Machine Learning Engineer Nanodegree
Capstone Project
Shamit Patel
May 30, 2017

Definition

Project Overview

One of the most intriguing applications of machine learning is home valuation, because a home is often the most significant purchase an individual makes in their lifetime [1] and an accurate valuation can be the difference between making a successful sale and not. Zillow's Zestimate was developed to provide consumers with as much as information about homes and the real estate market as possible [1] so that they can make highly informed decisions about purchasing a home. A Zestimate is an estimated home value based on 7.5 million statistical and machine learning models that analyze hundreds of features of every property [1]. Zillow has cemented itself as one of the most reliable sources for real estate information in the U.S. because it has constantly improved upon its margin of error in valuating homes [1].

This project is based on Kaggle's Zillow Prize: Zillow's Home Value Prediction competition, whose initial goal is to improve the Zestimate residual error, and ultimately to build a home valuation algorithm from the ground up [1]. All datasets for this project were obtained from [2], the competition's data webpage.

As described in [2], the data consists of three files:

- properties_2016.csv – list of all properties with their features for 2016
- zillow_data_dictionary.xlsx – definitions of all property features
- train_2016.csv – training set with transactions from 1/1/2016-12/31/2016

These datasets will be described in more detail later on in the report.

Problem Statement

In this project, I focused on the initial competition goal of predicting the log-error between Zillow's Zestimate and the actual sale price of a home, given all the features of a home. The log-error is defined as

$$logerror = \log(Zestimate) - \log(SalePrice)$$ [2].

Predicting this error is important because it allows us to precisely gauge how accurate our home valuation predictions (Zestimates) are.

The expected solution for this problem will be in the form of either a Generalized Linear Model (GLM) or a nonlinear regressor that will predict a log-error given a particular list of feature values. A Generalized Linear Model is a flexible generalization of typical linear regression that allows for response variables whose errors are not distributed normally [3]. GLMs that were explored were linear regression, polynomial regression, ridge regression, Bayesian ridge regression, lasso regression, Elastic Net, Least Angle Regression, Orthogonal Matching Pursuit, and Stochastic Gradient Descent Regression. Nonlinear regressors that were explored were

Support Vector Regression, Nearest Neighbors Regression, Decision Tree Regression, Random Forest Regression, AdaBoost Regression, Gradient Boosting Regression, and Multi-Layer Perceptron Regression. The model that performed best was chosen as the solution.

Metrics
All models were evaluated using the Mean Absolute Error between the predicted log error and the actual log error, keeping in line with Kaggle's evaluation procedure [1]. The mean absolute error (MAE) is the average of the magnitudes of the differences between the actual and predicted values [4]:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| = \frac{1}{n}\sum_{i=1}^{n}|e_i|\,.$$

This metric was chosen (1) to be consistent with the Kaggle competition's evaluation procedure and (2) because it is an objective and standard evaluation metric for regression models.

Analysis

Data Exploration
The features from the properties file (properties_2016.csv) are:

```
parcelid,airconditioningtypeid,architecturalstyletypeid,basementsqft,bathroomcnt,bedroomcnt,build
ingclasstypeid,buildingqualitytypeid,calculatedbathnbr,decktypeid,finishedfloor1squarefeet,calcul
atedfinishedsquarefeet,finishedsquarefeet12,finishedsquarefeet13,finishedsquarefeet15,finishedsqu
arefeet50,finishedsquarefeet6,fips,fireplacecnt,fullbathcnt,garagecarcnt,garagetotalsqft,hashottu
borspa,heatingorsystemtypeid,latitude,longitude,lotsizesquarefeet,poolcnt,poolsizesum,pooltypeid1
0,pooltypeid2,pooltypeid7,propertycountylandusecode,propertylandusetypeid,propertyzoningdesc,rawc
ensustractandblock,regionidcity,regionidcounty,regionidneighborhood,regionidzip,roomcnt,storytype
id,threequarterbathnbr,typeconstructiontypeid,unitcnt,yardbuildingsqft17,yardbuildingsqft26,yearb
uilt,numberofstories,fireplaceflag,structuretaxvaluedollarcnt,taxvaluedollarcnt,assessmentyear,la
ndtaxvaluedollarcnt,taxamount,taxdelinquencyflag,taxdelinquencyyear,censustractandblock
```

All these features are defined in detail in zillow_data_dictionary.xlsx.

A sample data instance from the properties file looks like:

```
11016594,1.0,,,2.0,3.0,,4.0,2.0,,,1684.0,1684.0,,,,,6037.0,,2.0,,,,2.0,34280990.0,-
118488536.0,7528.0,,,,,,0100,261.0,LARS,60371066.461,12447.0,3101.0,31817.0,96370.0,0.0,,,,1.0,,,
1959.0,,,122754.0,360170.0,2015.0,237416.0,6735.88,,,6.0371066461e+13
```

The training file looks like:

```
parcelid logerror transactiondate
11016594   0.0276        1/1/2016
…and 90,810 more instances through 12/31/2016
```

Clearly, not all the properties of a home are actually described, as can be seen from the sample data instance from the properties file above. So, I used data imputation to fill in these missing values. Specifically, I inserted a value of 0 for all missing values because this seems to be a standard procedure. Next, I removed the `censustractandblock` feature because there was already a `rawcensustractandblock` feature and these were basically the same thing. I also removed the `parcelid` feature because this is simply a unique identifier for the property and doesn't play any

other descriptive role. However, this was used as the index in my pandas DataFrame that stored all the features for all the properties. This left me with 56 features to use for training.

I observed that the following features are categorical:
`airconditioningtypeid, architecturalstyletypeid, buildingqualitytypeid, buildingclasstypeid, decktypeid, fips, fireplaceflag, garagetotalsqft, hashottuborspa, heatingorsystemtypeid, pooltypeid10, pooltypeid2, pooltypeid7, propertycountylandusecode, propertylandusetypeid, propertyzoningdesc, rawcensustractandblock, regionidcounty, regionidcity, regionidzip, regionidneighborhood, storytypeid, typeconstructiontypeid, unitcnt, yardbuildingsqft17, yardbuildingsqft26, taxdelinquencyflag`

Of these, four are alphanumeric:

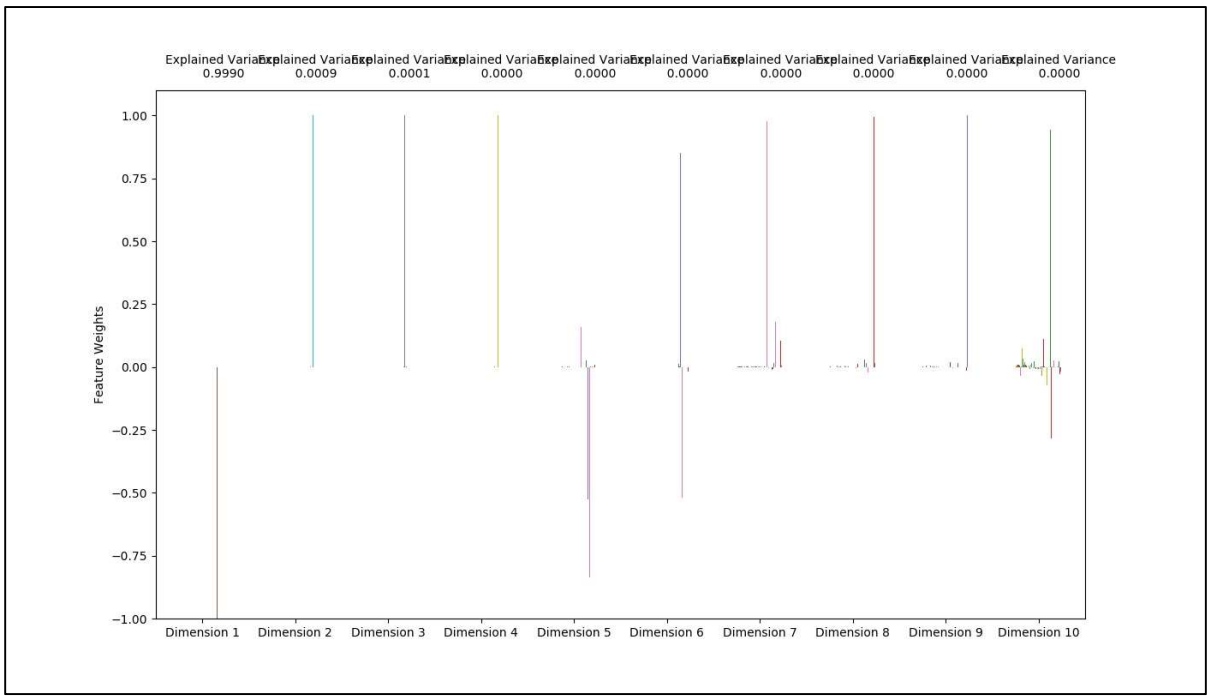
- `hashottuborspa (True/False)`
- `propertycountylandusecode (Example: 010D)`
- `propertyzoningdesc (Example: LARS)`
- `taxdelinquencyflag (Y/N)`

These four alphanumeric categorical features were converted to numerical values using pandas' to_numeric() function (for `hashottuborspa`) and the Categorical() function (for the remaining features). Finally, all noncategorical features were scaled to unit variance using sklearn's preprocessing.scale() function.

Data Properties:
As mentioned, the training file consists of 90,811 instances spanning the entire year of 2016. Given that I have 56 features, the curse of dimensionality is clearly at play here since I don't have enough training data to cover the entire feature space. So, I applied dimensionality reduction to the features, which will be discussed in more detail next.

Exploratory Visualization

Figure 1: Explained Variances of First 10 Principal Components of Features Data



As can be seen in Figure 1 on the previous page, the first 3 principal components of the features data together explain *all* of the variance in the data. This means that only a few particular features play a significant role in determining the log error between the Zestimate and sale price of a home. Since the first 2 principal components together explain 99.99% of the variance in the data, I chose those components when I did the dimensionality reduction so that I could capture the *majority* of the variance in the data yet at the same time avoid overfitting.

Algorithms and Techniques

The techniques I used to solve the problem of predicting log error between Zestimate and sale price consisted of two main phases: dimensionality reduction and regression.

Dimensionality Reduction:

I used Principal Components Analysis to determine the first 2 principal components of the features data. I then fit and transformed the features data to those 2 components and saved the results in separate variables reduced_training_features and reduced_testing_features, which would be used in the subsequent regression phase.

Regression:

I explored both GLMs and nonlinear regressors for the problem of predicting the log error between a Zestimate and the actual sale price of a home. The GLMs I explored were:

- Linear Regression: I used this to determine whether there is a linear relationship between the dependent and independent variables and if so, how strong that relationship is. This also served as a benchmark for the rest of the GLMs that I explored because all of those models were basically generalizations of linear regression.
- Polynomial Regression: I fit a degree-2 polynomial to determine if there is some nonlinear relationship between the log error and features and if so, what the strength of that relationship is. So, this model served as a good indicator for how the nonlinear regressors would perform.
- Ridge Regression: I explored this model (Tikhonov regularization) because it imposes a penalty on the size of the coefficients and can thereby address the problem of overfitting [5].
- Bayesian Ridge Regression: I used this technique because it estimates a probabilistic model of the regression problem [5], and therefore serves as a unique approach to regression relative to the other techniques. It also adapts to the data at hand [5].
- Lasso Regression: I used this model because it uses regularization to prevent overfitting, and I used parameters max_iter=2,500 and alpha=2 so I could learn as general of a model as possible.
- Elastic Net: I explored this algorithm because it is essentially a combination of Lasso and Ridge Regression. Specifically, it can learn a sparse model like Lasso, but still maintains the regularization ability of Ridge [5]. I used an l1_ratio of 0.5 to equalize the L1 and L2 penalties and thereby learn the best model possible.
- Least Angle Regression: I used this model because it is specifically intended for high-dimensional data, which is the case with this project. It is also as efficient as ordinary least squares [5].

- Orthogonal Matching Pursuit: I explored this algorithm because it imposes constraints on the number of nonzero coefficients [5]. That is, it uses regularization to prevent overfitting.
- Stochastic Gradient Descent Regression: I used this technique since it's an online algorithm that estimates the gradient of the loss each sample at a time, and thereby allows the model to learn local variations and structures in the data. It also works with data represented as dense numpy arrays of floating-point values for the features, which is exactly what I have [6].

The nonlinear regressors I explored were:

- Support Vector Regression: I used this technique (with default RBF kernel) because it can capture any nonlinear relationships in the data using the RBF (radial basis function) kernel.
- Nearest Neighbors Regression: This model was explored since it's simply a different and unique approach to regression where the target is predicted using local interpolation of the targets associated with the nearest neighbors in the training set [7]. This can allow the model to learn local structure in the data.
- Decision Tree Regression: I used this model because it can learn nonlinear relationships in the data, and I set min_samples_split=6 to prevent overfitting.
- Random Forest Regression: This technique was explored since it's an ensemble method that uses averaging to improve predictive accuracy and control overfitting [8].
- AdaBoost Regression: I used this model because it's an iterative algorithm that initially fits a regressor on the original dataset then fits additional copies of the regressor on the same dataset but where the instance weights are updated according to the current prediction error, so that subsequent regressors focus on more difficult cases [9]. This has the twofold benefit of initially controlling overfitting, but eventually capturing the local variations in the data also.
- Gradient Boosting Regression: This model was explored because it's an additive model that allows for the optimization of arbitrary differentiable loss functions [10]. So, like AdaBoost regression, it can learn a model that balances the bias-variance tradeoff very well. I used min_samples_split=4 to prevent overfitting.
- Multi-Layer Perceptron Regression: I used this model because it can learn very complex relationships between the output and input variables. It also uses backpropagation in the output layer, so it learns the best model possible [11]. I used max_iter=2500 and n_layers=75 so that the neural network could capture the most complex relationships from the data.

Benchmark

The benchmark model was simply a function that always predicts the mean of the log errors in the training data. That is, if the set of log errors in the training data is denoted by *E*, then the benchmark model is the following:

$$f(\boldsymbol{x}) = \frac{\sum_{e \in E} e}{|E|},$$

where $\boldsymbol{x}$ is a feature vector and *f* is the benchmark model. This benchmark is justified because the mean is a naive population estimator. The MAE of this benchmark is 0.0659848616942.

Methodology

Data Preprocessing
As discussed, preprocessing consisted of 4 main stages:

1. Filling in missing values with 0 using pandas' fillna() function
2. Feature selection (dropping `parcelid` and `censustractandblock` features)
3. Converting alphanumeric features to numerical values
4. Scaling noncategorical features to unit variance

Train/Test Split:
I trained on the instances with transactions from January-September 2016, and tested on the remaining instances (with transactions from October-December 2016). This particular split was chosen to keep in line with Kaggle's methodology.

Implementation
The sklearn library was used to implement all metrics and algorithms for this project.

Metrics:
I used the mean_absolute_error() function from the sklearn.metrics package to compute the MAE for a particular set of predictions.

Dimensionality Reduction:
I used the PCA() and fit_transform() methods from sklearn.decomposition to do the dimensionality reduction. Also, I used a slightly modified version of the visuals.py code from Udacity to generate Figure 1.

Generalized Linear Models:
The following sklearn methods were used to implement the GLMs:

- Linear Regression: sklearn.linear_model.LinearRegression
- Polynomial Regression: sklearn.preprocessing.PolynomialFeatures, sklearn.linear_model.LinearRegression
- Ridge Regression: sklearn.linear_model.Ridge
- Bayesian Ridge Regression: sklearn.linear_model.BayesianRidge
- Lasso Regression: sklearn.linear_model.Lasso
- Elastic Net: sklearn.linear_model.ElasticNet
- Least Angle Regression: sklearn.linear_model.Lars
- Orthogonal Matching Pursuit: sklearn.linear_model.OrthogonalMatchingPursuit
- Stochastic Gradient Descent Regression: sklearn.linear_model.SGDRegressor

Nonlinear Regressors:
The following sklearn methods were used to implement the nonlinear regressors:

- Support Vector Regression: sklearn.svm.SVR
- Nearest Neighbors Regression: sklearn.neighbors.KNeighborsRegressor
- Decision Tree Regression: sklearn.tree.DecisionTreeRegressor
- Random Forest Regression: sklearn.ensemble.RandomForestRegressor

- AdaBoost Regression: sklearn.ensemble.AdaBoostRegressor
- Gradient Boosting Regression: sklearn.ensemble.GradientBoostingRegressor
- Multi-Layer Perceptron Regression: sklearn.neural_network.MLPRegressor

Prediction/Scoring:
Prediction was performed using each of the models' predict() function and scoring was done using the score() function.

Complications:
The Support Vector Regression on the original features took forever and never finished. So, the results for this particular case are not available. However, this was expected given the nature of the algorithm and the problem (it's solving an optimization problem on a very large feature space). However, this model did work quickly using the reduced features.

Refinement
Refinement consisted of two main approaches:
- Dimensionality reduction
- Hyperparameter exploration

Dimensionality Reduction:
Initially, I fit all the models with the original feature data and observed the results. Then, I fit all of them with the reduced feature data and obtained better results in some cases.

Hyperparameter Exploration:
As discussed, I adjusted parameters for the following models:
- Lasso Regression: max_iter=2,500 and alpha=2
- Elastic Net: l1_ratio=0.5
- Decision Tree Regression: min_samples_split=6
- Gradient Boosting Regression: min_samples_split=4
- Multi-Layer Perceptron Regression: max_iter=2500, n_layers=75

These adjustments were done to optimally balance the tradeoff between bias and variance.

Results for GLMs (MAE of Benchmark: **0.0659848616942**):

	Mean Absolute Error	
Model	*Original Features*	*Reduced Features*
Linear Regression	0.066247558634	0.0660139278317
Polynomial Regression	0.0701624890728	0.066097340396
Ridge Regression	0.0662471912686	0.0660139278317
Bayesian Ridge Regression	0.0661563605476	0.0660139278317
Lasso Regression	0.0660015631779	0.0660133525174
Elastic Net	0.0660151223067	0.0660137840031
Least Angle Regression	26.4236181365	0.0660139278317
Orthogonal Matching Pursuit	**0.0659931202919**	**0.0659848573472**
Stochastic Gradient Descent	3.37277046534e+23	8.45108907272e+19

Results for Nonlinear Regressors (MAE of Benchmark: **0.0659848616942**):

	Mean Absolute Error	
Model	*Original Features*	*Reduced Features*
Support Vector Regression	N/A (Never finished)	0.0730565351188
Nearest Neighbors Regression	0.082520962392	0.106258698902
Decision Tree Regression	0.111798819389	0.281935880407
Random Forest Regression	0.0811745586131	0.10411552565
AdaBoost Regression	0.207601804071	0.372915012336
Gradient Boosting Regression	**0.0661077174404**	**0.068682837177**
Multi-Layer Perceptron Regression	47.7625242745	5340.8193883

Results

Model Evaluation and Validation

I chose Orthogonal Matching Pursuit as the final model because as can be seen from the tables above, it achieved the lowest MAE on both the original and reduced features amongst *all* the models, *and* it beat the benchmark MAE when using the reduced features. In addition, this model was chosen because of its robustness: despite reducing the dimensionality of the feature space, it performed *better*. This indicates that Orthogonal Matching Pursuit, like the rest of the GLMs, benefits from dimensionality reduction. This is in sharp contrast to the nonlinear regressors, which seemed to perform *worse* using reduced features. Intuitively, this dichotomy makes sense because GLMs are trying to learn essentially *linear* relationships from the data (which can easily be achieved after the feature space is reduced to two components), while nonlinear regressors are trying to learn *nonlinear* relationships from the data (which is harder to do when the feature space is reduced). The results from the Orthogonal Matching Pursuit model can be trusted to only a slight extent because it has only improved upon the benchmark very superficially.

Justification

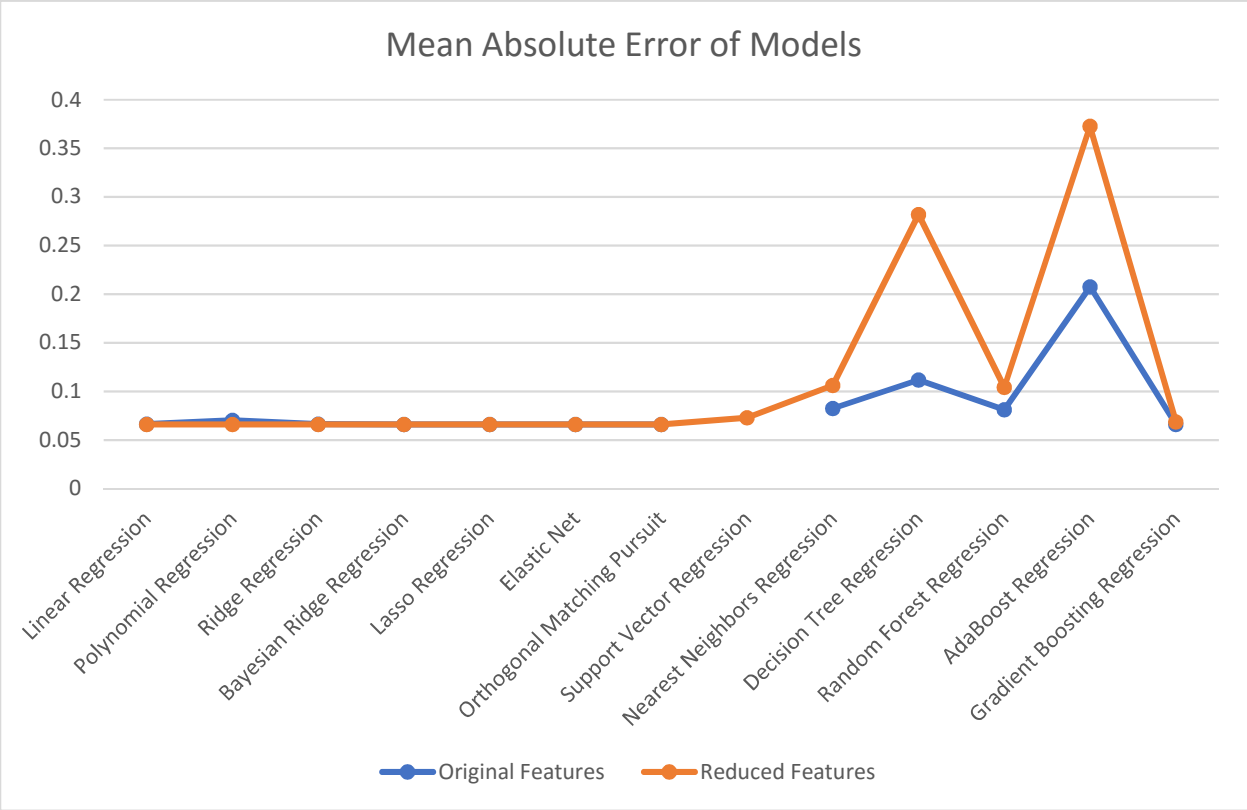
A two-tailed student's t-test with (1) the null hypothesis that the mean of the predictions of the O rthogonal Matching Pursuit model (using reduced features) is equal to the mean of the benchmar k predictions and (2) a significance level of 0.05 indicates that the final model is *not* significant e nough to have solved the problem. This is because the p-value of 0.99999999999965539 that we obtain from this test is *far* greater than 0.05, which indicates that we cannot reject the null hypothesis that the means are equal. In other words, the predictions of the Orthogonal Matching Pursuit model are essentially the same as those of the benchmark model (despite having a slightly better MAE). So, as discussed, the results of the final model can only be trusted to a slight extent because they don't represent a statistically significant improvement over the benchmark model.

Conclusion

Free-Form Visualization

I plotted the MAE for all models except Least Angle Regression, Stochastic Gradient Descent, and Multi-Layer Perceptron Regression because the MAEs for these models were way out of range compared to the MAEs of the other models. The plot can be seen in Figure 2 next.

Figure 2: Mean Absolute Error of Models



This visualization confirms what I have already discussed: that the GLMs performed better with the reduced features and the nonlinear regressors performed better with the original features. Again, this makes sense because GLMs are trying to learn linear relationships from the data, while nonlinear regressors are trying to learn nonlinear relationships from the data. This plot also shows the interesting trend that the results for the GLMs were much more *consistent* than the results for the nonlinear regressors. This can be explained by the fact that the nonlinear regressors were much more different from each other than the GLMs: the GLMs are all generalizations of *linear* regression, while the nonlinear regressors are simply arbitrary nonlinear models that aren't necessarily generalizations of some underlying model.

Reflection

The process for solving the problem of predicting the log error between a Zestimate and an actual sale price consisted of preprocessing the data, applying PCA to determine the principal components, applying GLMs and nonlinear regressors using both the original and reduced features to determine the optimal model, evaluating/validating/justifying the final optimal model, and further exploring these final results using a free-form visualization. An interesting aspect of this project was the PCA visualization because it indicated that the first two principal components *alone* accounted for 99.99% of the variance in the data, which I found astounding given the number of original features (56). This truly demonstrates PCA's ability to compress high-dimensional data into compact representations while still retaining maximum information. An aspect of this project I found difficult was finding a model that could beat the benchmark, but after much effort, I finally succeeded with the Orthogonal Matching Pursuit model using the

reduced features. This final model does meet my expectations for the problem because it did beat the benchmark. However, I'm not sure it can be trusted in a real-world situation because its improvement over the benchmark solution is not statistically significant.

Improvement

One aspect of the implementation that could be improved is the hyperparameter optimization. I could use sklearn's grid search feature to learn the best model for each type of regression. I would also like to see how deep neural networks would perform on this regression problem, because such networks are known to learn very complex representations and relationships from data that's fed to them. Both these improvements could potentially lead to much better results than the ones I obtained from my final model because they result in models that are robust to overfitting yet at the same time capture intrinsic local relationships in the data.

References

1. https://www.kaggle.com/c/zillow-prize-1
2. https://www.kaggle.com/c/zillow-prize-1/data
3. https://en.wikipedia.org/wiki/Generalized_linear_model
4. https://www.kaggle.com/wiki/MeanAbsoluteError
5. http://scikit-learn.org/stable/modules/linear_model.html
6. http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html
7. http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.htm
8. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
9. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html
10. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html
11. http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html