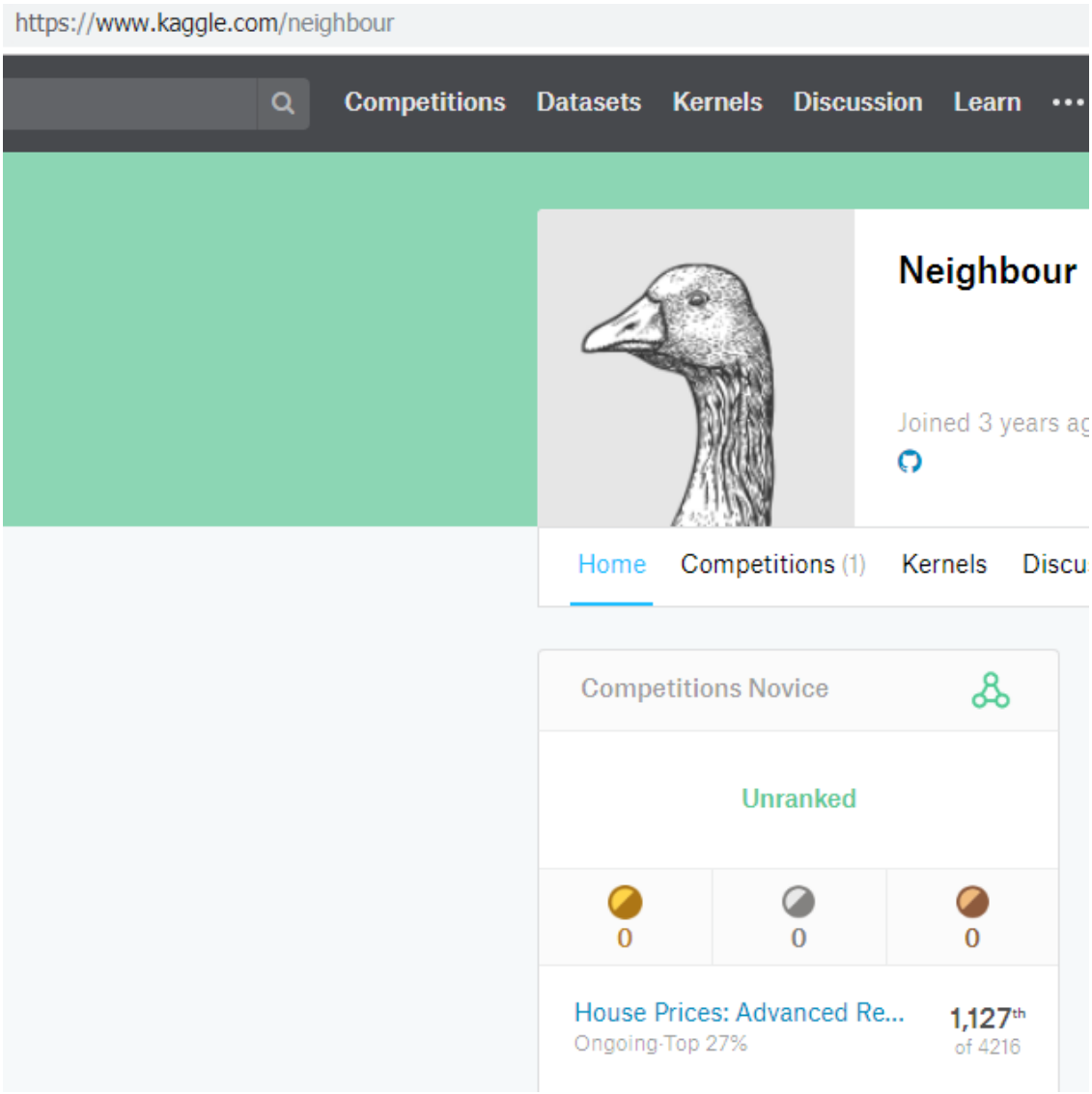
df_metrics_test:

	RMSLE	MAE	R2
LinearRegression, original Price, prep 1	0.186748	22468.135548	0.824071
LinearRegression, log Price, prep 1	0.147920	19003.727184	0.880870
AdaBoostRegressor, log Price, prep 2	0.123174	15174.593428	0.877152
RandomForestRegressor, log Price, prep 2	0.129829	16846.618036	0.849953
GradientBoostingRegressor, log Price, prep 2	0.112178	13768.037009	0.911669
LinearRegression, log Price, prep 2	0.126285	13774.472305	0.906580
Lasso, log Price, prep 2	0.193129	24632.278950	0.457415
Ridge, log Price, prep 2	0.111026	13281.719743	0.895906



1.5.2 Justification

Moving from tuned GradientBoostingRegressor model into the custom ensemble stacking model allowed me to jump from top 51% to top 27%, which is much better than the target stated in the capstone proposal, which was top 50%: Out[14]:

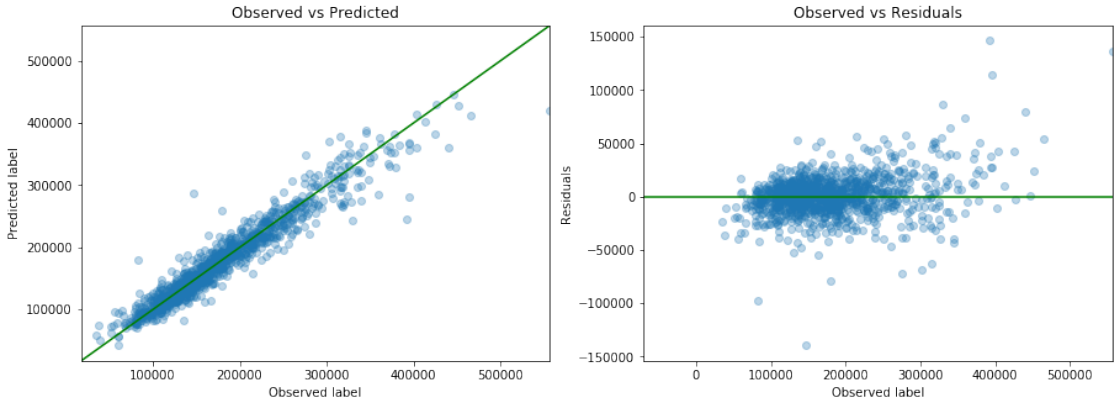


Also if we look at the models evaluation graphs with metrics above, we can see that the error of testing dataset and the error of training dataset converge at small enough value, meaning, that the final model is generalized enough, and is ready for unknown data.

1.6 Conclusion

1.6.1 Free-Form Visualization

Let's now depict the characteristics of final residuals and see how predictions are different versus observed label.




```
Residuals details:

count      1223.000000
mean        891.956966
std       18648.281361
min     -139743.619181
25%       -8359.821714
50%         908.978250
75%        9007.595612
max      146398.332502
Name: SalePrice, dtype: float64
```

We can see that the model is biased towards the underestimating the predictions: the predicted price is smaller than the true one. Also we can see that the predictions at the minimum of price are always higher than the true price (negative residuals), and the predictions at the maximum of price are always lower than the true price (positive residuals).

1.6.2 Reflection

The most time consuming part was data preparation. For example, it was a surprise that playing with the number of outliers influences the final model's error.

Another difficult part was dealing with overfitting. This required a lot of manual parameters tuning.

The most exciting part was observing the significant model's improvement after switching to custom ensemble stacking model.

1.6.3 Improvement

What could be done in this project is: more sophisticated conversion of categorical features into numeric, for example some categorical features assume linear increasing of their values. Also the model could be tuned in a way that the price prediction would be lowered downed at minimum of its value, and raised up at maximum of its value, this is according to the observation made in a Free-Form Visualization section.

```
Predicting the price for test data for final submission ..

Final CSV file:
```

	Id	SalePrice
count	1459.000000	1459.000000
mean	2190.000000	175772.863760
std	421.321334	76826.168739
min	1461.000000	44586.882250
25%	1825.500000	125586.838149
50%	2190.000000	154801.262096
75%	2554.500000	207635.631113
max	2919.000000	891588.503939

	Id	SalePrice
0	1461	122129.109719
1	1462	175634.657825
2	1463	184920.192933
3	1464	195543.151486
4	1465	194777.067915