Udacity Machine Learning Nanodegree: Sberbank Housing Price Prediction

Joe Dinius
Capstone Project

May 25, 2017

1 Introduction

Project Overview

The first project for this course was predicting housing prices in Boston based on a simple feature set. This project will build on some of the intuition that was built while completing that project.

The website Kaggle is currently hosting a competition that requires implementation of regression techniques like those used in the Boston housing prediction project. Sberbank, the Russian bank, along with Kaggle, is hosting a competition to predict Russian housing prices based on a dataset rich in features. According to the project introduction page, the Russian housing market is relatively stable when compared to the more volatile Russian economy as a whole. Having an accurate predictor would help to assure both investors and prospective property owners that they are making wise choices in how they invest in Russian real estate despite other potentially worrying economic indicators.

Problem Statement

Kaggle has provided a full dataset broken into training and test datasets. This dataset provides features from which a regression model will be used to predict housing prices based on the values of these features. A standard regression technique such as the one used for the Boston housing prediction project will most likely not perform well on the larger, more nuanced feature set that will be used for this project. A more flexible learner is needed. The package **XGBoost** is a fully integrated, flexible framework for performing gradient boosting regression using multiple simple learners, such as decision trees. The combination of simple learners into a more complex single learner has been very successful in both supervised machine learning problems. The reason **XGBoost** in particular is chosen as the starting point is its success in previous machine learning competitions.

The solution to this project has two parts: (1) tuning a regressor to minimize the prediction error of housing prices on validation data extracted from the training set provided and (2) predicting prices on the Kaggle-provided test set using regressors developed during the tuning process. The prediction data on the test set will be submitted to Kaggle for internal computation of a public score. The public score is used to rank solutions and will be used as a sanity check on the algorithm's performance. The desired public score is as close to 0 as possible; lower scores are better than higher ones.

Metrics

The primary metric here, as in most regression problems, is error on a test set with known output. The error metric used for optimization is the negative of the mean squared error

$$S_{\text{residual}} \equiv \text{mean squared error} = \sum_i \left(y_{\text{actual},i} - y_{\text{predicted},i}\right)^2,$$

where y_* represents the target variable of the regression. This provides a strong estimate of how well matched predictions are to actual observations. Since the error is averaged over all elements of a dataset (training or validation), this provides a quantitative assessment of model quality. The negative is chosen on account of the **sklearn** protocol where the negative is used in a maximization scheme; that is, minimizing error is equivalent to maximizing the negative of that same error.

Another metric will be used as well. In cross-validation efforts, the model's predictive power will be evaluated using the R^2 score, also known as the coefficient of determination. The mathematical formula for the R^2 score is

$$\begin{aligned} S_{\text{total}} &= \sum_i \left(y_{\text{actual},i} - \bar{y}\right)^2 \\ R^2 &\equiv 1 - \frac{S_{\text{residual}}}{S_{\text{total}}}, \end{aligned}$$

where

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_{\text{actual},i}$$

is the mean of the observed test target variable data. A score of 1 means that the model perfectly predicts the output for each given input, while a score of -1 means that the model is completely unable to predict output based on the given input. The goal is an R^2 score that is positive and as close to 1 as possible.

2 Analysis

As a starting point, this was used for reference.

Data Exploration

The dataset provided is captured in three csv files: train.csv, test.csv, and macro.csv. The columnar data of the csv files are described in data_dictionary.txt, please see this file for description of the individual features. The columns in train.csv and test.csv represent data about the properties themselves, along with timestamps; data such as square footage (size), number of rooms, and build year. Extrinsic properties, such as proximity to entertainment and cultural attractions, are also included in these csv files. The column **price_doc** denotes the sale prices of the properties, and is the target variable for the regression analysis. Data within macro.csv, as the name would indicate, presents broad macroeconomic indicators along with timestamps. These data include gross domestic product, currency exchange rates, and mortgage rates.

An initial sense of how the sales price, **price_doc**, is distributed is a good place to start the analysis. This can be accomplished with a simple histogram (see Figure 1):

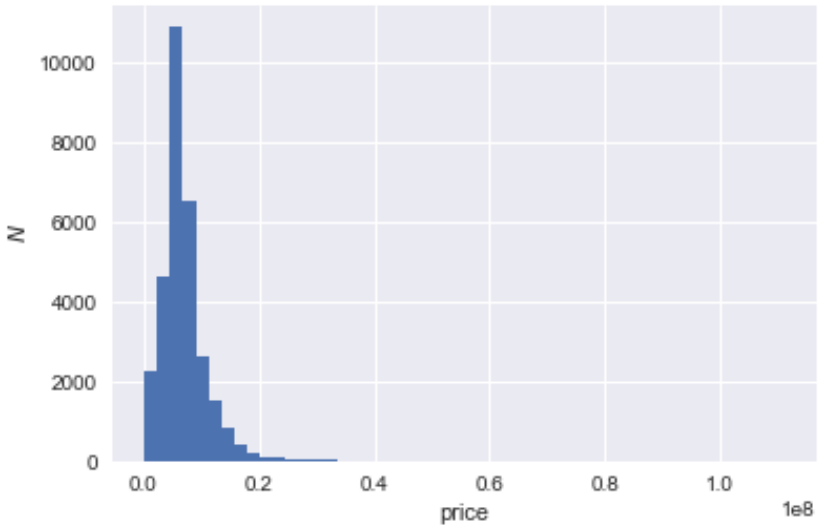

Figure 1: Histogram of sales price, p

The data appears to be positive-definite and may well be lognormal distributed. In regression analyses, when some variables appear to be lognormal, taking the log can reduce variation caused by outliers in the original distribution. The histogram is replotted after adding 1 (to avoid divergence of the logarithm function) and then taking the natural logarithm (see Figure 2):

The resulting histogram above seems to indicate a mostly normal distribution; there is apparent symmetry about a central value between 15 and 16. Going forward, the transformed price variable will be used as the new target variable for the regression analysis. This should enable the model to be more robust to variation caused by extrema in the price data.

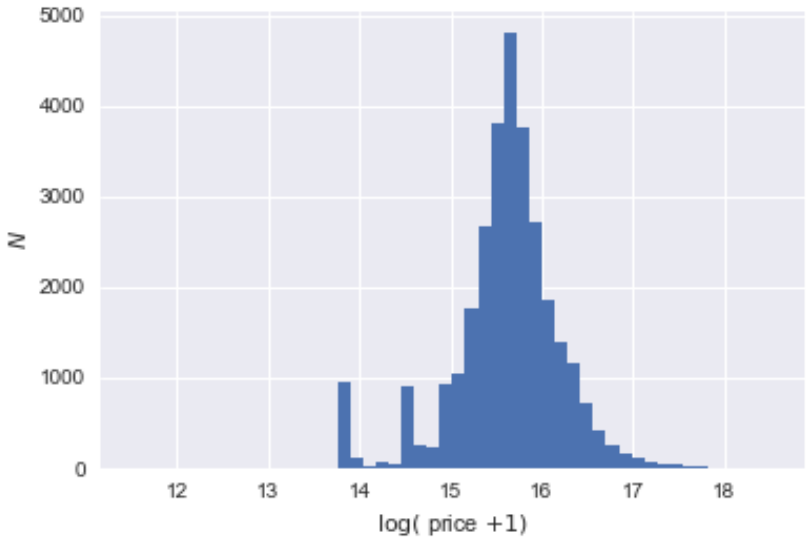

Figure 2: Histogram of logged sales price, $\log(1 + p)$

Next, explore the top-level statistics of the intrinsic properties of the real estate:

Table 1: Top-level statistics of intrinsic properties

Property (metric)	mean	median	standard deviation
full_sq	53.70	50.41	20.10
life_sq	32.66	30.40	22.74
kitch_sq	6.94	7.00	25.58
num_rooms	1.88	2.00	0.84

Table 1 variables show close connection between the mean and median values. What is surprising is the large variation in kitchen size. In fact, there is larger variance in this feature than in the full property size!

One might like to get a better idea of how some of these data are distributed. One such data trend that would of interest would be price versus the year the property was built (see Figure 3).

From the plot Figure 3, it is clear that a majority of properties sold have been built more recently (**build_year** $>$ 2013). However, it is interesting to note the large amount of construction in the late 1950's and early 1960's, and the subsequent dip in housing production culminating in a minimum in 1991, near when the Soviet Union collapsed. This is purely speculation at this point, and doesn't really hold much weight in the analysis that follows. However, it is not inconceivable that such a trend would positively correlate with macroeconomic indicators, such as GDP. Data related to that reported in Figure 3 would be the mean sale price versus the build year. Figure 4 shows this data, along with a third order regression with the 95% confidence interval.

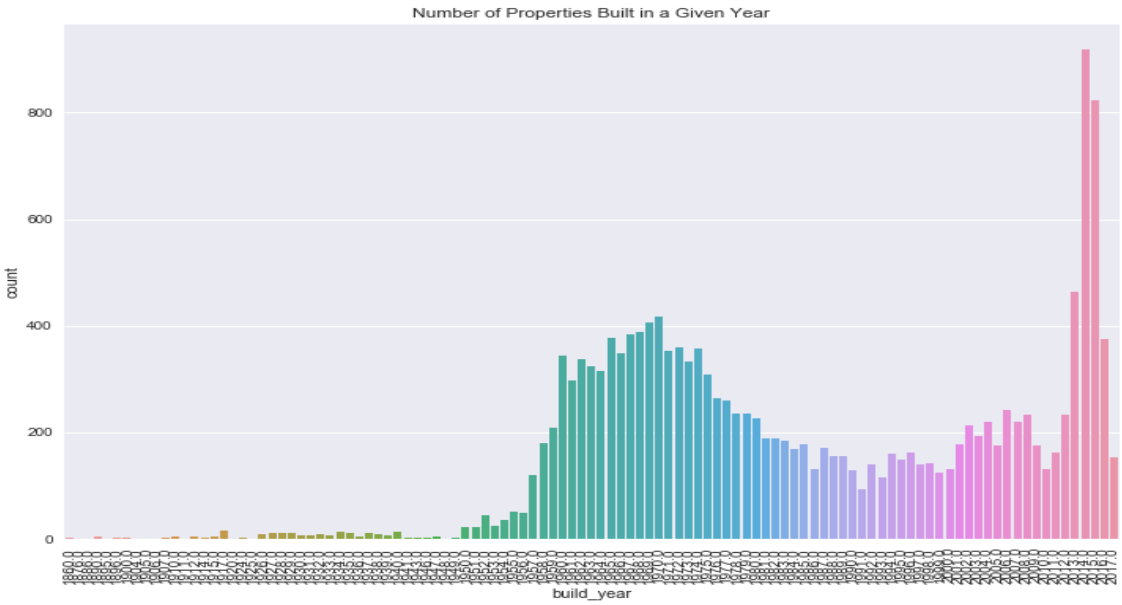

Figure 3: Frequency of properties built for a given year

Figure 4: Mean price, plus confidence interval, of price for a given year

Figure 4 demonstrates a more easily predictable trend for properties built more recently (**build_year** $>\approx 1945$). The inherent accuracy of such predictions can be seen from the tight variance bounds around the simple regression plotted alongside the raw data. The reduction in variability could be a result of more uniformity in housing projects during the Communist years, or for some other reasons. Again, this is merely an observation and doesn't impact the analysis that follows.

Another nice plot would be something that would show price as a function of the condition

of the property. A box-plot would work, and the **state** variable encodes the conditions of the properties.

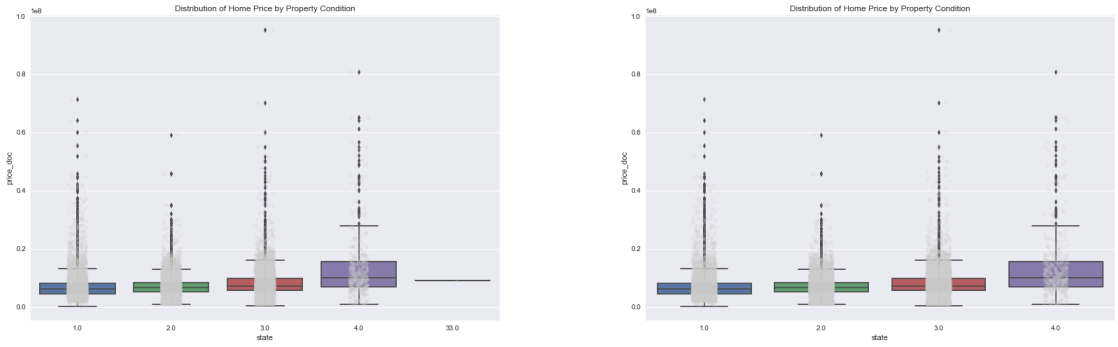

Figure 5: Box plots of condition versus price. Left: erroneous state 33, Right: corrected

Figure 5 demonstrates a few interesting points. The first is a data quality issue; namely, what does **state** = 33 mean? After looking into the dataset itself and at forums on the Kaggle competition page, this seems to be an invalid state. Given the frequency of occurrence and the easy-to-do data entry error of entering multiple of the same number in the same data cell, it seems the entry was made in error and the appropriate state in these instances should be **state** = 3. The original dataset can now be updated in light of this new information and the data can be replotted (see right side of Figure 5).

The trends observed tend to make sense. Price increases with increasing condition, or quality, of the property. Also, properties with the lowest and highest conditions occur less frequently than those properties in just so-so condition. Properties in the highest condition tend to have higher variability in price. This could be as a result of high quality properties having variability in size, number of rooms, and other intrinsic properties. There may be other trends as well; namely, extrinsic properties (e.g. proximity to shopping, grocery stores, schools, etc...) may have an impact on price.

Before moving on with the data exploration, another irregularity is present in the dataset. This was highlighted in the Kaggle forums for the competition and is pretty simple to fix. Some of the **build_year** data has been incorrectly entered and just needs to be updated with a valid date. Through inspection of the raw data, it's clear that much data is missing (indicated by **NA** in the cells). The most frequently missing data, along with the frequency missing, is plotted in Figure 6.

(It should be mentioned here that "raion" indicates an area around the property, kind of like a region would be in English-speaking nations.)

For some of the data, it is not particularly surprising that values are missing. Something like the number of hospital beds in a region may be a difficult number to come by in a real estate search. However, it is worth noting that a number of common property features are missing with surprisingly large frequency. Factors such as **state**, the condition of the property should definitely be recorded as part of any real estate transaction. If it were simply

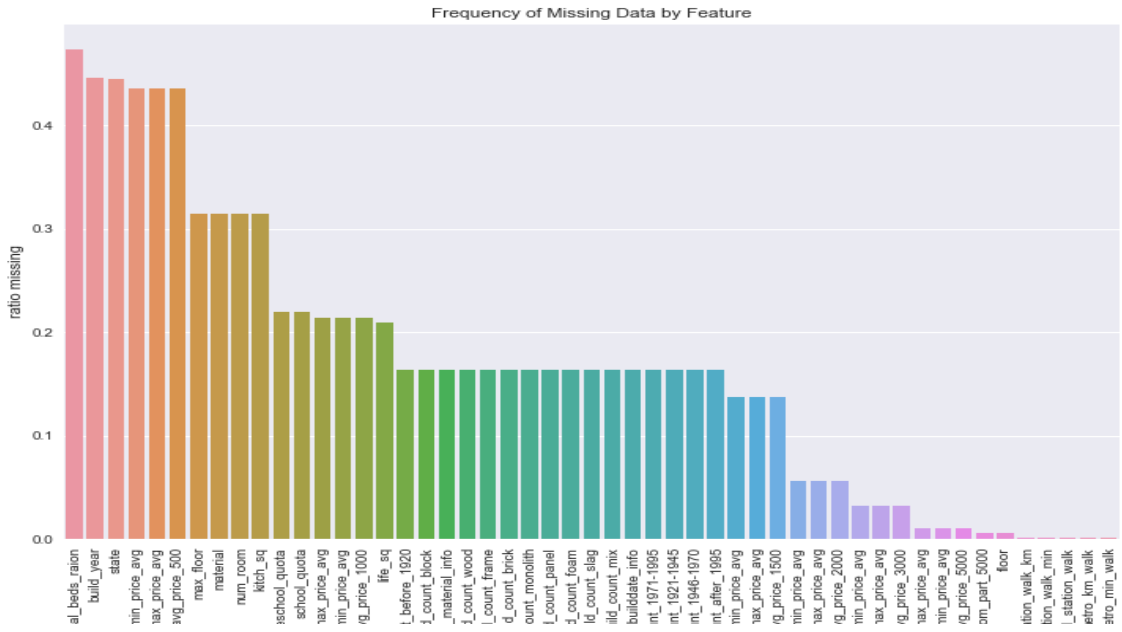

Figure 6: Frequency of missing data

missing because there is no property state (i.e. the property is a vacant lot), then this should be encoded as a separate class within the data.

Exploratory Visualization

A good first thing to consider is the importance of individual features. Arguably, the best starting point is how price relates to intrinsic features of properties. Heatmaps show how correlated one variable is to others and are great indicators of interrelatedness within data. The heatmap in Figure 7 shows interrelatedness between intrinsic data of properties.

The biggest correlations to price are seen in **full_sq** and **num_room**. This makes sense; increasing square footage and number of rooms should increase the sale price. Also, not surprisingly, the state and floor of the property (in the case of apartments) are also positively correlated with sale price.

In the heatmap presented in Figure 7, trends between intrinsic properties were made clear. These data showed a somewhat predictable trend: more square footage tended to mean a higher sales price. Less clear, however, is the correlation between extrinsic properties and price. Again, a heatmap will be illuminating here. Look at some of the more broad features, features pertaining to local population.

The variables shown in Figure 8 are mostly concerned with aspects of the regions themselves. These data include region population, along with a breakdown on working age (**work_***) and youth (**young_***) populations. Not surprisingly, there is moderate correlation

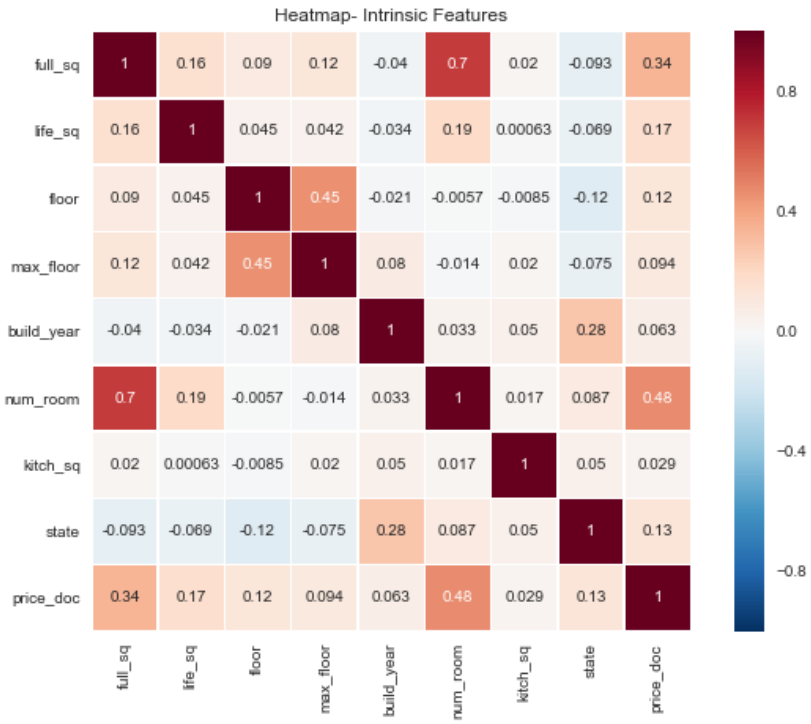

Figure 7: Intrinsic features heatmap

between the region name and the area of that region. This isn't, strictly-speaking, important to the analysis; it's just an observation. Also, the population characteristics are highly correlated to one another; that is, the working age male population of an area (**work_male**) is highly correlated with the working age female population (**work_female**) of that same area.

There doesn't appear to be a strong correlation to any of these variables with the sales price. Maybe other extrinsic features will show more correlation. There are so many of these features, and they can be broken down into categories: school, entertainment, and infrastructure.

Consider school characteristics first (see Figure 9):

The population of school-age children shows a moderate correlation with sales price, as expected. The degree of correlation is less than expected given naive assumptions about proximity to schools being a prime factor in real estate in the United States. Of note, as well, is the negative correlation of sales prices with distance to universities. It makes sense that, the farther one is from a university, the lower the sales price. This trend exists in the US as well, since real estate close to universities is very desirable, and fetches a larger sales price typically.

Next, consider entertainment features (see Figure 10):

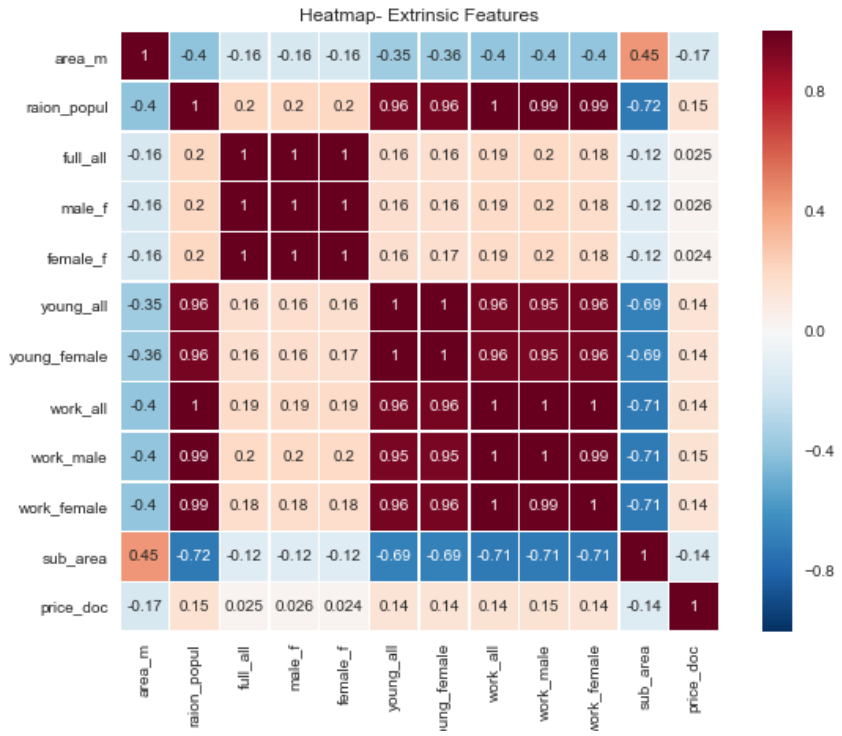

Figure 8: Extrinsic features heatmap

The data above again makes sense in the aggregate: the farther a property is from recreational attractions, like shopping centers or museums, the lower the sales price. Put another way, real estate near cultural and recreational attractions is more desirable and is typically more valuable.

Finally, infrastructure features are considered (see Figure 11):

The data above is surprising. Naively, one would assume that being closer to power generation stations would be undesirable and would show positive correlation with price. However, the data indicates otherwise. Other data, like correlation of price with distance to public transportation, makes sense in the context provided.

Next, consider correlation between price and macroeconomic indicators (see Figure 12):

This data is pretty interesting. As expected, intrinsic and extrinsic properties of the real estate are much more correlated to sales price than broader macroeconomic features. This is exactly as one would expect. It is surprising, though, that mortgage properties aren't more positively correlated with price. It would seem that lower mortgage rates might entice more buyers, and with more buyers, one might expect higher prices. However, the data shows almost no correlation to price. It is also interesting that prices aren't more negatively correlated with unemployment. Again, naively, it would seem that higher unemployment would mean lower prices on real estate transactions.

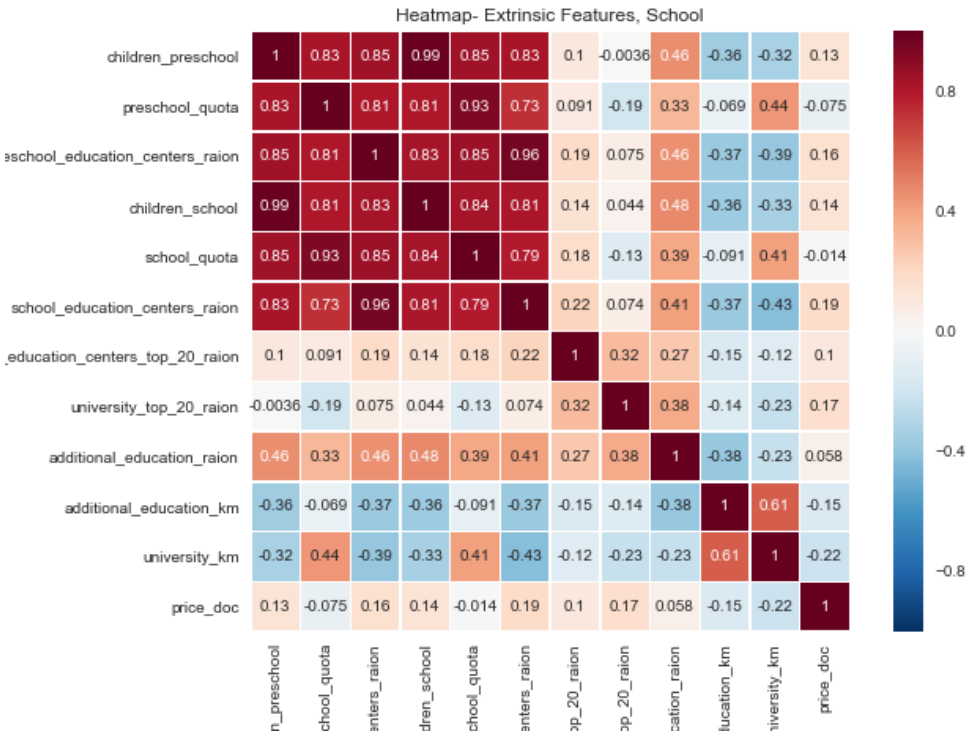

Figure 9: School feature heatmap

Algorithms and Techniques

As mentioned previously, **XGBoost**, along with the **sklearn** wrapper **XGBRegressor**, will be used for performing the regression analysis.

XGBoost uses tree ensembles: sets of classification and regression trees (CART). A CART is different from a decision tree in that decision trees provide only decision values, whereas CARTs provide real scores for each leaf. The tree ensemble then uses the sum of prediction of a target variable for multiple trees. **XGBoost** utilizes additive training (also known as boosting), where each new tree is added with full knowledge of what has already been learned. Put another way, the updated model at each training round incorporates the model learned from the previous state. To this previous model is added a new tree which decreases training loss and minimizes complexity of model; the complexity is controlled via regularization parameters that are hyperparameters of the model. See this resource for more details.

This package was chosen because of its success in machine learning competitions of various scales. The use of boosted trees will allow for a model that requires minimal feature engineering (see this); all that needs to be done is hyperparameter tuning. The package **sklearn** has two very nice optimization frameworks for estimator tuning: **RandomizedSearchCV** and **GridSearchCV**. **RandomizedSearchCV** allows for sampling the parameter space by

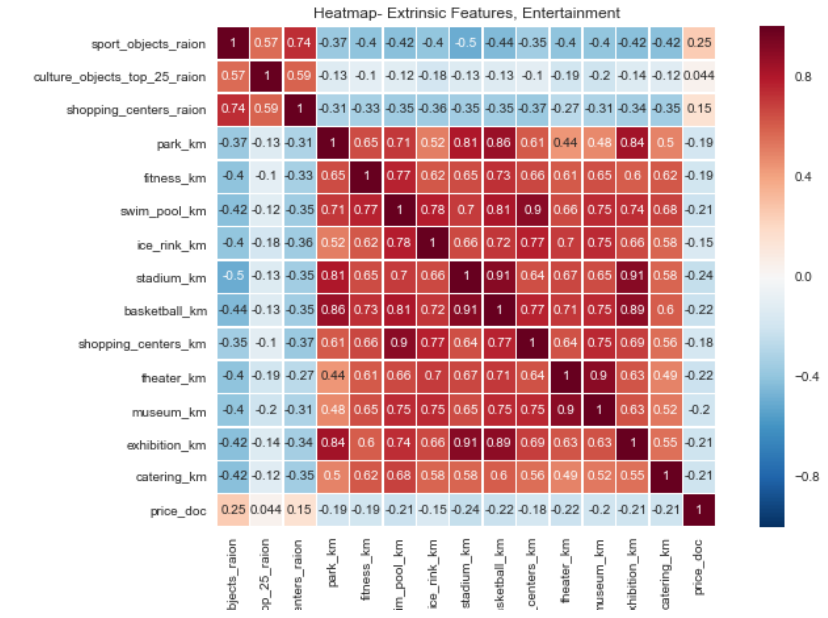

Figure 10: Entertainment feature heatmap

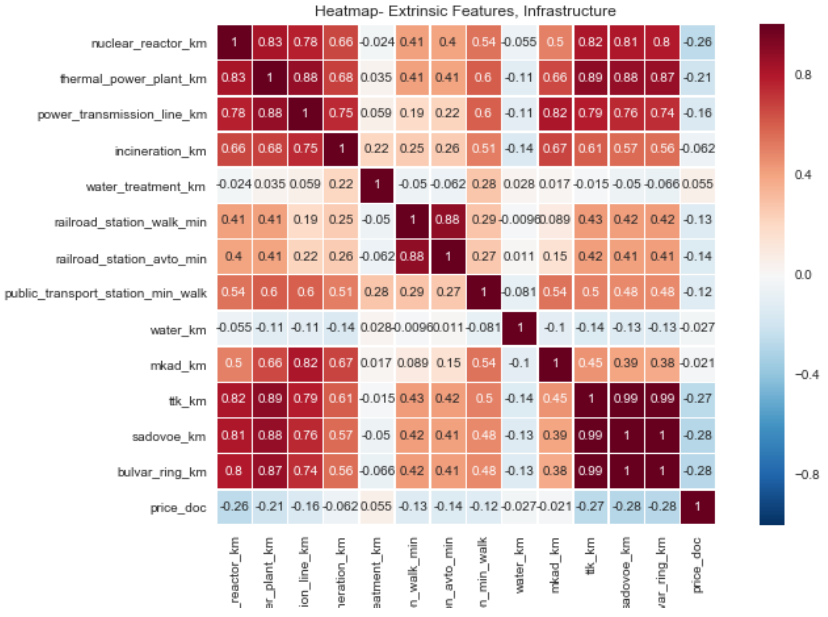

Figure 11: Infrastructure feature heatmap

random draws from probability distributions. The choice of distribution depends upon the parameter being tuned, however the uniform distribution is a great choice; this is because it allows for sampling of a spatial structure where each point has equal probability. The equal

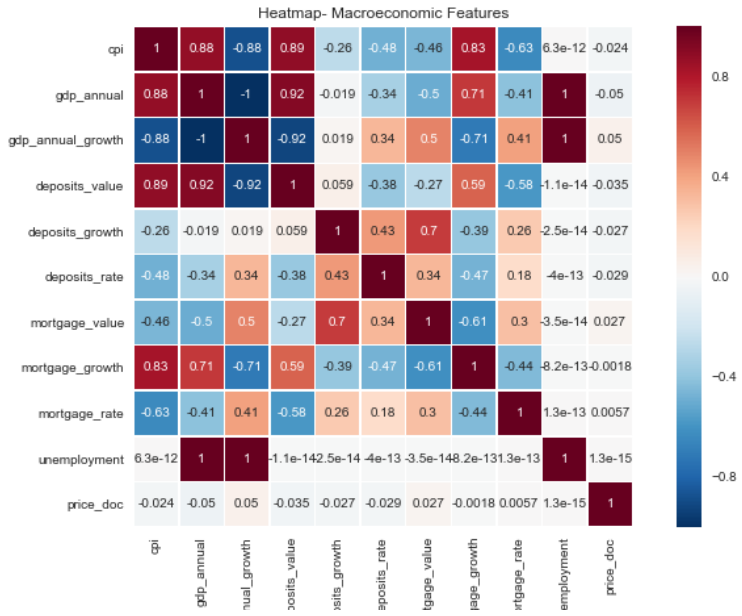

Figure 12: Macroeconomic feature heatmap

probability is desirable since, a priori, there are minimal assumptions made about model performance as a function of the hyperparameters.

As with all regression techniques, a model is developed here which minimizes error of a predicted target variable versus a known target variable from a training set comprised of feature vectors. How well the model performs is a function of several things; chief among them being choice of hyperparameters and structure of the input feature data. Both of these things will be discussed subsequently in the Methodology section.

Benchmark

No real established regression benchmark exists for this dataset. However, a simple process can be used to create a benchmark that can be built on through hyperparameter tuning. Random forest regressors provide a simple out-of-the box regressor that can be improved upon with more advanced regression techniques (e.g. **XGBoost**).

Random forest regression generalizes the idea of bootstrap aggregation to decision trees. Random subsets of the training data are sampled with replacement for each learning round and the predictions of each training round are averaged to prevent overfitting. Like **XGBoost**, random forest regression utilizes sets of CART. See this resource for more details.

Random forests provide comparable performance to boosting methods (see this) without having to perform much tuning to optimize. The **sklearn.ensemble** package has **RandomForestRegressor**. The regressor achieves an R^2 score of 0.39009 when trained on 70%

of the training data. Part of this project is to see how well different models perform with respect to .

The public leaderboard score for this submission was 0.41168, which is significantly worse than the current best score of 0.30728. But, as a first shot with no tuning, this isn't terrible. Another preliminary plot can be made now that this estimator has been created: feature importance can be estimated using the attribute **estimator.feature_importances_** (see Figure 13).

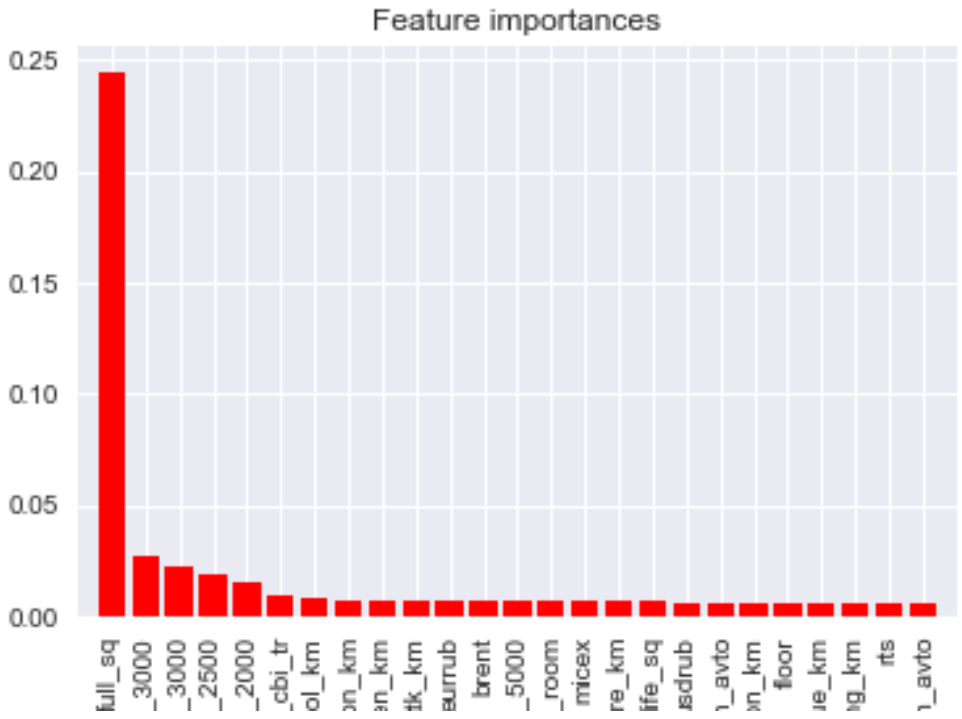

Figure 13: Feature importance

This makes sense, since one would expect that the most important features, with regard to sales price, would be the size of the property and recreational/entertainment features nearby (cafes, sports, etc...).

3 Methodology

As a starting point, this was used as a reference.

Data Preprocessing and Algorithm Implementation

There are a few different features that are not modeled as of yet, but could be important.

Such features include the month, day-of-week, and the year of the transaction. It would seem reasonable that some months would be better for purchasing property than others; real estate may be less desirable in winter months, for example. Also, the year of the transaction may be significant if there are significant economic or social trends that were important that year (e.g. the fall of the Soviet Union in the early-1990's).

The ratio of room space to total space may also be interesting. For instance, someone might not want a tiny kitchen in an otherwise large home. Also, apartments on the higher floors will most likely fetch higher prices than those on lower floors. Therefore, one should look at the ratio of the floor to the maximum number of floors in an apartment building. A small "fudge-factor" is added to the division in the **rel_floor** value computation to avoid divide-by-zero issues.

It should be clear from the heatmaps that the data is very collinear. This would seem to indicate that there might be a gain in model performance if the feature set was reduced to a combination of linearly independent pseudofeatures. This could be accomplished via principal components analysis. First, a baseline needs to be established to see if PCA might help. The R^2 score for the **XGBoost** baseline is 0.40971, with a public score of 0.39276. Both of these scores are an improvement over those obtained using the benchmark. In order to consider using PCA, there would need to be an improvement in R^2 score, the public score can be checked as an additional sanity measure.

Table 2: R^2 score vs. number of principal components- no whitening

# PC's	R^2
19	0.153
57	0.180
127	0.375
182	0.374
202	0.375
205	0.374
221	0.372
261	0.376
333	0.375
369	0.374

PCA didn't improve model performance (see Table 2). In fact, performance was degraded. It's clear that there seems to be nearly indistinguishable scores once the number of principal components exceeds some threshold (about 127 in this case). There is another PCA option that can be utilized to try that might improve performance: whitening. Whitening the data is a way to further remove correlation from feature vectors resulting from a principal components transformation.

Table 3: R^2 score vs. number of principal components- whitening

# PC's	R^2
19	0.157
57	0.182
127	0.375
182	0.373
202	0.373
205	0.373
221	0.372
261	0.374
333	0.375
369	0.374

The results are indistinguishable from the non-whitening results (see Table 3). Therefore it is concluded that it is not worth using principal components; stick with the dataset as-is and focus on tuning the hyperparameters for **XGBoost**.

Refinement

Following this procedure, the tuning process is as follows:

Table 4: Tuning Process

Step #	Process
1	Tune **max_depth** and **min_child_weight**
2	Tune **gamma**
3	Tune **subsample** and **colsample_bytree**
4	Tune regularization parameters
5	Tune **learning_rate**
6	Tune **n_estimators**

The functions **RandomizedSearchCV** and **GridSearchCV** functions from **sklearn.model_selection** are used for the optimization. The resource referenced also provides the range of values to use for the tuning. See this for more details on hyperparameter descriptions.

Step 1: Tune max_depth and min_child_weight

The parameters **max_depth** and **min_child_weight** are the number of layers of decision trees used and the minimum sum of instance weight in a tree's child, respectively. The public score is 0.34795, which is a significant improvement over the previous score of 0.39276. The R^2 score is 0.41821.

Step 2: Tune gamma

The hyperparameter **gamma** is the minimum loss reduction required to create a further partition. This can also be thought of as information gain required from a certain split in order to make that split.

The public score for this step is 0.34814. The best R^2 score is 0.41756. Both the public and R^2 scores are not significantly different than those from the previous step. Both are still an improvement over the random forest baseline.

Step 3: Tune subsample and colsample_bytree

The hyperparameters **subsample** and **colsample_bytree** are the subsample ratio of training data and the subsample ratio of columns when constructing a new tree, respectively. The first parameter is to prevent overfitting to the training data.

The public score for this step is 0.34924. The best R^2 score is 0.41667. Both the public and R^2 scores are not significantly different than those from the previous step. Both are still an improvement over the random forest baseline.

Step 4: Tune regularization parameters

The hyperparameters **reg_alpha** and **reg_lambda** are the L1 and L2 regularization terms, respectively. The function **GridSearchCV** is used due to the scale desired on the L1 term **reg_alpha**. It would be difficult to randomly sample and hit the desired scales to test for this hyperparameter.

The public score for this step is 0.35830. The R^2 score is 0.41689. The public score degradation is a more significant difference than those observed in previous steps, and therefore this step is not implemented in the final solution.

Step 5: Tune learning_rate

One potential issue with **XGBoost**, and boosting algorithms in general, is that the model can learn too quickly and overfit to the training data. Therefore, it is desirable to slow the learners and prevent this overfitting. The hyperparameter **learning_rate** (though it is called **eta** in the **XGBoost** package) represents this learning rate.

The public score is 0.33300 and the R^2 score is 0.42211. This is an improvement in both public and R^2 scores! In fact, this is better than any step so far, which is the ultimate goal.

Step 6: Tune n_estimators

The hyperparameter **n_estimators** is the number of trees to fit. The **GridSearchCV**

function is used here due to the scale desired to test.

The optimal number of estimators appears to be 500, as we've been using all along. Since the results of this step are equivalent to those of Step 5, the public score is not reported.

4 Results

Model Evaluation and Validation

The process outlined above for arriving at a final model is reasonable; model hyperparameters were chosen at each step that improve R^2 and public leaderboard scores over a benchmark model. However, a single validation set was used in the tuning. Now, it is necessary to test for generalization to unseen data; that is, using multiple training and validation sets and comparing model performance across these multiple sets. This can be done using cross-validation (here is another useful link). The package **sklearn.model_selection** has two utilities that will be needed to do this analysis: **learning_curve** and **ShuffleSplit**. The function **learning_curve** computes R^2 scores for training and validation scores across a cross-validation generator. The function **ShuffleSplit** creates cross-validation datasets based upon the desired number of instances, train-validation split ratio, and a random seed. The training and cross-validation scores for the regressor tuned in the last section are shown in Figure 14.

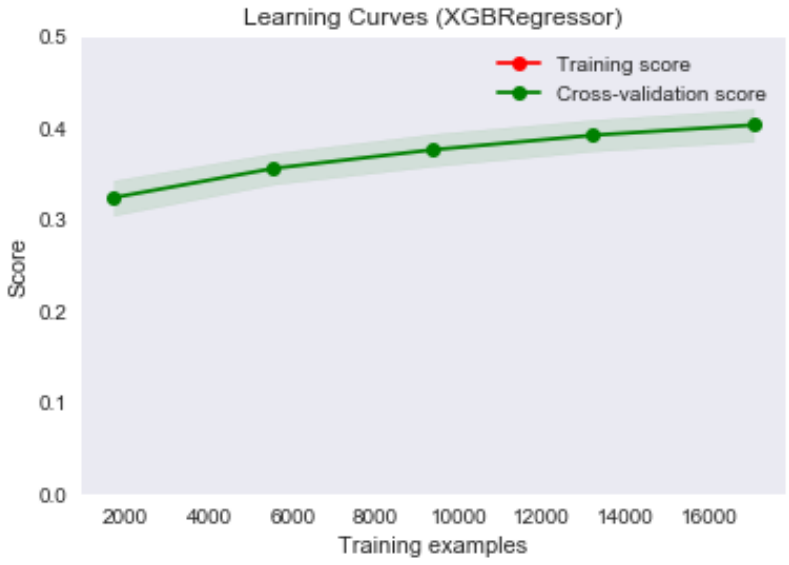

Figure 14: Cross validation- tuned XGBoost

The two mean scores lie directly on top of one another, indicating that the performance on the training and validation sets matches very closely, nearly perfectly, in fact. Any error would be well-within the 1-sigma uncertainty bounds, indicated by the green solid area

bisected by the mean scores. It can be concluded that the model generalizes very well to unseen data. The data appear to be approaching an asymptote, however this asymptote appears to be less than 0.5 even, indicating that housing price is not fully predictable given the feature set used. The plot also indicates that performance could be improved with more training examples.

Justification

To justify the use of this model, a simple comparison to the benchmark can be made. A good place to start would be to repeat the above cross-validation on the benchmark model.

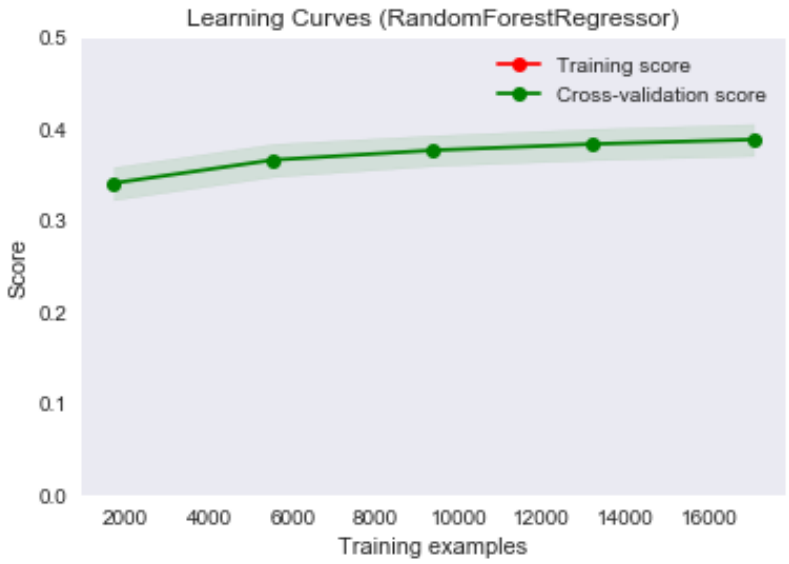

Figure 15: Cross validation- Random Forest

Figure 15 shows subtle differences from the cross-validation plot for the tuned regressor. First, there are some similarities. Most notably, this regressor would perform better with more training results. The asymptotic value that the data appears to be approaching is lower than that of the tuned regressor plot; the slope of the mean score line is flatter in this plot. It appears as though maximum value of the R^2 score for this regressor will occur for a smaller training sample size than the maximum obtained for the tuned regressor. As in the tuned regressor case, the model generalizes well to unseen data.

The final solution is significant enough to solve the problem at hand. The model gives reasonable performance given the complexity of the data and the minimal addition of features extracted from the raw data. Practically, there is only so much training data that is available. If more data were available, the model performance would be a little better, but nowhere near 1, as indicated by the cross-validation plot. This solution is favorable over more complex solutions (such as stacked regressors) since it is easy to implement, easy to interpret, and provides adequate performance.

5 Conclusion

Free-Form Visualization

There is one table that hasn't been shown yet, but summarizes the analysis quite well.

Table 5: Performance summary

Regressor	R^2 score	Improvement	Kaggle Public Score	Improvement
Random Forest	0.39009	N/A	0.41668	N/A
XGBoost Baseline	0.40971	+0.050	0.39276	-0.046
XGBoost Step 1	0.41821	+0.069	0.34795	-0.155
XGBoost Step 2	0.41756	+0.071	0.34814	-0.154
XGBoost Step 3	0.41667	+0.068	0.34924	-0.152
XGBoost Step 4	0.41689	+0.069	0.35830	-0.130
XGBoost Step 5	0.42211	+0.082	0.33300	-0.191
XGBoost Step 6	0.42211	+0.082	0.33300	-0.191

Table 5 shows improvement of each step over the benchmark. The **Improvement** columns show the results of the relative improvement over the benchmark, computed as

$$\text{Improvement} = \frac{s_{\text{step}} - s_{\text{benchmark}}}{s_{\text{benchmark}}},$$

where s_* denotes a particular score, either R^2 or the Kaggle public score. These columns can be interpreted, after multiplying by 100, as the percent relative change in the scores from the benchmark compared to each step. The sought after trends are a positive change in R^2 score and a negative change in the Kaggle public score. The table shows that, through tuning, an 8% positive change in R^2 score, along with a large 19% relative change in public score, were achieved.

Reflection

A model was developed for predicting the sales price of properties within the Russian housing market based upon features provided by Sberbank, a Russian bank. Plots were presented, and analyzed, showing trends in both micro- and macroeconomic features, and the relationship of these features to sales price. Minimal feature engineering was performed on the dataset to provide a few additional, potentially important features. A benchmark model based on an untuned random forest regressor was presented as an initial first cut at a solution. Based upon previous analyses and their results, **XGBoost** was selected, and tuned, as the final solution for this problem. **XGBoost** provided a nice framework to build upon since it requires minimal feature engineering and scaling to achieve best results.

Quite surprising was the fact that PCA didn't really improve performance, despite the high degree of multicollinearity observed in the data. Perhaps naively, one would think that

creating a new pseudofeature set that orthogonally spans the feature space would allow for greater predictive power of the underlying model. However, the results seem to confirm the previous notion that decision trees are insensitive to principal component reduction.

This project presented some difficulties in interpreting the data provided. There were many features, and many of these features seemed very interrelated. There was also much data missing. Fortunately, there are good standalone packages within **sklearn** for dealing with such cases. In the end, it was great that there exists a simple-to-use framework, **XGBoost**, that made working with the data very easy.

The final model fits expectations, and the process followed for tuning was very straightforward. A similar process could be used for solving problems of a similar type: data analysis, data reduction/scaling, feature engineering, and regressor development are the common steps for any regression analysis. Depending upon the data involved, it may not be necessary to use as strong a learner as **XGBoost**; a simple linear regressor may be adequate if the degree of nonlinearity in the target variable is low. Given the data observed in the cross-validation plots and the amount of training data available, it seems that this model performs about as well as can be expected for a single regressor approach.

Improvement

It would seem that better performance might be achieved if multiple learners were combined. Some algorithms may achieve better performance for some regions in the feature space while others might perform better elsewhere. The package **mlxtend** provides a **Python** framework for doing stacked regression. Stacked regression provides a way of combining multiple regressors into a single predictor, and has been shown to work well in other Kaggle competitions. There is a trade-off here, though. Typically, the added complexity comes with relatively modest improvements. This added complexity may obscure any interpretation of the model; functional relationships between features and target variables may be more difficult to establish despite any observed increases in predictive power (i.e. R^2 score).