Predicting Sales Prices of the Houses Using Regression Methods of Machine Learning

Parasich Andrey Viktorovich, Parasich Viktor Aleksandrovich, Kaftannikov Igor Leopoldovich, Parasich Irina Vasilevna
South Ural State University
Chelyabinsk, Russian Federation
parasich_av@yandex.ru, pva16@yandex.ru, kil7491@gmail.com, lirina.lirina1959@gmail.com

***Abstract*—In this article we will describe our solution for "House Prices: Advanced Regression Techniques" machine learning competition, which was held on Kaggle platform. The competitor's goal was to predict house's sale price by their attributes like house area, year of building, etc. In our solution, we use classic machine learning algorithms, and our original methods, which will be described here. At the end of the competition, we took 18th place among 2124 participants from whole world (top 1%).**

Keywords—machine learning; Kaggle; neural networks; boosting; regression

I. INTRODUCTION

The Ames Housing Price data set [1] recently released on Kaggle [2] is "a modernized and expanded version of the often cited Boston Housing dataset". It covers all the recorded house sale price in Ames, IA from January 2006 to July 2010. With 79 explanatory variables describing almost every feature of residential homes, we aimed to apply data imputation, feature engineering and machine learning modeling to achieve a better predictive accuracy on the housing price.

Each house description consist of such attributes as house area (called 'GrLivArea'), garage capacity ('GarageCars'), overall quality estimation of house and kitchen('OverallQual', 'KitchenQual'), distinct of the city ('MS_Zoning'), data about neighborhood, type of sale ('SaleType'), year of building ('YearBuilt'), and another similar attributes.

The dataset contains 1460 observations in the training set and 1459 observations in the test set. There are 46 categorical variables including 23 nominal and 23 ordinal ones, and 33 numeric variables in the dataset. The training set also has the sale price as response while the test set does not.

Solutions are evaluated on Root-Mean-Squared-Error (RMSE) between the logarithm of the predicted value and the logarithm of the observed sales price. (Taking logs means that errors in predicting expensive houses and cheap houses will affect the result equally).

Our solution used Python programming language with Pandas, NumPy, Sklearn and XGBoost libraries.

The work was supported by Act 211 Government of the Russian Federation, contract № 02.A03.21.0011

II. EXPLARATORY DATA ANALYSIS

Density plots of the numerical features and target variable (sale price) indicates that the features are skewed (see Fig. 1 and Fig. 2). First we will transform the skewed numeric features by taking feature_new = ln(feature + 1) - this will make the features more normally distributed and eliminate heteroscedasticity, which is critical to algorithm's accuracy.

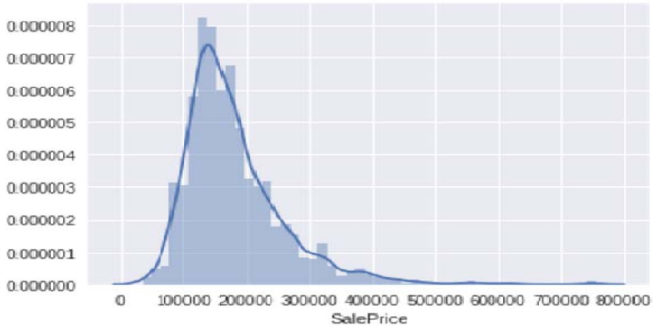

Fig. 1. Distribution of target variable before log transformation.

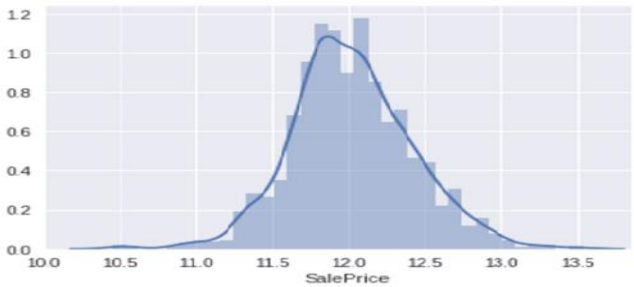

Fig. 2. Distribution of target variable after log transformation.

Viewing the first five rows of the data indicates that there are columns that have missing values. The categorical variables with the largest number of missing values are: 'Alley', 'FirePlaceQu', 'PoolQC', Fence, and 'MiscFeature'. The missing values indicate that majority of the houses do not have alley access, no pool, no fence and no elevator, 2nd garage, shed or tennis court that is covered by the 'MiscFeature'.

978-1-5386-7531-1/18/$31.00 ©2018 IEEE

We can see that one of the features ('GrLivArea') have largest correlation with the target variable (sale price) (see Fig. 3).

The numeric variables do not have as many missing values but there are still some present. There are 259 missing values for the 'LotFrontage', 8 missing values for 'MasVnrArea' and 81 missing values for 'GarageYrBlt'.

This dataset contains many imbalanced categorical variables (majority of samples has one of values of this variable, and vary small count of samples has another values of this variable). This can lead to overfitting on these features, so these variables need to be identified and removed.

We need to fill up missing values to allow our models work correctly. All numeric features were cleaned with the median value, and all categorical missing values were cleaned using the mode. Replace missing numerical values by zero leads to worse accuracy. It may be better to replace missing variables with values of respective variables from nearest neighbors of samples.

III. Feature Engineering

First, we exclude indistinguishable variables from the model. For example, we drop 'Street' variable because there are no significant differences. In addition, some variables are not our main points, such as 'sale month' and 'sale year'; some are too detailed or complicated, such as 'fence type' and 'roof material'. These variables are dropped from the model. Lastly, we changed the formats of nominal and ordinal variables that are consist of various levels.

We use one-hot encoding of categorical features to include there in linear regression algorithms like Lasso regression and Elastic Net regression.

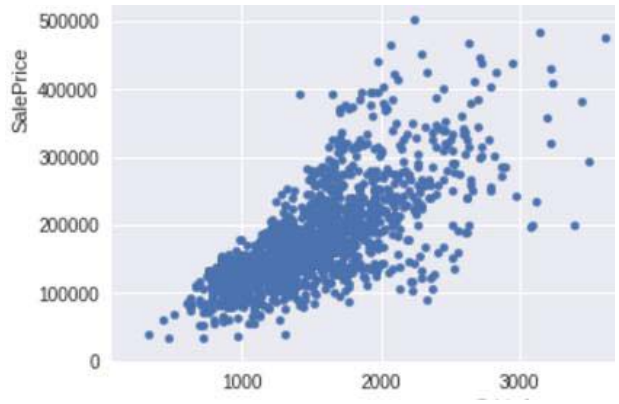

Fig. 3. Joint distribution of target variable (sale price) and house living area.

Also we need to normalize all numerical variables to be placed into the same range to ensure a stable convergence of weight and biases. In standard normalization procedure variables are given to the range [0, 1]. This methodology isn't robust to outliers, because of maximum or minimum values may be a outliers, therefore we use 75-percentile value instead of maximum value and 25-percentile value instead of minimum value in our normalization procedure to ensure robustness to outliers.

For include more descriptive information in our model we use polynomial features: we add products of area features ('LotArea', 'GrLivArea', 'TotalBsmtSF', 'GarageArea', 'BsmtUnfSF') and some other features ('OverallQual', 'OverallCond', 'ExterQual', 'ExterCond', 'BsmtCond', 'GarageQual', 'GarageCond', 'KitchenQual', 'HeatingQC', 'bad_heating', 'MasVnrType_Any', 'SaleCondition_PriceDown', 'Reconstruct', 'ReconstructAfterBuy', 'Build.eq.Buy') to existing features.

To enable our model for especial treating of extreme values of some important variables ('OverallQual', 'OveralCond', 'KitchenQual'), we include in our model additional features, which indicate presence of these extreme values, because extremely high and extremely low values of these variables can be more important for buyer. These nonlinear effects cannot be modeled using only linear features representation. We also add this new features to polynomial features construction.

IV. Basic Machine Learning Algorithms

As a base part and initial step of our solution, we use Lasso regression algorithm [3]. Lasso (least absolute shrinkage and selection operator) is a regression analysis method that performs both variable selection and regularization. Lasso algorithm can be threat as standard linear regression with weights *L1* penalty as regularizer.

Standard linear regression can be written in a form:

$$y_i = \beta_1 x_i^1 + \beta_2 x_i^2 + \cdots + \beta_p x_i^p + \varepsilon_i, \quad i \in 1, \ldots, n,$$

where y_i is prediction for i sample, x_i^q is value of q feature for i sample, β is weight coefficient for q feature, n is total samples count.

The objective function, minimized during Lasso training, can be written in a form:

$$\hat{\beta} = \min_{\beta \epsilon R} \left\{ \frac{1}{N} \|y - X\beta\|_2^2 + \alpha \|\beta\|_1 \right\},$$

where β is a weights vector, α is a regularization parameter.

Using weights L1 penalty forces certain coefficients to be set to zero. Therefore, some of the variables would not have influence on prediction. So Lasso algorithm can be used as a feature selection algorithm to automatically drop noisy features. Important features would have large weight coefficient. Accuracy of Lasso algorithm significantly depends on chooses of regularization parameter. Small values lead to overfitting, but large values lead to pure accuracy. Optimal value can be found using cross-validation. We choose α = 0.0003.

Another algorithm, that we used, was Elastic Net regression [5]. This algorithm is very similar to Lasso regression algorithm, except of using L2 penalty term together with L1 penalty term. So the objective function, minimized during training, can be written as:

$$\hat{\beta} = \min_{\beta \epsilon R} \left\{ \frac{1}{N} \|y - X\beta\|_2^2 + \gamma \|\beta\|^2 + \alpha \|\beta\|_1 \right\},$$

where β is a weights vector, α and γ is a regularization parameters. We use α = 0.005 and γ = 0.13 in our solution for Elastic Net regression.

In addition, we apply gradient boosting [6] of regression trees in our solution. To implement gradient boosting, we use Python XGBoost library [7]. We train 100 trees with maximum depth 20. Disadvantage of tree-based algorithms for regression problems is piecewise-constant form of prediction that gave up. This leads to some decrease of prediction accuracy. Another potential disadvantage is absence of generalization ability for diapason of target variables that was not occur in training set (linear regression based algorithms possess this ability). Advantages of tree based models that they can use more complex decision rules and can better deal with particular cases in data (linear regression based models trained to solve common case).

As another part of our algorithm, we employ multilayer perceptron regressor (some kind of neural network regressor). We observed, that neural network needs to very careful tuning of learning hyper parameters. With improper choose of such parameters, neural network can give several times less accuracy. We came to a 3-layer perceptron (140, 70, 25 neurons on each layer respectively) with logistic activation function and LBFGs [8] optimizer.

Ensembling of different models is an easy and widely used way to improve prediction accuracy. To improve accuracy of single models we use ensemble of Lasso, Gradient Boosting, ElasticNet and neural network regressors. Ensemble's prediction takes a form:

$$e(x) = 0.47 * p_{Lasso}(x) + 0.245 * p_{Boosting}(x) + 0.225 * p_{ElasticNet}(x) + 0.06 * p_{NeuralNetwork}(x),$$

where $e(x)$ – answer of ensemble of models for sample x, $p_{Lasso}(x)$ – prediction of Lasso model for sample x, $p_{Boosting}(x)$ – prediction of boosting model for sample x, $p_{ElasticNet}(x)$ – prediction of Elastic Net model for sample x, $p_{NeuralNetwork}(x)$ – prediction of neural network for sample x.

Weights of the models in ensemble were adjusted by cross-validation.

V. Residual Regressor

Lasso regression gives averaged optimal answer in common case, but does not recognize particular cases. To address this problem we decided to use "residual regressor" – algorithm that uses the residuals of target variable and previous algorithm's prediction for each sample of training set as a target variable for training. We decided to use regression forest algorithm as a residual regressor due to its robustness to overfitting and predictability of hyperparameters influence on learning process. Only accurately tuned regression forest give us a quality improvement.

$$y_{trainRC}(x) = y_{train}(x) - p(x),$$

where $p(x)$ – prediction of initial algorithm for sample x, $y_{train}(x)$ – known target variable for training set sample, $y_{trainRC}(x)$ – residual's value.

Vector of $y_{trainRC}$ was used for training of residual regressor here. Final answer of algorithm would be calculated in form:

$$y(x) = p(x) + c_r(x),$$

where $c_r(x)$ – residual regressor prediction on sample x.

Training on different transformation of target variables may help to enhance models diversification and therefore to improve ensemble accuracy. To apply this idea for residual regressor we decide to train another residual regressor, which would predict the residual multiplication coefficient, and training target values for him would be calculated as a result of division known target variables by values, predicted by first level model. So the final answer takes the form:

$$y(x) = 0.85 * (e(x) + c_1(x)) + 0.15 * (e(x) * c_2(x)),$$

where $e(x)$ – answer of ensemble of models for sample x, $c_1(x)$ – prediction of addition residual regressor, $c_2(x)$ – prediction of multiplication residual regressor.

VI. Dropout For Lasso Training

During our investigation of Lasso regression training we can see that some features with obscure meaning (like product of GrLivArea by overall_poor_condition) receive big weights coefficient, and some variables that by meaning should have positive weight coefficient, receive negative coefficient, and vice versa, especially polynomial features. That can be explained as some features compensate negative side effect of another features, so some features form a "bunch of features" during gradient descent process in Lasso training. As gradient descent performs coordinate-wise, decreasing of any feature's weight coefficient in a bunch leads to decrease target function, so bunches of features becomes a stable structure. But this makes training and proper regularization harder (as bunch of features makes a significant contribution into L1 penalty term), and makes feature selection and feature engineering harder so on. This is very hard to add more complex features in the algorithm without accuracy decreasing, when this phenomenon occurs.

This phenomenon is very similar to neuron's co-adaptation that occurs in deep neural networks (neurons adopting to compensate errors of other neurons). In deep neural networks training, "dropout" procedure [4] applied to prevent this phenomenon and improve generalization ability. So similar procedure can be applied to Lasso training for enable to improve model accuracy by adding more complex and descriptive features. We can assign some coefficients to zero on some training iterations, which should destroy feature bunches, but informative single features should alive.

TABLE I. LASSO REGRESSION TRAINING RESULTS WITH DROPOUT AND EARLY STOPPING

Dropout rate p	Iterations between applying dropout q	Total iterations count	Cross-validation error
0	0	15000	0.10265
0	0	150	0.10244
0.2	5	3000	0.10317
0.1	1	3000	0.10821
0.1	5	3000	0.10221
0.1	10	3000	0.10111
0.1	20	3000	0.10134
0.05	10	4000	0.10095

Another advantage should be a knocking out of local minima, that is useful for any local optimization algorithm. Lasso regression training can be regarded as simplified particular case of neural network training, so very similar regularities are present in both of these algorithms.

Therefore we decide to implement dropout for Lasso regression training. In our dropout implementation after every q iterations some randomly selected portions of model's weight coefficients (with respect to dropout ratio p) set to zero, and Lasso training continue with previously selected weight coefficients, with the exception of zeroed ones. Results are shown in Table 1. We can see that dropout can significantly decrease regression error on independent test data (best decreasing is about 0.0015).

We also observed that Lasso regression works better with early stopping (another technique, very similar to neural network training). This technique consists in stop of training when cross-validation error of model begins to increase. In our experiments, we found that 150 iterations of training give best result. Therefore, we compare ordinary Lasso training with early stopping against Lasso with dropout and early stopping applied simultaneously.

VII. LOGIT TRANSFORMATION

Due to unbalancing phenomena in the tree training procedure, tree's answers tend to average target value and residual curve takes the near-sigmoidal form (see Fig. 4). To address this issue we perform logit transformation of tree's answers. Logit function is the inverse function for sigmoid function, so after applying this transformation to tree's answers residuals curve takes more proper form, which is reflected in stable accuracy improving.

For calculate transformation, we firstly normalize our predictions to lie into [0, 1] segment:

$$pn(x) = \frac{p(x) - \min(p(x))}{\max(p(x)) - \min(p(x))},$$

where $p(x)$ – predictions for sample x, $pn(x)$ – predictions, normalized to lie into [0, 1] segment, $min(p(x))$ – 1-percentile for all $p(x)$, $max(p(x))$ – 99-percentile for all $p(x)$.

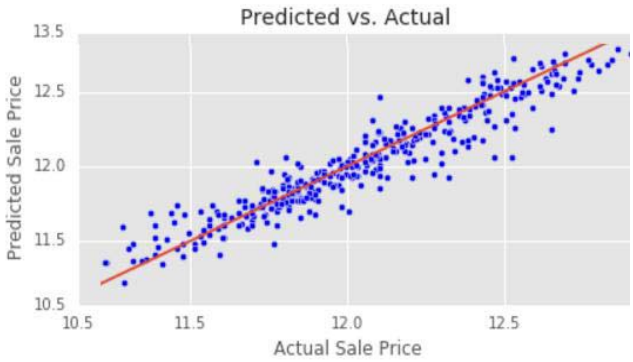

Fig. 4. Residuals distribution for XGBoost predictions in log scale.

We use percentiles here due to unrobustness default minimum and maximum operations to outliers. After prediction's normalization we calculate logit transform using logit function formula:

$$\text{logit}(\text{p}(\text{x})) = \log_C \frac{pn(x)}{1 - pn(x)},$$

where C – adjusted constant, determining the intensity of the transformation.

As we want to perform small output transformation, we need to use very large C value. In our solution, we set $C = 5 * 10^{68}$. Value of C can also be tuned using cross-validation.

VIII. NEURAL NETWORK MACHINE

Neural network consists of a set of individual neurons, and each neuron takes response for some case recognition. In case of small training set size, large network size, or insufficient regularization, some neurons may learn very particular consistent patterns, which are true only on training set, and lead to improper predictions on unseen data. All of this leads to overfitting or excessive regularization, which worsens the final accuracy. However, these patterns will be destroyed, if we retrain our model on previously unseen portion of data. To address this problem, we perform following procedure: firstly we split training data for two parts of equal sizes, then we train our neural network on a first subset of data, after that we re-train our neural network on second subset of data, starting with previously learned weights. But after this procedure neural network may destroy false particular consistent patterns learnt from first subset, but learn new particular consistent patterns for second subset of data. Therefore, we iterate this procedure several times, and train neural network on full portion of the data in the end of each iteration. This allow us to enlarge neural network size without risk of overfitting and therefore to increase final quality. Further quality improvement with this technique can be obtained by non-random splitting the data on two subsets, which leads to eliminating more stable lie patterns. Initially this technique is proposed to solve cross-dataset generalization problem.

TABLE II. CROSS-VALIDATION RESULTS FOR SOME VARIANTS OF OUR SOLUTION

Algorithm	*CV score*	*CV score standard deviation*
Lasso	0.11139	0.0106
XGBoost	0.13058	0.0108
XGBoost with logit transform	0.12986	0.0107
ElasticNet	0.11203	0.0107
Neural network	0.11787	0.0095
Neural network machine	0.11695	0.0096
Ensemble of Lasso, XGBoost, ElasticNet and Neural network machine	0.11125	0.0109
Ensemble + residual regressor	0.11108	0.0120
Ensemble + ensemble of residual regressors	0.11093	0.0120
Full solution without polynomial features	0.11127	0.0114

IX. EXPERIMENTAL RESULTS

In this section, we will evaluate accuracy of basic models of our algorithm separately, accuracy of ensemble of this models, and accuracy with our improvements. For evaluating accuracy of the algorithms, we use 5-fold cross-validation. Results are shown in Table 2. As we can see, standard deviation of the results is very high because of small dataset size.

X. CONCLUSION

We describe our solution for house's sales price prediction problem for one of the Kaggle competitions, in which we combine standard machine learning algorithms with our original ideas like residual regressor, logit transform and neural network machine. At the end of the competition, we took 18th place among 2124 participants from whole world (top 1%). The best way to improve result is to train more base models for ensembling (about 50 models instead of six models in our solution, like this do another contestants), which requires more computational resources. Another problem with many models ensembling is how to make errors of these models uncorrelated, otherwise we would not receive quality improvement. One of the ways to achieve models decorrelation is to learn same models on different feature sets.

REFERENCES

[1] D. De Cock "Ames, Iowa: Alternative to the Boston housing data as an end of semester regression project" J. of Statistics Education. vol. 19, No. 3, 2011.

[2] https://www.kaggle.com/c/house-prices-advanced-regression-techniques

[3] R. Tibshirani "Regression shrinkage and selection via the lasso" J. of Roy. Soc. London. Series B (Methodological), pp. 267-288, 1996.

[4] N. Srivastava, G.E. Hinton, A. Krizhevsky, I. Sutskever, R.R. Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting" The J.l of Machine Learning Research, Vol. 15, No. 1. pp. 1929–1958, 2014.

[5] H. Zou, T. Hastie "Regularization and variable selection via the elastic net" J. of Roy. Soc. London. Series B (Statistical Methodology), Vol. 67, No. 2, pp. 301-320, 2005.

[6] J. H. Friedman "Greedy function approximation: a gradient boosting machine" pp. 1189-1232, 2001.

[7] https://github.com/dmlc/xgboost

[8] M. W. Schmidt et al. "Optimizing Costly Functions with Simple Constraints: A Limited-Memory Projected Quasi-Newton Algorithm" AISTATS, Vol. 5, pp. 456-463, 2009.