Machine Learning Engineer Nanodegree

Capstone Project

David Havera
May 4, 2018

I. Definition

(approx. 1-2 pages)

Project Overview

In September of 2017, I competed in the Zillow's Home Value Prediction competition that was hosted by Kaggle. This was my first competition and I was very proud to score within the top 70th percentile. My approach was very simple. First, I removed categorical variables. Second, I completed the following data preparation steps via a pipeline:

- Selector - Converted DataFrame to a NumPy array
- Imputer – Replaced missing values with median
- Standard Scaler - Not needed for decision trees, but needed for other algorithms
- SelectKBest - Selects the most important features
- PCA – Feature reduction into correlated components.
- Random Forest Regressor - Used to make predictions

I chose the Random Forest Regressor because of the following benefits.

- Limited parameter tuning
- Fast and versatile
- Great for feature selection because algorithm evaluates a lot of variations

Next, I ran a grid search to optimize the pipeline parameters for KBest and PCA. Finally, I used a random search to tune the Random Forest Regressor because I discovered that it can be more effective than a grid search [1].

For the capstone project, I will update the benchmark model to evaluate its performance against several other regression algorithms. Furthermore, I will create a

template that I can use to approach supervised learning regression problems in the future. The template will have the following steps:

- Exploratory data analysis: I will include visualizations such as heat maps
- Data Cleaning (scaling, imputing, etc): I will include a pipeline for the numeric features
- Feature Engineering: I will evaluate the performance with and without feature engineering
- Kfold cross validation: I will evaluate models using cross validation
- Untuned/Tuned algorithm comparisons with box plots: Box plots will allow me to review the results of the kfold cross validation scores variability and average
- Algorithm tuning using grid and random search: I will tune all the algorithms to evaluate the kfold cross validation scores
- Learning curve analysis for top performing algorithms: Learning curves will be used to evaluate a model's bias and variance
- Stacking of top performing algorithms: Stacking can improve model performance by combining top performing models

This template is critical to me professionally because I have just been promoted to an Analytics Engineer at GE Aviation. I am responsible for using machine learning algorithms to deliver cost-out projects at GE Aviation. My cost-out target for 2018 is $150 million. To achieve this target, I analyze the manufacturing cost for engine parts to determine if there are ways to make the parts cheaper at our factories. The research for aircraft engine parts is difficult because GE Aviation has more than 40 pillar IT systems across 65 factories. The task of making better engines for a cheaper cost requires a lot of data munging to succeed. However, there is a huge opportunity in this area because the information has never been consolidated and reviewed by machine learning algorithms.

I manage an offshore team of data scientists, so it's critical that we are all on the same page as we progress toward our common goal. I need a standard machine learning template that I can implement as a starting point as we are assigned new productivity projects each quarter. The template that I create during my capstone will be operationalized with my overseas team and used to investigate areas of savings. The template will also ensure that we are evaluating the machine learning algorithms globally with the same practices such as using learning curves.

Problem Statement

"Zillow Prize, a competition with a one million dollar grand prize, is challenging the data science community to help push the accuracy of the Zestimate even further.

In this competition, you are going to predict the logerror between their Zestimate and the actual sale price, given all the features of a home for the months in Fall 2017. The log error is defined as

$$logerror = log(Zestimate) - log(SalePrice)$$

and it is recorded in the transactions training data. Submissions are evaluated on **Mean Absolute Error** between the predicted log error and the actual log error." [2]

Metrics

This competition is based on Mean Absolute Error (MAE) of the forecasted versus actual sale price. The final model will have the lowest MAE and will be scored as a "Late Submission" on Kaggle's website. Additionally, final model will have the lowest MAE after kfold cross validation and will illustrate a learning curve with low variance and bias.

II. Analysis

(approx. 2-4 pages)

Data Exploration

There are 60 columns in the training dataset broken out into the following data types: float64(53), object(5), datetime64[ns](1), and int64(1).

The data types are misleading because there are categorical variables in the float64 datatype. More specifically, latitude, longitude, regionidcity, regionidcounty, regionidneighborhood, and regionidzip are all categorical variables describing location that are of data type float 64. The features for location are all high cardinality items with more than 15 unique values. I chose regionidcity because it had the least amount of unique values at 179.

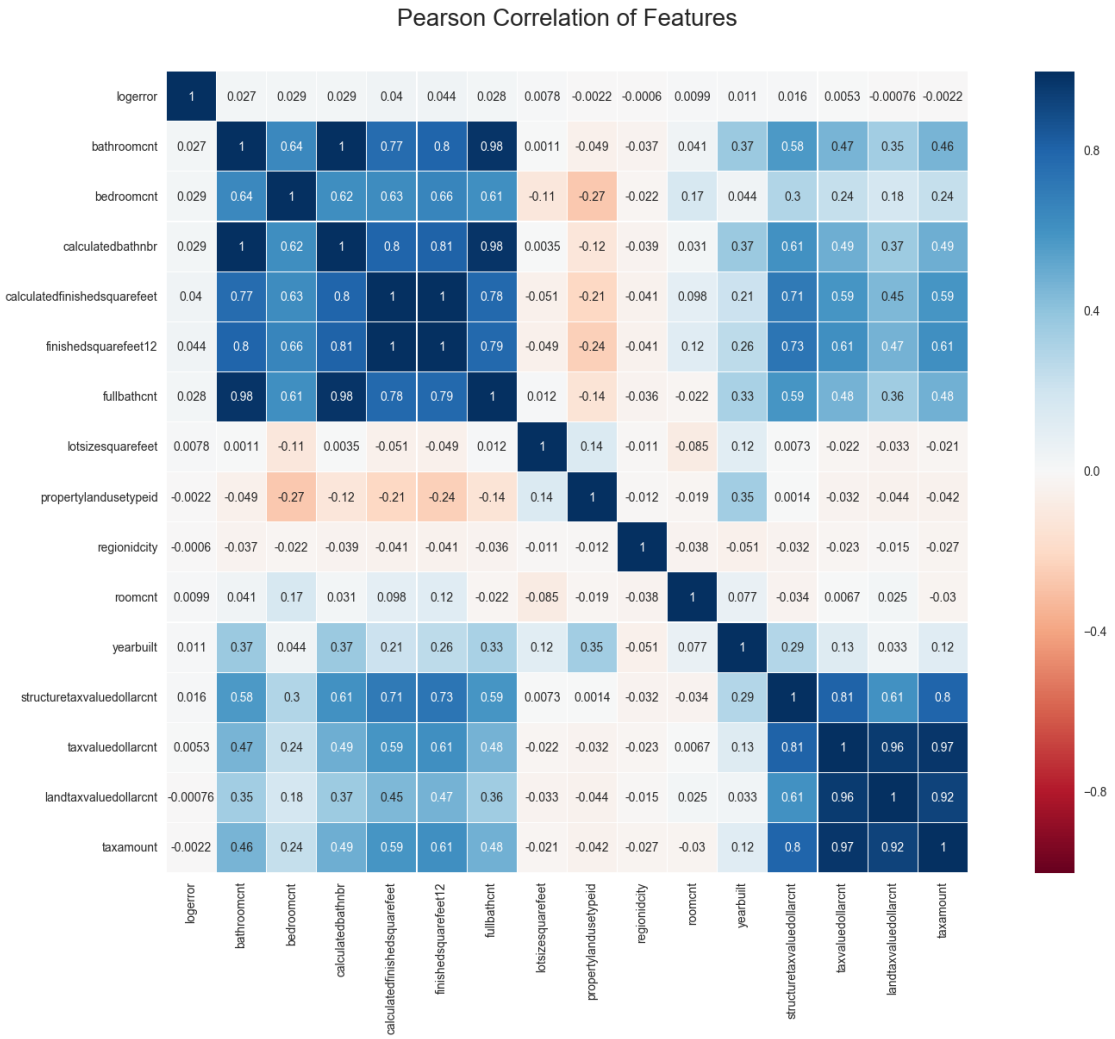
The training data also contains a high level of NANs. I have set the threshold to remove any column with over 15% NANs and 33 columns were removed. Additionally, 'latitude' and 'longitude' have over 126,000 unique values, so I decided to remove these features.

Also, 'propertycountylandusecode' is of datatype object and was removed due to high cardinality of 90 unique values. Furthermore, I removed ‘assessmentyear’ because properties were either assessed in 2015 or 2016 and properties would be reassessed at the time of purchase. Next, I removed ‘rawcensustractandblock’ and ‘censustractandblock’ due to high cardinality of over 57,000 unique values. Finally, I removed ‘parcelid’ and ‘fips’ as these are just a reference number for Zillow. I also removed ‘regionidcounty' and ‘regionidzip’ because I chose to use ‘regionidcity’ as the feature for location.

Exploratory Visualization

I used a heatmap to review relationships of features. This helped me to further divide the data from 16 columns into 7 categories.

1. Target Variable: ‘logerror’
2. Room Count: ‘bathroomcnt’, ‘bedroomcnt’, ‘calculatedbathnbr’, ‘fullbathcnt’, & ‘roomcnt’
3. Square Footage: ‘calculatedfinishedsquarefeet’, ‘finishedsquarefeet12’, & ‘lotsizesquarefeet’,
4. Taxes: ‘taxamount’, ‘landtaxvaluedollarcnt’, ‘taxvaluedollarcnt’, ‘structuretaxvaluedollarcnt’
5. Location: ‘regionidcity’
6. Property Type: ‘propertylandusetypeid’
7. Age: ‘yearbuilt’



Algorithms and Techniques

I evaluated a variety of algorithms to provide a depth of analysis. The algorithms that I chose are as follows:

LASSO regression: I chose this regularized linear model to constrain weights and limit overfitting. LASSO uses L1, norm of the weight vector, for regularization. LASSO also helps with feature selection by eliminating weights of the least important features [3].

KNeighborsRegressor: I chose this algorithm to evaluate if neighboring data points could be used to predict the target variable.

BENCHMARK MODEL (Random Forest Regressor): This was the algorithm that I used in the previous Zillow competition and the parameters were random_state = 7 and n_estimators = 10.

Gradient Boosting Regressor: This algorithm combines weak learners (performance slightly better than random chance) into a single predictive algorithm. Gradient Boosting has 3 elements:

1. Loss Function: A loss function is chosen and optimized [4].
2. Weak Learner: Regression trees are used output values for splits and the output is added together. This allows subsequent outputs to be added and the residual to be corrected. Trees are constructed in a greedy manner and choose the best split points based on purity scores such as Gini [4].
3. Additive Model: Trees are added one at a time, and existing trees in the model are not changed. A gradient descent procedure is used to minimize the loss when adding trees [4].

I chose default of 10 n_estimators to be similar to the benchmark model. I also chose a max_depth of 5 to limit the complexity of the untuned gradient boosting regressor [4].

Benchmark

I will include the benchmark algorithm in all my upcoming analysis and comparisons. I have created new features and cleaned the data differently during my capstone project, but keeping the baseline algorithm parameters the same, I can evaluate a benchmark against new algorithms.

III. Methodology

(approx. 3-5 pages)

Data Preprocessing

Imputing

I filled NANs for columns that I would use for feature engineering. More specifically, I filled NANs in the 'regioncityid' and 'yearbuilt' columns with median. I did this because I felt the mean could be a number that doesn't represent a categorical data point. Also, I used mean for imputing true numeric columns such as 'calculatedfinishedsquarefeet' and 'lotsizesquarefeet'.

Outliers

I used the tukey method to identify outliers for 'logerror'. My definition of an outlier is +-1.5 *IQR to identify and remove extreme outliers [3].

Categorical Variables

Next, I evaluated categorical variables to determine the best approach to include them in machine learning models. The 16 features illustrated in the heatmap are all float64. However, 'yearbuilt', 'regionidcity', 'propertylandusetypeid' are categorical variables and have 139, 179, and 15 unique values respectively. One hot encoding is the most widespread approach for dealing with categorical variables but it only works well for fields with up to 15 different values [5].

I decided to use binning instead of one-hot encoding for 'yearbuilt' and 'regionidcity'. I converted 'propertylandusetypeid' to string and one-hot encoded it.

I used a supervised ratio for 'regionidcity' to prepare the 179 unique values for machine learning. The supervised ratio transformed each city into a percentage of positive instances of that city [3]. I further added random noise with laplace distribution to avoid potential leakage of the training data. Finally, I binned the supervised ratio with the laplace noise by quantile which created 4 columns that represented the regionidcity supervised ratio [4]. I explored feature hashing for 'regionidcity' but with the loss of the original feature information, I would not be able to understand the drivers within this feature. Finally, I one-hot encoded the four bins created for 'regionidcity'.

Next, I binned the 'yearbuilt' into quantiles using qcut. This created a categorical object variable for 'yearbuilt' quantiles, which I one-hot encoded into 4 columns representing the quantiles bins for 'yearbuilt.'

After one-hot encoded these three categorical variables: 'propertylandusetypeid', 'quantile_bin_region', and 'yearbuilt_quantile'. I evaluated model performance with and without the categorical variables. A random forest model scored (0.081022) mean absolute error without the categorical variables. This score improved slightly to (0.080652), so I kept the one-hot encoded categorical variables in the data set.

Feature Creation

I created a feature 'total_square_footage' by adding 'calculatedfinishedsquarefeet and 'lotsizesquarefeet'. This feature will help reduce the number of columns passed to the machine learning algorithms, which will speed up the training time and can improve the model fit.

I also added polynomials to evaluate if there was a relationship between the numeric features. Adding polynomials decreased the score to (0. 081547), so I excluded polynomials from the final dataframe.

Testing/Training Sets

I used train_test_split to divide my data into testing and training sets for algorithm testing. After that I created a pipeline to impute and scale the data to ensure the algorithms results would not be skewed.

Feature Selection

I created a function called train_predict() to assist during feature selection by comparing algorithm performance across several feature selection methods.

I evaluated the following feature selection methods:

- SelectKBest (SKB)
- Feature Importance (Random Forest)
- Select from Model (LASSO)

The best performance was SelectFromModel of the Random Forest.

Dimensionality Reduction

First, I used PCA to determine if dimensionality reduction could be applied to the RF data set. I chose a PCA threshold of 95% because I wanted to minimize information loss. PCA decreased accuracy from (0.080652) to (0.084723). The problem with PCA is that it makes a parametric assumption about the shape of the data. Therefore, it relies on a predetermined equation and will always output the same features each time it is run [9].

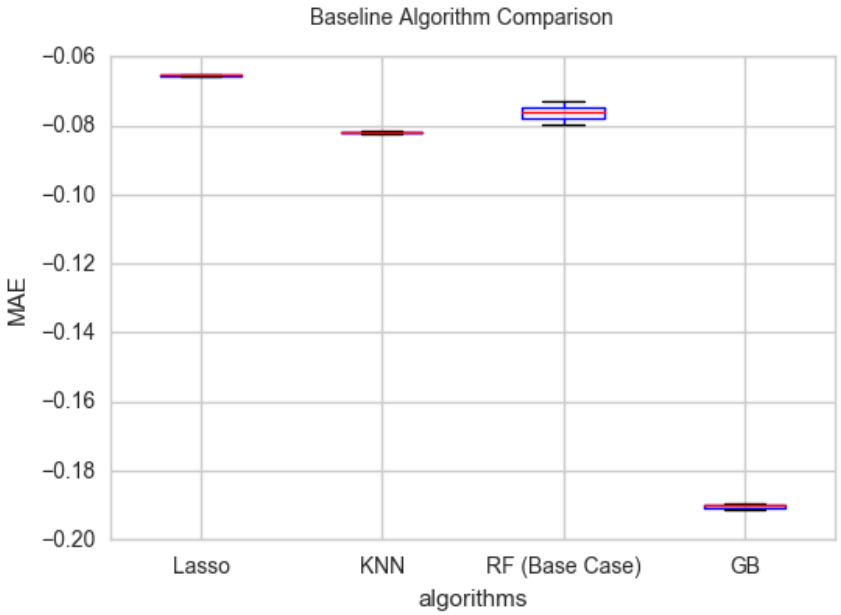
Since PCA resulted in a decrease in MAE, I tried using feature learning algorithms to reduce dimensionality. These algorithms don't make any assumptions about the shape of data and rely on stochastic learning. Restricted Boltzmann Machines (RBM) are an algorithm in the feature learning category and can be implemented in scikit-learn. More specifically, the algorithms use epochs to look at data points, extract features, and converge on a solution [9].

I used gridsearch on RBM and discovered that 2 components resulted in a significant improvement of MAE from (0.080652) to (0.07709). Therefore, I used RBM as the dimensionality reduction method for the RF data set.

Implementation

I created a pipeline to evaluate the best parameters in the models. This pipeline evaluates the models with cross validation and mean absolute error scoring. For the cross validation I used the following parameters: n_splits=3, shuffle=True, and random_state=7. Finally, I displayed the results of these pipelines on a box plot to evaluate the model scores across quantiles.

The graph illustrates that Lasso had the strongest performance and that KNN, RF, and Gradient Boosting can all benefit from tuning.



Refinement

I used grid/random search to tune various parameters for each algorithm.

KNN_clf_tuned = KNeighborsRegressor(n_neighbors = 9)

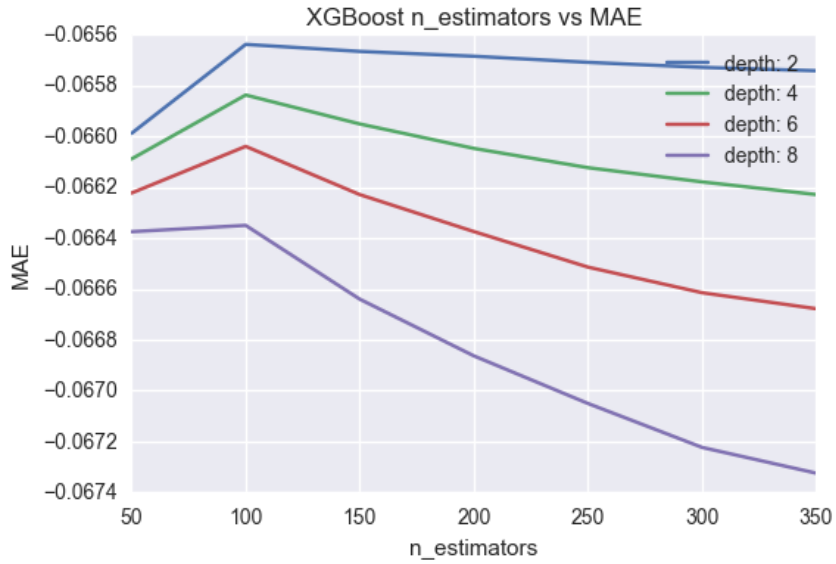
RF_clf_tuned = RandomForestRegressor(n_estimators= 189, max_depth = 3, min_samples_split = 2)

GB_clf_tuned = XGBRegressor(n_estimators = 100, subsample= 1, learning_rate = 0.1, max_depth= 2)

However, Lasso has limited ability to tune and did not show an improvement with changes to Alpha.

XB Boost Tuning (Trees and Depth)

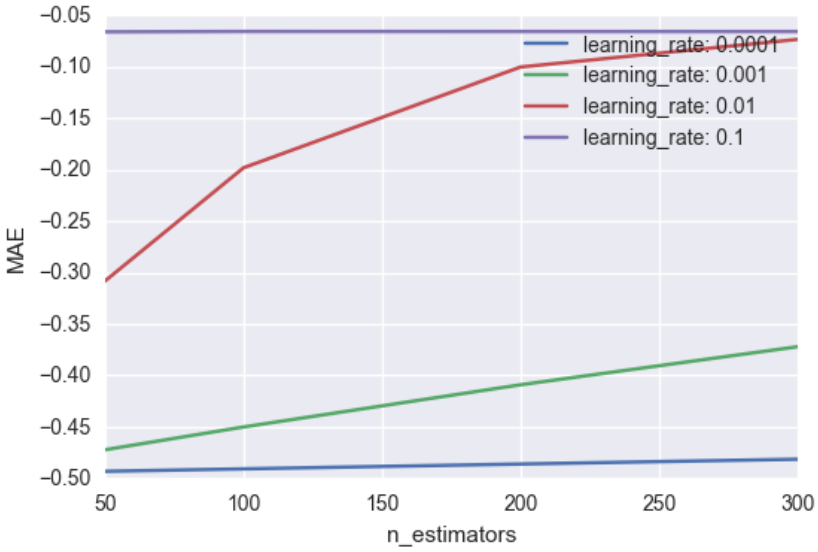
I spent a lot of time on parameter tuning for XG Boost. I first tuned max_depth and n_estimators (Number of trees). Increasing n_estimators will make the model slow to overfit. In contrast, increasing max_depth results in more complex trees, which can lead to overfitting so a shorter depth is preferred [4]. I graphically showed the grid search, which illustrates that the optimal MAE score is at 100 n_estimators and max_depth = 2.



XB Boost Tuning (Learning Rate)

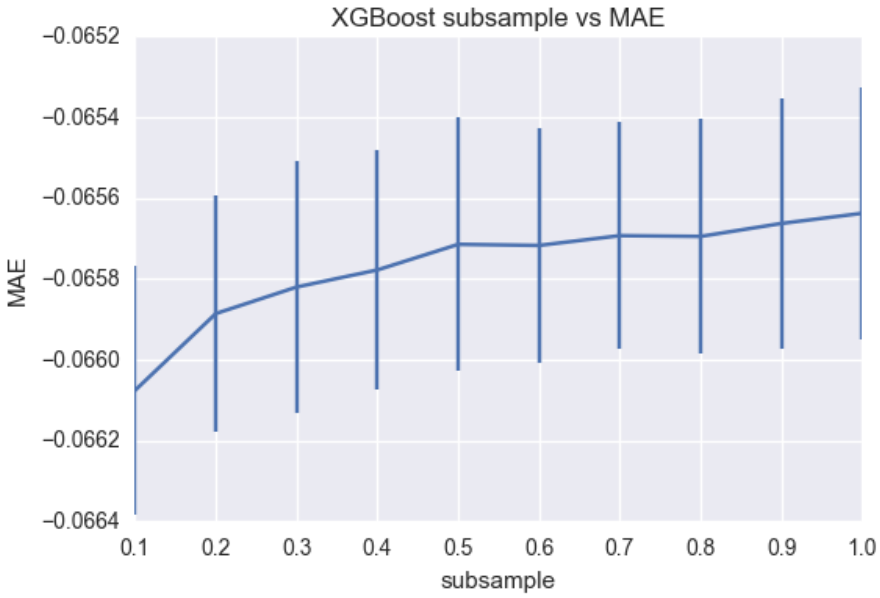
Next, I tuned the learning rate because gradient boosted trees are quick to learn and overfit training data. The learning rate is an effective way to slow down learning and to avoid overfitting. In gradient boosting, trees are created and added to the model sequentially. New trees are created to correct the residual errors in the predictions from the existing sequence of trees, which can result in model quickly learning and quickly overfitting. The learning rate acts as a weighting for corrections to new trees when they are added to the model [4].

The graph below illustrates that a learning rate of 0.1 is optimal given a low number of trees (100 to 300) and max depth of 2.



XB Boost Tuning (Subsample)

Bagging is a technique to subset rows (selected at random) of the training data to train individual trees. Bagging is conducted without replacement. A random forest uses subsets of rows in the training data to calculate the split point, but this process can also be used in gradient boosting because it is a greedy procedure. More specifically, new trees are added to correct the residual error of the existing model. Each decision tree is created using a greedy search procedure to select split points that best minimize an objective function. A greedy approach can result in redundancy because the same attributes and the same split points can be used over and over again. Bagging addresses this problem by introducing randomness that will allow for different trees to be created and added to the model [4].



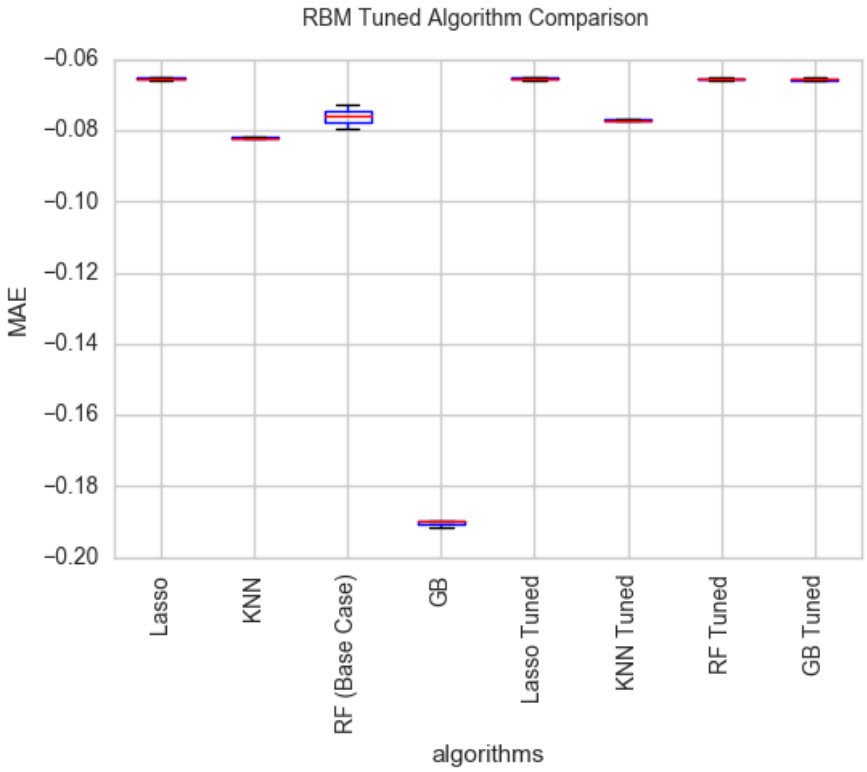
The graph above actually shows that my XGBoost model does not benefit from subsampling of rows.

IV. Results

(approx. 2-3 pages)

Model Evaluation and Validation

I added another pipeline to evaluate the best parameters in the models. This pipeline evaluates the models with cross validation and mean absolute error scoring and I also displayed the results of these pipelines on a box plot.



This graph shows that all algorithms performance improved from parameter tuning excluding Lasso.

In fact, XG Boost had the biggest increase in performance of 66% and our base case algorithm performance improved by 14%. The tuned algorithm performance is much more tightly clustered around 0.6 MAE.

MAE Scores	**Training**		
Algoritm	Untuned	Tuned	V%
Lasso	0.0655	0.0655	0%
KNN	0.0820	0.0771	6%
RF (Base Case)	0.0765	0.0656	14%
XG Boost	0.1903	0.0656	66%

I selected the tuned models for Lasso, RF, and XG Boost to evaluate on the test set. I threw out KNN because it was higher than the three other algorithms.

MAE Scores	**Training**			**Test**
Algoritm	Untuned	Tuned	V%	Tuned
Lasso	0.0655	0.0655	0%	0.0667
KNN	0.0820	0.0771	6%	
RF (Base Case)	0.0765	0.0656	14%	0.0668
XG Boost	0.1903	0.0656	66%	0.0668

The results were fantastic for all algorithms on the test data set and in fact mirrored the performance expected from the training set.

I chose to stack these algorithms to see if I could get a better performance. I found using the stacking regressor from mlxtend.regressor much easier than writing a class to weight algorithms and I used the following formula.

stack_clf = StackingRegressor(regressors=[GB_clf_tuned, RF_clf_tuned], meta_regressor=LASSO_clf)

Stacking didn't result in a great improvement but it did score better than RF and XGBoost.

MAE Scores	**Training**			**Test**
Algoritm	Untuned	Tuned	V%	Tuned
Lasso	0.0655	0.0655	0%	0.0667
KNN	0.0820	0.0771	6%	
RF (Base Case)	0.0765	0.0656	14%	0.0668
XG Boost	0.1903	0.0656	66%	0.0668
Stacking				0.0667

Justification

Tuning the baseline RF model's parameters resulted in significant increase in performance. Additionally, it was very beneficial to evaluate more than just one algorithm to see if a more optimal solution exists.

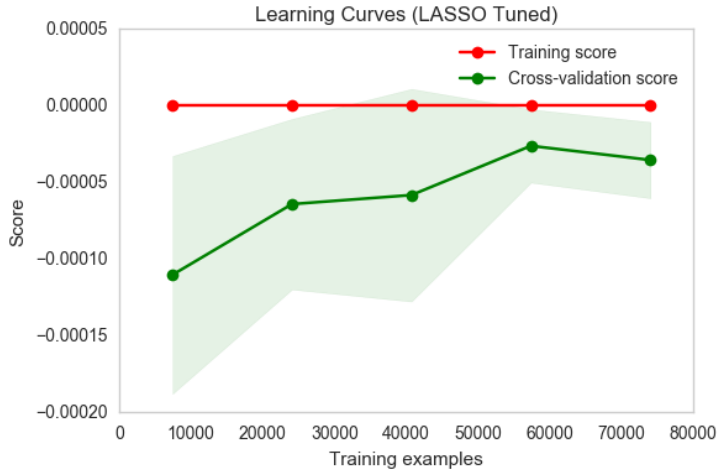
V. Conclusion

(approx. 1-2 pages)

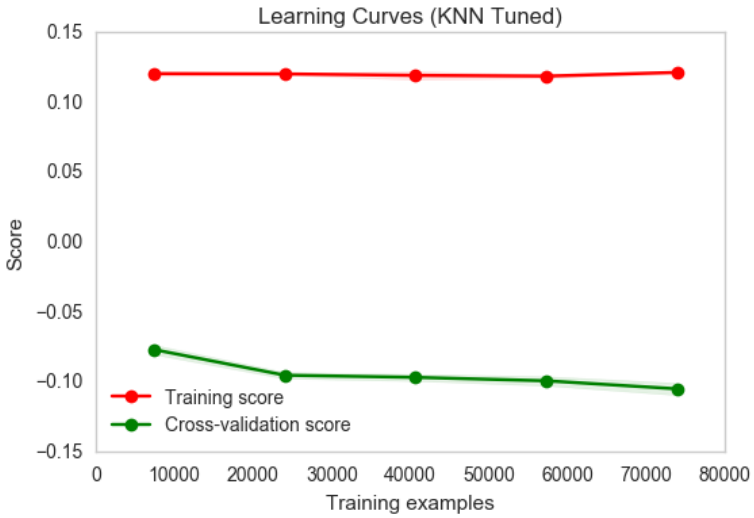
Free-Form Visualization

I evaluated all the algorithms learning curves. I find learning curves to be a very effective way to evaluate if a model is robust and will have good performance.

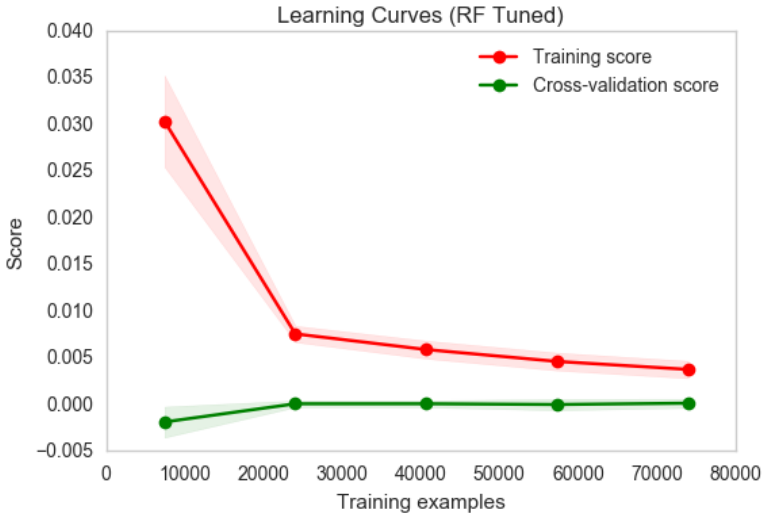
The Lasso learning curve shows that the model has good performance. The error is decreasing throughout training samples.



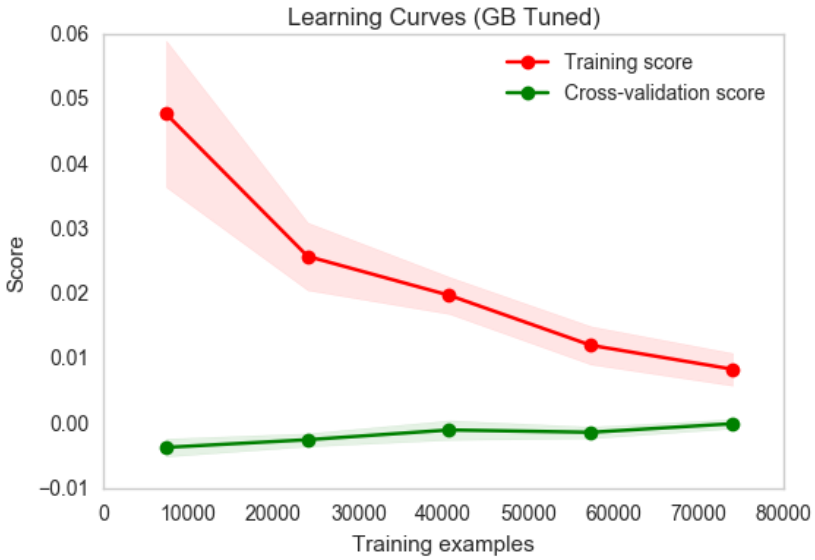
In contrast, the KNN model exhibits high variance and is overfitting the training data.



The tuned base model also shows great promise as the curves converge toward a very low error.



Finally, the XGBoost model also shows great potential because the error is steadily reduced with increasing training examples.



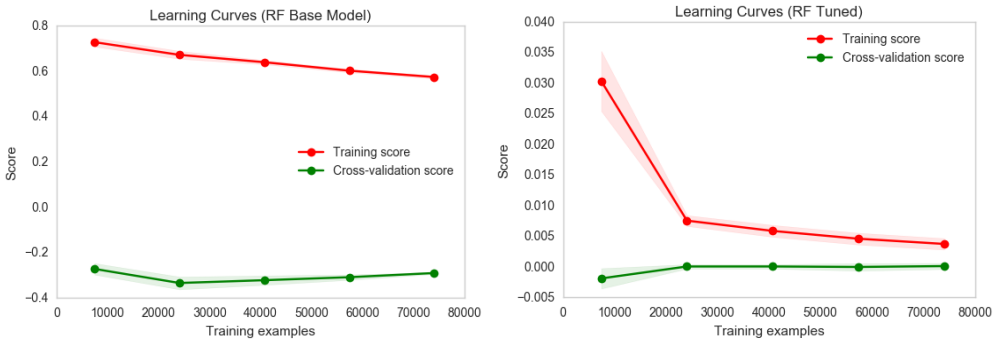
Reflection

My base model was a basic Random Forest with limited tuning and limited data preparation. I took this analysis so much farther with NA removal, binning, one-hot encoding, feature creation, feature selection, dimensionality reduction comparison, and algorithm comparison. I also tuned the algorithms in more depth as illustrated with XGBoost.

As shown in the learning curve comparison, I have 3 models that are excellent choices for evaluating new data. Additionally, these models all scored very well on the test data and during cross validation as illustrated in the box plots of the tuned algorithms.

Additionally, I only used one method of feature selection for the base model (SelectKBest). SelectKBest relies on human input to select the number of features. However, the SelectFromModel Random Forest feature selection that I chose for the capstone project is an improvement because it relies on a hard-minimum (median) and removes the human element.

Additionally, the learning curves below show the impact of parameter tuning on the base model. The base model had high variance and overfit the training data. Tuning key parameters of the base model resulted in a 14% improvement in cross validation score on the training data and an improved learning curve.



I found the hidden categorical variables that were of float data type to be difficult to incorporate. I tried several methods one-hot encoding, hashing, and binning and ultimately decided on binning + one-hot encoding. I felt good that adding these categorical variables improved baseline performance but I will continue to research if there is a better way to handle high cardinality categorical variables.

In conclusion, the process that I have laid out for evaluation of data and model selection is a road map that can be used for future regression problems.

Improvement

There are many improvements that I would love to test if time was permitting. First, is there a better way to test for outliers than Tukey? Also, should I expand outlier selection to other columns than the target variable?

Second, did I rule out polynomials too soon? More specifically, should I have included polynomials in feature selection before determining that they weren't worthwhile?

Third, are there other algorithms that I should have included? Neural networks require limited data manipulation and can handle a lot of features. Perhaps a neural network would have performed better.

Finally, a better solution exists because many scores are higher on the Kaggle board than I scored. However, I am proud to say that I have significantly improved my understanding of machine learning from my baseline model to the capstone project.

Bibliography

[1] A. E. Deeb, "Rants on Machine Learning," Medium, 22 Jun 2015. [Online]. Available: https://medium.com/rants-on-machine-learning/smarter-parameter-sweeps-or-why-grid-search-is-plain-stupid-c17d97a0e881. [Accessed 18 Febr 2018].

[2] "Zillow Prize: Zillow's Home Value Prediction (Zestimate)," Kaggle, [Online]. Available: https://www.kaggle.com/c/zillow-prize-1#description. [Accessed 18 Feb 2018].

[3] A. Géron, Hands-On Machine Learning with Scikit-Learn and Tensorflow, O'Reilly Media, Inc, 2017.

[4] J. Brownlee, Gradient Boosted Trees with XGBoost and scikit-learn, 2017.

[5] "Highlighting Outliers in your Data with the Tukey Method," Bacon Bits, [Online]. Available: http://datapigtechnologies.com/blog/index.php/highlighting-outliers-in-your-data-with-the-tukey-method/. [Accessed 25 4 2018].

[6] D. Becker, "Using Categorical Data with One Hot Encoding," Kaggle, [Online]. Available: https://www.kaggle.com/dansbecker/using-categorical-data-with-one-hot-encoding. [Accessed 26 April 2018].

[7] J. Moeyersoms, "Data Mining Tip: How to Use High-cardinality Attributes in a Predictive Model," KD Nuggets, [Online]. Available: #https://www.kdnuggets.com/2016/08/include-high-cardinality-attributes-predictive-model.html. [Accessed 15 March 2018].

[8] A. Zheng and A. Casari, in *Feature Engineering for Machine Learning*, O'Reilly Media, Inc, 2018.

[9] S. Ozdemir and D. Susarla, Feature Engineering Made Easy, Packt Publishing, 2018.