PROJECT #1 – REGRESSION ANALYSIS

Prepared by : Kisha Taylor
Date : 12th September 2019

Problem Statement

We will predict the median price of a house based on given variables such as the age of the house, the number of rooms, town crime rate per capita etc. This is a multi-variate regression problem where we will predict a quantitative target variable based on a set of attributes (predictors). In regression, we measure the relationship between the quantitative dependent (target) variable and independent predictor variables. We are using the Boston Housing Market dataset where the target variable is the median housing price.

Methodology

1. **Problem Statement/Definition**
2. **Data Collection**
   - Dataset: Boston Housing Market readily available
3. **Data Preparation**
   (i) Data Exploration
   (ii) Data Cleaning
   (iii) Split into Train and Test
   (iv) Data Analysis
   (v) Feature Generation &/Or Feature Selection
   (vi) Data Preprocessing
   - Scale features where appropriate
4. **Train Model**
5. **Validate Model & Tune Model hyperparameters**
6. **Test Model assumptions (Eg. assumptions of a regression model)**
7. **Select best model**
8. **Report results**
9. **Conclusion**

List of models used:

1. **Linear Regression**
   a. using subset of features

2. **Polynomial Linear Regression**
   a. using subset of features

3. **Penalized Linear Regression - Lasso Regression**
   a. using subset of features
   b. using all features
   c. using polynomial transformation on all features

4. **Decision Regression Tree**

Ensemble Models:

5. **Random Forest**
6. **Gradient Boosted Tree**

So, technically more than 8 models were evaluated as the polynomial features explored were

to N polynomial degrees where N ranged from 2 to 5 (inclusive). There were 6 distinct types of models evaluated as noted above.

The dataset had the following variables:

CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO B LSTAT MedianPrice

where the Median price was the target variable.

METHODOLOGY

Problem Statement

We defined the problem from a business perspective and defined it further from an analytics perspective. So, we started out by identifying what we wanted to achieve, which was predicting the housing prices based on particular set of variables then we had to identify the kind of analytical technique that would be used to achieve these results. In this case we recognized that it was a regression problem and we would explore several regression models.

> *We will predict the median price of a house based on given variables such as the age of the house, the number of rooms, the location etc. This is a multi-variate regression problem where we will predict a quantitative target variable based on a set of attributes (predictors). In regression we measure the relationship between the quantitative dependent (target) variable and independent predictor variables. We are using the Boston Housing Market dataset where the target variable is the median housing price.*

Data Preparation

- **Data Collection & EDA**

As shown above, we collected the data then performed exploratory data analysis where we tried to understand the variables used, the values for the variables, the data types for each variable. The distribution for each variable. The frequency or range of variable values. We tried to identify missing values. In this case, there were no missing values.

- **Data Cleaning**

We checked number of missing variables for each feature (i.e entire dataset) initially when we started our data exploration. As mentioned, our dataset does not have missing values. If it did, we could handle via imputation (eg. Fill in with mean values) or delete row with missing variable. We would not want to discard rows with missing values if there were more than just a few as that would be throwing out valuable information since the other attribute value would be discarded.

Transform data to dummy variables if categorical.

If any of the independent variables are categorical, we need to transform the values into numerical values using dummy variables. In this case, all variables were already numeric.

Check outliers

Detected using 1.5 * IQR factor as threshold for deviation. That is, if the value falls outside the range of + or - 1.5*IQR where IQR is the Interquartile range (Q3-Q1) then it is an outlier.

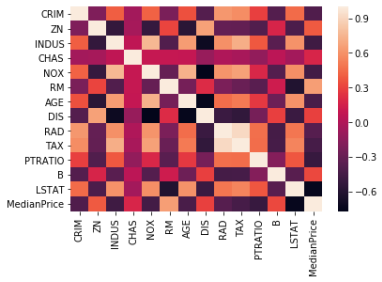
Note that it may be necessary to eliminate outliers depending on model being used. Eg. Linear models do not handle outliers well.

- **Split Data into Training and Test dataset**

We partition the dataset into two separate datasets. One for training and the other for test dataset. We do this my randomly selecting a specific percentage (%) of rows in the original dataset for each. In our case we used 70% for training and 30% for our test set. This is important because we need to ensure that in our model building process there is no influence at all by the test data.

- **Data Analysis**

For this we focused on analyzing the relationship between the predictor variables and the dependent (target) variable by generating a correlation matrix. This is performed as a pre-step to model building and forms apart of the feature selection process.



- **Feature selection**

Having identified that there are highly correlated predictors we will apply another technique known as the Variable Inflation Factor (VIF) technique to identify highly correlated predictor variables by assessing the correlation between that feature being assessed and all the other predictor features. So, the Pearson's correlation assesses the correlation only between pairs of variables while the VIF assesses multiple variables versus the feature in question.

Using the Pearson's correlation technique, we identified the following features that should be dropped:
'DIS', 'RAD', 'AGE', 'TAX', 'NOX', 'INDUS', 'RM', 'LSTAT'

Variance Inflation Factor (VIF)

VIF tells us how much the variance in the model has been inflated consequent on multicollinearity on the model. VIF = 1 means no correlation at all. if VIF is between 1 & 5: this means there is moderate correlation while VIF > 5 means high correlation exists.

Note: The VIF for a particular feature is calculated by regressing the feature against all the other features. The formula is : $VIF_j = 1/(1-Rsq_j)$. So, $feature_j$ is the dependent feature and all the other features are predictor variables in the model. Based on the formula, if Rsq_j approaches 1 then the value of VIF approaches infinity. If Rsq =0 (other features do not influence any variance in $feature_j$, the dependent var) then the VIF = 1.

After dropping redundant features:

VIF factor	features
1.059015	CHAS
2.020560	CRIM
2.236746	ZN

```
3.723832      RAD
3.934894      DIS
4.244625    LSTAT
```

Feature Scaling

The features were scaled prior to training. This was necessary because we were using a Linear Regression model for the first model which would not handle features that have different scales very well. The model would be bias toward the features that have a higher scaling which may not necessarily reflect its predictive power.
We used Standard Scaler which transforms the values such that the mean = 0 and the standard deviation= 1.
Formula: Scaled x = (x-mean)/Std Dev.

The following Regression models were explored:

1. Linear Regression

Linear regression measures the relationship between a quantitative dependent variable and independent predictor variables (either quantitative or categorical).

One of the key assumptions of linear regression is that the relationship between the dependent and the independent variables is linear with respect to the coefficients/parameters of the independent variables. So, if the coefficient of x_1 = 0.3 then this means for a one unit change in the independent variable the dependent variable changes by 0.3 while holding all other variables constant. The total change in the dependent variable is additive over the change influenced by each independent variable. So, to calculate the total change in y (dependent variable) we will multiply the coefficient of each variable by the value of each xi and sum them.

So, $\mathbf{y = \beta_o + \beta_j X}$ or in pure matrix notation $\mathbf{y = \beta_j X}$ where β is the feature coefficient matrix and j = 0 to n number of features and X is the feature matrix. Note $x_o = 1$ always and β_o represents the intercept term.

We are trying to estimate the coefficients such that the total (sum) squared error is minimized. The error is defined as the difference between the actual value and the predicted value of y.
RSS (Residual Sum of squares) = $1/N \sum(y - y\hat{})2$ (Error / Loss function)
This is also referred to as the OLS (Ordinary Least squared method). Other loss functions can be minimized however OLS is the most popular for regression. Essentially, this means that we are trying to find a straight line, not just any straight line, but one that minimizes the squared errors, in other words minimizes the extent to which our predictions on average are wrong.

The assumptions in Linear Regression:

- **#1**: Linear relationship exists between independent variables and response (or dependent) variables.
- **#2**: No multicollinearity, already checked and action taken resulting in the removal of redundant vars.
- **#3**: Constant variance of the errors- Homoscedasticity.
- **#4**: The residuals must not be correlated i.e no autocorrelation must exist.
- Optional - Assumption #5: The response variable and the residuals must be normally distributed.

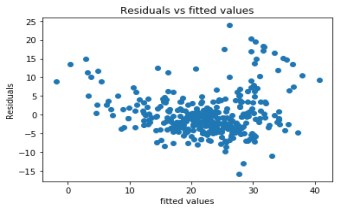
In this project, we assessed the validity of our linear model based on all four (4) key assumptions and found that all four were in violation. Hence, we could not (no matter what the performance results reflected) choose the linear regression model as these results would be misleading and unreliable.

#1: Linear relationship exists between independent variables and response (or dependent) variables.

X- violation

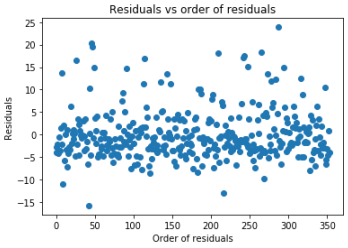
#3: Constant variance of the errors.

X- violation



Note : No symmetry about the invisible horizontal line at zero.

#4: The residuals must not be correlated i.e no autocorrelation must exist.



Note : No symmetry about the invisible horizontal line at zero.

#2: No multicollinearity, already checked and action taken resulting in the removal of redundant vars.

Note that this model used a subset of the original features derived after using Variable inflation Factor technique to filter out multicollinear variables, a necessary step to avoid a violation of one of the key assumptions in Linear regression listed above as assumption#2.

Subset of features used to train model:

CRIM ZN CHAS DIS RAD LSTAT

Results:

MSE	: 34.66
Rsq.	: 0.62
Adj-R-sq.	: 0.61

2. Polynomial Linear Regression

This is still linear regression however where the features have been transformed to polynomial terms eg. X_i^2 X_i^3 X_i^4 X_i^5 where X_i is set to X_i^2 or X_i^3 etc.

This is an efficient way of fitting a curvilinear line. Note, the function is still linear with respect to the coefficients.

Note we explored polynomial to the degree N = 2, 3, 4 and 5. Based, on the results on the test set we chose polynomial N=2, a quadratic equation. These polynomial linear models all used the same subset of features as the previous linear model.

CRIM ZN CHAS DIS RAD LSTAT

Results

MSE	R-sq	Adj_R_sq	Poly degree N
25.7874	0.7186	0.707	2
43.9924	0.52	0.5002	3
547.321	-4.9716	-5.2187	4
16,921.90	-183.627	-191.2668	5

3. Lasso Penalized Regression

This is a statistical technique to avoid over-fitting and it is being used now because we want to explore the possibility of using more features by letting the model innately (naturally) choose for us.

The model shrinks the coefficients (possibly even to zero) by adding a penalty term to the Loss function (OLS). This penalty term is defined as the sum of Beta (feature vector) squared multiplied by alpha (a control variable with values greater than or equal to 0). As alpha approaches infinity, the coefficients shrink towards zero. As alpha approaches 0, the coefficients get larger (vs. the former case). Note at alpha = 0, the penalty term has no effect.

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}|\beta_j| = \mathrm{RSS}+\lambda\sum_{j=1}^{p}|\beta_j|.$$

Ref. for eqn.: Introduction to Statistical Learning in R (ISLR)

Hyperparameter Tuning

Note that in Linear Regression there were no hyperparameters to tune. However, in Lasso Regression we have one hyper-parameter to tune. The penalty term is defined as the sum of Beta (feature vector) squared multiplied by alpha (a control variable with values greater than or equal to 0). This control variable "alpha" is our hyper-parameter. As alpha approaches infinity, the coefficients shrink towards zero. As alpha approaches 0, the coefficients get larger (vs. the former case). Note at alpha = 0, the penalty term has no effect and the coefficients will thus be equal to that of a regular linear regression model. The closer alpha moves towards infinity the greater number of coefficients will be reduced to a zero value.

Cross-validation and tuning being performed together.

Cross validation is a technique which estimates the performance of the model on unseen data.
We are using k-fold cross validation where we divide the data into k parts and use one part for testing/validation and all others for training. We repeat the process such that all k parts are used for validation. Note that the tuning is also performed here. For each value of our hyperparameter we conduct k-fold cross validation. The best estimator is the parameter values in this case the coefficients and the hyperparameter values that give the best results.

Predictor variables used in training

(i) The model was trained using a subset of features matching the ones in the first linear model.

CRIM ZN CHAS DIS RAD LSTAT

After tuning the best estimator for alpha = 0.1

Results :

MSE : 35.13
R-Sq. : 0.62
Adj-R-Sq. : 0.60

(ii) Then all features were used :

CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO B LSTAT

Results:

MSE : 20.85
R-Sq. : 0.77
Adj.-sq. : 0.75

(iii) We also used polynomial transformations of all the original variables.
Results:

MSE	R-sq	Adj_R_sq	alpha - hyperparam.	# predictors with non-zero coeff.	NCoeff.	Poly degree N
20.8531	0.7725	0.7546	0.1	11	14	1
12.8183	0.8601	0.8297	0.1	27	105	2
10.9604	0.8804	0.843	0.1	36	560	3
10.6477	0.8838	0.8345	0.1	45	2380	4
1.2643	0.888	0.8121	0.1	61	8568	5

Results (shown above) do no reflect a significant improvement in Adj-R-sq. as the polynomial degree N increases above N=2. Hence, we have chosen N=2, avoiding an unnecessary increase in model complexity for a marginal improvement in model fit.

Note the hyper-parameter alpha has a value = 0.1 for all polynomial models after tuning.

4. Decision Tree

Decision Trees are hierarchical models that employ a top down greedy approach (locally optimal solution) where the input space is partitioned at each step. The algorithm works recursively. Decision Trees can be used for both regression and classification. A tree consists of decision nodes (including the root node), branches (with respect to a node branches are the values for the attribute) and leaves. It has a strong appeal in the business setting given its ease of interpretation. Further, model development is simplified given limited pre-processing required. Unlike linear regression it is robust to outliers. It does not require any scaling of the input data (predictors) and can handle missing data.

How does it work exactly?
It recursively searches the input space to determine the best predictor where best is defined as the predictor that if used to split the dataset will result in the target variable being homogenous or as close to homogeneity as possible versus the other predictors. The goal is to find the features that have the best predictive power, that is, are the best at predicting the target value. In the case of classification, this (homogeneity) is measured based on an impurity test such as Entropy. Other measures such as Gini Index can also be used. Essentially, we evaluate the candidate predictors based on their information gain, that is, the extent to which they reduce class impurity (or increase in homogeneity). The predictor that has the highest reduction in class impurity implies it has the highest information gain. So, if we split on this predictor we would be close to segmenting the data instances into distinct exclusive classes. In our project, since this is a regression problem the Decision Tree algorithm used MSE (mean squared error) to assess impurity. Rather than computing the probability of an attribute value (as in classification) we simple sum over the MSE for each attribute value.

Key elements
- Determining criteria for “best split” among candidate predictors (that is measure for “goodness of a split”)
- When to stop building the tree

Note, Trees are unstable learners and hence prone to overfitting. One way to avoid overfitting is to fully grow the tree and prune after based on the impact of each branch on increase homogeneity. Another way is to deliberately not fully grow the tree and not try to achieve pure homogeneity.

The model was trained using all original features in the data set.

CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO B LSTAT

We tuned the three (3) hyperparameters below :¶

These all help to control the tree full growth in an attempt to avoid overfitting.

- **max_depth** : represents the maximum levels the tree will have where a child of the root would be on level one and a child of the that node would be on level 2.
- **min_samples_split** : represents the minimum number of samples (i.e instances at a decision node) that must be present for a split to be allowed.
- **min_samples_leaf** : represents the minimum number of leaves (i.e instances at a leaf node) that must be present after a split. Otherwise the split will not be allowed.

Note min_samples_split counts the instances at the node before the split ensuring it meets the min criteria while the min_samples_leaf counts the instances that would be present at the next level in the tree after the split. It is possible to have the former be satisfied and the latter not satisfied resulting in the split being disallowed.

Ref. : Great explanation available here :
https://stackoverflow.com/questions/46480457/difference-between-min-samples-split-and-min-samples-leaf-in-sklearn-decisiontre

Hyper-parameters tuned and the results of the best estimators were:

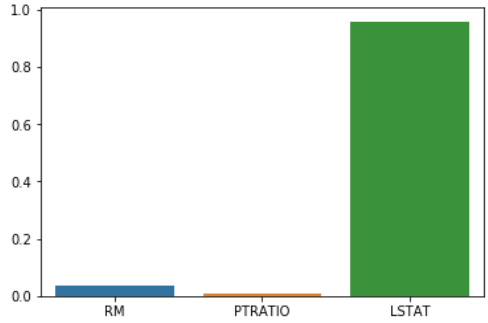
Max_depth =3
min_samples_split =35
min_samples_leaf =35

(same hyperparameters used for Decision Trees above, results differ slightly)

Results:

MSE	: 28.9203
R-Sq.	: 0.6845
Adj-R-sq.	: 0.6547

Feature Importance

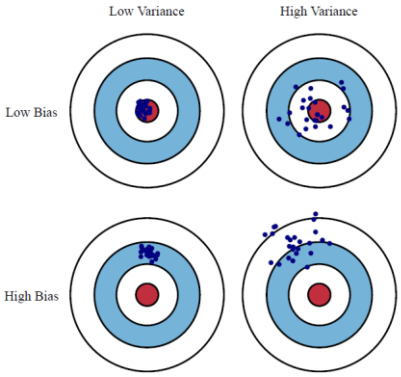


This reflects LSTAT as having the highest feature importance value while RM and PRATIO have significantly lower values. All other features are assumed not important.

5. Random Forest

The basic concept of ensemble learning is that we make predictions based on the predictions of multiple learners. This is in contrast to the use of just one learner. The purpose of ensemble learning is to improve model performance. It does so by reducing variance error of unstable models (high variance & low bias learners). This is an ensemble learner which uses the decision tree as its base learner.

Note, let's go back to some fundamentals in model building to ensure we grasp the concept of ensemble learning. Prediction Errors can be decomposed into two main components : reducible and irreducible error. Now, reducible error can be further decomposed into bias and variance error.
Total reducible error = bias^2 + variance. Note, as the name implies irreducible error is error outside of our control due to sampling error or input error. Reducible error can be impacted by the model builder. However, when bias is impacted it typically also has the opposite impact on variance. As a model increases complexity, it tends to decrease its bias and increase its variance. Bias is how accurate the model is at making a prediction. Variance, on the other hand, is how different the prediction is when there is a small change in the dataset (measures how sensitive the model is to small changes in the dataset). See diagram below that explains this well.



Ref. : http://scott.fortmann-roe.com/docs/BiasVariance.html

When a model's complexity is low it means the model is simple and has made a lot of assumptions about the underlying pattern of the data. So, errors occur if there is a discrepancy

between the data and the assumptions o the model. For example, if the underlying pattern is a non-linear pattern and we are trying to fit a linear model then error due to our bias will occur because the model cannot adjust to the pattern of the data as it lacks flexibility due to its simplicity. This is called underfitting the data.

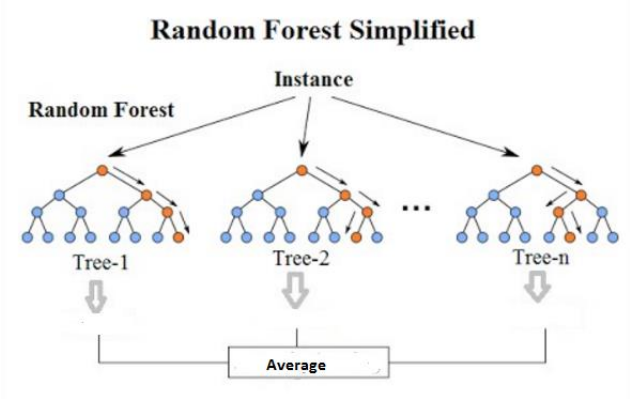
Now, over-fitting occurs when the model is very complex and even learns the noise in the data rather than the true underlying pattern. It fits the data too closely and therefore will not be good at predicting unseen data. Slight changes in the dataset will lead to different predictions and thus led to variance error. So, therein lies the classic bias-variance dilemma where an attempt to reduce bias by increasing the model's complexity increases the variance error of the model also. As the model complexity increases the bias error decreases more rapidly than the increase in variance error. However there comes a point at which the change in errors are the same (in opposite directions). This is the optimal (ideal) point. After this point the variance error increases more rapidly than the decrease in the bias error leading once again to an overall increase in total model error. See the graph below.

Ref.: https://towardsdatascience.com/holy-grail-for-bias-variance-tradeoff-overfitting-underfitting-7fad64ab5d76

Now, let us get a bit more into the concepts related to ensemble learning. Let's start with bootstrapping. Bootstrapping is resampling with replacement. This is done typically when we have limited data to work with and need to ensure we have a sufficient sample size. Bagging is short for bootstrap aggregation when we aggregate over the bootstrap samples. Aggregation can be in the form of a simple or weighted average etc. for regression type problems or a voting where the mode or most frequent vote is used.

How does this relate to ensemble methods? In random forest we use bootstrap aggregation (Bagging) to derive the final prediction using multiple learners. We randomly subset the dataset for each learner and we even further randomize by our selection of candidate features on which to split on. Not all features are considered for a best split only a randomly chosen set of features. Bagging reduces the variance of the model given that we average over the individual learners.

Note in Random forest we generate the base learners in parallel, one independent of the other. This is in contrast to gradient boosting which we will delve into in the next section below.



Ref. : https://commons.wikimedia.org/wiki/File:Random_forest_diagram_complete.png

The model was trained all original features in the dataset.

CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO B LSTAT

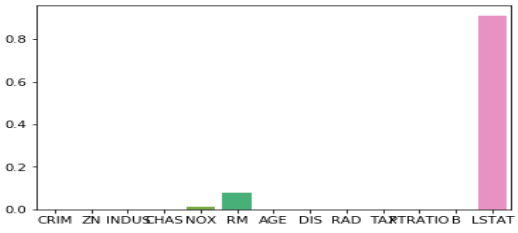
Hyper-parameters tuned and the results of the best estimators were:

Max_depth =1
min_samples_split =35
min_samples_leaf =35
(same used for Decision Trees above)

Results:

MSE	: 29.5144
R-Sq.	: 0.678
Adj-R-sq.	: 0.6476

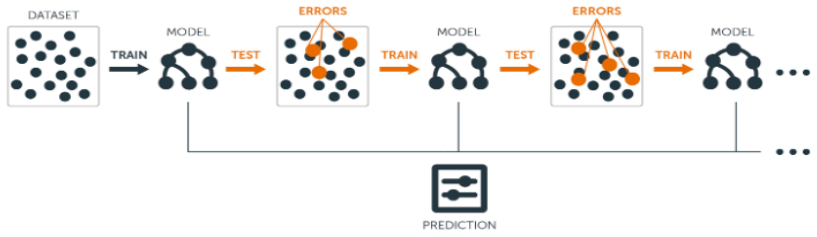
Feature Importance- Random Forest



Similar to the feature importance results for Decision Trees, the feature importance results for Random Forest also reflect LSTAT and RM as the two most important features as shown in the graph above.

6. Gradient boosted Tree

The gradient boosting algorithm is also an ensemble learning algorithm that in contrast to the random forest trains its learners sequentially and thus there is a dependency among all learners. The subsequent learners are trained based on the errors made by the previous learners. We use the entire dataset for the very first learner then after training we test that learner. Any observations that did not have the correct prediction are flagged as errors and those observations are weighted more heavily than others. The next learner is fed training data via bootstrapping. Since the observations that had errors are more heavily weighted, we expect the sample drawn will likely include those observations. We essentially model the error of the previous learner. See diagram below which illustrates the sequential nature of the model building process for gradient boosted models.



Ref. : https://towardsdatascience.com/simple-guide-for-ensemble-learning-methods-d87cc68705a2

The m+1 (subsequent) learner is trying to model the errors of the previous learner. By doing so, it can adjust the previous learners result by its error to derive the actual prediction.

See formulas below and explanation from the following source:

Ref. : https://en.wikipedia.org/wiki/Gradient_boosting)

$$F_{m+1}(x) = F_m(x) + h(x).$$

$$F_{m+1}(x) = F_m(x) + h(x) = y$$

or, equivalently,

$$h(x) = y - F_m(x).$$

Loss function :

$$\frac{1}{2}(y - F(x))^2$$

Note the gradient in gradient boosting stems from the fact that the estimate of h(x) (the error, which is being modelled) is actually the negative gradient of the loss function,

computed by taking the first derivative of the loss function with respect to F(x) multiplied by -1.

The model was trained using all original features in the dataset.

CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO B LSTAT

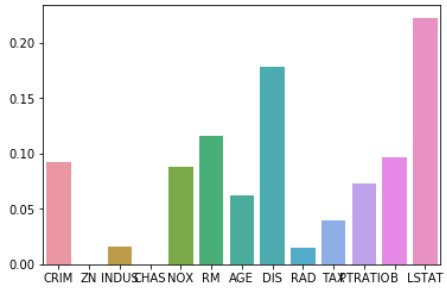
Hyper-parameter results:

max depth: 3
min samples leaf: 35
min samples split: 35
max leaf nodes: 6

Results:

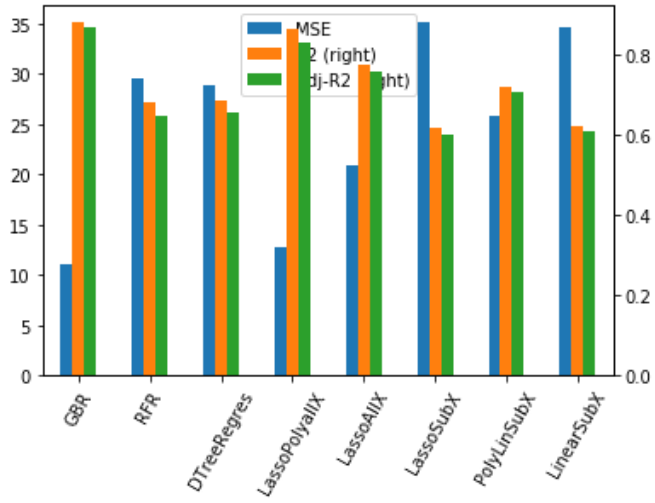
MSE	**:** 11.144
R-Sq.	**:** 0.8784
Adj-R-sq.	**:** 0.867

Feature Importance



LSTAT and DIS then RM emerged as the most important features using Gradient Boosted Trees. This is in contrast to the Decision Tree and Random Forest results which reflected RM as #2 in its rank of importance rather than #3.

RESULTS



	ModelNames	MSE	R2	Adj-R2
0	GBR	11.144007	0.878413	0.866959
1	RFR	29.514394	0.677982	0.647647
2	DTreeRegres	28.920281	0.684464	0.654740
3	LassoPolyallX	12.818300	0.860100	0.829700
4	LassoAllX	20.853133	0.772481	0.754605
5	LassoSubX	35.126830	0.616747	0.600889
6	PolyLinSubX	25.787400	0.718600	0.707000
7	LinearSubX	34.664314	0.621794	0.606144

The results above reflect the Lasso Regression (Penalized linear model) as the worst performing model when a subset of features was used. However, the performance of this Lasso Regression model improved significantly when all features are used and the model innately selects the best features by how it values the coefficient o the features (so zero coefficient means no importance, the lower the coefficient, the lower the variable importance and vis-versa). Note also that when Lasso Regression was used on all the features that had been transformed to a polynomial of degree N= 2, the performance as measured by the Adj-R^2 improved from 0.75 to 0.83.

The Linear model had the 2nd to worst performance with an Adj-R^2 = 0.61. Note that this is not surprizing as after testing for linear regression assumptions it was evident that the underlying pattern in the data from our residual vs fitted plot was non-linear. So, we were forced to explore other models that could handle non-linear data.

Surprizing Results for Random Forest (RF) and Decision Tree (DT)

The performance of the Random Forest and Decision Tree was surprisingly low as reflected in the Adj-R^2 of 0.65 for each.

So, the two outstanding performers were the Gradient boosted machine and the Lasso Regression where polynomial transformations (of degree N=2) were used on all features. Results, based on the Adj-R^2 reflected 0.87 and 0.83 respectively.

CONCLUSION

Gradient Boosted Machine emerged as the best performing model as it had the best MSE, R2 and Adj-R^2 of all the models. **Note however, I would opt. to use Penalized Linear Regression - Lasso** where the polynomial transformations of all features were used rather than Gradient boosted machine since there was only a marginal improvement in model performance yet GBM poses a greater challenge with respect to interpretability in a business context.