Semester Project

Shreyas Rewagad [1], Jivitesh Poojary [2]

All the work herein is solely ours

Abstract

The ever-changing market condition warrant for an adaptive gauging tool that would help predict the property evaluation from time to time. In this paper, we have enlisted various solutions that could resolve the problem of property evaluation. The paper discusses the techniques employed to predict the house prices of individual residential property in Ames, Iowa from 2006 to 2010 for which the data is made available from Kaggle. Replicating the methods demonstrated in this paper, one should be able to acurately predict the property's value if conferred with the necessary data.

Data Science, School of Informatics and Computing, Indiana University, Bloomington, IN, USA
srewagad@umail.iu.edu[1], jpoojary@umail.iu.edu[2]

Contents

1. Introduction

The data set describes the sale of individual residential property in Ames, Iowa from 2006 to 2010. This data is obtained directly from the Ames Assessor's Office which is used for tax assessment purposes but lends itself directly to the prediction of home selling prices. The main objective of this project is to predict the selling price of homes.

2. Background & Process Overview

House Price prediction is one of the necessary activities in real estate business, this task is provides vital data to government, real estate agencies, etc. allowing them to gauge and strategize their business.

To efficiently predict the correct property value, we have taken a thorough look at the dataset and treated it for missing value. Intuitively, after imputing the missing values; a feature selection is being done to reduce the dimensions and only accept the features that aid in the price prediction.

As regression and any prediction technique works best on numerical data, categorical attributes are converted into numerical. This phase redoubled the attribute count and as all the different levels would not process the same significance in predicting the 'SalePrice', hence a second feature selection was done, this also helped on cutting down on the computation time.

The data thus obtained was suitable for building various regression models. For each model to make a confirm prediction; process of cross-validation was designed which incorporated sampling the dataset into a train and a test set. and training the predictor function iteratively for various small samples of the dataset. This also helps us to reduce overfitting and to tune the predictor function as the learning rate of the predictor function is reduced.

In addition to this, the error of the above process is calculated and parameter tuning is done so that the error is minimized. The different models and the entire approach is explained in detail in the forthcoming topics.

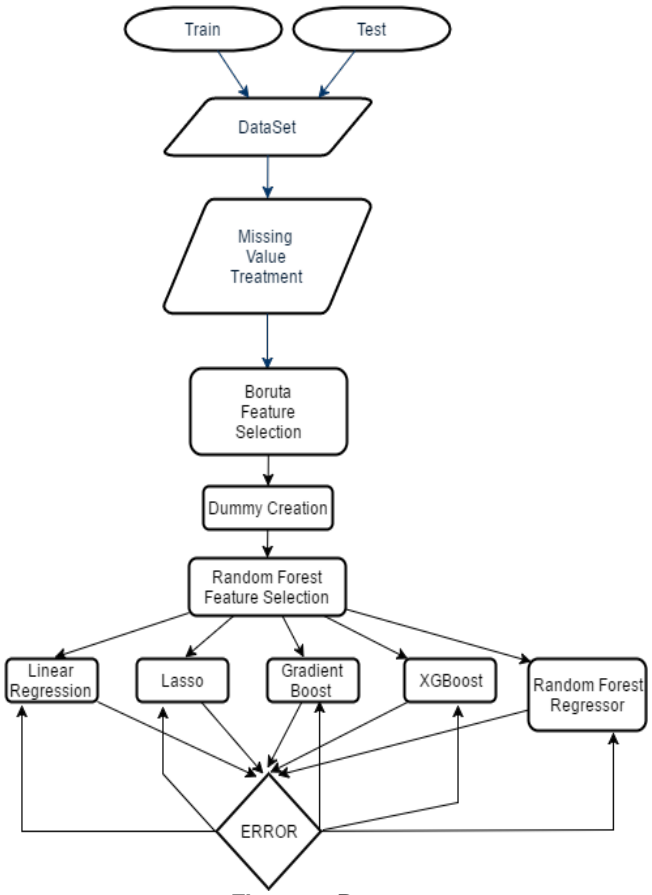

Figure 1. Process

3. Exploratory Data Analysis & Data Preprocessing

3.1 Variable Analysis

The dataset contains necessary descriptive variables that depict the physical attributes of residential homes in Iowa sold between year 2006 and 2010. The attributes (square footage, number of bedrooms and bathrooms, size of lot, etc.) accurately delineate the property to the perspective home buyer. The data set contains 2919 observations.
Total 80 variables out of which:

1. Nominal: 23
2. Ordinal: 23
3. Discrete :14
4. Continuous:20

3.2 Missing Values Treatment

1. The dataset contains categorical variables with a specific level - 'NA', this essentially means that the entity that the variable specifies is not present on the property. The implementation is being done in python which treats 'NA' as missing values. Replacing all these variable levels as 'NONE'. Below are such variables:
2. Missing values in categorical variables are treated by imputing respective mode values. Below is the list of such variables:

Table 1. Variables with Level as 'NA'

Alley	GarageType	BsmtQual
GarageFinish	BsmtCond	GarageQual
BsmtExposure	GarageCond	BsmtFinType1
PoolQC	BsmtFinType2	Fence
FireplaceQu	MiscFeature	

Table 2. Categorical variables with missing values

MSZoning	Utilities	Exterior1st
Exterior2nd	MasVnrType	Electrical
KitchenQual	Functional	SaleType

3. Imputing missing values for some of the variables involve a level of domain understanding, such as if garage doesn't exist then there would be zero garage cars, similar with the basement, pool and other property attributes that are missing for a reason. These are imputed in the most logical, listing them below:

Table 3. Imputing missing values

Variable	Condition	Imputed
MasVnrArea	MasVnrType==NONE	0
BsmtFinSF1	BsmtFinType1== NONE	0
BsmtFinSF2	BsmtFinType2 == NONE	0
BsmtUnfSF	BsmtQual == NONE	0
BsmtFullBath	BsmtQual == NONE	0
BsmtHalfBath	BsmtQual == NONE	0
TotalBsmtSF	BsmtQual == NONE	0
GarageYrBlt	GarageFinish == NONE	YearBuilt
GarageCars	GarageFinish == NONE	0
GarageArea	GarageFinish == NONE	0
LotFrontage	No Street	0

4. Feature Selection

Three main goals of feature selection are:

1. Simplification of the model for the interpretation.
2. To reduce the training time for the model.
3. To enhance the generalization by reducing overfitting.

4.1 Stepwise

Stepwise[2][3] regression includes regression models in which the choice of predictive variables is carried out by an automatic procedure. The stepwise[2][3] feature selection can be done in 3 ways, forward, backward and both. In Backward Stepwise Regression, it begins with the full least squares model containing all 79 predictors and the label/prediction variable - SalePrice, and then iteratively removes the least useful predictor, one-at- a-time with the selection criteria depending on the Adjusted R2, AIC or BIC scores. Akaike

information criterion is a measure of the relative quality of statistical models for a given set of data. Based on AIC the stepwise[2][3] feature selection took in the linear model and eliminated the below features: Out of total 79 features available, 49 were selected building the model.

Table 4. List of Rejected Features after Step - Backward iterative elimination

Rejected Features		
LotFrontage	Alley	LotShape
HouseStyle	Exterior2nd	ExterCond
Foundation	BsmtFinType1	BsmtFinType2
TotalBsmtSF	Heating	HeatingQC
CentralAir	Electrical	LowQualFinSF
GrLivArea	BsmtHalfBath	HalfBath
FireplaceQu	GarageType	GarageYrBlt
GarageFinish	PavedDrive	OpenPorchSF
EnclosedPorch	3SsnPorch	MiscFeature
MiscVal	YrSold	SaleType

4.2 Boruta

Boruta[1] is a feature selection algorithm. It works as a wrapper algorithm around Random Forest. This technique achieves supreme importance when a data set comprised of several variables is given for model building.

4.2.1 Working of Boruta

1. It adds randomness to the given data set by creating shuffled copies of all features.
2. Then, it trains a random forest classifier on the extended data set and applies a feature importance measure to evaluate the importance of each feature where higher means more important.
3. At every iteration, it checks whether a real feature has a higher importance than the best of its shadow and constantly removes features which are deemed highly unimportant.
4. The algorithm stops either when all features gets confirmed or rejected or it reaches a specified limit of random forest runs.

4.2.2 Result of Boruta on the given Dataset

Summary: Rejected = 19, Tentative=14, Confirmed=46 Note: Tentative attributes have importance so close to their best shadow attributes that Boruta[1] is not able to decide with the desired confidence in default number of random forest runs.

In the above plot Red, yellow and green boxplots represent Z scores of rejected, tentative and confirmed attributes respectively. The tentative attributes will be classified further as confirmed or rejected.Classification is done after comparing the median Z score of the attributes with the median Z score of the best shadow attribute.

Out of 14 tentative features 3 are classified are rejected and 11 are classified as confirmed by the Boruta[1]:

Table 5. List of Rejected Features after initial run(Boruta)

Rejected Features		
Condition2	X3SsnPorch	MoSold
BsmtFinSF2	MiscFeature	MiscVal
Street	BsmtHalfBath	Utilities
LotConfig	BsmtFinType2	PoolQC
Heating	RoofMatl	PoolArea
SaleType		

Table 6. List of Tentative Features after initial run(Boruta)

Tentative Features		
LotShape	BsmtExposure	Condition1
EnclosedPorch	MasVnrType	LotFrontage
BsmtCond	RoofStyle	Functional
ScreenPorch	Fence	LandSlope
SaleCondition	Alley	

1. Rejected: BsmtCond, EnclosedPorch, LandSlope
2. Confirmed: LotShape, Condition1, MasVnrType, Functional, Fence, SaleCondition, BsmtExposure, LotFrontage, RoofStyle, ScreenPorch, Alley
3. Summary: Rejected = 22 Confirmed = 57.

4.3 Boruta over traditional feature selection algorithms

Boruta[1] follows an all-relevant feature selection method where it captures all features which are in some circumstances relevant to the outcome variable. In contrast, most of the traditional feature selection algorithms follow a minimal optimal method.They rely on a small subset of features which yields a minimal error on a chosen classifier.

5. Dummy Variables

5.1 Purpose of Dummy variables

Dummy variables are the numeric stand-ins for qualitative facts in a regression model. In regression analysis, the dependent variables may be influenced not only by quantitative variables but also by qualitative variables. It takes the value 0 or 1 to indicate the absence or presence of some categorical effect that may be expected to shift the outcome. After treating all the variables for missing values, we have created dummies for the categorical variables. Total variable count after creating dummies is 224.

6. Experimental Section

After removing Cleaning data and selecting features we are building below models on the dataset:

6.1 Linear Regression

Simple linear regression is a statistical method. It allows us to summarize and study relationships between two continuous

Table 7. List of Confirmed Features after initial run(Boruta)

Confirmed Features		
MSSubClass	MasVnrArea	FireplaceQu
YearBuilt	TotalBsmtSF	GarageQual
BsmtUnfSF	X2ndFlrSF	LotArea
X1stFlrSF	HalfBath	HouseStyle
FullBath	Fireplaces	BsmtFinSF1
GarageCars	GarageArea	HeatingQC
OpenPorchSF	BldgType	GrLivArea
MSZoning	ExterQual	KitchenAbvGr
LandContour	BsmtFinType1	GarageType
YearRemodAdd	BedroomAbvGr	GarageCond
OverallQual	Exterior1st	Foundation
CentralAir	BsmtFullBath	KitchenQual
GarageYrBlt	PavedDrive	Neighborhood
OverallCond	Exterior2nd	BsmtQual
Electrical	TotRmsAbvGrd	GarageFinish
WoodDeckSF		

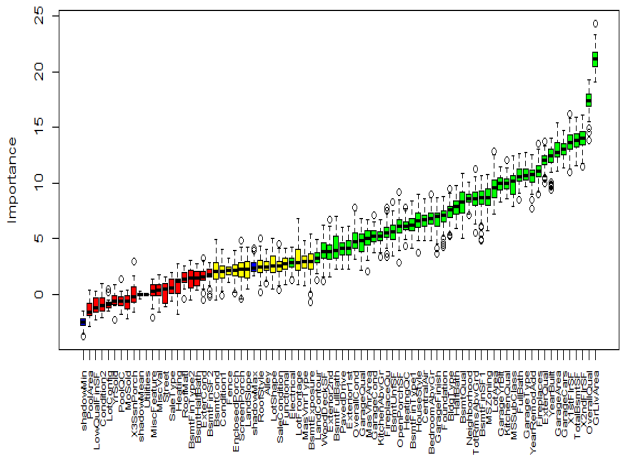

Figure 2. Graph of the importance of the features

variables. A predictor that has a low p-value is a meaningful addition to your model. As the changes in the predictor's value are related to changes in the response variable. A large (insignificant) p-value suggests that changes in the predictor are not related to the changes in the response.

We have applied sklearn.linear_model.LinearRegression. With below parameters:

1. Fit intercept: whether to calculate the intercept for this model. If set to false, no intercept will be used in calculations.
2. Normalize: If True, the regressors X will be normalized before regression. This parameter is ignored when fit intercept is set to False.

For fit intercept = True/False and normalize = False we are getting low error value that is 0.00.

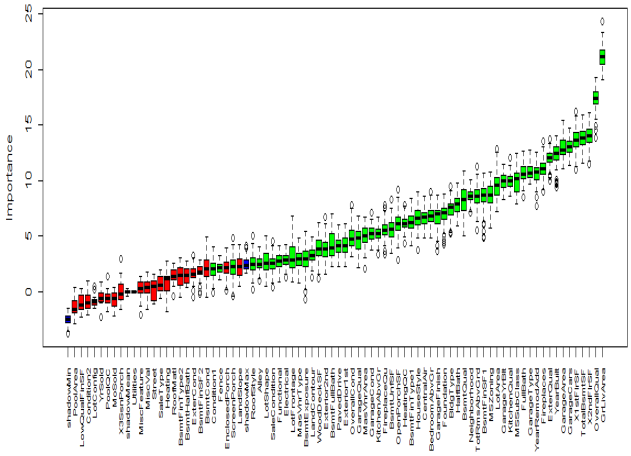

Figure 3. Boruta result plot after the classification of tentative attributes

6.2 Random Forest

Random forest[6] is like bootstrapping algorithm with Decision tree (CART) model. Random forest[6] tries to build multiple CART model with different sample and different initial variables. For example, it will take a random sample of 100 observation and 5 randomly chosen initial variables to build a CART model. It will repeat the process (say) 10 times and then make a final prediction for each observation. Final prediction is a function of each prediction. This final prediction can simply be the mean of each prediction.

We have applied sklearn. ensemble. RandomForestRegressor. With below parameters:

1. n_estimators: The number of trees in the forest.
2. min_samples_split: The minimum number of samples required to split an internal node.

From the Random Forest Error graph we can clearly see that there appears no particular trend, the error rate is relatively same. Also the small error rate of $\approx$ 0.02 makes us think that the model might be overfitted to the training data and might not do well with test data. The lowest error of 0.004, occurs when n_estimators = 270 and min samples split = 7.

6.3 Least absolute shrinkage and selection operator

It is a regression analysis method which performs both variable selection and regularization to enhance the prediction accuracy and interpretability of the model. We have applied sklearn.linear_model.Lasso[10] with below parameters:

1. Normalize: If True, the regressors X will be normalized before regression. This parameter is ignored when fit intercept is set to False
2. Alpha: Constant that multiplies the L1 term. Defaults to 1.0. alpha = 0 is equivalent to an ordinary least square, solved by the Linear Regression object.

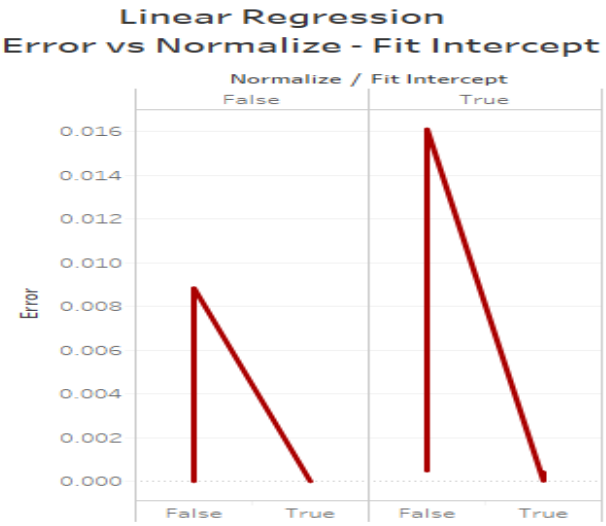

Figure 4. Plot for Linear Regression Error against fit intercept and normalize

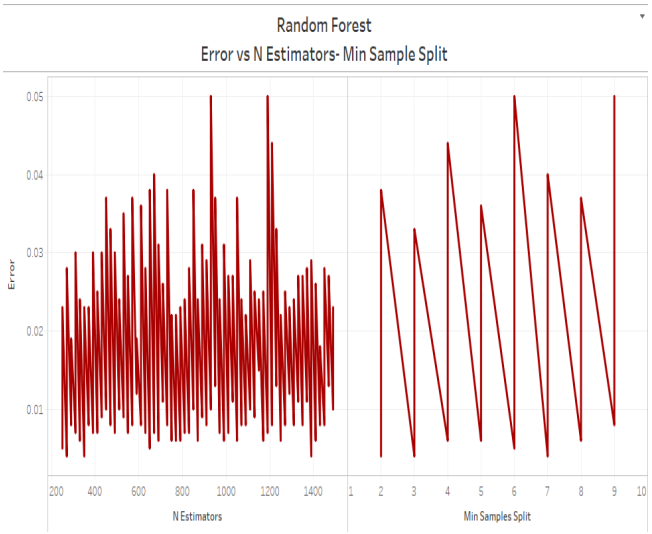

Figure 5. Plot for Random Forest Error against N Estimators and Min Sample Split

For any value of Normalize and Alpha we are getting low error value that is 0.00; though performing cross validation, value of error motivates us to think that the model greatly overfits the train data and tuning the parameters doesn't lead us anywhere. Thus disregarding this approach.

6.4 Ridge Regression

Ridge[7] Regression is a measure taken to alleviate multicollinearity amongst regression predictor variables in a model. Often predictor variables used in a regression are highly correlated. In such cases, the regression coefficient of any one variable depend on which other predictor variables are included in the model, and which ones are left out. (So, the predictor variable does not reflect any inherent effect of that predictor on the response variable, but only a marginal or partial effect, given whatever other correlated predictor variables are included in the model). Ridge[7] regression adds a small bias factor to the variables to alleviate this problem.

We have applied sklearn.linear_model.Ridge with below parameters:

1. Normalize: If True, the regressors X will be normalized before regression. This parameter is ignored when fit intercept is set to False.
2. Alpha: Regularization strength; must be a positive float. Regularization improves the conditioning of the problem and reduces the variance of the estimates. Larger values specify stronger regularization.
3. Solver: 'auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag'
   (a) auto chooses the solver automatically based on the type of data.
   (b) 'svd' uses a Singular Value Decomposition of X to compute the Ridge[7] coefficients.
   (c) 'cholesky' uses the standard scipy.linalg.solve function to obtain a closed-form solution.
   (d) 'sparse_cg' uses the conjugate gradient[8] solver as found in scipy.sparse.linalg.cg.
   (e) 'lsqr' uses the dedicated regularized least-squares routine scipy.sparse.linalg.lsqr.
   (f) 'sag' uses a Stochastic Average Gradient descent.

For Normalize = False/True, Alpha = 0 and Solver = Auto /svd / cholesky we are getting low error value that is 0.00. The model appears true to real, error spikes if the data is normalized, this is understandable as data of higher order is scaled and reduced, which negatively affects the prediction.

6.5 Gradient Boost

Gradient boosting[8] is a technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

We have applied XGBoost[9] with below parameters:

1. loss: It refers to the loss function to be minimized in each split. It can have various values for classification and regression case.
2. n_estimators: The number of sequential trees to be modeled.
3. learning rate: This determines the impact of each tree on the final outcome. The learning parameter controls the magnitude of this change in the estimates.
4. Subsample: The fraction of observations to be selected for each tree.

For subsample = 0.7, loss = Huber, n_estimators = 1000 and learning rate = 0.1 we are getting low error value that is

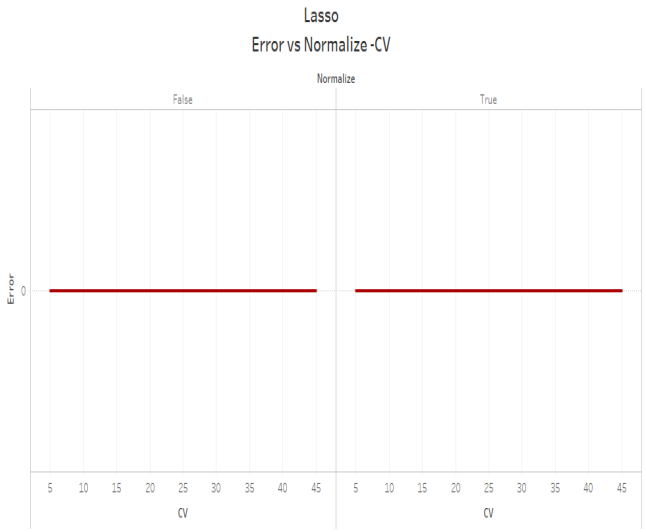

Figure 6. Plot for Lasso Regression Error against CV and Normalize

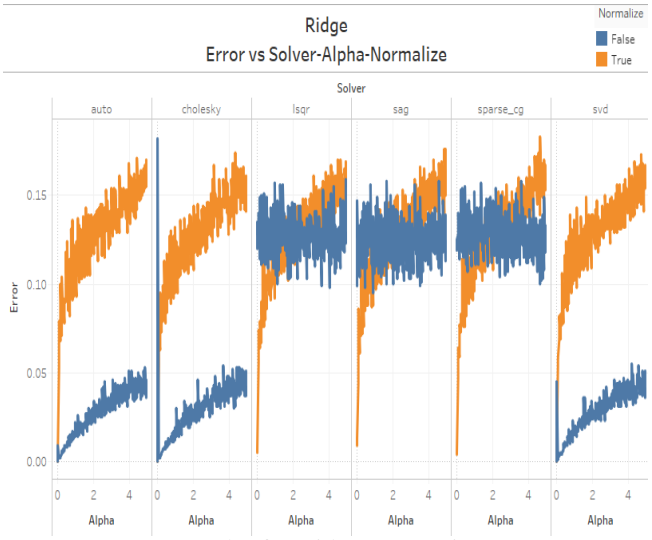

Figure 7. Plot for Ridge Regression Error

0.01. The error tends to reduce as the learning rate increases and the number of tree estimators increases.

6.6 XGBoost - Extreme Gradient Boosting

XGBoost is an optimized distributed gradient boosting system designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost[9] provides a parallel tree boosting(also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.

We have applied XGBoost with below parameters:

1. subsample: Denotes the fraction of observations to be randomly samples for each tree.
2. max_depth: The maximum depth of a tree.
3. learning rate: This determines the impact of each tree on the final outcome. The learning parameter controls

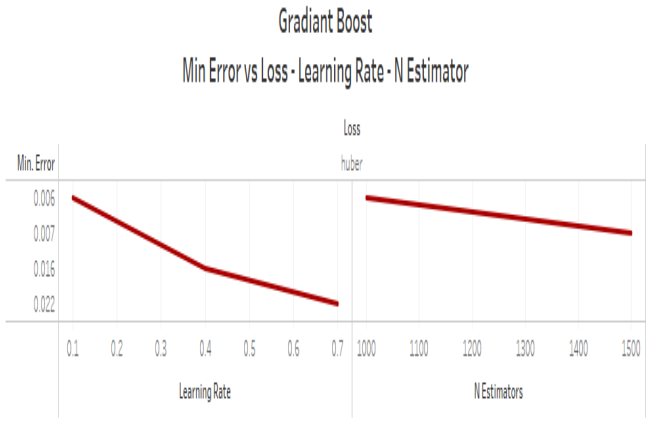

Figure 8. Plot for Gradient Boost Error against subsample, loss, n_estimators and learning rate

the magnitude of this change in the estimates.

For subsample = 0.7, learning rate = 0.1 and max_depth = 26 we are getting low error value that is 0.01. The error falls inline with the methodology of the model. Increase in learning rate reduces the error, In the same manner an increase in the size of the helps in better understanding of the target variable thus reducing the error rate.

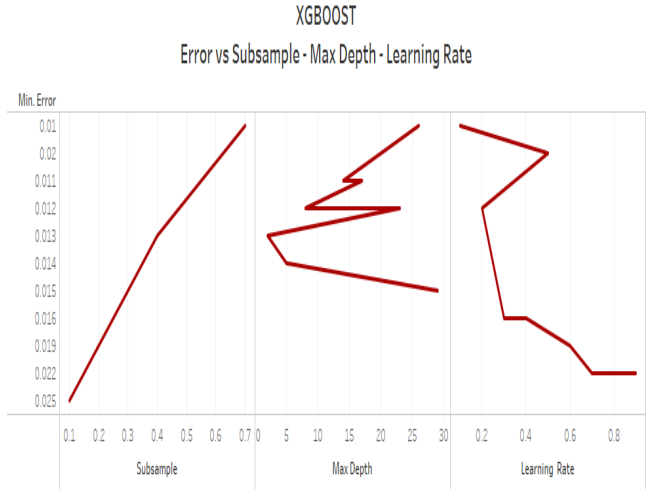

Figure 9. Plot of Error for XGBoost against subsample, learning rate, max_depth.

6.7 Tools & Infrastructure

Tasks were performed on the Sharks (SoIC server) and the Karst (IUB University Server), configuration listed in the table.

Table 8. Table of tools and infrastructure

Name	Type	Specification
Karst	Server	16 Cores x86 64 architecture
Sharks	Server	32 Cores x86 64 architecture
Python	Platform	2.7.3 64-bit version
Spyder	IDE	Version 2.3.9
Linux	OS	Red Hat Enterprise Linux Server release 7.3 (Maipo)
PuTTY	SSH Tool	Version 0.67
WinSCP	FTP tool	Version 5.9.3
R	Analysis Tool	Version 3.2.2
RStudio	IDE for R	Version1.0.44

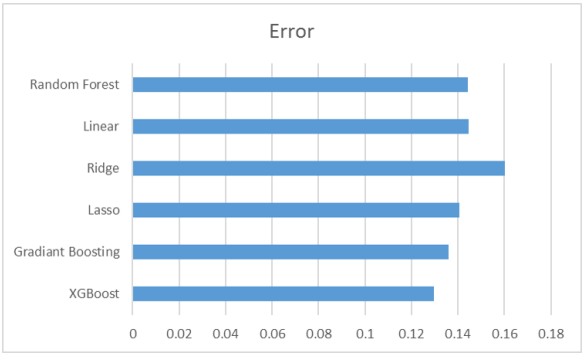

Figure 10. Model Comparison

Conclusion

The Chart (Figure 10) represents the error in the actual test data. Judging by the error it is definitely clear to us that the models - Lasso[10], Linear[5], Random Forest[6] and Ridge[7] are overfitting the train data and hence they perform poorly. The Gradient Boost[8] and XGBoost[9] though perform better, accuracy remains a concern.

6.8 Analysis & Future Work

Let us examine our approach, each model implemented was tuned for accuracy on the train data. Subsequently, performing cross validation to resolve the issue of overfitting, for which Error chart(Figure 10) state otherwise.

We could say that the feature selection phase should be further explored. Indeed, in the process of feature selection after the dummy[4] creation, we have sampled the train data and performed the random forest[6] feature selection on the various samples of data, yielding a list of features selected for every sample; finally getting the union of the feature set. This ensures that the features we select are only the once who contribute significantly towards predicting the 'SalePrice'.

We still might perform regression by using different kinds of techniques. But that the error on either of the existing techniques does not significantly vary encourages us to rethink the way we could have made refinements to our dataset.

Perhaps we might achieve better accuracy if a deep and through domain understanding of the real estate condition is done. This might help us correctly weigh the categorical variables. Understanding the domain would not only help us deduce ranks for levels in the category but also help us neglect the significant predictors that act otherwise.

Acknowledgments

The authors wishes to thank Prof. Mehmet Dalklic for numerous helpful discussions and comments.
Also, thanking Hasan Kurban, lead AI, for his guidance and valuable suggestions.
We also want to extend thanks to the Kaggle community for their varied analysis and approaches that inspired us to solve the problem.

References

[1] R Document: Boruta Package, https://cran.r-project.org/web/packages/Boruta/Boruta.pdf

[2] R Document: Stepwise Package, https://stat.ethz.ch/R-manual/R-devel/library/stats/html/step.html

[3] Implementation Strategy: Step Function. http://www.utstat.toronto.edu/ brunner/oldclass/appliedf11/handouts/2101f11StepwiseLogisticR.pdf

[4] Dummy Creation: get_dummies http://pandas.pydata.org/pandas-docs/version/0.18.1/generated/pandas.get_dummies.html

[5] Linear Regression http://scikit-learn.org/stable/modules/linear_model.html

[6] Random Forest Regression http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

[7] Ridge Regression http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

[8] Gradient Boosting Regressor http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html

[9] Boosted Trees http://xgboost.readthedocs.io/en/latest/model.html

[10] Lasso Regression http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoLarsCV.html