Capstone Project: Zillow House Pricing Competition

Diego Gallegos Garcia

Project Overview

House Price Prediction has been an important problem to solve in machine learning in the development of several algorithms. Some important examples have been Extreme Machine Learning (Guang-Bin Huang, 2006) with Feedforward Neural Network that analyzes California's housing prices. Also tree methods have given this problem a lot of attention. Also Expert Systems with Applications (Byeonghwa Park, 2015) showcases the usage of boosting and tree methods to predict house prices in Fairfax County, Virginia. The datasets have always the same shape, a series of features such as square footage to number of bedrooms and a sales price. Making the objective to predict the sales price for new additions to dataset. This has been a major improvement for many real estate companies sucha as Zillow, Trulia and Redfin. Most academic papers use MLS listing to train on their model as they are true data with a few years old.

Problem Statement

The Capstone Project for Machine Learning is about the Zillow House Prediction Kaggle Competition. The competition posed the problem to improve the algorithm that Zillow uses to predict house prices based on a dataset that contains a previous period of house sales. The dataset not only contains house sales but also a series of features about the homes such as square footage, number of bedrooms and bathrooms along with the location of the house. The house prices that are going to be assessed is the last quarter of 2017 which was available after the public dataset phase was done. The fact that we have the sales which is the variable that must be predicted, it indicates that the problem at hand is classified as a supervised learning problem. A small set of algorithms have been chosen to be used on the competition such as Gradient Boosting Decision Trees and to improve the accuracy of the algorithm meta ensemble (stacking) has been implemented. The competition uses the logarithmic error between the Zillow House Sale algorithm prediction (Zestimate) and the real house sale price. Its worth to mention that real house sale prices are not available to competitors for some of the months. (Zillow, 2018)

Metric

The target variable can be defined as follows:

$$Logerror = \log(Zestimate) - \log(SalePrice)$$

An important remark is to know a little about Kaggle competitions dynamics. The first stage of the competition consists in submitting prediction against the public dataset which contains limited data regarding newer house sales events and the private dataset that has the whole universe of training examples that assesses the ability of the predictor generalize to newer data. This is a very important step as there is a very thin line between creating accurate models and creating models that can only adapt to data that it already knows. The metric being used to access all submissions is the Mean Absolute Error (MAE) between the predictions and the real value.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

The other most used metric is RMSE (Root mean squared error), they are similar in that both are in the range of positive real numbers. But they differ in that RMSE tends to give more weight to large errors while MAE is steady. Therefore, it can generate outliers that later have to be processed, in this specific dataset as seen in Figure 2 the distribution of many features has a lot of variance and skew distributions. MAE will favor the predicted value steadiness as RMSE will tend to overshoot from the real value. RMSE implications not only describes average error but also implications that are harder to interpret and understand for specific datasets.

Data Exploration

During the preprocessing phase, the variable types are considered to chose what techniques to apply. The chosen technique depends whether the model being used is tree-based or non-tree based. As the model being used is a tree based model, the numerical values are not scaled (min-max or standard scaling).

First lets see the provided dataset by the organizer. The datasets are provided by the Kaggle Competition It consists in:

- **properties_2016.csv** - all the properties with their home features for 2016.
- **train_2016.csv** - the training set with transactions from 1/1/2016 to 12/31/2016
- **sample_submission.csv** - a sample submission file in the correct format.

Properties dataset contains all the training features of the parcels. The shape of the data is 2,985,217 x 58. There 58 columns where 57 represent the training features and one is the parcel id. Some of the most important features are:

- transaction_month: month when the transaction took place.
- latitude: Latitude of the middle of the parcel multiplied by 10e6
- longitude: Longitude of the middle of the parcel multiplied by 10e6
- calculatedfinishedsquarefeet: Calculated total finished living area of the home
- structuretaxvaluedollarcnt: The assessed value of the built structure on the parcel
- landtaxvaluedollarcnt: The assessed value of the land area of the parcel

- yearbuilt: The Year the principal residence was built
- lotsizesquarefeet: Area of the lot in square feet
- taxvaluedollarcnt : The total tax assessed value of the parcel
- regionidzip: Zip code in which the property is located
- finishedsquaredfeet12: Finished living area
- bedroomcnt: Number of bedrooms in home
- bathroomcnt: Number of bathrooms in home including fractional bathrooms
- rawcensustractandblock: Census tract and block ID combined - also contains blockgroup assignment by extension

The rest of the explanation features can be obtained on the **zillow_data_dictionary.csv** file in the datasets on the kaggle data section.

Training dataset contains 90,275 training examples:

- Parcelid: the id of the property to be referenced on the properties dataset.
- Logerror: the target variable which is the logerror between the real price and Zestimate (Zillow's prediction)
- Transactiondate: the date when the sale was made.

	parcelid	logerror	transactiondate
0	11016594	0.0276	2016-01-01
1	14366692	-0.1684	2016-01-01
2	12098116	-0.0040	2016-01-01
3	12643413	0.0218	2016-01-02
4	14432541	-0.0050	2016-01-02

calculatedfinishedsquarefeet	finishedsquarefeet12	finishedsquarefeet13	finishedsquarefeet15
NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN
73026.0	NaN	NaN	73026.0
5068.0	NaN	NaN	5068.0
1776.0	NaN	NaN	1776.0

Figure 1 Peek for dataset of features

Figure 2 Values for all feature variables and its distribution

Exploration Visualization

The first step towards exploring data that was taken, was to analyze the target variable. The logerror distribution graph helps assess the skew-ness and normality of the target variable.

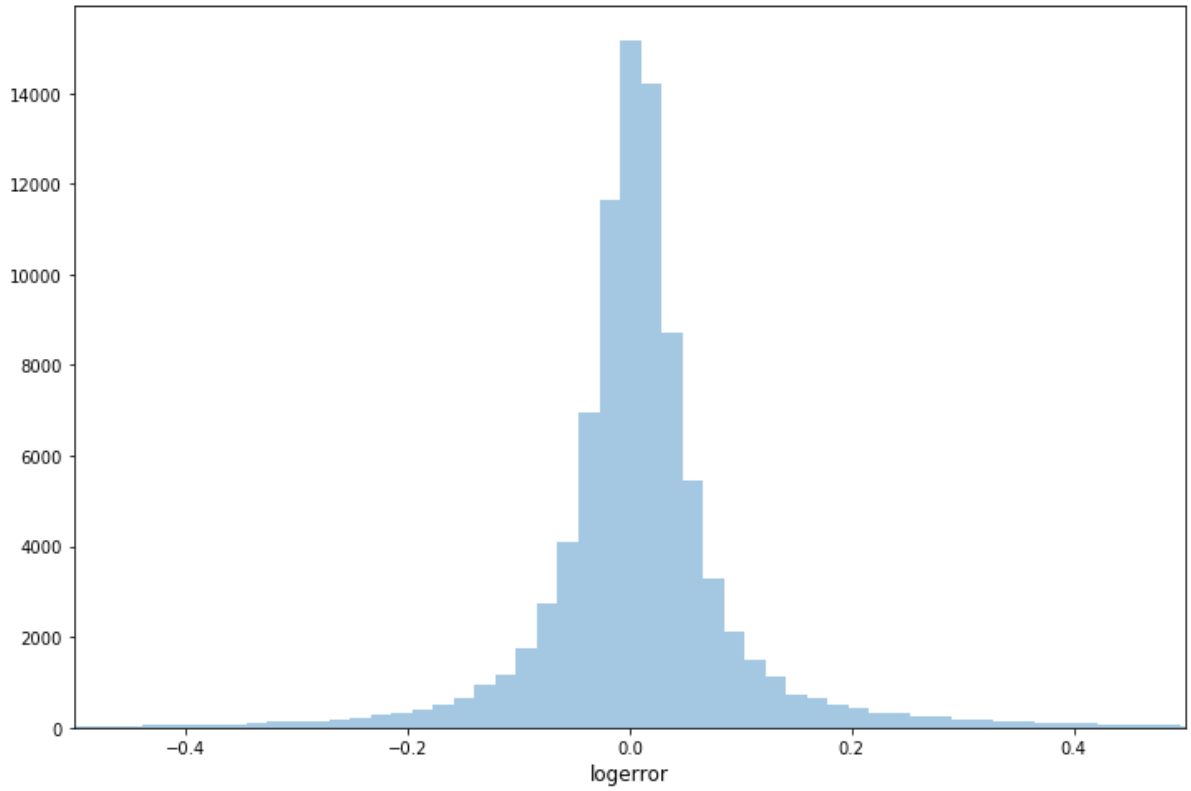

Figure 3 Logerror normal distribution

The data distribution across the dates of interest contains enough data points for each month to make learn accurately the last quarter of the year. One important thing to notice is the small amount of data points for the the last quarter, this is done on purpose as it was revealed for assessing the algorithms predictive power during the private dataset phase.

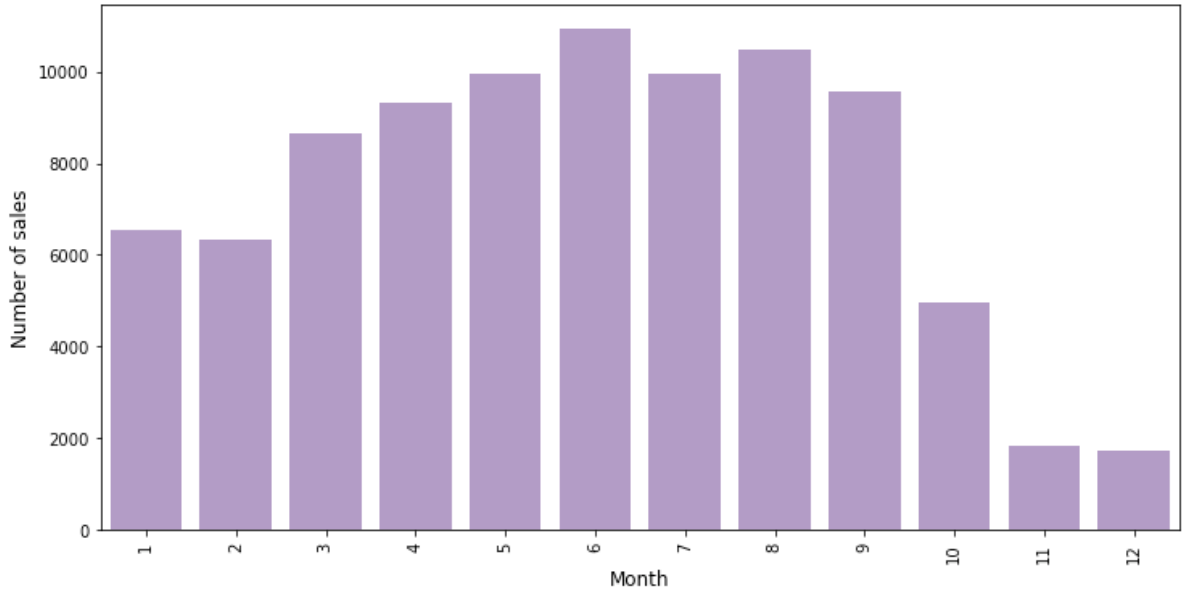

Figure 4 Monthly Sales Prices

Benchmark

The comparison will be between different boosting methods such as lightgbm and xgboost . The first benchmark used as a baseline was using sklearn model to test the accuracy with more advanced and engineered libraries.

Algorithm	Scored (MAE)
AdaBoost	0.104
Gradient Boosting Regressor	0.0242

This first solutions were not submitted as the MAE given on the first iterations the more advance libraries were far superior. Next, the initial submission scores for individual algorithms.

Algorithm	Public Score	Private Score
Xgboost	0.0795558	0.0894925
Lightgbm	0.0648173	0.0756822

This will be compared to the final ensemble solution.

Data Preprocessing

Next, the training features must be analyzed to find out what pre-processing step do they need. First, the amount of values that are null are plotted to know the sparsity of the dataset. As it can be seen, there are variables that for most of the training examples the values are null. Training features with sparsity greater than 80% were discarded from the dataset.

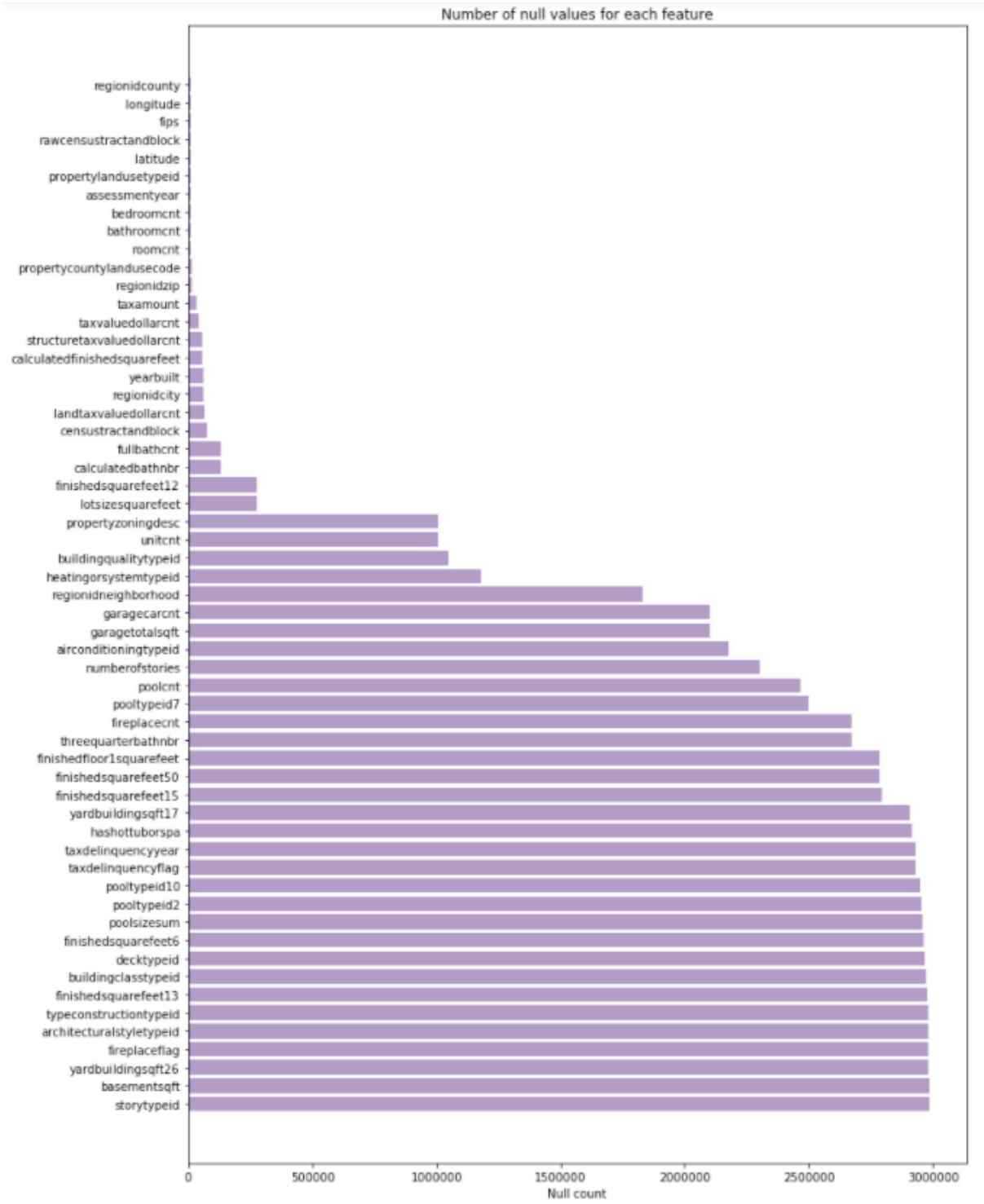

Figure 5 Number of missing values per feature

The next step, towards pre-processing the training features is imputation or also called handling missing values. Some of the methods tried were:

1. Filling the value with a number outside of the missing value range. (-1, -9999)
2. Fill the value for mean and median
3. Reconstruct value by interpolation.

4. Create a column that is called **isnull** (Boolean) that indicates whether a value is null or not.

We made experiments for the first two methods and stick with filling values with mean values. Filling values with median led to the same results.

Correlation

Correlation is used to investigate the relationship between continuous variables. This is analysis was conducted only to compare it with the results of the Feature Importance Analysis that uses Xgboost (Boosting Trees) to calculate by entropy techniques what features have greater weights towards the decision of predicting house pricing. As it can be seen on the plot of correlation of variables with the target variable it can give us an insight of which variables are important than others to predict the logerror correctly. Correlation must be handled with caution as it is known that correlation does not imply causation and that why it is used a comparative and exploratory analysis.

The values with greater positive and negative correlation are plotted.

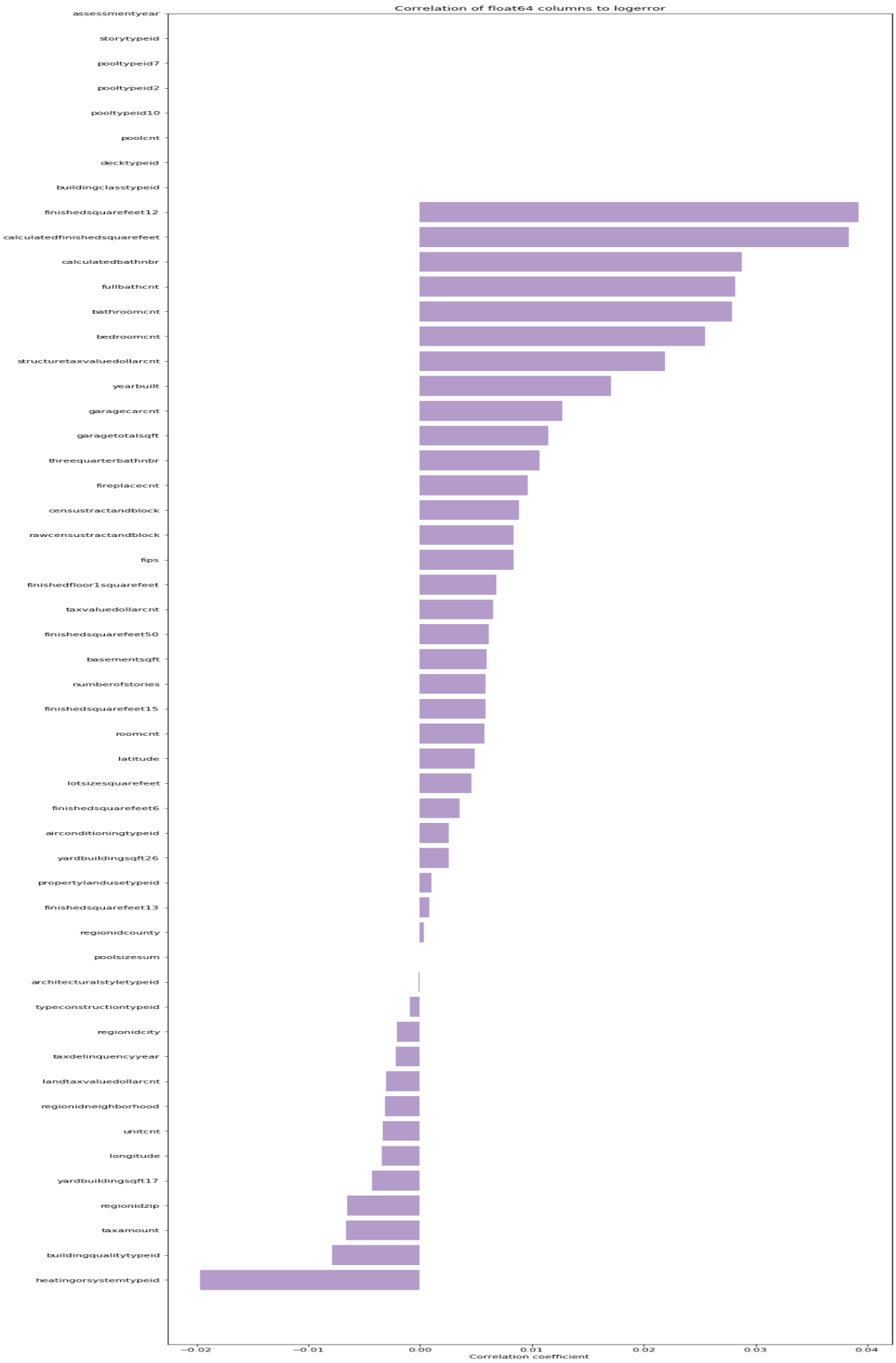

Figure 6 Correlation of variables with respect to target

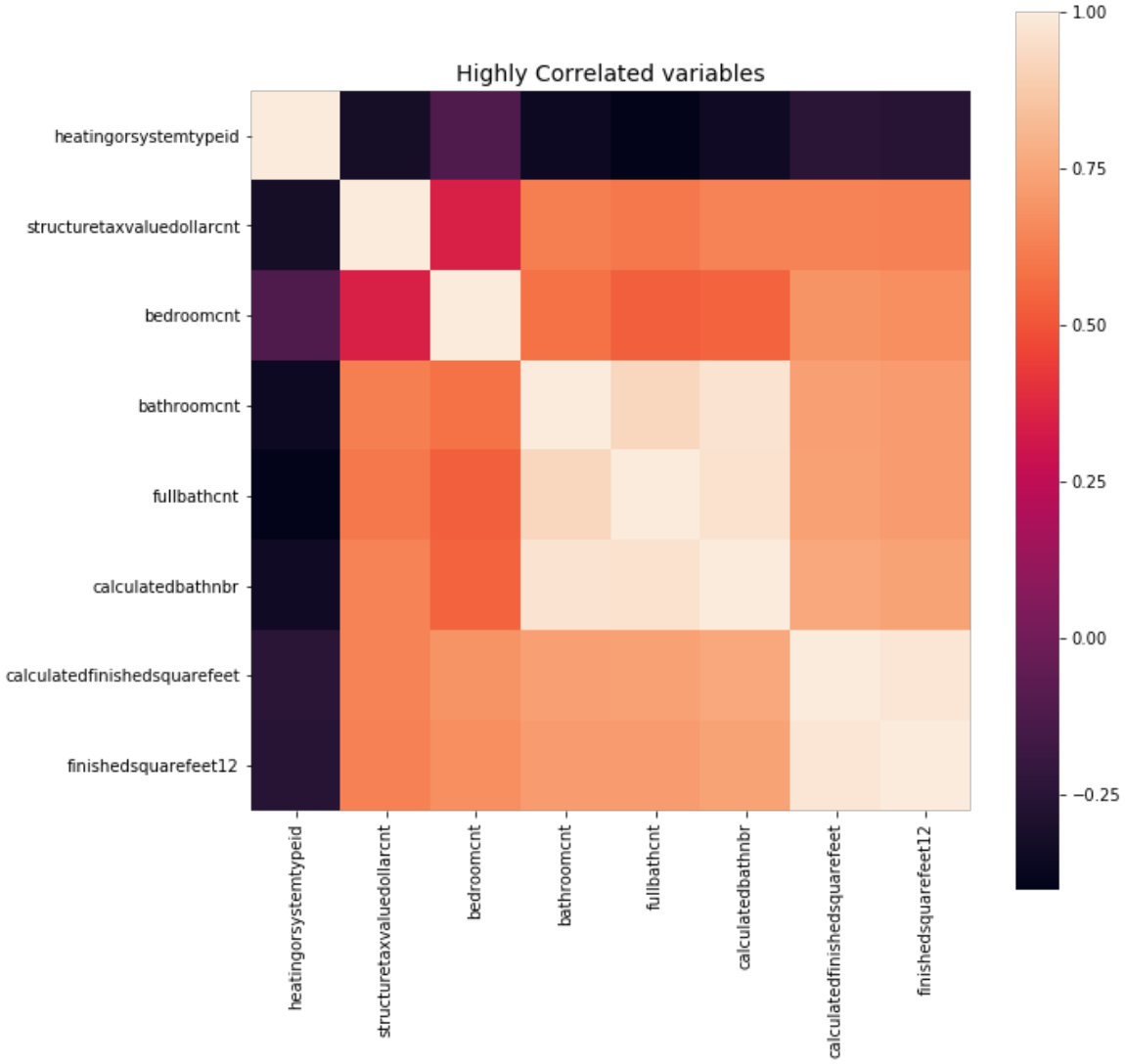

Figure 7 Most correlated features heat map with r/target

Feature Importance

Now, based on the correlation plots and its results it is desired to compare it with a most robust features importance algorithm based on decision trees. Decision trees use entropy as its voting mechanism to determine on which variable to split. Entropy obtains the best split based on the which split conserves the most information. Which can be defined as follows, where H denotes the entropy and p for the probability of occurring the event x where x is training feature and then the sum of all features it is taken.

$$H(X) = \sum_{i}^{x} -p_i(x)\, log_2(p_i(x))$$

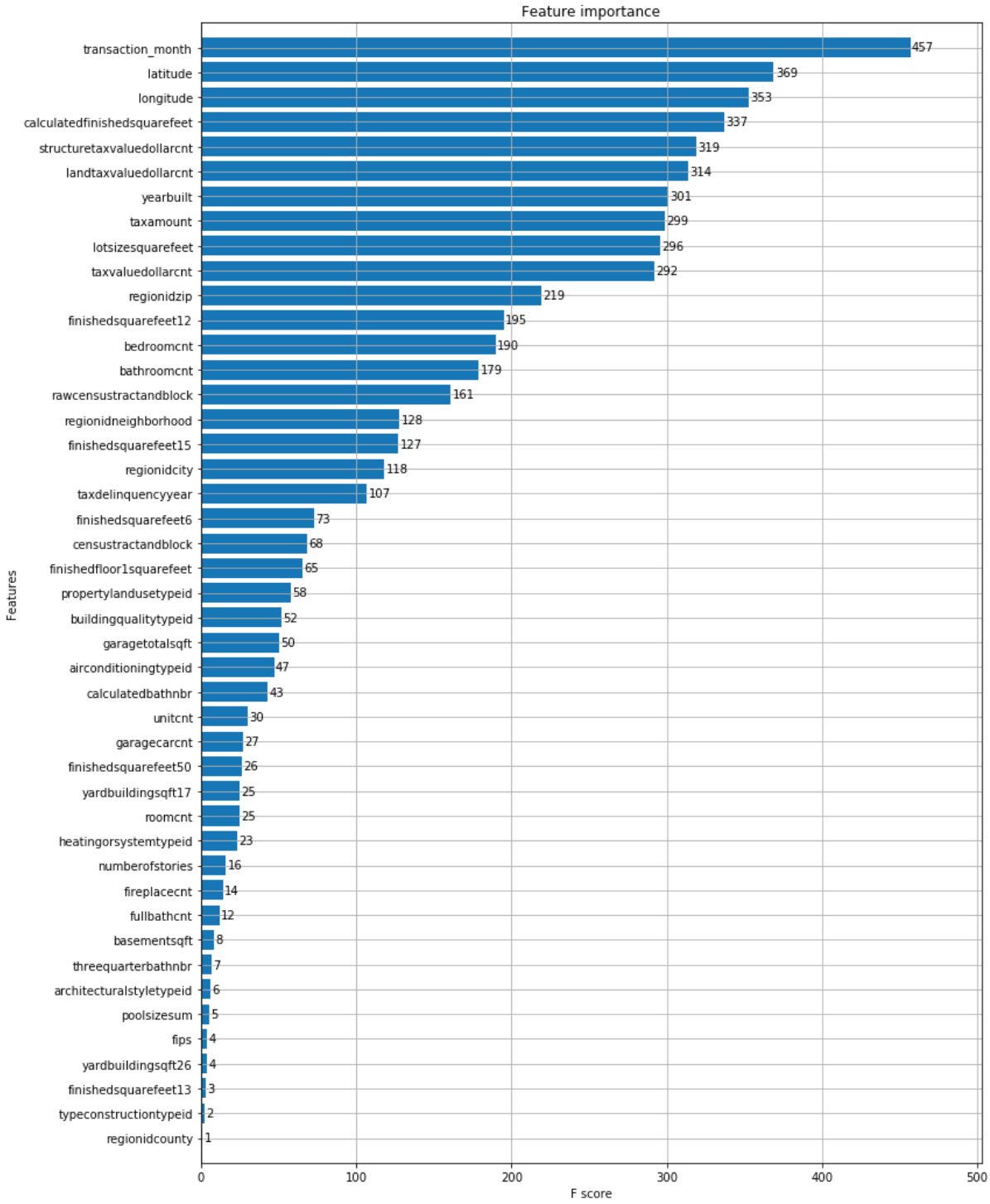

Figure 8 XGB feature importance

Feature Construction

As there were some variables discarded to high count of null values, new variables were constructed. The constructed features are:

Table 1 Engineered features

N-life	Amount of years since property was built to today
N-LivingAreaError	Error between real to theoretical area
N-LivingAreaProp	Ratio of constructed area
N-LivingAreaProp1	Ratio between permiter and area
N-ExtraSpace	Non constructed area
N-ExtraSpace-2	Difference between perimeter and area
N-TotalRooms	Bathrooms multiplied by bedrooms
N-AvRoomSize	The estimated size of each room in the house
N-ExtraRooms	Rooms that are bedrooms
N-ValueProp	Ratio of built home cost to land cost
N-GarPoolAC	Property has garage, pool or hottub and AC
N-location	Sum of coordinates
N-location-2	Multiply coordinates
N-location-2round	Round location-2
N-latitude-round	Round latitude to 4 decimals
N-longitude-round	Round longitude to 4 decimals

One important thing to notice is that most important features are used to produce new feature variables.

Implementation

A brief description of the algorithms used is presented next to provide a background on them. First is important to notice difficult aspects during the implementation, basic libraries such as sklearn or R packages tend to be insufficient to handle sparse and unprocessed data. This packages will require to be fed highly processed data to throw accurate values. Data was processed but sophisticated packages as xgboost, lightgbm and keras has important utility methods that takes preprocessed data and further analyzes for outliers, missing values and malformed data. Another tough part in the pipeline was data analysis, imputation was something that worried us a lot as literature recommends a careful analysis as it can have important effects on predicted data. Imputation was tested on several methods and variance was insignificant from each other. Training models was the easiest part as it depends on well prepared data. Exploratory data visualization was determinant in answering important questions that helped fix several issues with data as outliers and missing values.

OLS

Ordinary Least Squares or Linear Regression is a method that uses a polynomial to fit a curve in data. This line does not have to be perfect but be good enough to generalize on new data. It optimizes this line by using a loss function in this case mean squared error.

The following figure explains what a good line fit looks like and what does choosing a very simple model or complex models looks like.

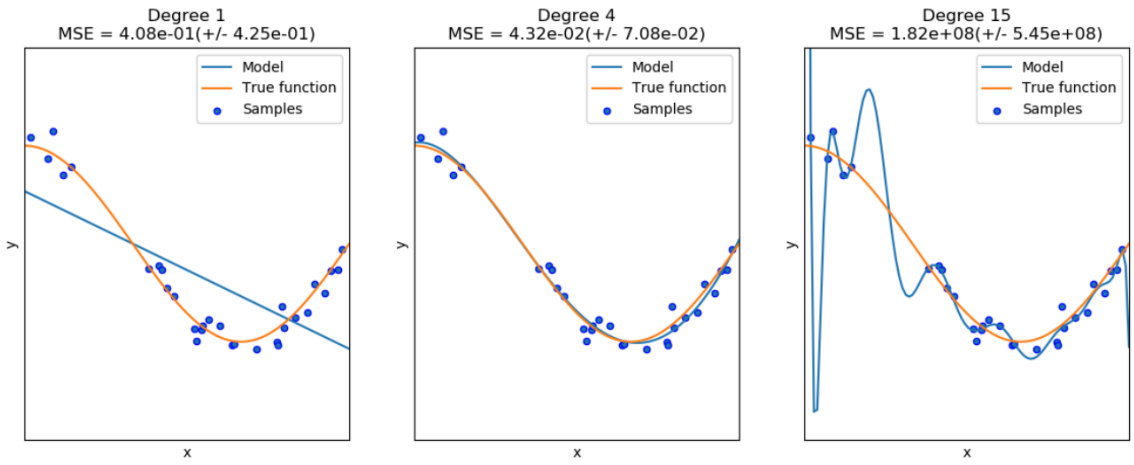

Figure 9 Overfitting vs Underfitting

The line in the middle accurately fits the data. Left is an example of underfitting which is a consequence of a simple models. Right curve is an example of trying to hard to fit data (overfitting) that as result poorly predicts on unseen data.

In this project, as data is complex OLS would have had a very poor performance. But when aggregating the output that comes out of other algorithms, in our case, gradient boosting and neural networks. The data is a summary of a very complex model into a very simple explanation. OLS is the model used over the weighted sum of the output of the mentioned models.

Gradient Boosting Trees

Boosting works by sequentially applying a classification algorithm to reweighted versions for the training data and then taking majority vote of the sequence of classifiers thus provided. (Jerome Friedman, 2000)

Gradient boosting is base on this same principle. We'll start over a simplification of the using decision trees to solve a regularization problem. Let suppose we want to fit a decision tree to a dataset which is defined as follows:

$$F_1(x) = y$$

where F_1 represents the model and x is matrix that contains the training features. Then based on additive models, in this case an additive regression model.

$$F(x) = \sum_{j=1}^{p} f_f(x_j)$$

Where f_j is a separate function for each of the p feature variables x_j, technically its said that f_j belongs to a small, prespecified subset of the feature variables. In this case, we can say that p decision trees are fitted to dataset x. Now, lets explain what are this functions f_j over which x is fitted. Various algorithms have been used for additive models and one of them that is illustrative of them is a backfitting iterative algorithm where the result of fitting the model is subtracted to the real value. So for example, for the first fitted model F_1 , we have the following:

$$h_1(x) = y - F_1(x)$$

where h_1 is called the residual. The next model will be trained on this residual value and then added back to the first trained model. This continues iteratively and we can formally express this update rule as follows:

$$f_j(x_j) \leftarrow y - \sum_{k \neq j} f_k(x_k)$$

Remembering that when we defined the additive regression model we started from j = 1 until p. So this sort of approach takes weak learners (algorithms that has the probability to have better performance than chance). This is what gradient descent boosting knows as partial residuals.

This is a simplified cost function where gradient descent is being applied.

This composes the theoretical framework of XGBoost and Lightgbm which uses optimization beyond the scope of this paper. One more important thing to mention is the whole picture with respect to lost function as it not single-handedly depends on the model but also handling bias. This is solved by adding regularization terms and this boosting libraries have a well thought way of handling this:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

$$\text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

In this equation taken from the original XGBoost paper (Tianqui Chen), here L represents the loss function and l is the model, the new part is Ω where is represent the regularization term. Regularization is a way of artificially prevent overfitting by discouraging complex predictions. On most reading, L1 and L2 methods are used (Nagpal, 2017) but this libraries uses the number of leaves (T) and weights that need to be learned to have an adaptative way of preventing overfitting on each iteration. Another important features, is the way it selects split candidates. On most academic introduction to decision trees entropy or gini measures are taught but the way XGBoost does is it depends on the the current

and previous trees. Also, they try to optimize resoures such as cache and core usage.

Neural Network

A neural network was also used to train the dataset. Neural networks simulate the learning process of human neurological system and it tries to model the activation process of neurons when transmitting data. On figure X, we can see a representation of an artificial neural network where it receives an input (in orange), in our case, the dataset. The neural net is composed of different layers. There are three types of common layers, input, output and hidden layer. Each circle represents a neuron which is interconnected to other neurons, what is called as fully connected. This neurons have an initial weight assigned (normally initialized from a random normal distribution centered with zero mean). The blue neurons belong to hidden layers. The output of each hidden layers takes each value from previous layers and process this values through activation functions. This activation functions are what allow neural networks to model more complex functions, not only linear.

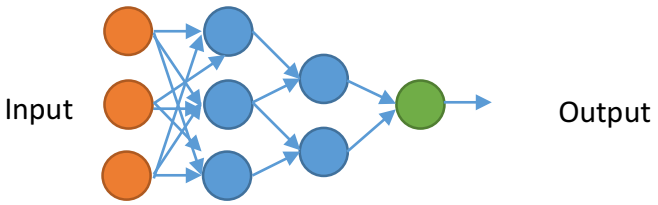

Figure 10 Neural Network Layers

In our case, we use rectified linear units (R). The algorithms that takes care of calculating the forward pass of values from the input to the output is called **feedforward propagation**. If we called X the input and W the initial weights of each neuron. The output of the first layer will be:

$$h_1 = R(Wx)$$

The output of this hidden layer then becomes on the output of the next hidden layers until the output is calculated. The following step, is to calculate the error or loss function which is the target to be optimized. Optimization has to deal with calculating derivatives and is what **backpropagation algorithm** is about. It calculates the derivative of the loss function (L) with respect to the weights that we want to optimize it values as it is the only independent parameter. This involves calculating the intermediate derivatives of this hidden layers, so it will look as follows:

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial R}\frac{\partial R}{\partial h_n}\frac{\partial h_n}{\partial W}$$

Where h_n represents the feedforward values of the n hidden layers. After the error or cost is calculated, then the weights have to be updated accordingly.

$$W \rightarrow W - c * \frac{\partial L}{\partial W}$$

where c represents a constant which is a ratio of the learning rate and the number of features. Notice the simplification on the operations, which are considered to be tensors.

In our case we used Keras, which is a framework that constructs a computational graph which enables the easy calculation of derivatives of any neural network configuration. Keras is built on top of a backend which can be Tensorflow or Theano.

Parameters and Architectures

Following the parameters optimized for decision tree algorithms are described:

Parameter	Description
Xgboost	
eta	learning rate, manages algorithm's convergence speed
max_depth	Maximum tree depth for base learners
subsample	Subsample ratio of the training instance
colsample_bytree	Subsample ratio of columns when constructing each tree.
objective	Learning task
eval_metric	Evaluation metric to assess algorithm's predictions.
Lightgbm	
max_bin	Max number of bins that feature values will be bucketed in.
learning_rate	learning rate, manages algorithm's convergence speed
metric	Metric for loss function
sub_feature	Subsample ratio of columns when constructing each tree.
num_leaves	Number of leaves in one tree.
bagging_fraction	Feature selection without resampling
min_hessian	Minimum child weight, deals with over-fitting.
min_data	Minimum number of data in a leave, deals with over-fitting
bagging_freq	Frequency for bagging, at every k iteration

The neural network architecture is composed as follows:

Dense Layer	400 x Input Features Length
PReLU	Activation: Rectified Linear Unit
Dropout: 40%	Dropout Layer to avoid over-fitting
Dense Layer	160 x 400
PReLU	Activation: Rectified Linear Unit
Normalization	Normalizes activation function weights for each batch.

Dropout: 60%	Dropout Layer to avoid over-fitting
Dense Layer	64 x 160
PReLU	Activation: Rectified Linear Unit
Normalization	Normalizes activation function weights for each batch.
Dropout: 50%	Dropout Layer to avoid over-fitting
Dense Layer	26 x 64
PReLU	Activation: Rectified Linear Unit
Normalization	Normalizes activation function weights for each batch.
Dropout: 60%	Dropout Layer to avoid over-fitting
Final Dense Layer	Which is the output of the neural network

Then it uses the Adam Optimizer, instead of the usual Stochastic Gradient Descent, over a MAE loss function.

All these trained models then are averaged to obtain the final solution explained next. Hyperparameter optimization was a hard stage because of the time it tooked to train. The average ran took 2 days in a 64GB RAM machine on a cloud service. This limited the amount of time we had to go over several ranges of values.

Meta Ensembling

This term refers to the combination of several models to obtain a single prediction. Primarily, this is based on the assumption that no model is perfect but also finds inspirations in boosting which takes several weak learners to obtain strong learner. In practice, specially in Kaggle competitions there have been several implementations of stacking or meta ensembling. In our case, we trained four models. The models are

- Lightgbm (Lightgbm Github)
- Xgboost ((Xgboost Github)
- Neural Network
- Ordinary Least Squares

The final solution consists in a weighted average of the output of all this models. The winning team final submission had a score of 0.0740861253. Our final submission had 0.0754311 which was a result of stacking models and hyper parameter optimization. We had over 70 submissions where we tweak weights and hyperparameters. When the winning solution was published it used a similar stacking method to combine several models and retraining the output of other models to have a more robust model which is something very similar as boosting methods do that they train more models based on previous outputs.

Weight optimization for stacking is a very complex mathematical problem. Weights are also hyperparameters and as such they can optimized using known strategies such as K-Fold. The data was separated into 3 folds. This was chosen because of training time. Then grid search was used to find the final values.

Weight	Initial	Final
Lightgbm	0.33	0.62
Xgboost	0.33	0.32
Neural Net	0.34	0.06

The hyperparameter training for stacking is extremely time consuming and other approaches can be used as quadratic optimization and using the lead submission as a target value for the output of the model.

Refinement

K fold cross validation was used to optimize hyperparameters in the case of gradient boosting (Jain, 2016). Sklearn provides GridSearchCV functions that implements k-fold over the training data over a grid of parameters. This process takes days on regular CPUs which made us to use cloud providers. Specifically, we used hetzner.com machine SX131 which has 64GB RAM, 8 cores and Xeon CPU. On a regular, macOS device with 16RAM it can take over 3 days to run just one grid search for a couple of parameters. A code example is left on the final notebook.

The values for the parameters tuned is presented in the following table. It contains information about initial values as well as the ranges over grid search was run over.

Parameter	Values	Ranges	Initial
Xgboost			
eta	0.02	0.001 – 0.1	0.1
max_depth	4	1 - 8	8
subsample	0.7	0.1 – 0.9	0.3
colsample_bytree	0.7	0.1 – 0.9	0.3
Lightgbm			
max_bin	10	5 - 12	6
learning_rate	0.0021	0.0001 – 0.01	0.001
sub_feature	0.5	0.1 - 1	0.8
mum_leaves	512	128 - 610	256
bagging_fraction	0.85	0.1 - 1	0.3
min_hessian	0.05	0.01 – 0.1	0.1
min_data	500	400 - 620	500
bagging_freq	40	30 - 50	40

To compare the initial values which were presented in the Benchmark and the ones on the Results. The Benchmark section has values for individual algorithms. The hyperparameter optimization was not used for those

submissions as it was implemented in the final stage when stacking was implemented.

Results

Model Evaluation and Validation

The parameters for each algorithm were discussed in the Implementation and Refinement section as well as the process for optimization. These optimization helped in the last phase of submissions. Earlier models during the competition were not passed through optimization as the CPU resources did not allow us to ran our scripts fast enough. When a robust model was implemented, we invested in cloud resources. The Benchmark section contains information regarding the scores for individual algorithms used. The range of hyperparameters started from default values to values used by other competitors and shared through kernels. Model improvement and hence our score boosted when choosing a meta ensemble of those models including a deep neural network.

The reason as why complex representation of models such a meta ensemble has a better performance than individual models is a hard question. One hint as why this works for some models is that better data representation can occur between layers of stacking, similar as neural nets. In our case, individual models can also be seen as weak learners and summing up several weak learners can form a strong learner.

The following scores are for the public and private scores of the competition which gives us an assessment of generalization of the algorithm.

Meta Ensemble		
Public Score	**Private Score**	**Difference**
0.0643694	0.0754324	~0.01

Initial benchmarks did not consider any stacking without hyperparameter optimization but comparing this to individual model submissions can let us the the superior performances that it offers without having to worry about overfitting.

Justification

The difference in the Mean Absolute Error varied by 1.5% with respect to the private score. These values help us understand the robustness of the model. The private score denotes the model trained with the data provided for the competition and the public score represents the value of unseen data trained with the private score model. The final submission was very satisfying as the variation of unseen data was very low as the we expected more than a 5% variation.

We can conclude that for parameters given and model chosen the final submission is superior to non-stacked models benchmarked.

Conclusion

Free-Form Visualization

Various models were discussed until this point. The final solution looks as follows:

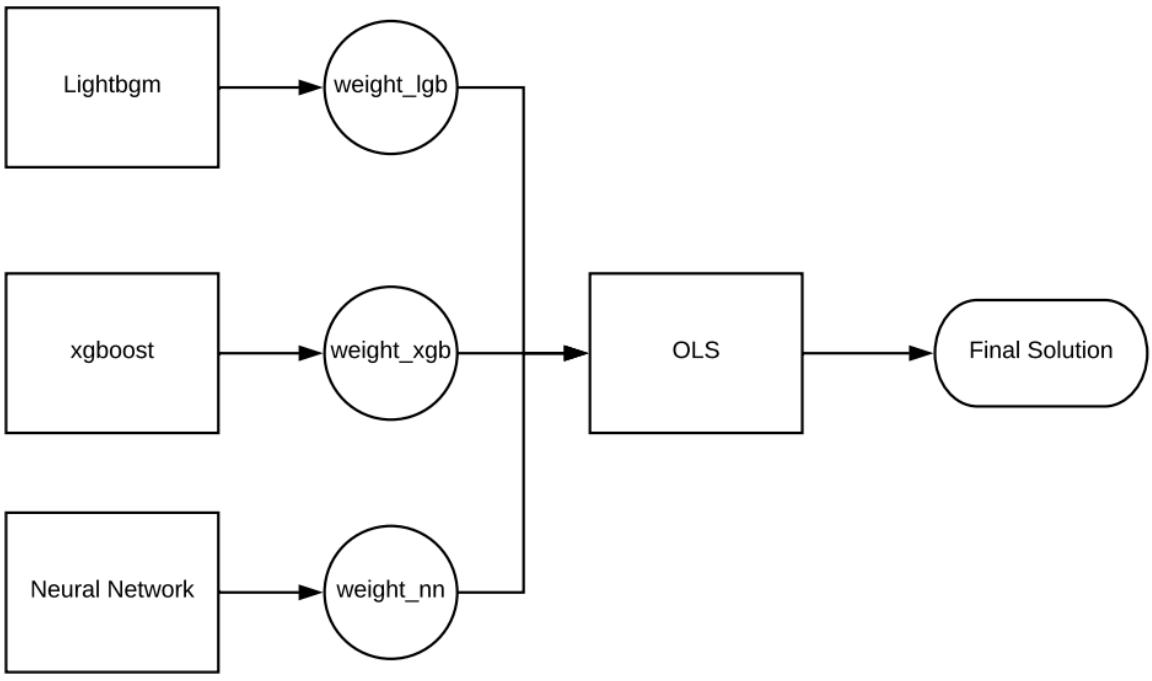

Figure 11 Final Solution diagram

It took over 50 submissions to find a local minimum for the point (weight_lgb, weight_xgb, weight_nn). Then the output of this models was trained by an ordinary least squares model, this is the second ensemble phase that gave us the final solution. Further ensemble methods can be applied but that will be left for improvement.

Reflection

Participating in a very diverse community as Kaggle. I had the opportunity to discover practical approaches towards solving machine learning problems. Also, collaboration was a key factor to have finish the competition with a satisfying result for being the first time participating. The kernels that other competitors

share was a baseline form learning new ways to answer questions about data and learning the practical explanation for many algorithms used. Being exposed to boosting methods has been a great exposure to professional machine learning solutions.

The challenges that the project had as any machine learning project were the data analysis phase that consists of data cleansing, data exploration and exploration visualization because there were many missing values and choosing the right strategy.

The whole pipeline can be seen in the following figure.

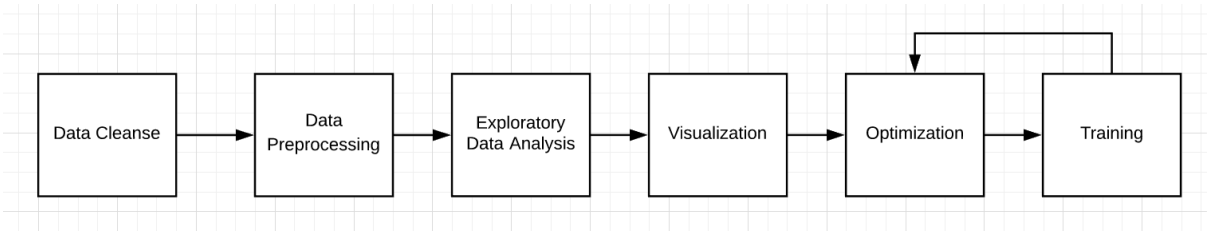

Figure 12 Data Pipeline

Data wrangling and cleansing is the most important part in the pipeline. Everything can go wrong if data is not diligently inspected and cleansed. Data preprocessing also treats data to be fed adequately to machine learning algorithms. In this stage, outliers are striped off and prepared to be analyzed and obtained accurate statistics that can provided explanations for questions. Visualization can also make summaries of important assumptions for features. Feature importance graphs were very important to prune features and use only important ones. This also benefitted feature engineering to create new features. Then training stage starts as well as optimization, this is an iterative process where tweaking takes place in each submission.

Improvement

The lightgbm solution was very similar to the final solution of the ensemble, so it will be very important to tune lightgbm's weight in order to score better on the private score. Also another improvement will be to train the deep learning neural net with different weight initiation strategies and different optimization algorithms.

Lightbgbm		
Public Score	**Private Score**	**Difference**
0.0648173	0.07575	~0.01

The stacking model can also be more complex, different layers can be added and trained in samples. Figure 11 shows only one layer but the output of base models can also be the input of similar models with different parameters optimized for

that specific layer. Kaggle has a very nice article that depicts the solution used for the Netflix competition which used a multi-layered stacking solution.

Bibliography

Byeonghwa Park, J. K. (2015). *Expert Systems with Applications.* Elsevier.

Guang-Bin Huang, Q.-Y. Z. (2006). *Extreme Learning Machine: Theory and Applications.* Singapore: Nanyang Technological University.

Jain, A. (2016). *Xgboost Hyperparameter Optimization Article*. Retrieved from Analytics Vidhdya: https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

Jerome Friedman, T. H. (2000). Additive logistic regression: a statistical view of boosting. *Annals of Statistics* .

Lightgbm Github. (n.d.). Retrieved from https://github.com/Microsoft/LightGBM

Nagpal, A. (2017, October 13). *Towards Data Science*. Retrieved from L1 and L2 Regularization Methods: https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c

Tianqui Chen, C. G. *XGBoost: A Scalable Tree Boosting System.* University of Washington. University of Washington.

Xgboost Github. (n.d.). Retrieved from https://github.com/dmlc/xgboost

Zillow. (2018). *Kaggle*. Retrieved from Zillow's Home Value Prediction: https://www.kaggle.com/c/zillow-prize-1