UTSAV PATEL

Indiana University-Bloomington
2661, E 7th Street, Apt. E, Bloomington,IN-47408
utpatel@iu.edu | utsavpatel0307@gmail.com | 312- 774-3172

Zillow Project Report:

- **Exploratory Data Analysis from pandas_profiling library:**

Overview

Dataset info

Number of variables	24
Number of observations	11588
Total Missing (%)	5.3%
Total size in memory	2.1 MiB
Average record size in memory	192.0 B

Variables types

Numeric	20
Categorical	2
Boolean	0
Date	0
Text (Unique)	0
Rejected	2
Unsupported	0

- `BGMedRent` has 2631 / 22.7% missing values Missing
- `BGMedYearBuilt` has 247 / 2.1% missing values Missing
- `BGPctVacant` has 4132 / 35.7% zeros Zeros
- `GarageSquareFeet` has 2841 / 24.5% missing values Missing
- `TransDate` has a high cardinality: 128 distinct values Warning
- `Usecode` has constant value 9 Rejected
- `ViewType` has 8956 / 77.3% missing values Missing
- `ZoneCodeCounty` has a high cardinality: 178 distinct values Warning
- `censusblockgroup` has constant value 530000000000 Rejected

Data Preprocessing :

- **Checking Missing Values:**

Train Missing Values :

	Missing Values	% of Total Values
ViewType	8956	77.3
GarageSquareFeet	2841	24.5
BGMedRent	2631	22.7
BGMedYearBuilt	247	2.1
BGMedHomeValue	6	0.1

Test Missing Values:

	Missing Values	% of Total Values
SaleDollarCnt	4402	100.0
ViewType	3404	77.3
GarageSquareFeet	1138	25.9
BGMedRent	963	21.9
BGMedYearBuilt	62	1.4
BGMedHomeValue	7	0.2

Note here that in above test missing values, SaleDollarCnt will obviously be missing in the test file, so no interpretation should be done from it.

- ## Check categorical variables:

```
dftest_X.dtypes=="object"
```

PropertyID	False
TransDate	True
censusblockgroup	False
ZoneCodeCounty	True
Usecode	False
BedroomCnt	False
BathroomCnt	False
FinishedSquareFeet	False
GarageSquareFeet	False
LotSizeSquareFeet	False
StoryCnt	False
BuiltYear	False
ViewType	False
Latitude	False
Longitude	False
BGMedHomeValue	False
BGMedRent	False
BGMedYearBuilt	False
BGPctOwn	False
BGPctVacant	False
BGMedIncome	False
BGPctKids	False
BGMedAge	False

Thus, there are only 2 categorical variables: TransDate and ZoneCodeCounty

- ## Treating categorical Variables

1. **TransDate:**
   How I treated TransDate and Why I dropped TransDate Column:
   First approach that came to my mind was to create bins based on months and that lead to below counts:

Train File:

TransDate	SaleDollarCnt	count
2015-04-30	614949.954518	1671
2015-05-31	614010.690454	2116
2015-06-30	637301.737849	1934
2015-07-31	601015.601758	2275
2015-08-31	613819.837924	1888
2015-09-30	602209.944249	1704

Test File:

TransDate	SaleDollarCnt	count
2015-10-31	NaN	1852
2015-11-30	NaN	1034
2015-12-31	NaN	1450
2016-01-31	NaN	66

While in test we had different months. Also, there was no specific pattern that we could infer from above figure. So, I thought of creating new features based on day of transaction (Sunday, Monday, ..). But, again no useful pattern was found in that approach. So, I decided to **drop TransDate** Column.

2. **ZoneCodeCounty :**

Clearly One Hot Encoding was best approach for this variable.

ZoneCodeCounty Unique Values in **Train : 178**
ZoneCodeCounty Unique Values in **Test : 143**

After One Hot Encoding of Train and Test file:

Total Train Columns: 199
Total Test Columns : 164

In order to keep train and test files in same dimensions, I fetched columns which were in testfile and not in trainfile and created new columns in trainfile. After that I dropped train columns which were not in test file. Now, at the end of this step, **we have 164 rows in both train and test**.

3. **"Censusblockgroup" and "Usecode" – Dropped :**

Censusblockgroup and Usecode were same in both the files(train and test). Thus, I **dropped it**. Now there are 162 columns in train and test.

- ## Creating new Columns:

New columns were created from columns which had missing values. If there is missing values in a column, the corresponding data will have 1 if data is missing and 0 if data is not missing.

```
dftrain_X['Missing ViewType']=(np.isfinite(dftrain_X['ViewType'])==False)
dftrain_X['Missing ViewType']= dftrain_X['Missing ViewType'].astype(int)
dftrain_X['Missing GarageSquareFeet']=(np.isfinite(dftrain_X['GarageSquareFeet'])==False)
dftrain_X['Missing GarageSquareFeet']= dftrain_X['Missing GarageSquareFeet'].astype(int)
dftrain_X['Missing BGMedYearBuilt']=(np.isfinite(dftrain_X['BGMedYearBuilt'])==False)
dftrain_X['Missing BGMedYearBuilt']= dftrain_X['Missing BGMedYearBuilt'].astype(int)
dftrain_X['Missing BGMedRent']=(np.isfinite(dftrain_X['BGMedRent'])==False)
dftrain_X['Missing BGMedRent']= dftrain_X['Missing BGMedRent'].astype(int)
dftrain_X['Missing BGMedHomeValue']=(np.isfinite(dftrain_X['BGMedHomeValue'])==False)
dftrain_X['Missing BGMedHomeValue']= dftrain_X['Missing BGMedHomeValue'].astype(int)
print("Shape of train is : ",dftrain_X.shape)
dftrain_X.head()
```

Missing ViewType	Missing GarageSquareFeet	Missing BGMedYearBuilt	Missing BGMedRent	Missing BGMedHomeValue
1	0	0	0	0
0	0	0	0	0
1	0	0	0	0
0	1	0	0	0
0	1	0	0	0

- ## Scaling the Data :

  Used sklearn.preprocessing.StandardScaler() to scale the dataframe.

- ## Dealing with Missing Values :

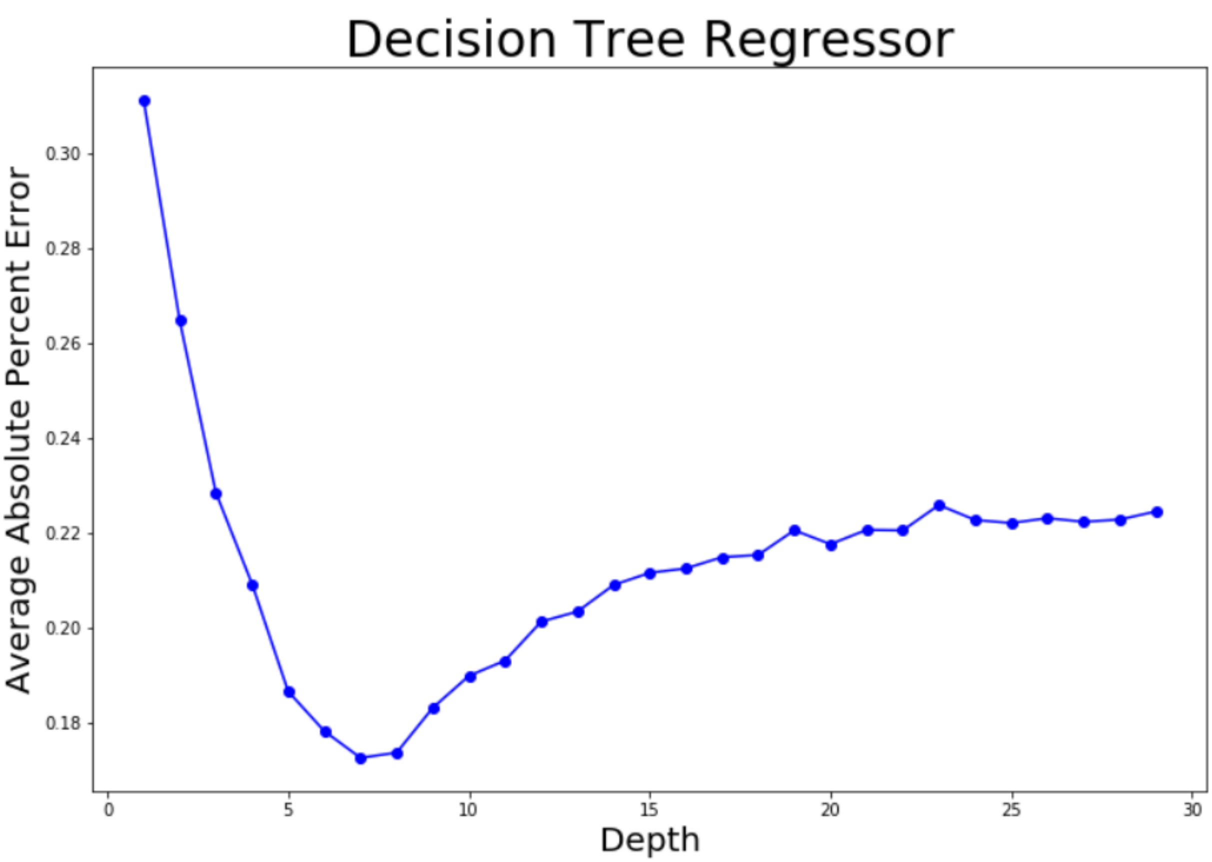
  Many ways were tried for dealing with missing values. Those techniques are **Forward Filling, Backward Filling, Mode imputation, Mean Imputation, KNN imputation , MICE imputation**. Out of all these, the most beneficial technique was KNN imputation.

  ### KNN Imputation:

  Used KNN from fancyimpute library to impute Missing values from nearest matching 3 observations.

- # Machine Learning Models:

➢ **Depth=7 was best for all ML Tree based Algorithms.**



After Performing all kinds of ML Algorithms for Regression, I came to conclusion that mostly tree based learning algorithms were performing a lot better than other. An another intuition was all tree based learning algorithms uses weak trees and the optimal depth at which all the algorithms were performing better was at depth = 7.

What Machine Learning Models were tried and Why I Rejected them :

Ordinary Least Squares Regression: Rejected due to Underfitting
KNN : Rejected due to Overfitting
Support Vector Machine : Rejected due to Low Performance
Decision Trees: Rejected due to Low Performance
Random Forest: Used
Gradient Booster: Used
AdaBoost : Rejected as XGBoost was performing a way better than this.
XGBoost: Used
LightGBM: Used
Perceptron : Rejected due to Overfitting
Neural Network : Rejected due to Overfitting

What approach was used for reducing Overfitting and make Robust Model:

K Fold Cross Validation technique was used for reducing the model overfitting and dealing with varied data . **5 fold Cross Validation was used**.

Model Building :

```
model1 = LGBMRegressor(boosting_type='dart', max_depth=8, learning_rate=0.13, n_estimators=600,objective='regression',metric='mape
model2= LGBMRegressor(boosting_type='dart', max_depth=9, learning_rate=0.19, n_estimators=400,objective='regression',metric='mape
model3 = LGBMRegressor(boosting_type='dart', max_depth=8, learning_rate=0.24, n_estimators=350,objective='regression',metric='mape
model4 = LGBMRegressor(boosting_type='dart',num_iterations=900 ,max_depth=7, learning_rate=0.09, n_estimators=1000,objective='regr
model5 = LGBMRegressor(boosting_type='dart', num_leaves=29,min_data_in_leaf=15 ,max_depth=7, learning_rate=0.09, n_estimators=1000
model6 = LGBMRegressor(boosting_type='dart', num_leaves=25, max_depth=7, learning_rate=0.13, n_estimators=1000,objective='regressi
model7 = ensemble.GradientBoostingRegressor(n_estimators= 300, max_depth=7,learning_rate=0.054 , loss= 'ls',random_state=30)
model8 = XGBRegressor(max_depth=8,learning_rate=0.04, n_estimators=500,booster='gbtree',random_state=30)
model9 = XGBRegressor(max_depth=8,learning_rate=0.081, n_estimators=100,booster='gbtree',random_state=30)
model10 = RandomForestRegressor(max_depth=18, random_state=30,n_estimators=91)
```

Stacking of Models:

Note that We achieved lowest testing error of 12.9 % from LightGBM model (5th model) but as we had very small amount of data, thus in such cases it may happen that if we use one machine learning model only and it does not deal better with the unseen test dataset, we may get very poor testing accuracy. Thus, We used Stacking approach into account for this type of problem. Thus, we are almost sure that our average of 10 best models will never overfit data. Also, after combining all 10 models prediction, we get testing accuracy of 12.8% which is lower than our best models accuracy and training accuracy reduced to a great extent from 9.06% of our previous best model to just 7.1% which is a lot improvement as we have values in 10e+5 to 10e+6 range.

All 10 models was made from best parameter tuning.

10 Models which were stacked together are below:

Model	No	Training Error	5 Fold Testing Error	No of estimators	Max depth	Learning rate	Num of iterations	No of leaves	Minimum data in leaf
LightGBM	1	0.091082	0.130032	600	8	0.13	100	31	20
LightGBM	2	0.091730	0.129483	400	9	0.19	100	31	20
LightGBM	3	0.087806	0.130125	350	8	0.24	100	31	20
LightGBM	4	0.091184	0.130137	1000	7	0.09	900	31	20
LightGBM	5	0.090642	**0.129125**	1000	7	0.09	100	29	15
LightGBM	6	0.086398	0.129469	1000	7	0.13	100	25	20
Gradient Booster	7	0.075843	0.13391	300	7	0.054	-	-	-
XGBoost	8	0.063819	0.1341148	500	8	0.04	-	-	-
XGBoost	9	0.0798823	0.1346995	100	8	0.081	-	-	-
Random Forest	10	0.052911	0.144339	91	18	-	-	-	-

Why our final model will never overfit and will be a great Predictor?

1. We used almost all models which are **Ensemble Learners**, which are the best learners when it comes to learning from weak learners and making a single strong learner. So, even if one learner makes mistake it is not likely that all 1000 learners will make mistake.
2. We used **Stacking** of Different Models. Suppose Random Forest Regressor would not perform better on unseen test data, we have other 9 learners which will not undergo overfitting.
3. We used **Cross Validation** Approach with not large value of K, which may lead to overfitting or thinking that our model is very good and would surely perform better on test data. Thus, instead of much usual value of k=10, I used k=5 which makes sure that we are serving main purpose of K Fold Cross Validation, which is to reduce Overfitting.

Thanks and Kind Regards,

Utsav patel

utpatel@iu.edu

312-774-3172