Project Report

House Prices: Advanced Regression Techniques

Project Team,

Teja Kiran Chunduri(txc163430)

Mohana Prudvi Dongala(mxd165330)

Sowmya Bhupathiraju (sxb162130)

1. **INTRODUCTION AND PROBLEM DESCRIPTION**

Buying a house is a dream of almost every person. House Prices depend on numerous factors in different economic conditions. Generally, when we plan to buy a house, we do analysis on numerous factors to see whether the price for the house is fair and a worthwhile investment. This is no less than prediction. In big data and machine learning terms, this is a regression problem for the machine.

The problem here is to build models to predict the house prices in the Ames City based on various features about the house. The data for this competition comes from Ames City Housing data which was provided by the Assessor's Office Records System.

They have provided us with

- Train.csv
- Test.csv
- Data description file

2. RELATED WORK

The concept of prediction using regression has been used extensively from the past. For reference, we have gone through various blogs and discussions that are similar to our problem.

One such approach was done by Kevin Wrong who tried to predict the house prices on Ames Dataset. We studied how preprocessing was done on the data, like correlations, removing attributes etc. We opine that the model can further be developed by doing more feature engineering.

Ricky Yue and Jurgen De Jager have published the Advanced Regression Modelling on House Prices. They have cross validated the predicted values from each of the L algorithms to form a new N*L matrix. They used the ensemble models to generate predictions on the test. They have not done much of the data analysis or the preprocessing.

Kenneth Richard Corsini has done a regression model on the Cobb County, Georgia. He has considered various building characteristics as attributes and done extensive data analysis. His model helped to develop a model for determining changes in certain attributes of residential real estate. He used basic models for the prediction.

Our first motto on mind is to go through each and every attribute and iteratively test and build our final model. Instead of the existing approach of removing attributes solely on one factor like correlation, we will be using various tools like correlation, histograms, distribution so as to come to a conclusion.

3. DATASET DESCRIPTION

The raw dataset has been modified by Dean De Cock with variables that are needed by the buyer before purchasing the house. These include quantitative and qualitative features of the houses.

The training data set given for the given competition had training values where each has one ID and predicted cost column.

- Total features: 79 attributes and 1 predicted value (SalePrice)
- Total number of training instances: 1460
- Total number of testing instances: 1460

Attributes and their significance

Slno	Attribute	Description
1.	Sale Price	the property's sale price which must be predicted in dollars.
2.	MSSubClass	The class of building
3.	MSZoning	Zoning classification
4.	LotFrontage	The length of the street connected to the property
5.	LotArea	Lot size(square feet)
6.	Street	Type of road access
7.	Alley	Type of alley access
8.	LotShape	shape of property
9.	LotContour	contour of the property
10.	Utilities	utiliies available in the property
11.	LotConfig	Lot configuration
12.	LandSlope	Land Slope of property
13.	Neighbourhood	Physical locations(In Ames City)
14.	Condition1	Closeness to main road or railroad
15.	Condition2	Closeness to main road or railroad (if a second is present)
16.	BldgType	Dwelling type
17.	HouseStlye	Dwelling type
18.	OverallQual	Overall quality of the material used and finish
19.	OverallCond	Overall condition rating
20.	YearBuilt	Original construction date
21.	YearRemodAdd	Remodel date
22.	RoofStyle	Type of roof
23.	RoofMatl	Roof Material
24.	Exterior1st	Exterior covering material on house
25.	Exterior2nd	Second exterior covering material on house
26.	MAsVnrType	Masonry veneer type
27.	ExterQual	Exterior material quality
28.	ExterCond	Present condition of the exterior material

29.	Foundation	Type of foundation
30.	BsmtQual	Height of the basement
31.	BsmtCond	Condition of the basement
32.	BsmtExposure	Walkout level basement walls
33.	BsmtFinType1	Quality of basement finished area
34.	BsmtFinSF1	Type 1 finished(sq. feet)
35.	BsmtFinType2	Quality of second finished area (if present)
36.	BsmtFinSF2	Type 2 finished(sq. feet)
37.	BsmtUnfSF	Unfinished basement area(sq. feet)
38.	TotalBsmtSF	Total basement area(sq. feet)
39.	Heating	Type of heating
40.	HeatingQC	Heating quality and condition
41.	CentralAir	If Central air conditioning is present
42.	Electrical	Electrical system
43.	1stFlrSF	First Floor area
44.	2ndFlrSF	2ndFlrSF: Second floor area
45.	LowQualFinSF	Low quality finished area
46.	GrLivArea	Above grade living area(sq feet)
47.	BsmtFullBath	Number of full bathrooms in Basement
48.	BsmtHalfBath	Number of half bathrooms in Basement
49.	FullBath	Full bathrooms above grade
50.	HalfBath	Half bathrooms above grade
51.	Bedroom	Number of bedrooms
52.	Kitchen	Number of kitchens
53.	KitchenQual	Kitchen quality
54.	TotalRmsAbvGrd	Total rooms above grade
55.	Functional	Home functionality rating
56.	Fireplaces	Number of fireplaces
57.	FireplaceQu	Fireplace quality
58.	GarageType	GarageType
59.	GarageYrBlt	Year of garage construction
60.	GarageFinish	Interior finish of the garage
61.	GarageCars	Number of cars in garage capacity
62.	GarageArea	Size of garage(sq. feet)
63.	GarageQual	Garage quality
64.	GarageCond	Garage condition
65.	PavedDrive	Paved driveway
66.	WoodDeckSF	Wood deck area(sq. feet)
67.	OpenPorchSF	Open porch area(sq. feet)
68.	EnclosedPorch	Enclosed porch area(sq. feet)
69.	3SsnPorch	Three season porch area(sq. feet)
70.	ScreenPorch	Screen porch area(sq. feet)

71.	PoolArea	Pool area(sq. feet)
72.	PoolQC	Pool quality8
73.	Fence	Fence quality
74.	MiscFeature	Miscellaneous feature
75.	MiscVal	$Value of miscellaneous feature
76.	MoSold	Month Sold
77.	YrSold	Year Sold
78.	SaleType	Type of sale
79.	SaleCondition	Condition of sale

Table 3.1:List of all attributes

5.PRE-PROCESSING TECHNIQUES

Preprocessing is the most critical part of model building. Understanding the root of your model i.e., the data. We have observed that the data provided to us was very less. This meant we need to tune the data in such a way that even such small amount of data helps in building a good model.

- Imputing Null Values: The first and the most prominent observation that we made from the dataset is the vast number of null values in the dataset. Here, eliminating a data point is not an ideal option because of the little data available to us. To further understand these NA values, we have studied all the columns and their significance of the presence of a NA value in them. Few columns exhibited that NA value is a significant level in them while others were simply missing data. The approach we applied for each column is different and completely derived from the meaning of that column. Then imputed it making the data point useful.

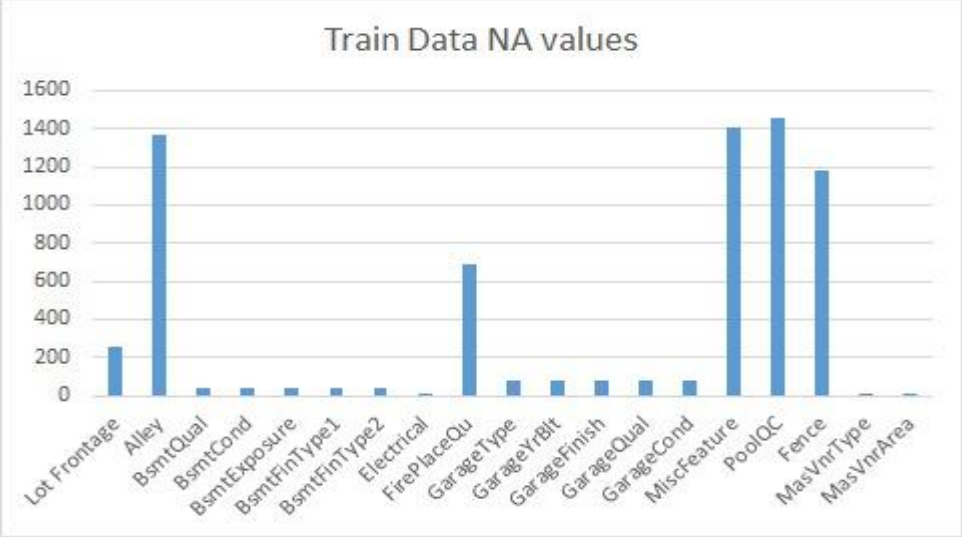


Fig 5.1:Train Data NA values

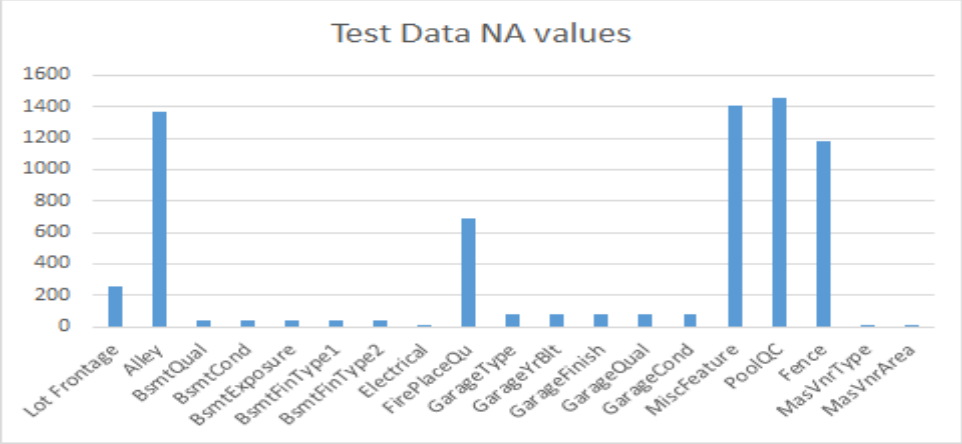


Fig 5.2:Test Data NA values

We observed that the columns without NA values in train dataset had NA values in test dataset. This meant simply handling these NAs from the perspective of train data was not enough. To build a stronger model, the data handling should be able to tackle the unknown test data.

However, we also kept in mind that it is always good to drop the columns with too many NA values. We accomplished it by checking the results after dropping each of these columns.

We have tabulated few columns to show how we handled the NA values:

#	Attribute	Conclusions
1.	MSZoning	Missing values. Should be handled , Most Freq. value =>(RL)
2.	Lot Frontage	Missing values. Should be handled , Average value =>(68)
3.	LotArea	Missing values. Should be handled , Average value =>(10516)
4.	Street	Missing values. Should be handled , Most Freq. value =>(Pave)
5.	Alley	Significant (there is no alley access)
6.	BsmtQual	Significant (there is no basement)
7.	BsmtCond	Significant (there is no basement)
8.	BsmtExposure	Significant (there is no basement)
9.	BsmtFinType1	Significant (there is no basement)
10.	BsmtFinType2	Significant (there is no basement)
11.	Electrical	Missing values. Should be handled , Most Freq. value =>(SBrKr)
12.	FirePlaceQu	Significant (there is no fireplace)
13.	GarageType	Significant (there is no garage)
14.	GarageYrBlt	Significant (there is no garage)
15.	GarageFinish	Significant (there is no garage)
16.	GarageQual	Significant (there is no garage)
17.	GarageCond	Significant (there is no garage)
18.	MiscFeature	Significant (there is no additional feature)

19.	PoolQC	Significant (there is no pool)
20.	Fence	Significant (there is no Fence)
21.	MasVnrType	Significant(no veneering done)
22.	MasVnrArea	fill with 0
23.	BsmtFinSF2	fill with 0
24.	BsmtUnfSF	fill with 0
25.	TotalBsmtSF	fill with 0
26.	BsmtFullBath	fill with 0
27.	BsmtHalfBath	fill with 0
28.	KitchenQual	Missing values. Should be handled , Most Freq. value =>(TA)
29.	Functional	Missing values. Should be handled , Most Freq. value =>(Typ)
30.	SaleType	Missing values. Should be handled , Most Freq. value=>(WD)

Table 5.1:List of all attributes where we handled NULL

- Removing Attributes:

  We identified a few columns to be constant columns i.e., most of the values were same. We then checked the correlation of these rows with the SalePrice. We found a few columns which had a very little correlation. But before going ahead and deleting these columns, we used trial and error method by deleting each column and checking the change in the RMSE value. We removed those columns which had a negligible impact on the RMSE values.

List of Such Columns
RoofMatl
Street-removed
Alley
Utilities
Heating
KitchenAbvGrd
WoodDesckSF
OpenPorchSF
EnclosePorchSF
SSN Porch
ScreenPorch
PoolArea
Condition2-removed
PoolQC
Fence-removed
MiscFeature-removed
CentralAir
BsmtHalfBath

- Adding Columns:

  We have combined the columns having similar significance and forming a new column preserving their contribution to the model. Further analysis was done on individual columns as well.
  In case of columns having information like area and number of columns we have added multiple columns like first floor area and ground floor area to make total area. Similarly we have modified Number of Bedrooms and bathrooms (Half and Full Baths). We also used correlation before adding such columns whether their contribution was similar. For example, combining columns which had close correlation and all having either positive or negative correlation makes sense than randomly adding just by their meaning.

- Columns like Areas and Numbers:

  To further visualize the data we plotted histograms and by this we wanted to understand the distribution of the feature values.
  Few of the features available had numerous levels and had to be handled in order to be a useful contribution to our prediction model.
  Some of them were:

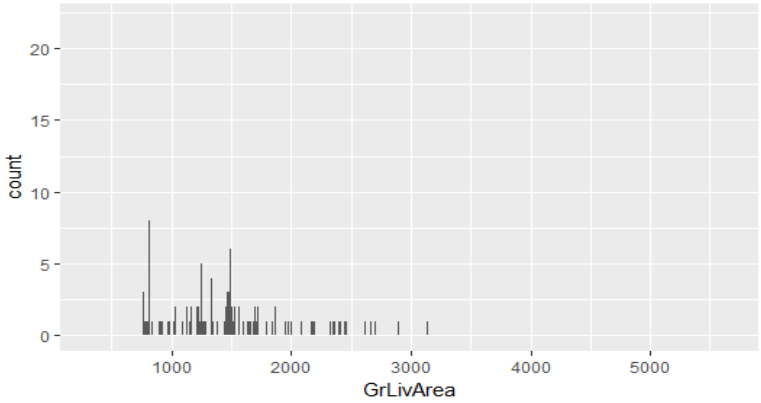


Fig 5.3:Histogram of GrLivArea

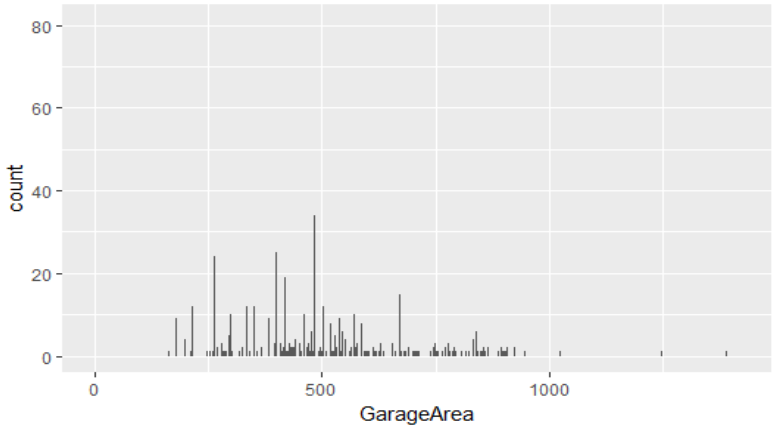


Fig 5.4:Histogram of GarageArea

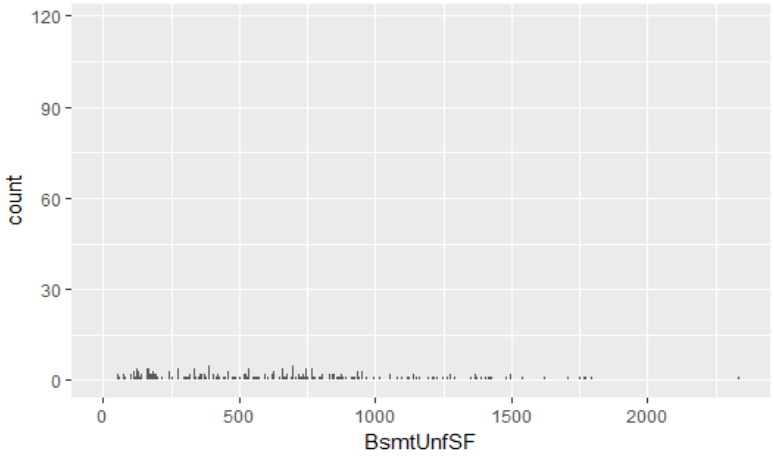


Fig 5.5:Histogram of BsmtUnfSF

Since this is a housing dataset most of the features emphasized on details like area and numbers. We can see from the above figure that the GrLivArea distribution does not appear ideal but however the Living Area can affect your sale price and need to be modified. For this, we had to come back to checking correlation with the SalePrice. There were features with values which had a high range like 0-215245.

We proceeded on the strategy that ideally their distribution should be normalized. For this we analyzed the ranges of such columns. Then we tried to understand number of data points lying between these ranges by visualizing them.

Going through all this, we considered that log value of these columns will help us normalize their distribution which prevents our data to be right or left skewed.

This was finally done on all the numerical attributes.

- In Case of Years:

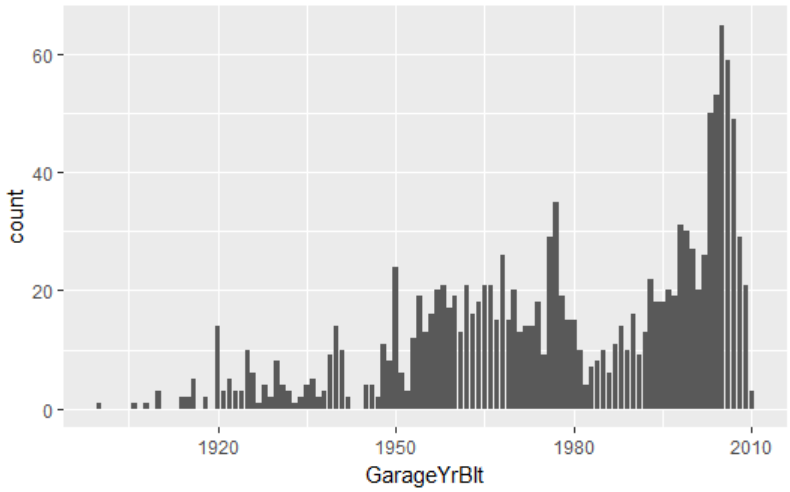


Fig 5.6:Histogram of GarageYrBlt

The approach for columns like area and number may not be suitable for columns such as 'year built' as they show case numerous levels. They are to be handled in a different manner. We approached this by changing them to bucketed columns. One such column was YearSold.

- Noisy Values:

  While going through data set, we found that the column GarageYr had the value 2207. This value clearly is invalid. We had gone through other columns just to make sure that they did not contain such values.
  Column – Garage year built- value: 2207
  We handled this by replacing this value by most frequent value - 2007. Also we checked with the data point (values like year built) whether replacing it with 2007 made sense.

- New columns:

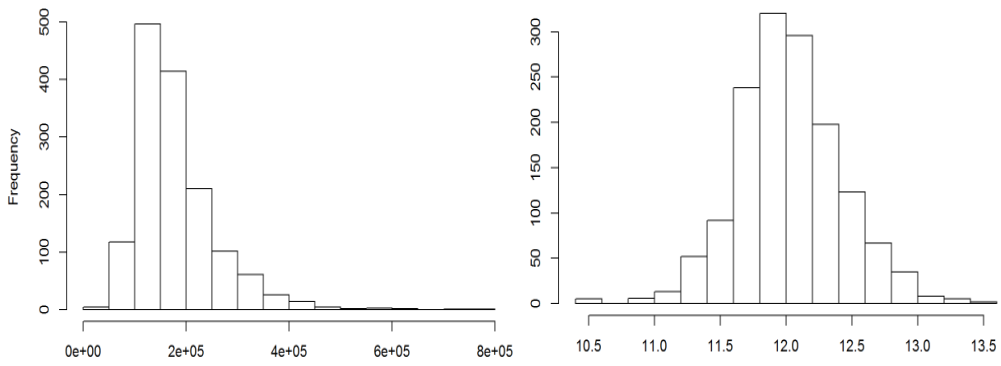
  Crafting new columns could help us utilize the hidden and indirect meaning from existing column.
  Such Columns were:
  - Reconstructed: if YrSold< YearRemod we set the value to "1" else the value to "0".
  - BuiltEquivalent – If YearSold < YrBuilt we set the value to "1" else the value to "0".

- Modification on Target Column

  On plotting the Target column SalePrice the distribution showed that it was more left skewed. This may lead to an unbalanced model. So as to avoid this we derived the value

of Log plus one on the column and then inversed the column of predicted values on exponent of column minus one.

We can see below the change is distribution before and after this modification:



Before

After

Fig 5.7:Histograms of Saleprice

6.PROPOSED SOLUTION AND METHODS

We started with the Linear regression as it is easiest to use and interpret. As this gave a good fit for the data, we assumed the data to be linear in nature. Then we processed with the other linear regression techniques like Lasso. The other algorithms that we implemented were Random Forest, Gradient Boosting which also gave satisfactory results.

Various techniques were available in MLlib of Spark which could help us in improving the model. We used all these techniques in our model.

They were:

- ParamGrid: Helps in model selection and hyperparameter tuning. It helps to reduce the strain on the user part by automatically helping in parameter tuning.

```
val paramGrid = new ParamGridBuilder()
  .addGrid(lr.regParam, Array(0.1, 0.01, 0.05, 1))
  .addGrid(lr.fitIntercept)
  .addGrid(lr.elasticNetParam, Array(0.0, 0.5, 1.0, 0.0001, 0.0005, 0.00075))
  .build()
```

- CrossValidation: Training and testing on the same data often results in overfitting. In order to avoid this we generally split the data into k smaller sets where it is trained on (k-1) datasets and tested on one dataset

```
val cv = new CrossValidator()
  .setEstimator(pipeline)
  .setEvaluator(new RegressionEvaluator)
  .setEstimatorParamMaps(paramGrid)
  .setNumFolds(5)
```

- Train validation Split: This is used to find out the error on the training data

```
val trainValidationSplit = new TrainValidationSplit()
  .setEstimator(lr)
  .setEvaluator(new RegressionEvaluator)
  .setEstimatorParamMaps(paramGrid)
  .setTrainRatio(0.8)
```

- Pipelines: This is used to implement various stages in the algorithm. The input dataframe is transformed as it passes through each stage. All the stages run in order in the pipeline

```
val pipelineIndex = new Pipeline().setStages(indexers)
val transformedIndex = pipelineIndex.fit(housingCastedIdx).transform(housingCastedIdx)
val columnDroppedIdx = categoricalAttributes.toList
val filteredDFIdx = transformedIndex.select(transformedIndex.columns .filter(colName => !columnDroppedIdx.contains(colName)) .map(colName => new Column(colName)): _*)
val colsIdx= filteredDFIdx.dtypes
val colnamesIdx = colsIdx.filter(x => x._1 != "SalePriceNew").filter(x =>x._1!= "Id").map(_._1)
```

- String indexer: This is used to convert the string data into numerical indices. This is done based on the label frequencies i.e. the most frequent value gets a zero.

```
val categoricalAttributes = housingCastedIdx.dtypes.filter(_._2 == "StringType") map (_._1)
val nominalAttributes = housingCastedIdx.dtypes.filter(_._2 == "IntegerType") map (_._1)
val indexers = categoricalAttributes.map( c => new StringIndexer().setInputCol(c).setOutputCol(s"${c}_idx"))
```

- Vector Assembler: Vector assembler helps to combine the given columns to a single column. It is helpful for combining distinctive features generated by transformers.

```
val features =new VectorAssembler().setInputCols(colnamesIdx).setOutputCol("features")
val output = features.transform(finalDFIdx)
val regModelData = output.select("label", "features")
```

We tried to include PCA in our implementation but it somehow affected our model adversely.

We followed the following strategy during the implementation and testing phase:

- We prepared a Preprocessed Data frame after implementing our planned preprocessing methodology.
- This was converted to Vector of features and label. We used String indexer and Vector assemblers on the dataset. Also, few algorithms required labelled point. So, we prepared out Data frame accordingly.

- After this we implemented the above mentioned four algorithms to draw conclusions for best algorithm.
- Then we used Pipelines to implement stages of our algorithms. For model selection (Hyper Parameter Tuning) we made use of ParamGrid Builder.
- In our implementation we then pushed our training dataset to Train Validation Split. This split our test dataset to train and test in the ratio of 80:20. From this we obtained the RMSE and R2 values. We selected the best sets of Parameters (2 to 3 best values) using the RMSE and r2.
- The above obtained set of parameters were used for our final model Building. Now we trained our model on the whole train dataset.
- Now we run our algorithm on Test Data Set to generate our Submission file for the competition.

7.EXPERIMENTAL RESULTS AND ANALYSIS

Linear Regression

Slno	ElasticNetparam	Fit Intercept	regParam	RMSE	R2
1.	0.5	True	1.0	0.3659	0.8123
2.	0.5	False	1.0	0.3148	0.8360
3.	0.00075	False	1.0	0.2561	0.8368
4.	0.0	False	0.05	0.18410	0.8716
5.	0.5	true	0.01	0.1148	0.8806
6.	0.0005	False	0.05	0.2096	0.8610

Table 7.1: Linear Regression Results

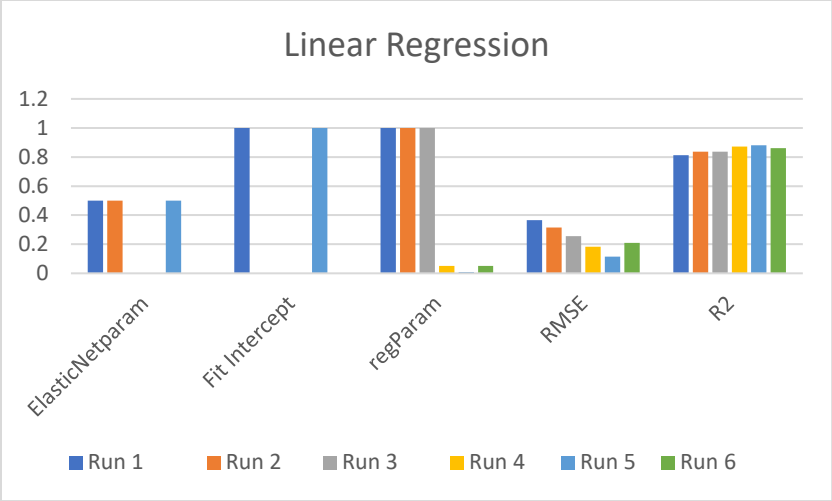


Fig 7.1:Linear Regression Results

By implementing the above tuned parameters on the testing data, we got lowest error for the parameters of the 5 th run, which is 0.1148

Lasso Regression

Slno	ElasticNetparam	Fit Intercept	regParam	RMSE	R2
1	1.0	true	0.01	0.1061	0.8966
2	1.0	True	0.1	0.1820	0.8812
3	1.0	True	0.05	0.1396	0.8896
4	1.0	True	0.01	0.107	0.80061

Table 7.2: Lasso Regression Results

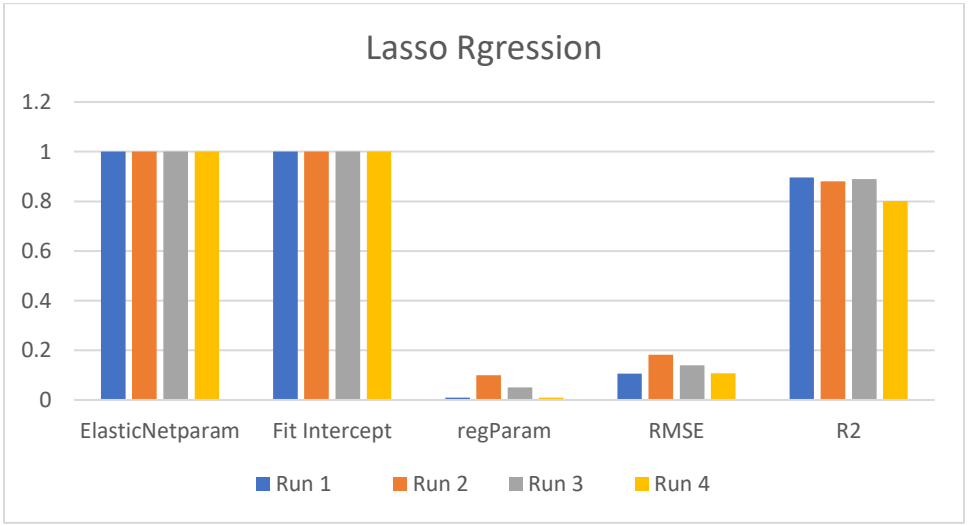


Fig 7.2:Lasso Regression Results

By implementing the above tuned parameters on the testing data, we got lowest error for the parameters of the 1st run, which is 0.1061

RandomForest Regression

Slno	maxdepth	numTrees	RMSE	R2
1	4	2	0.18637	0.7881
2	4	5	0.1729	0.7886
3	4	250	0.1479	0.7918
4	8	10	0.15600	0.8718
5	4	100	0.1483	0.8007
6	4	550	0.1482	0.8129

Table 7.3: RandomForest Regression Result

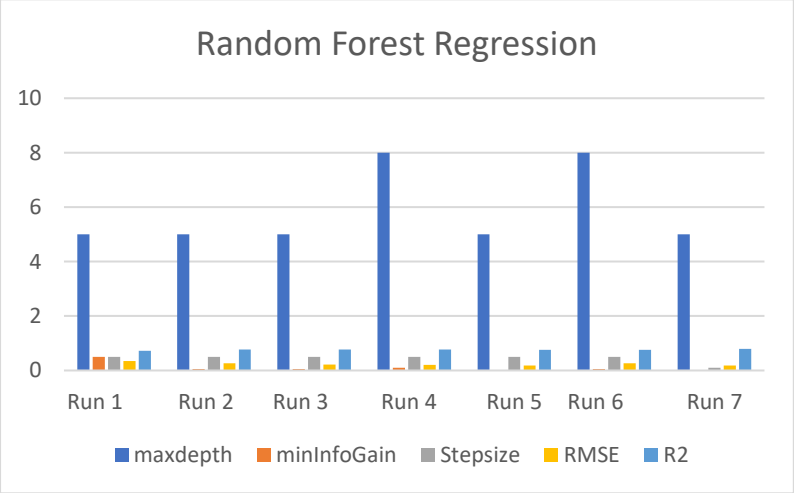


Fig7.3:LinearRegression Results

Gradient Boost Regression

Slno	maxdepth	minInfoGain	Stepsize	RMSE	R2
1	5	0.5	0.5	0.3488	0.7211
2	5	0.05	0.5	0.2588	0.7645
3	5	0.05	0.5	0.2223	0.7702
4	8	0.1	0.5	0.2087	0.7716
5	5	0.01	0.5	0.18713	0.7585
6	8	0.05	0.5	0.2631	0.7571
7	5	0.0	0.1	0.1828	0.7885

Table 7.4: Gradient Boosting Regression Result

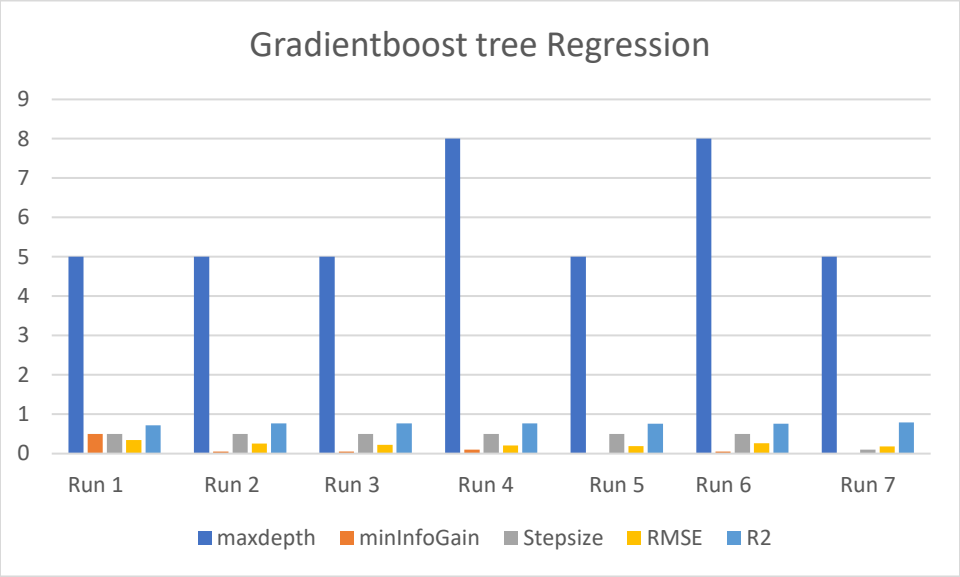


Fig 7.4:GradientBoost Regression Results

By implementing the above tuned parameters on the testing data, we got lowest error for the parameters of the 7th run, which is 0.1828

8.CONCLUSION

After running all our algorithms on different parameter grids. We obtained the following best values for each of them.

Models	Parameters	RMSE	R2	RMLSE(values obtained on submission file on Kaggle)
Linear Regression	Elastic Netparam:0.5 Fit Intercept:true regParam:0.01	0.1148	0.8806	0.1368
Lasso Regression	**Elastic Netparam:1.0 Fit Intercept:true regParam:0.01**	**0.1061**	**0.8966**	**0.13386**
Random Forest Regression	maxDepth:4 numTrees:250	0.1479	0.8416	0.20699
GradientBoost tree Regression	maxDepth:5 MinInfoGain:0.0 Stepsize::0.1	0.1828	0.8286	0.2303

Table 8.1: Summary of results

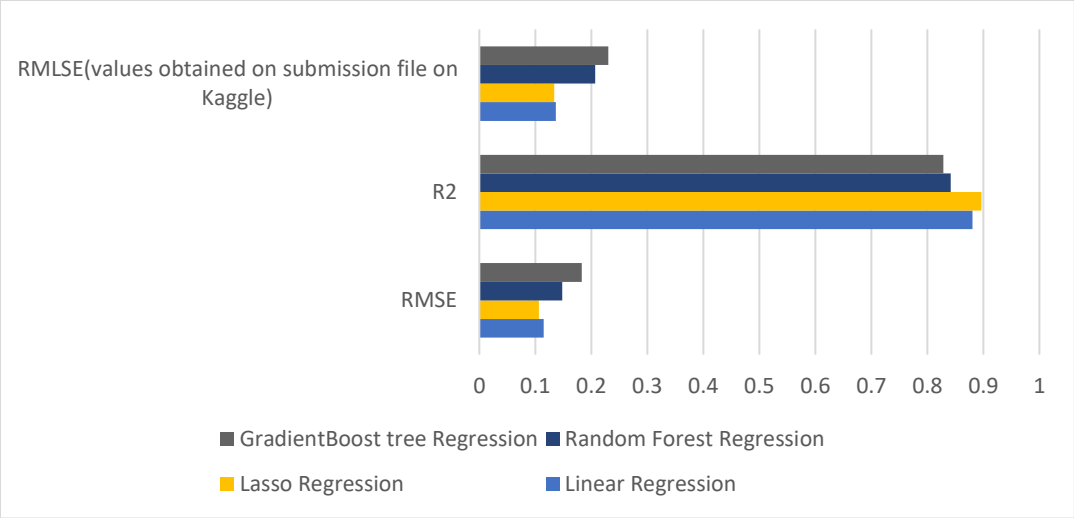


Fig 8.1:Comparison of All algorithms

The best model built so far was LASSO regression with the above mentioned paramgrid.

We achieved our best RMLSE value of 0.133 in the competition where the leading team obtained RMLSE value of 0.10.

9.FUTURE WORK

In the future, we plan to improve our model further by implementing a few ideas we have in mind

- On analyzing predicted SalePrice values, we found that there are certain outliers, which affects the model. And since, we got the best results with Linear methods and most of the Linear Regression methods are sensitive to outliers, identifying and handling the outliers would increase our model accuracy.

- New Columns: We could achieve adding new columns from a few attributes. However, we could concentrate on this dimension of analysis to add more meaningful columns.

10.CONTRIBUTION OF TEAM MEMBERS

Understanding the datasets	Teja, Sowmya, Mohana
Preprocessing data	Teja, Sowmya, Mohana
Status report	Teja, Mohana
Proposed solutions	Teja, Sowmya, Mohana
Linear Regression	Mohana
Lasso Regression	Sowmya
RandomForests, Gradient Boosting tree	Teja
Report	Teja, Sowmya, Mohana

11.REFERENCES

[1] “Communicating Scala Objects”, Bernard Sufrin

[2] “Scala Cookbook” ,Alvin Alexander

[3] https://spark.apache.org/docs/1.1.0/mllib-guide.html

[4] “Predicting Ames Housing Prices”, Kevin Wong

[5] https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data