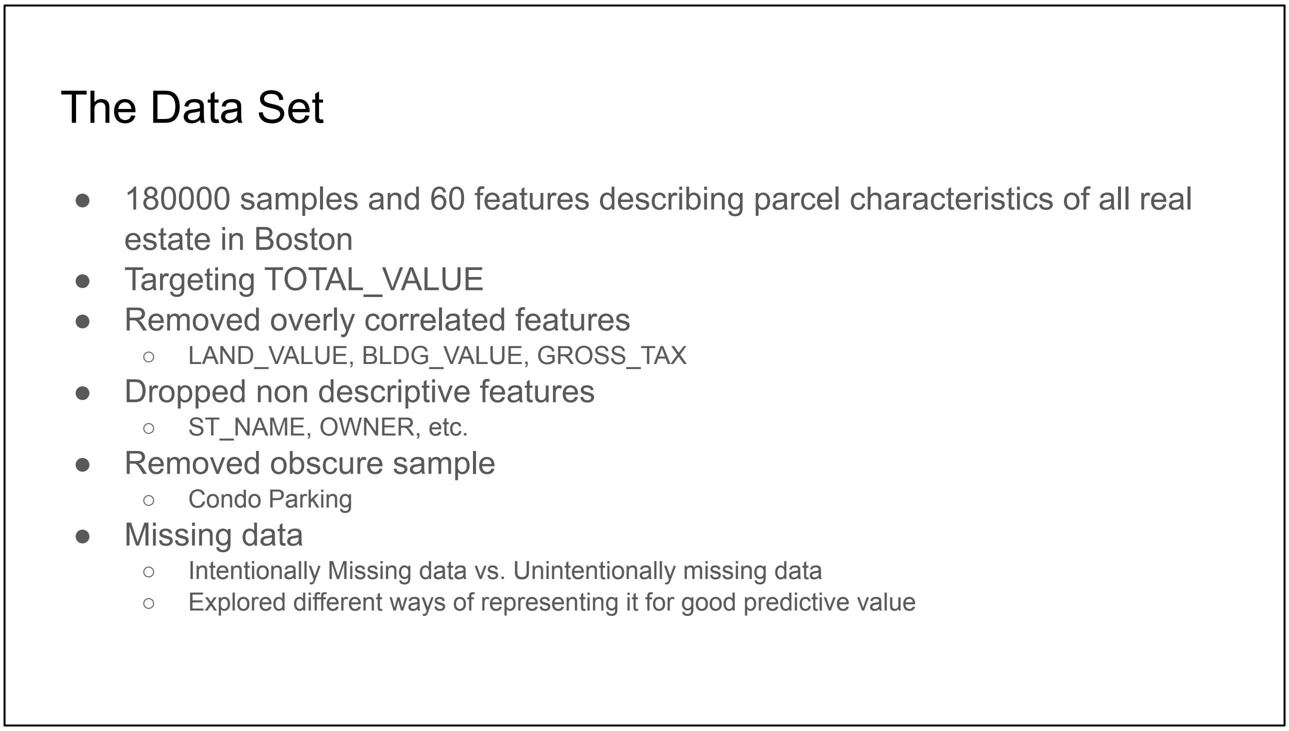
Boston Housing Market Analysis

EEECE 5644

Sean Klein and Bradley Bares

Data Set: https://data.boston.gov/dataset/property-assessment/
Code: https://github.com/bradleybares/eece5644-final-project



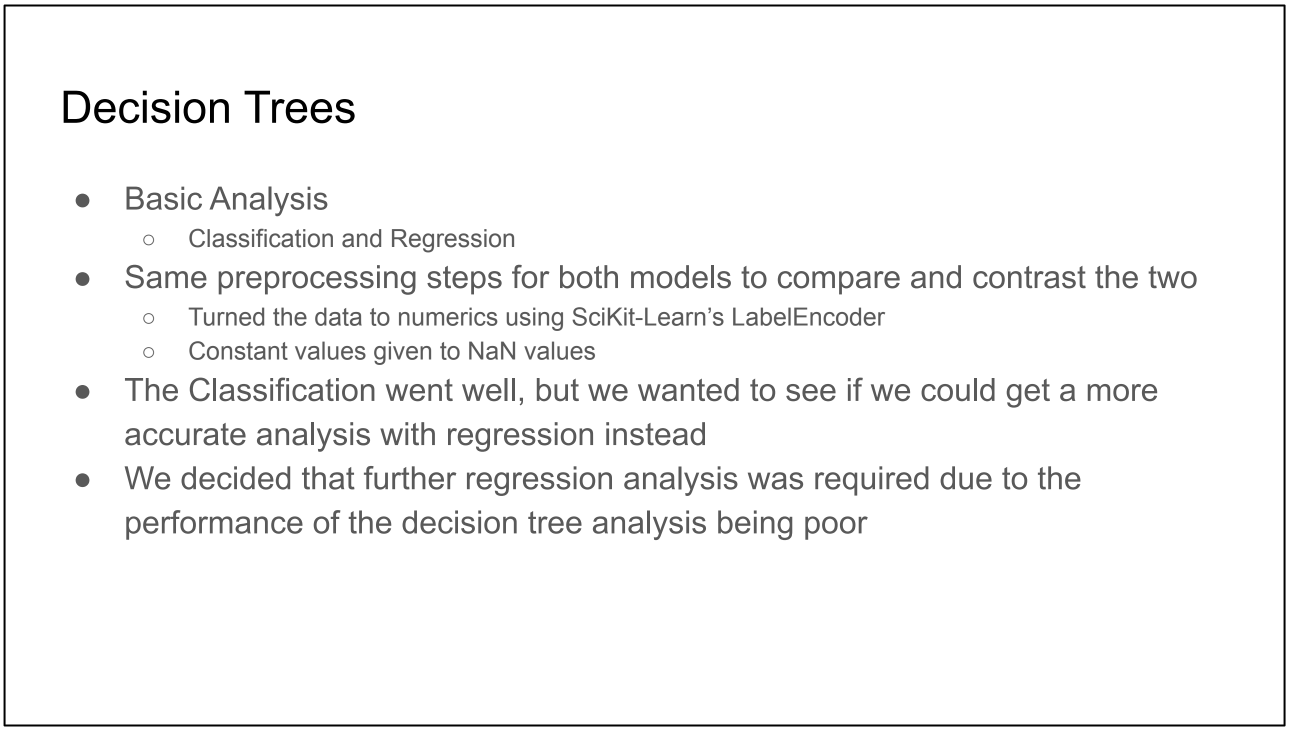
The Data Set

- 180000 samples and 60 features describing parcel characteristics of all real estate in Boston
- Targeting TOTAL_VALUE
- Removed overly correlated features
    - LAND_VALUE, BLDG_VALUE, GROSS_TAX
- Dropped non descriptive features
    - ST_NAME, OWNER, etc.
- Removed obscure sample
    - Condo Parking
- Missing data
    - Intentionally Missing data vs. Unintentionally missing data
    - Explored different ways of representing it for good predictive value

The Boston Property Assessment department produces yearly a large and rich data set describing all the real estate in Boston. Totalling 180000 samples and 60 different features our goal was to estimate the total value of a property given it's feature set and determine the most important features.

The features that we thought of as unfulfilling were features that could easily replace our target variable for analysis. For example, BLDG_VALUE had a correlation to the TOTAL_VALUE of .9715 meaning that it is heavily correlated to the data. The features that we thought had little to no correlation were ones like ST_NAME and OWNER, where if you had more contextual information they might be strong indicators of housing value, but with just the dataset alone, they were lacking in predictive power. They would also lose all of their meaning when we encode them as integers which needed to be done in order to use SciKit-Learn. There were also some columns that had too many missing values (Like BTHRM_STYLE3). These columns were more representative of other features so they were dropped. Some property types were exceptional offenders of inaccurate or missing data (Condo Parking) so they were removed.

The data set was rampant with missing data so even with removing the columns with the highest amounts of missing data, we still had a lot of preprocessing left to do. The specific ways we went about handling this were unique to the associated model.



Decision Trees

- Basic Analysis
  - Classification and Regression
- Same preprocessing steps for both models to compare and contrast the two
  - Turned the data to numerics using SciKit-Learn’s LabelEncoder
  - Constant values given to NaN values
- The Classification went well, but we wanted to see if we could get a more accurate analysis with regression instead
- We decided that further regression analysis was required due to the performance of the decision tree analysis being poor

To start the analysis of the data set, we decided to go with two different kinds of Decision Tree.

The ways we handled the missing data was: 1) estimate the values of the data so that is either replaced with a value of 0 or the average values of its specific column, and 2) replace the missing value with -1 to signal to the models that the value was in fact missing so it would not affect the data as much as potentially improperly estimating the value.

SciKit-Learn does not work well with non numerics so we had to use an encoder to change all of the data values of all other types into numerics to run the trees.

We started with the classification tree because it was a low risk/low return form of analysis which would provide us with a stepping stone for our further analysis. We were able to attain fairly good results with the classification.

Then we moved onto our first kind of regression, with the regression tree. The results that were achieved did not hit our desired goals so we knew we had to try other forms of regression.

Decision Tree Classification (R2 Score: .7876)

- We split the data set up into

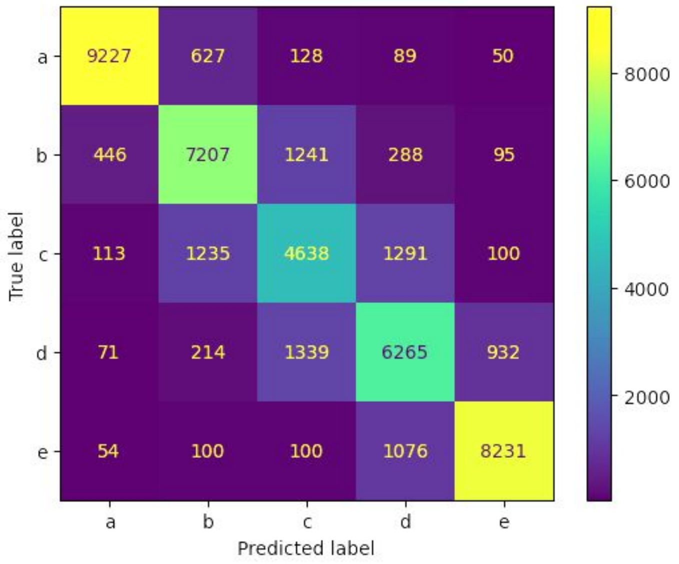
  A: 0 - 350,000

  B: 350,000 - 550,000

  C: 550,000 - 700,000

  D: 700,000 - 1,000,000

  E: 1,000,000 - Highest Total Value
- The depth after trial and error testing that performed the best was 17
- The figure to the right is the confusion matrix constructed by the results of the decision tree on the test set compared to the actual values



	a	b	c	d	e
a	9227	627	128	89	50
b	446	7207	1241	288	95
c	113	1235	4638	1291	100
d	71	214	1339	6265	932
e	54	100	100	1076	8231

In order to do classification on the data set, we had to construct our own classes for analysis. Since for our regressions, the target value was TOTAL_VALUE, we used that column to create a new one by grouping the values into evenly sized range buckets and assigning a letter to it.

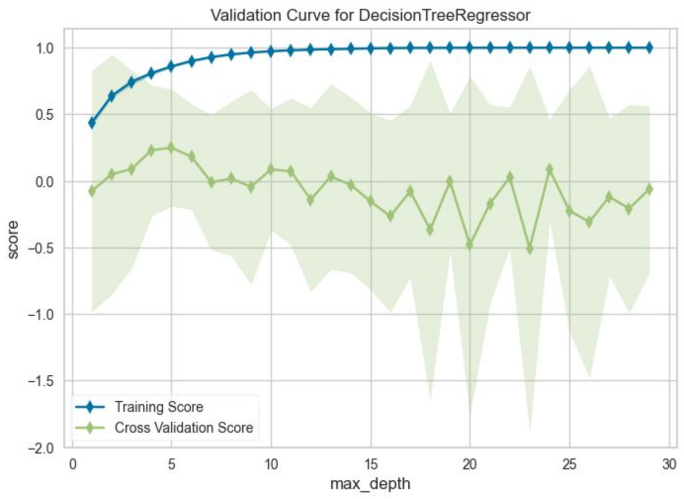
At first we aimed to make these equally distanced in values, however the size of the splits ranged vastly, so through some guess work, we were able to figure out splits that were more separated more evenly.

There was no easy way to figure out the ideal depth for this tree other than trial and error, we wanted to ensure that the R2 score returned from the test set was maximized and we were not overfitting the model on the test set.

The bright and more colorful diagonal of the confusion matrix represents all of the values which were properly assigned, while the more dark and purple squares are the values where the predicted and true values were misaligned. You can see that the highest misaligned values grow greater as they get closer to their actual value, which shows that even the misaligned values show a trend towards the true value.

Decision Tree Regression (R2 Score: .6556)

- Determining tree depth was a difficult process (random_state of the graph unknown)
- For random_state = 42, through trial and error, we found that a tree depth of 15 worked best



The regression underperformed in comparison to the classification. It also proved to be harder to analyze for things like what the ideal depth is.

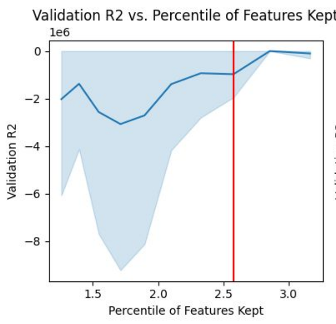
We implemented yellowbrick’s validation curve which allows you to examine the R2 scores of the model in comparison to the maximum depth hyperparameter. As you can see the graph does not have a consistent cross validation curve which means that finding the ideal depth is a near impossible task.

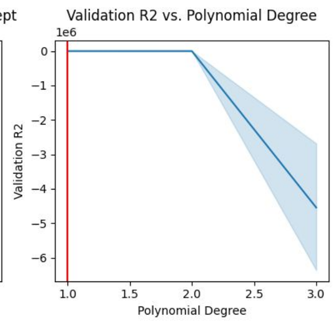
We ended up going with a depth of 15 by making educated guesses based on the graph and looking at the scores that were achieved. We did not test any depth past 40, as we believe that at that depth it would surely be overfitted due to the amount of data points that we had available to us.

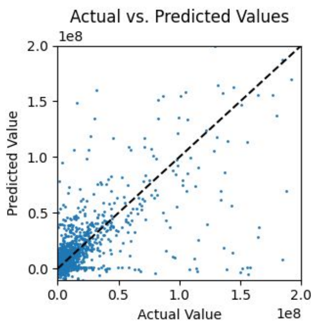
With a score of .6556, we decided to turn our direction towards other forms of regression to see if we could get better models that represented the data in a more accurate way. This led us to trying out the following two other forms of regressions.

Linear Regression (R2 Score: .6909)

- Imputed NaN Categoricals with mode & Imputed Numericals with mean
- Encoded Cats with OHE
- GridSearchCV Hyperparameter Selection
    - Percentile: 2.58 - 14 Features
    - Degree: 1







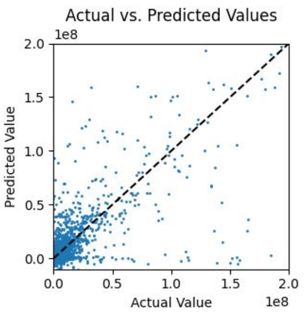
Linear Regression is the classic regression tool and we thought it would be a good fit for the dataset given the abundance of linear correlations between features and Property Value.

The best performing preprocessing imputed NaN categorical values with the most frequent category and the numerical values with the average value. The categorical values were then encoded using the OneHotEncoder. A pipeline was then run to select the most correlative features, create a polynomial feature set and run a linear regression.

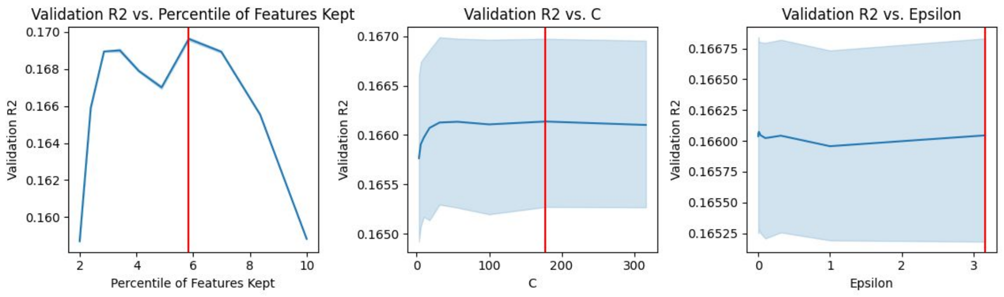
The hyperparameters for this pipeline were the percentile of features to keep and the polynomial degree. Using grid search cross validation we found the best percentile of features to be 2.58 or 14 features and the best degree to be 1. This makes sense as the correlation drops off sharply among the features and the features are mostly linearly correlated.

With these hyperparameters the model had an R2 score of .6909, a solid performance for a basic regression technique.

LinearSVR (R2 Score: .7113)



- Imputed NaN Categoricals as ‘NA’ & Numerical as -1
- Encoded Cats with OHE & Scaled Nums
- GridSearchCV Hyperparameter Selection
  - Percentile: 5.84 - 33
  - Regularization Parameter (C): 178
  - Loss Parameter (Epsilon): 3.16

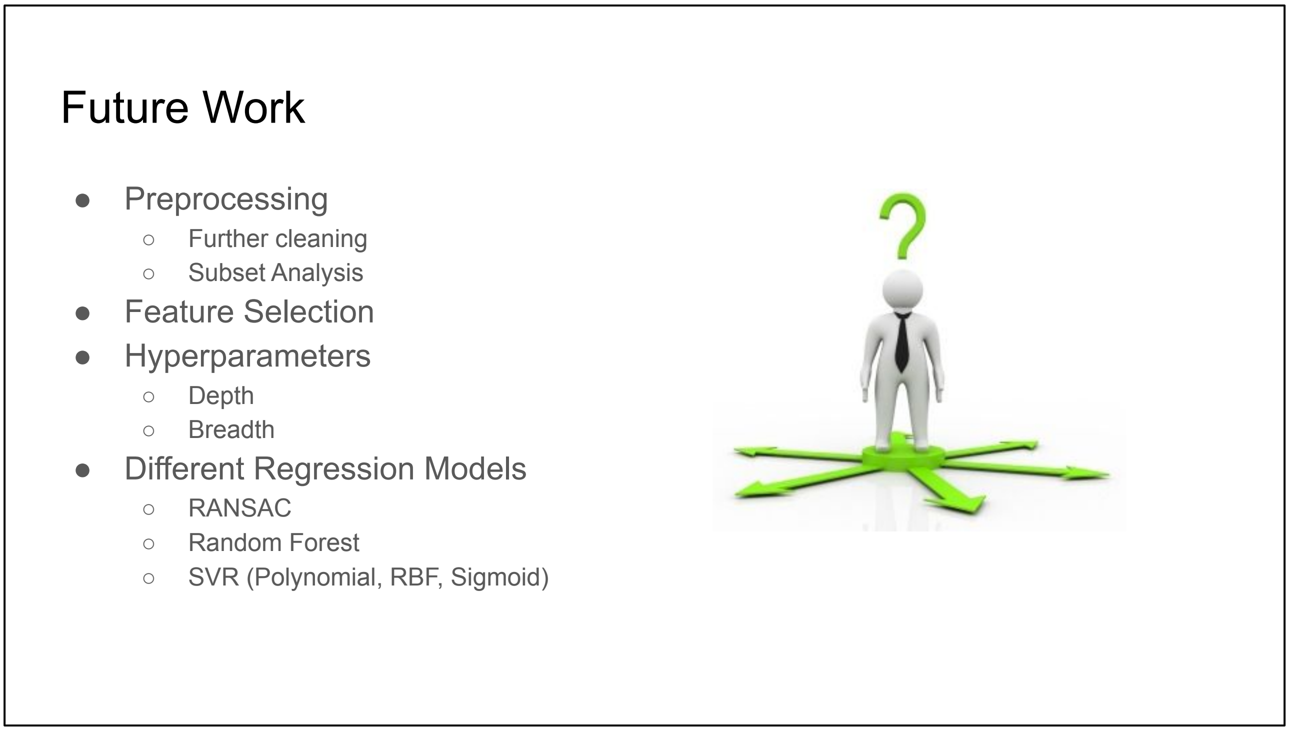


For a more advanced regression technique we selected Linear Support Vector Regression as we believed that a linear kernel would perform well with this dataset and that the advanced computations would be more accurate than Linear Regression.

The best performing preprocessing imputed NaN categorical values with ’NA’ and the numerical values with -1 to indicate a lack of information. The categorical values were then encoded using the OneHotEncoder. The numerical values were then normalized and scaled to unit variance. A pipeline was then run to select the most correlative features and run a Linear SVM Regression.

The hyperparameters for this pipeline were the percentile of features to keep, the regularization parameter (C), and the loss parameter (Epsilon). Using grid search cross validation we found the best percentile of features to be 5.84 or 33 features, the best regularization parameter to be 178, and the best loss parameter to be 3.16. This makes sense because LinearSVR is capable of determining a more complex association between features meritting the additional features and the larger than default parameter are in line with the large dataset.

With these hyperparameters the model had an R2 score of .7113, a clear victory over the other regression techniques.



Future Work

- Preprocessing
  - Further cleaning
  - Subset Analysis
- Feature Selection
- Hyperparameters
  - Depth
  - Breadth
- Different Regression Models
  - RANSAC
  - Random Forest
  - SVR (Polynomial, RBF, Sigmoid)

As with most Machine Learning projects there is room for improvement and given extra time, more work to be done. Given more time we would explore further preprocessing, feature selection, hyperparameters, and different regression models.

Considering preprocessing we would purse further cleaning the data, imputing more sensible values than the broad imputation that we performed and we would evaluate the performance of the regression models on subsets of the data. Given the vast differences in certain properties, regression could be improved by first creating subsets of properties.

Moving to feature selection we only really considered filtering methods, more specifically features that had high linear correlation with property value. Alternate methods of filtering take into account information gain and variance of the features.

When revisiting the model hyperparameters we would want to explore other tunable parameters and further tune the existing parameters. This would likely require new methods of hyperparameter selection that are more efficient than attempting every combination with cross validation as GridSearchCV does.

Finally and most obviously we can pursue other regression models. Some mainstream models to consider are RANSAC, Random Forest, and SVR with non-linear kernels.