Midterm: Working With Large Data Sets and Machine Learning

Team 4: Emily Strong and Raksha Kaverappa

Code Repository: <https://github.com/erstrong/INFO-7390-ADS-Fall-17-TeamNo.4/tree/master/Mid-Term>

**Part 1: Data Ingestion, EDA, Wrangling**

AWS Bucket: <https://s3.console.aws.amazon.com/s3/buckets/zillowdata-esrkoutput>

EDA Notebook: <https://github.com/erstrong/INFO-7390-ADS-Fall-17-TeamNo.4/blob/master/Mid-Term/Part%201%20Exploratory%20Data%20Analysis.ipynb>

**Design and Implementation:**

Our first step in data ingestion was to read the data for properties and transactions from both years into data frames. We then used right joins to match properties to transactions for each year, thus removing from the set any properties that did not sell in the given year. For the data cleansing, removing the properties that didn't sell before performing calculations for missing values could affect the calculations, however the remaining sets are so large (2016 n = 90275, 2017 n = 77613) we decided this was unlikely and properties that sold are more likely to resemble other properties that sold.

**Data Cleansing Analysis:**

Our first step in data cleansing was to determine the percent missing data for each feature. The following features have more than 80% missing data and cannot be calculated so we are dropping them:

* 'architecturalstyletypeid'
* 'basementsqft'
* 'buildingclasstypeid'
* 'decktypeid'
* 'threequarterbathnbr'
* 'finishedfloor1squarefeet'
* 'finishedsquarefeet6'
* 'finishedsquarefeet13'
* 'finishedsquarefeet15'
* 'finishedsquarefeet50'
* 'poolsizesum'
* 'storytypeid'
* 'typeconstructiontypeid'
* 'yardbuildingsqft17'
* 'yardbuildingsqft26'

Since in part 4 we need to use Latitude and Longitude to calculate distances, we chose to drop any rows with NaN in those columns. After doing this, the following features did not have any missing data and were not eliminated for other reasons: bedroomcnt, fips, parcelid (key), propertylandusetypeid, assessmentyear, taxamount, taxvaluedollarcnt, transactiondate.

The following features represent physical attributes of the properties and have skewed data because of the real-world frequencies and differences between condos/apartments, small houses, large houses, etc. For these we have substituted missing values with the median:

* 'calculatedbathnbr'
* 'calculatedfinishedsquarefeet'
* 'numberofstories'
* 'yearbuilt'
* 'taxvaluedollarcnt'
* 'structuretaxvaluedollarcnt'
* 'landtaxvaluedollarcnt'
* 'taxamount'

There were several properties that had outliers for calculatedfinishedsquarefeet with values under 100 sq ft. In looking at these properties, their values for other features were inconsistent and often outliers as well, such as 20 bathrooms for a 66 sq ft house. We thus dropped these properties from the set.

Our analyses for the remaining features are:

|  |  |
| --- | --- |
| **Feature** | **Decision** |
| 'airconditioningtypeid' | Per the 2005 study cited in this article most homes in the Los Angeles area do not have air conditioning so we are replacing missing values with the value for "None": <http://articles.latimes.com/2006/jul/28/business/fi-air28> |
| 'bathroomcnt' | Per the data dictionary this is redundant with calculatedbathnbr and the two columns are highly correlated. This column has no missing values but it does have 0s which we assume is impossible, and the 0s match to NaNs in the other column. We are dropping this one. |
| 'buildingqualitytypeid' | We are replacing missing values with the mean. |
| 'finishedsquarefeet12' | This has the same definition as calculatedfinishedsquarefeet, the two are highly correlated, and this column has more missing data so we are dropping it. |
| 'fireplacecnt' | There is more than 80% missing data, but there are no 0s. We assume a NaN is a 0. |
| 'fireplaceflag' | There is more than 80% missing data and count does not match that for 'fireplacecnt'. Recalculating based on 'fireplacecnt'. |
| 'fullbathcnt' | There is more than 80% missing data and the column is correlated with calculatedbathnbr so we are dropping it. |
| 'garagecarcnt' | There are no 0s so we are replacing missing values with 0. |
| 'garagetotalsqft' | For properties with a garagarcarcnt > 0 we are replacing missing values and 0s with the median. For properties with a garagecarcnt=0 we are setting this value to 0. |
| 'hashottuborspa' | Though there is more than 80% missing data, there are no False values, and the count for True matches pool type 2 + pool type 10. We are filling missing values with False. |
| 'heatingorsystemtypeid' | This is categorical so we are replacing the missing values with the mode. |
| 'lotsizesquarefeet' | Data is skewed so we are replacing missing values with the median. There are two outlier values (3589145 and 6971010) that repeat suggesting these numbers are automatically generated by some sort of data entry error. Replacing these two values with the median. |
| 'poolcnt' | There are no 0s, and the count is slightly over the sum of counts for pool types 2 and 7. We are filling missing values with 0. |
| 'pooltypeid10' | This is a one-hot of pool type. We are combining the pool types into a single column for our exploratory data analysis, calculated from poolcnt and hashottuborspa. |
| 'pooltypeid2' | This is a one-hot of pool type. We are combining the pool types into a single column for our exploratory data analysis, calculated from poolcnt and hashottuborspa. |
| 'pooltypeid7' | This is a one-hot of pool type. We are combining the pool types into a single column for our exploratory data analysis, calculated from poolcnt and hashottuborspa. |
| 'propertycountylandusecode' | Since zoning is done by individual counties, these descriptions would not generalize. There are also too many unique values for this to be useful as a categorical variable. We are dropping this column. |
| 'propertyzoningdesc' | Since zoning is done by individual counties, these descriptions would not generalize. There are also too many unique values for this to be useful as a categorical variable. We are dropping this column. |
| 'rawcensustractandblock' | We are converting this to a "calculated" value to get rid of meaningless trailing digits. There are no missing values after the lat/lon NaNs are dropped. |
| 'censustractandblock' | Some of the values are impossible and this is redundant with 'rawcensustractandblock'. We are dropping this column. |
| 'regionidcounty' | This is redundant with the FIPS codes. We are dropping this column. |
| 'regionidcity' | Per the data dictionary not all properties will have a value in this column. We are dropping this column. |
| 'regionidzip' | The values in this column are incorrect/impossible. We are recalculating as a new column 'zipcode' and dropping this one. |
| 'regionidneighborhood' | There are too many unique values for this to be useful as a categorical variable and the values are Zillow's internal coding, they do not have real-world meaning that can be matched to other data sets. We are dropping this column. |
| 'roomcnt' | There must be at least as many rooms as there are bathrooms and bedrooms. We are calculating missing values as the sum of those columns. |
| 'unitcnt' | Data is skewed so we are replacing missing values with the median. Outliers appear to be the result of user error – the properties are condos and planned developments based on the propertylandusetypeid, and the unitcnt appears to be for the entire development whereas all other data reflects individual units. We are changing outliers to the median. |
| 'taxdelinquencyflag' | There are only 'Y' values, so though there are more than 80% missing data, we assume those are all 'N' and are filling missing values with that. Confirmed that 'Y' count matches the count for 'taxdelinquencyyear' |
| 'taxdelinquencyyear' | More than 80% have NaN but the count matches that for 'taxdelinquencyflag', so these NaNs are accurate – these properties have never had a value for this column. Due to the small size of the data we are dropping this column. |

**Part 2: Build A Prediction Model**

Model Notebooks: <https://github.com/erstrong/INFO-7390-ADS-Fall-17-TeamNo.4/tree/master/Mid-Term/Part2-PredictionModels>

For each of our prediction models we split the data into 80% train, 20% test.

**Multiple Linear Regression Design:**

We took five approaches to multiple linear regression. First we simply ran a regression with all of the numerical data and one-hots for Boolean and categorical data to get baseline RMSE and MAE results to which we could compare other models. Then we used a Lasso method to select features, which gave marginal improvement in the MAE but had a worse RMSE. Then we tried a recursive feature elimination, which gave worse results in both metrics.

Unimpressed by the automated results we were getting, we did a manual backward selection using MAE for the test data as the metric for deciding whether to keep or remove a variable, which finally gave us improvements in both metrics over the baseline. Since the order in which we eliminated variables could have confounded the results and some of the changes in MAE were very small relative to the value, we finally did a forward selection using only those variables chosen through backward selection and adding them in reverse to the order in which we had removed them. This gave us the best results overall, though there was only a 4% improvement in RMSE and a 1.3% improvement in MAE over the baseline.

In comparing the RMSE and MAE for the train versus test sets, there is little difference in the values suggesting the model isn't overfitting the data, the lack of improvements are simply because the variables do not have linear relationships with the logerror. This is consistent with the fact that none of the variables had a strong correlation with logerror during our EDA.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method** | **RMSE Train** | **RMSE Test** | **MAE Train** | **MAE Test** |
| Basic multiple linear regression | 0.16360 | 0.17031 | 0.069055 | 0.070423 |
| Lasso | 0.16386 | 0.17111 | 0.068954 | 0.070343 |
| Recursive Feature Elimination | 0.16396 | 0.17132 | 0.069239 | 0.070646 |
| Backward Selection | 0.16575 | 0.16336 | 0.069117 | 0.069596 |
| Forward Selection After Backward | 0.16580 | 0.16339 | 0.069027 | 0.069504 |

**Random Forests Design:**

We took six approaches to using random forests, but unfortunately each of these performed worse than the linear regression baseline. In our first attempt, we used the default settings. In our second we increased the number of trees generated to 20. In the third we increased it to 100. In our fourth we set the maximum number of features per tree to 20. In our fifth we increased the number of trees generated to 200. This trial gave the best performance overall among the random forest attempts based on MAE, but it performed worse than any of the linear regressions.

In our sixth attempt, we used recursive feature elimination with the number of trees generated set to 65, and the RFE set to select the 20 most important features. For our seventh attempt, we used only those 20 features, removing all other feature columns, and generated 100 trees. This gave us the best RMSE of the random forest implementations and the second lowest MAE, and the RMSE was better than the linear regression baseline but it did not perform better than the manual feature selection approaches.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameters** | **RMSE Train** | **RMSE Test** | **MAE Train** | **MAE Test** |
| Defaults | 0.075247 | 0.18177 | 0.03447 | 0.084781 |
| n\_estimators=20 | 0.068879 | 0.17712 | 0.031854 | 0.080630 |
| n\_estimators=100 | 0.062874 | 0.17355 | 0.028374 | 0.075758 |
| n\_estimators=100, max\_features=20 | 0.062310 | 0.17233 | 0.028146 | 0.075140 |
| n\_estimators=200, max\_features=20 | 0.061535 | 0.17222 | 0.027465 | 0.074336 |
| RFE with n\_estimators=65, max\_features=20 | 0.063359 | 0.17403 | 0.029016 | 0.076600 |
| Rank 1 features, n\_estimators=100 | 0.063219 | 0.16730 | 0.028663 | 0.074580 |

**Neural Networks Design:**

For our Neural Network model, we tested with 5 different approaches. For all the approaches implemented, we assigned our activation function to be ‘relu’ and our solver to be ‘adam. ’In our first approach, we had 3 hidden layers having 100 neurons in the first layer, 50 in the second and 20 in the third with learning a rate as constant. In our second approach, we changed the three hidden input layers to (500,250,100) and changed the learning rate to adaptive. For our third approach, we increased our hidden layers to (1000,750,500) and retained the parameters we had for the previous approach.

For our fourth approach, we increased the hidden layers to (10000,5000,2000). We got the lowest MAE for the fifth approach which was the best model with Neural Network. But when we compared it to our Linear regression model, we observed that the linear regression model with manual feature selection performed slightly better that Neural Network model.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameters** | **RMSE Train** | **RMSE Test** | **MAE Train** | **MAE Test** |
| [100,50,20]  Learning rate: Constant | 0.166314 | 0.1718 | 0.072642437057 | 0.0720487971513 |
| [500,250,100]  Learning rate: Adaptive | 0.173828021605625 | 0.175893991649261 | 0.0710263205695 | 0.0719059421754 |
| [1000,750,500]  Learning rate: Adaptive | 0.166025771287562 | 0.160106218334866 | 0.0692888577424 | 0.0733719476469 |
| [10000, 5000, 2000]  Learning rate: Adaptive | Memory error | 0.16631 | Memory error | 0.069980 |

**Analysis:**

Among the models that we created, the multiple linear regression generated by backward selection followed by forward selection is the best fit for the data based on mean absolute error. However, the fact that there were no significant differences between the models we developed suggests that none of these are appropriate models for the data. One of the commonalities among the models was that they tended to select location features. This suggests that spatial analysis might be more appropriate for this data set, possibly revealing clustering of log errors in regions with either high or low density of transactions. A number of the location features would make this particularly appropriate for GIS analysis since shapefiles for census tracts, zipcodes, etc are publicly available. Another possibility is that the data that will help improve the log error is external to Zillow. Since they readily have available to them all of the data in this set, they likely have already optimized the Zestimate as much as they can based on these features. Neighborhood demographics (racial and ethnic composition, income and education levels, etc) could reveal patterns in the log error and could be matched to groups of properties by census block (the smallest unit in which census data is aggregated).

**Part 3: Model Deployment**

API Notebook: https://github.com/erstrong/INFO-7390-ADS-Fall-17-TeamNo.4/blob/master/Mid-Term/Part3-API/Azure.ipynb

**Design and Implementation:**

The Multiple Linear regression model provided the lowest errors out of the three models we created. We selected the columns to best predict the log error, applied linear regression to obtain a score of 0.13 in the final output. Our MAE and RMSE errors were same as that was observed in the python code. We deployed the model using azure web service command. We consumed the API in our python notebook and observed the resulting log error for the input provided.

**Part 4: Geospatial Search**

API Notebook: https://github.com/erstrong/INFO-7390-ADS-Fall-17-TeamNo.4/blob/master/Mid-Term/Part4-GeospatialSearch/Part%204%20Geospatial%20Search.ipynb

**Design and Implementation:**

We exported the parcelid, latitude and longitude columns from our cleaned data set and used that to create a single-table MySQL database. We modified the query from Alexander Rubin's "Geo (proximity) Search with MySQL" for implementing the Haversine formula to only use the provided latitude and longitude, not a maximum distance.

We confirmed that the query worked in our MySQL database, and then built a simple Flask app to create an API. We created a python notebook to consume the API, converting the JSON results to a data frame that can be printed as a table and mapping the results. We managed to deploy our MySQL database on AWS but did not successfully the deploy the Flask app.