Linear Regression versus Random Forest Regression for House Prices in California

Project Report

Jeremiah Hawkinson
ID: 006002668

December 2022

School of Computer Science
California State University San Bernardino

Professor Yan Zhang

Abstract

Linear Regression and Random Forest Regression are two well known machine learning algorithms. In this report they are both tested and compared on a medium data set of California house prices by district from a 1990 census to see which of the algorithms performs best in predicting a house's price. The data was scaled using standardization, null values were filled using the median of its own attribute, and unnecessary attributes were dropped to best suit our prediction algorithms. The base parameters of the sklearn library for python was used to complete these tests. It was discovered that Random Forest Regression is superior at predicting the price of a house in California.

1 Introduction

There are many machine learning algorithms available for us to use. In this paper we will be testing the effectiveness of Linear Regression and Random Forest Regression in predicting house prices of California as shown in a 1990 census, found from the database of Kaggle. They will be tested using the same data and compared to each other to determine which machine learning algorithm does the best job in predicting the prices of houses in California.

2 Machine Learning Algorithms

2.1 Linear Regression

Linear regression analysis is used to predict the value of one variable based on other variables. The variable you want to predict is called the dependent variable. The variable(s) you are using to predict the other variable's value is called the independent variable.

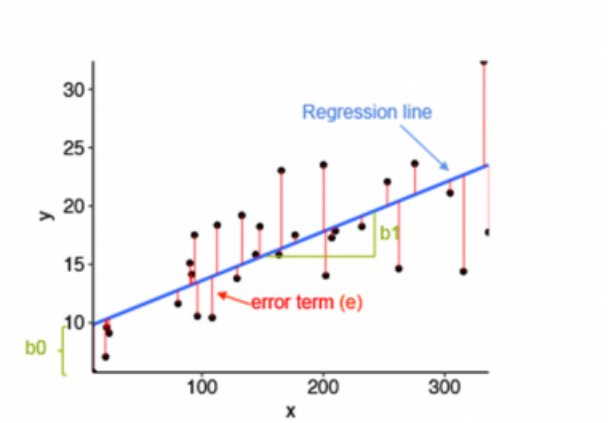

Figure 1: Example of Linear Regression.

Linear Regression estimates the coefficients of the linear equation, involving one or more independent variables to predict the value of the dependent variable. It fits a straight line or surface that minimizes the discrepancies between predicted and actual output values. A simple linear regression calculator uses a "least squares" method to discover the best-fit line for a set of paired data. You then estimate the value of x (dependent variable) from y (independent variable).

2.2 Random Forest Regression

Random Forest Regression is a type of classification that uses a large number of decision trees that work together as an ensemble. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees, then uses averaging to increase predictive accuracy and help control over-fitting of the data set.

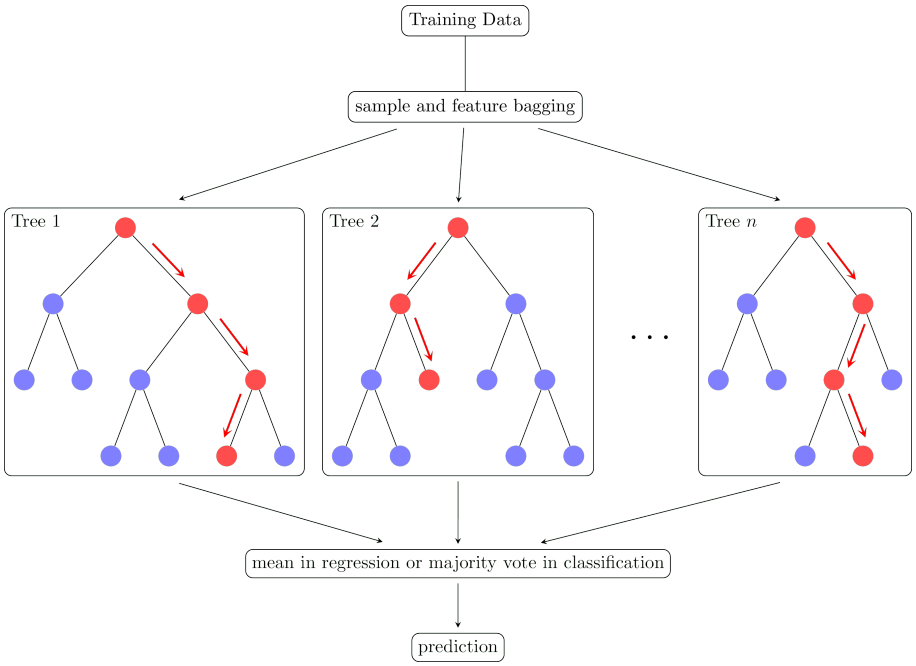

Figure 2: Example of a Random Forest decision trees.

3 Data Set and Experiment

3.1 Data Set: California Housing Prices

The data set includes the information for house prices for California districts derived from a 1990 census.

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

Figure 3: Header of the California House Prices Data Set

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

Figure 4: Information on the columns and data types of the data set.

There are 10 columns in the data set and 20640 entries for us to test. Small grievances such as the "ocean proximity" column being a categorical object instead of numerical will require us to decide on an approach method to prepare the full data set for testing. Another issue shown in Figure 4 shows that the column "total bedrooms" has 207 null values that also need to be addressed before testing.

3.2 Correlation Matrix and Feature Engineering

Using the correlation matrix it is possible to determine what attributes have the largest affinity with the attribute that is trying to be predicted. In this case "Median House Value" is the attribute which is being predicted.

```
#checking correlation between each attribute and median_house_value
corr_matrix = cali_housing.corr()
corr_matrix.median_house_value.sort_values(ascending = False)
```

```
median_house_value    1.000000
median_income         0.688075
total_rooms           0.134153
housing_median_age    0.105623
households            0.065843
total_bedrooms        0.049686
population           -0.024650
longitude            -0.045967
latitude             -0.144160
Name: median_house_value, dtype: float64
```

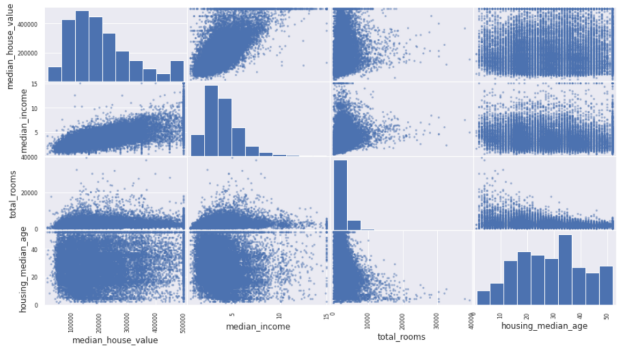

Figure 5: Histogram of the top four correlated attributes.

Seeing how "total rooms" is valued so highly in correlation to the value of a house, a step further can be taken to possibly increase the rate of our prediction by adding some new feature attributes. Using the attributes "total rooms" and "households" we can find out the number of rooms per household in a district to help better represent our data. Another attribute that was created was "people per household" using the "population" and "household attributes. These added attributes can be seen in the correlation matrix of Figure 6

```
#seeing how our implemented features correlate to median_house_value
corr_matrix["median_house_value"].sort_values(ascending = False)
```

```
median_house_value       1.000000
median_income            0.688075
rooms_per_household      0.151948
total_rooms              0.134153
housing_median_age       0.105623
households               0.065843
total_bedrooms           0.049686
people_per_householed   -0.023737
population              -0.024650
longitude               -0.045967
latitude                -0.144160
Name: median_house_value, dtype: float64
```

Figure 6: New attribute "rooms per household" is third in correlation.

3.3 Data Preprocessing

Moving back to the small grievances stated in Section 3.1. It was necessary to fix the null values within attribute "total bedrooms". This was done by using the sklearn simple imputing function of python to get the median of each attribute and filled any null values within the data set using them. The categorical value of "ocean proximity" was dropped from the data set as it would contribute little to overall prediction of house values and could not be used as a non-numerical

value.

The range of the data in each value varies greatly in this data set so it is necessary to scale it. The data was scaled using standardization, and the attributes were split into "x" and "y" to prepare it for our machine learning algorithms. For our Linear Regression algorithm the "median household value" was dropped and put into "y" while the other attributes were called into "x" to fuel our prediction. The last instance of data preprocessing was splitting our data set into a train and test set using the sklearn library. There is a test set of 20 percent, numbering 4,128 instances, and a training set of 80 percent, numbering 16,512 instances.

```
house_data.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	rooms_per_household	people_per_household
0	-1.327835	1.052548	0.982143	-0.804819	-0.972476	-0.974429	-0.977033	2.344766	2.129631	0.628559	-0.049597
1	-1.322844	1.043185	-0.607019	2.045890	1.357143	0.861439	1.669961	2.332238	1.314156	0.327041	-0.092512
2	-1.332827	1.038503	1.856182	-0.535746	-0.827024	-0.820777	-0.843637	1.782699	1.258693	1.155620	-0.025843
3	-1.337818	1.038503	1.856182	-0.624215	-0.719723	-0.766028	-0.733781	0.932968	1.165100	0.156966	-0.050329
4	-1.337818	1.038503	1.856182	-0.462404	-0.612423	-0.759847	-0.629157	-0.012881	1.172900	0.344711	-0.085616

```
x = house_data.drop(["median_house_value"], axis = 1).values
y = house_data['median_house_value'].values
```

```
# import the train_test_split
from sklearn.model_selection import train_test_split

# split the data into train and test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.20, random_state = 20)
```

Figure 7: Scaled data and training/test sets.

3.4 Algorithm Parameters

The parameters used for the machine learning algorithms in our experiment were both the base parameters given by the imported library sklearn. A random state of 20 was given as a parameter to each for easier replication of the experiment. Random Forest Regression used n = 100 for its estimation.

4 Evaluation and Comparison

4.1 Linear Regression

The Linear Regression test score achieved by our machine algorithm managed to reach 62.7 percent using our test size and reached 63.9 percent with our training set as seen in Figure 8. The coefficients from the training and test sets are shown and a comparison of the predicted values versus actual values are shown in Figure 9 The root mean squared error came out to 0.6248 which

is acceptable given the complexity of the data set and our scores for Linear Regression.

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(x_train, y_train)
predict_lin_reg = lin_reg.predict(x_test)
```

```
lin_reg.score(x_train, y_train)
```

```
0.6387103045467208
```

```
lin_reg.score(x_test, y_test)
```

```
0.6273690058384889
```

	coeficients
longitude	-0.751766
latitude	-0.805346
housing_median_age	0.124663
total_rooms	-0.133770
total_bedrooms	0.236905
population	-0.454423
households	0.382632
median_income	0.635594
median_house_value	0.035585
rooms_per_household	0.012953

	y_test	y_pred
0	-0.773495	-0.694754
1	0.739597	0.664016
2	-0.654771	-0.343350
3	0.511680	0.361657
4	0.028981	0.199198

Figure 8: Scores for the training and test sets. Linear Regression coefficients for predicted results and actual results.

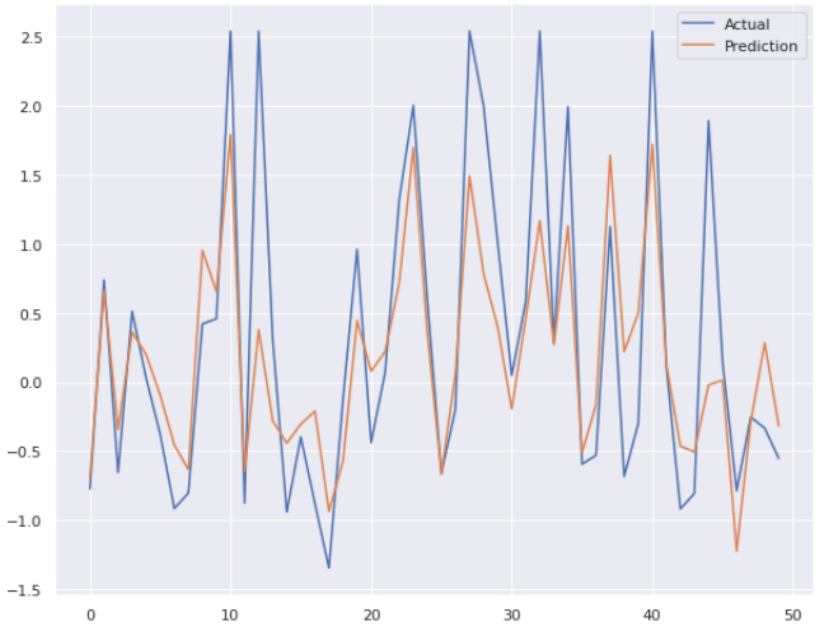

Figure 9: Actual values versus predicted values.

4.2 Random Forest Regression

The Random Forest Regression algorithm was ran using base sklearn parameters with n=100 for estimators and random state=20. This gave a large improvement over the Linear Regression scores with an 81.5 percent using the test set and a 97.3 percent using the training set. The root mean squared error was also minimally 0.4396 showing considerable results.

Figure 10: Scores for the Random Forest Regression algorithm.

4.3 Comparison

Random Forest Regression performed considerably better than Linear Regression in predicting the median house value of houses in California using our data set. Obtaining accurate results 19 percent more often, and at a lower margin of error.

	R^2 Score	Root Mean Squared Error
Random Forest	0.815492	0.439649
Linear Regression	0.627369	0.624796

5 Conclusion

In conclusion Random Forest Regression is significantly better at predicting the house prices of California using this data set than Linear Regression. Random Forest Regression had a 81.5 percent chance of correctly predicting while Linear Regression achieved a 62.7 percent chance. Both of these test are sufficient in attempting to predict something as complex as the housing market of California but Random Forest Regression does it more effectively.