Machine Learning Engineer Nanodegree

Capstone Project

Avito Demand Prediction Challenge

Sachin Lamba

Mar, 2019

I. Definition

Project Overview

Today, any website we visit always tried to recommend something depending on our taste (or we can say history). Machine Learning is a field to check from data what history can say about our future tastes.

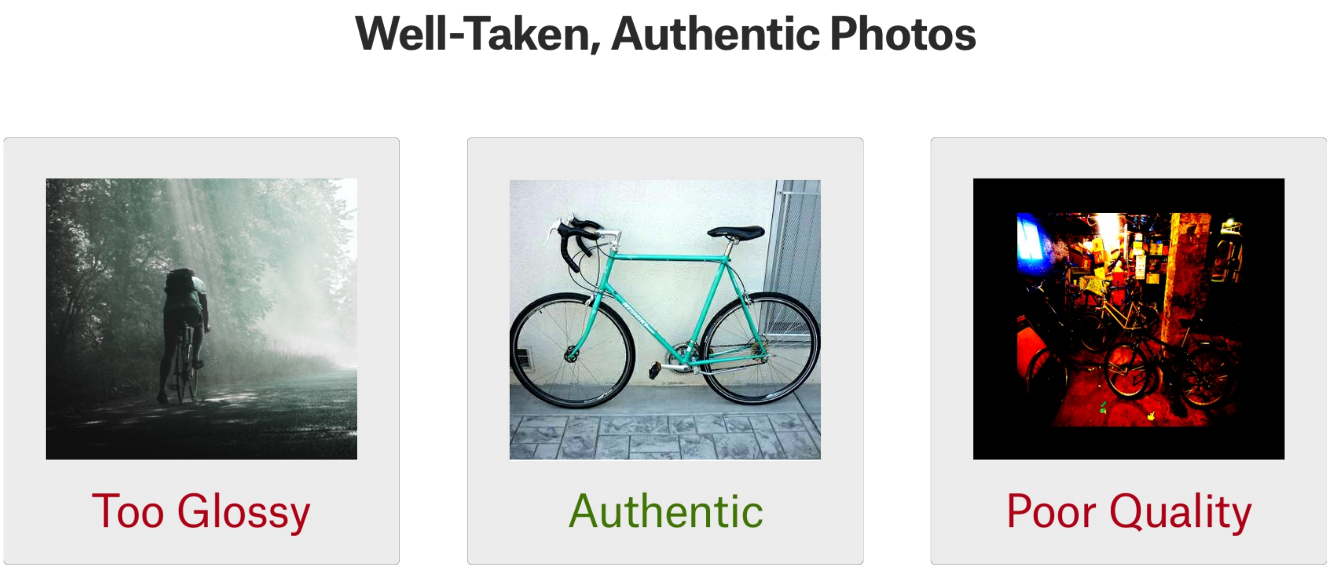
In India, Some companies like OLX are helping their users to sell their products online. But What Sellers (Customer for company) don't know about it is that what details they are filling, can make their items price go up or down. Machine Learning can help in predicting the likelihood of the price depending on the info provided by the seller and directly suggest that seller what need to be updated to make more effective product sell.

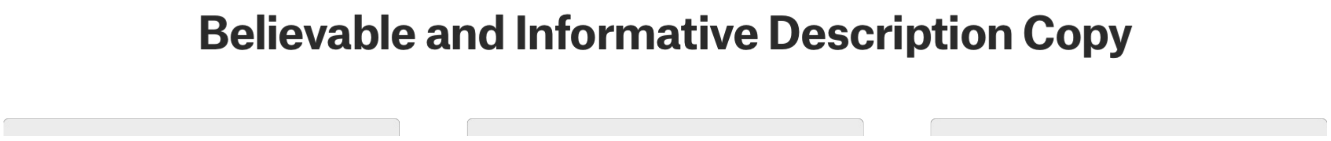
Similar to this situation, Avito (Russian based company) created a challenge on kaggle (https://www.kaggle.com/c/avito-demand-prediction) to predict demand for an online advertisement based on its full description, its context and historical demand for similar ads in similar contexts.

I want to check how provided details of a product can affect its demand in live market. It is like a recommended system, except that the user will get a prediction of how well their product is and how well he defined it for a profit.

Problem Statement

When selling used goods online, a combination of tiny, nuanced details in a product description can make a big difference in drumming up interest. Details like:





Description:
AMAZING WATCH FOR SALE!!!!

DON'T MISS THIS DEAL. IT'S THE DEAL OF THE CENTURY!!

Unlikely

Description:
I have an adjustable Chaleur D'Animale Watch for sale.

It's never been worn and still in the original box. Battery included.

Informative

Description:
fancy watch for sale

no low ball offers, cash and carry

Poor Quality

And, even with an optimized product listing, demand for a product may simply not exist–frustrating sellers who may have over-invested in marketing. Avito, Russia's largest classified advertisements website, is deeply familiar with this problem. Sellers on their platform sometimes feel frustrated with both too little demand (indicating something is wrong with the product or the product listing) or too much demand (indicating a hot item with a good description was underpriced). In this Kaggle competition, Avito is challenging to predict demand for an online advertisement based on its full description (title, description, images, etc.), its context (geographically where it was posted, similar ads already posted) and historical demand for similar ads in similar contexts. With this information, Avito can inform sellers on how to best optimize their listing and provide some indication of how much interest they should realistically expect to receive.

I want to take a subset of their data to test and verify how prediction by details help a user to get best price of their product.

I treated this problem as Regression one. As the value can be in [0,1] both inclusive, as we can never be sure 100% of ad demand.

By treating this problem as Regression, I had **different options** in choosing from algorithms like XGBoost, CatBoost, LightGBM, AdaBoost. First, I tried the training dataset with default paramters of each algorithm to fit and checked on which of them gave better combination of `timing + accuracy`. Later by using the best algorithms in this combination, I tried with tuned hyper-paramters(adjsuted static & dynamicly generated multiple times) so that I can generate my submission file for those tuned models and checked its accuracy values by kaggle submission. We have a image showing how I did on accuracy.

Metrics

Metrics are used to evaluate how well our algorithms are working.

We know that `RMSE` penalizes the predictions more heavily than `MAE`. Generally, **RMSE will be higher than or equal to MAE.**

However, even after being more complex and biased towards higher deviation, loss function defined in terms of RMSE is smoothly differentiable and makes it easier to perform mathematical operations compare to MSE *making it default choice for any models.*

R Squared & Adjusted R Squared are often used for explanatory purposes and explains how well our selected independent variable(s) explain the variability in your dependent variable(s)

`Adjusted R²` does a better job than `RMSE` whose scope is limited to comparing predicted values with actual values. Also, the absolute value of RMSE does not actually tell how bad a model is. It can only be used to compare across two models whereas Adjusted R² easily does that.

However, if you care only about prediction accuracy then RMSE is best. It is computationally simple, easily differentiable and present as default metric for most of the models. Read More

As per the Kaggle challenge and also because of its simplicity with the problem as regression type, I will be using the following metric:

$$ RMSE = {\sqrt{ \sum_{i=1}^n (y - y_i^)^2 \over n } } $$

Where

$$ predicted Value => y_i^ $$

$$ original Value => y $$

II. Analysis

```
In [1]:
```

```
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt
```

Data Exploration

The dataset is composed of multiple CSV files but I will be mainly using train.csv & test.csv file. It has been obtained from a Kaggle Competition provided by Avito.

The most important file is the train.csv file which has 18 columns containing user id, category, region, price, image flag etc along with the target variable deal_probability and has 1503424 rows. The test.csv is similar to the previous file discussed but does not have our target variable and we have to use these to predict the destination and has 508438 rows. We have a good amount of data to work with to produce meaningful models.

- train.csv - Train data
  - item_id - Ad id
  - user_id - User id
  - region - Ad region
  - city - Ad city
  - parent_category_name - Top level ad category as classified by Avito's ad model
  - category_name - Fine grain ad category as classified by Avito's ad model
  - param_1 - Optional parameter from Avito's ad model
  - param_2 - Optional parameter from Avito's ad model
  - param_3 - Optional parameter from Avito's ad model
  - title - Ad title
  - description - Ad description
  - price - Ad price
  - item_seq_number - Ad sequential number for user
  - activation_date- Date ad was placed
  - user_type - User type
  - image - Id code of image. Ties to a jpg file in train_jpg. Not every ad has an image
  - image_top_1 - Avito's classification code for the image
  - deal_probability - The target variable. This is the likelihood that an ad actually sold something. It's not possible to verify every transaction with certainty, so this column's value can be any float from zero to one
- test.csv - Test data. Same schema as the train data, minus deal_probability

In [2]:

```
all_train_dataset = pd.read_csv("./inputs/train.csv")
all_train_dataset.head()
```

Out[2]:

	item_id	user_id	region	city	parent_category_name	category_name	param_1
0	b912c3c6a6ad	e00f8ff2eaf9	Свердловская область	Екатеринбург	Личные вещи	Товары для детей и игрушки	Постельные принадлежности
1	2dac0150717d	39aeb48f0017	Самарская область	Самара	Для дома и дачи	Мебель и интерьер	Другое
2	ba83aefab5dc	91e2f88dd6e3	Ростовская область	Ростов-на-Дону	Бытовая электроника	Аудио и видео	Видео, DVD и Blu-ray плееры
3	02996f1dd2ea	bf5cccea572d	Татарстан	Набережные Челны	Личные вещи	Товары для детей и игрушки	Автомобильные кресла
4	7c90be56d2ab	ef50846afc0b	Волгоградская область	Волгоград	Транспорт	Автомобили	С пробегом

In [3]:

```
all_test_dataset = pd.read_csv("./inputs/test.csv")
all_test_dataset.head()
```

Out[3]:

	item_id	user_id	region	city	parent_category_name	category_name	param_1	param
0	6544e41a8817	dbe73ad6e4b5	Волгоградская область	Волгоград	Личные вещи	Детская одежда и обувь	Для мальчиков	Обувь
1	65b9484d670f	2e11806abe57	Свердловская область	Нижняя Тура	Хобби и отдых	Велосипеды	Дорожные	NaN
2	8bab230b2ecd	0b850bbebb10	Новосибирская область	Бердск	Бытовая электроника	Аудио и видео	Телевизоры и проекторы	NaN
3	8e348601fefc	5f1d5c3ce0da	Саратовская область	Саратов	Для дома и дачи	Бытовая техника	Для кухни	Вытяж
4	8bd2fe400b89	23e2d97bfc7f	Оренбургская область	Бузулук	Личные вещи	Товары для детей и игрушки	Детские коляски	NaN

Exploratory Visualization

Abstract Data:

- Training Data have about 1503424 rows.
- Testing data have 508438.
- **Splitting our dataset in train:validation bins in ratio of ~~0.75:0.25~~ 0.85:0.15 .** As I go through my work, I updated my ratio for final run to this value.
- Deal_probability is the targeted column.

With multiple runs, I found differnt column names which have NaN values inside, so that I can fill those in my preprocessing step with empty strings or zeros.

Columns `param_1` , `param_2` , `param_3` , `description` , `image` contain NA values. In pre-processing step I filled them with some default values depending on their values values/types.

Check different paramters and dtypes for columns for better understanding of data types.

In [12]:

```
# all_train_dataset.info()
for col in train_input.columns:
    if(col == "param_1" or col == "param_2" or col == "param_3" or col == "description" or col == "imag
e"): # as NA exist in these columns
        print("Column Name:", col, "    #Unique Values(+1 extra):", len(np.unique(train_input[col].fill
na("missing"))))
    else:
        print("Column Name:", col, "    #Unique Values:", len(np.unique(train_input[col])))
```

```
Column Name: item_id     #Unique Values: 1503424
Column Name: user_id     #Unique Values: 771769
Column Name: region     #Unique Values: 28
Column Name: city     #Unique Values: 1733
Column Name: parent_category_name     #Unique Values: 9
Column Name: parent_category_name      #Unique Values: 9
Column Name: category_name      #Unique Values: 47
Column Name: param_1      #Unique Values(+1 extra): 372
Column Name: param_2      #Unique Values(+1 extra): 272
Column Name: param_3      #Unique Values(+1 extra): 1220
Column Name: title      #Unique Values: 788377
Column Name: description      #Unique Values(+1 extra): 1317103
Column Name: price      #Unique Values: 102368
Column Name: item_seq_number      #Unique Values: 28232
Column Name: activation_date      #Unique Values: 21
Column Name: user_type      #Unique Values: 3
Column Name: image      #Unique Values(+1 extra): 1390837
Column Name: image_top_1      #Unique Values: 115650
```

In [4]:

```python
train_output = all_train_dataset['deal_probability'].astype('float32')
train_input = all_train_dataset.drop(['deal_probability'], axis=1)
# del all_train_dataset['deal_probability'] ###### can be used
test_input = all_test_dataset.copy()
```

In the `next cell`, we can check on how much variation we have in our targeted variable.

In [29]:

```python
train_output.describe()
```

Out[29]:

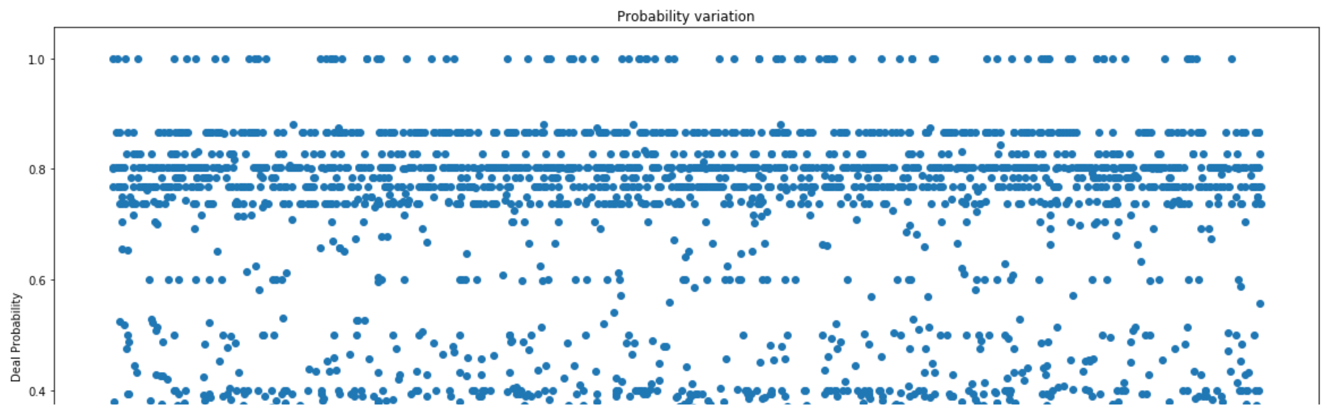
```
count    1.503424e+06
mean     1.389905e-01
std      2.585520e-01
min      0.000000e+00
25%      0.000000e+00
50%      0.000000e+00
75%      1.508700e-01
max      1.000000e+00
Name: deal_probability, dtype: float64
```

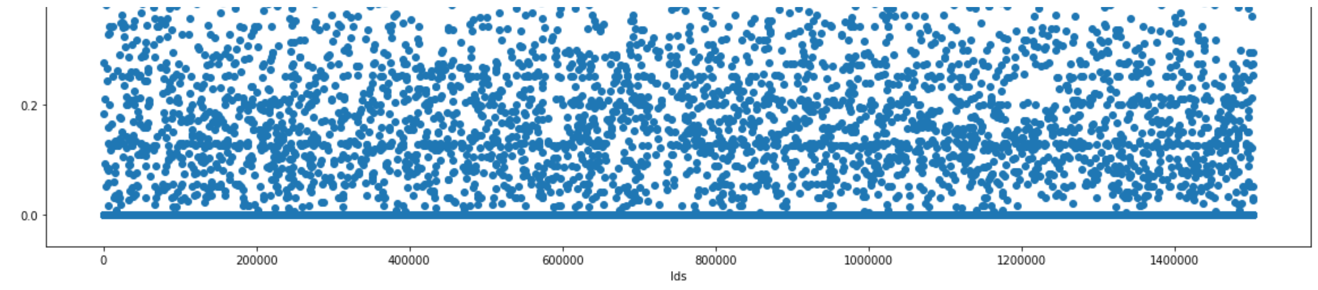
To check how the dataset vary with different `deal_probability`. In below graph, we can check it with 1% of the dataset that its more towards low range in `deal_probability`.

Also, In the `following cell` by plot where we can see that max values lie in lower part of range [0, 1].

In [13]:

```python
plt.figure(figsize=(20, 10))
dataToShow = 0.01 # percentage
indexes = np.random.randint(0, len(train_output), size=int(dataToShow * len(train_output)))
x_select = train_output.iloc[indexes]
plt.xlabel("Ids")
plt.ylabel("Deal Probability")
plt.title("Probability variation")

plt.scatter(indexes, x_select)
plt.show()
```





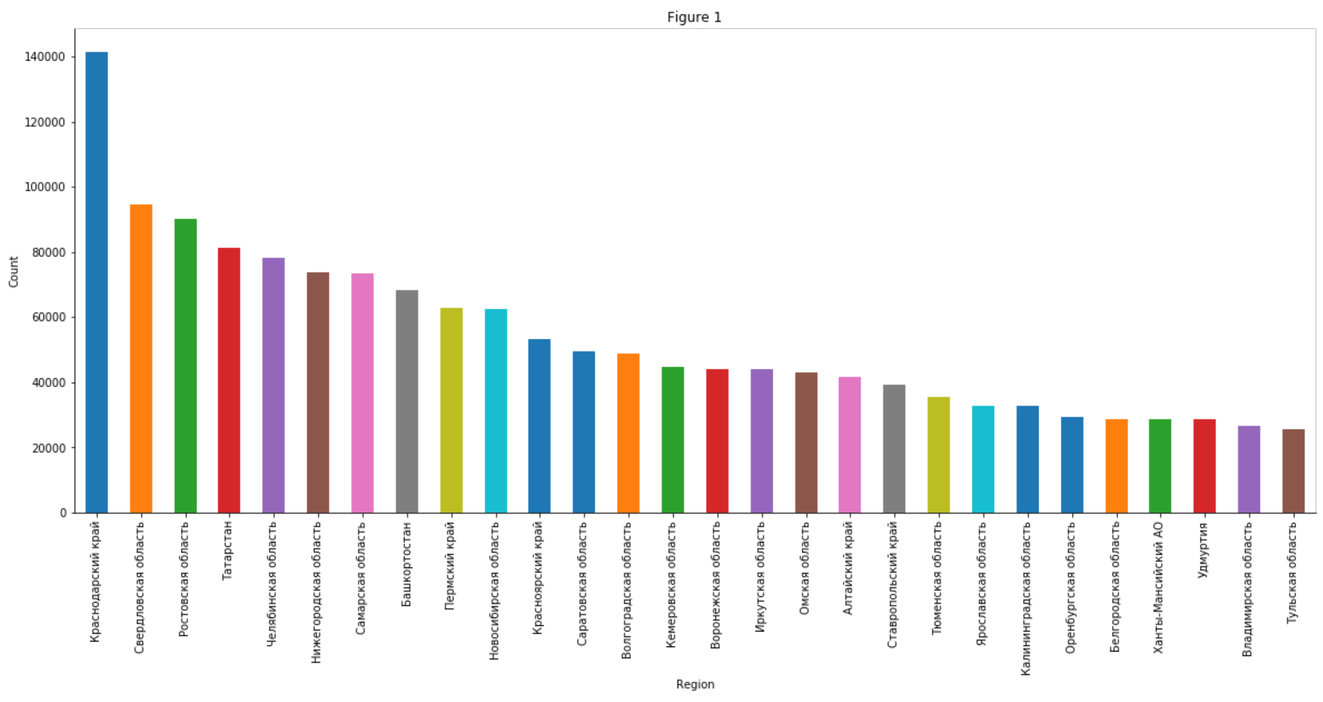
Below 3 graphs shows that `region` *(figure 1)*, `Category Name` *(figure 2)* and `Parent Category Name` *(figure 3)* have few unique values which can be encoded to numeric data for model training.

Variation in data :

- Figure 1 show that approximately all data is distributed in multiple Regions.
- Figure 2 can be interpreted as 2 CategoryNames contain quite data points compare to all others. If we include top 10, it will be more than 50% of data points depending on Category Names.
- Figure 3 can be interpreted as only 1 ParentCategoryNames contain about half data points compare to all others.

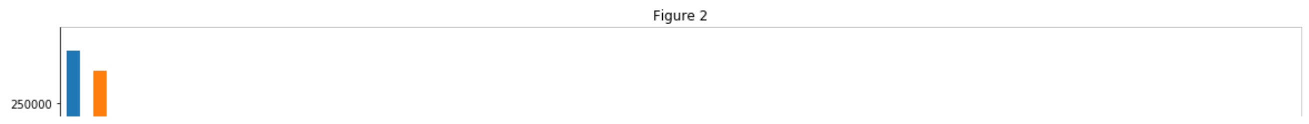
In [33]:

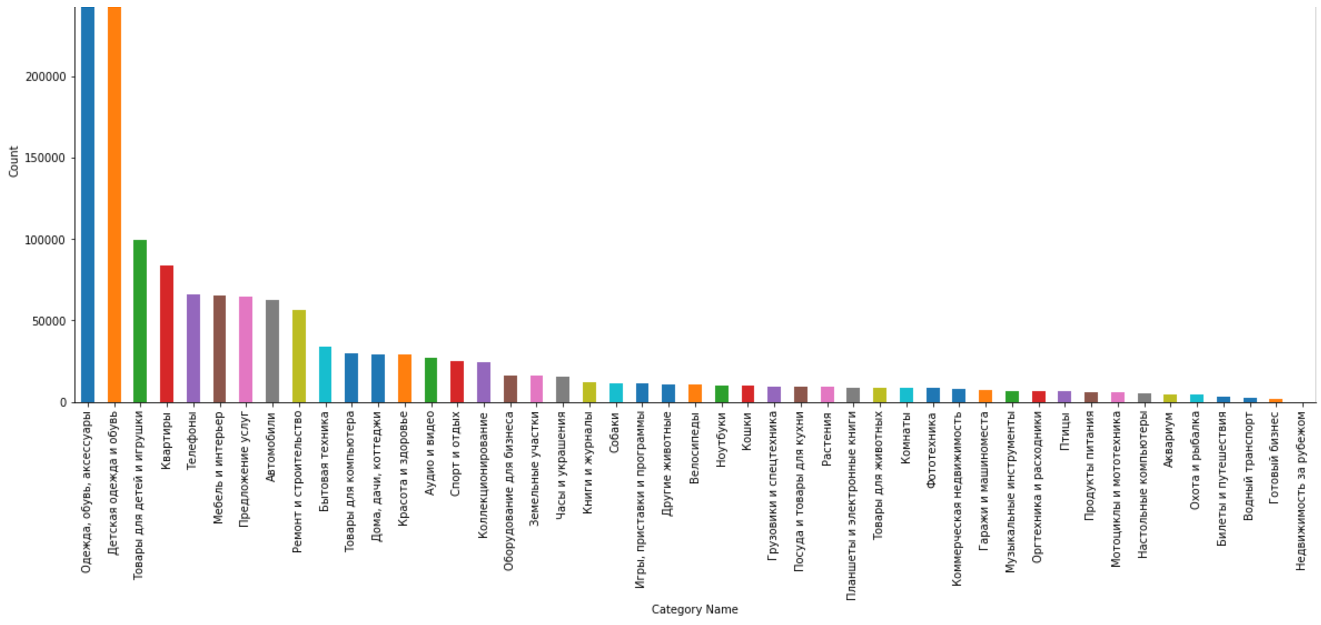
```
plt.figure(figsize=(20, 8))
all_train_dataset.region.value_counts(dropna=False).plot(kind='bar', rot=0)
plt.title('Figure 1')
plt.xlabel('Region')
plt.ylabel('Count')
plt.xticks(rotation=90)
sns.despine()
```



In [34]:

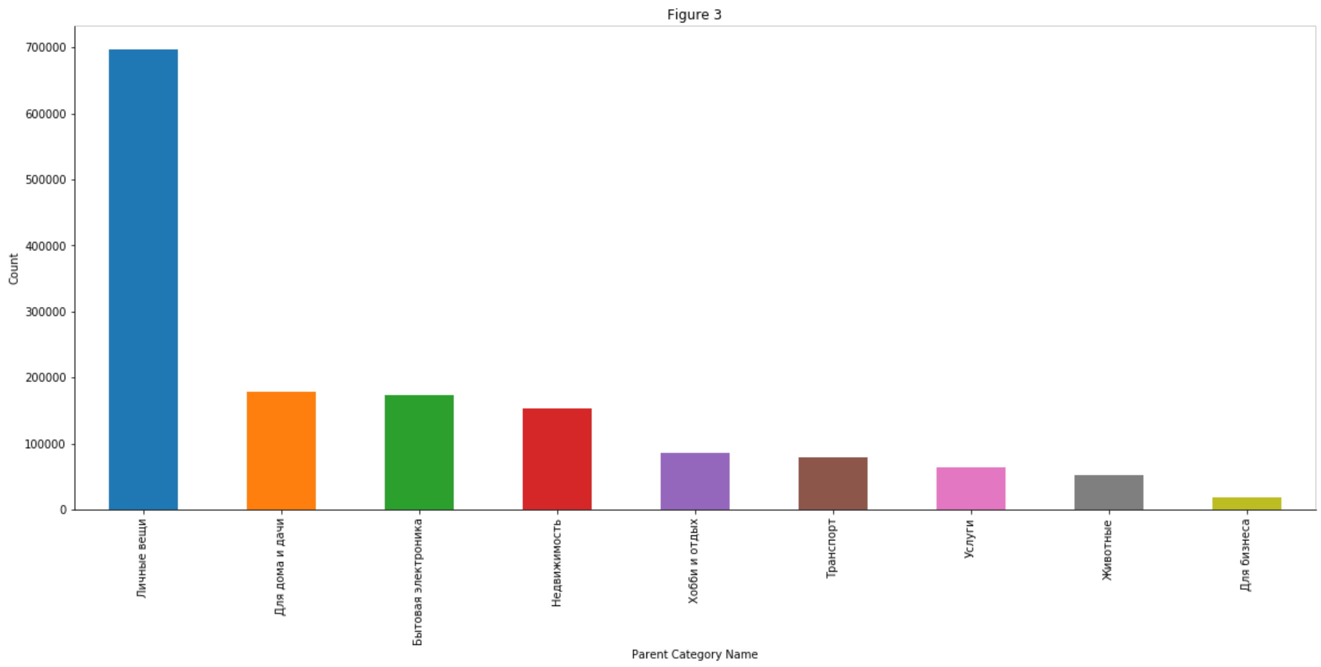
```
plt.figure(figsize=(20, 8))
all_train_dataset.category_name.value_counts(dropna=False).plot(kind='bar', rot=0)
plt.title('Figure 2')
plt.xlabel('Category Name')
plt.ylabel('Count')
plt.xticks(rotation=90)
sns.despine()
```





In [35]:

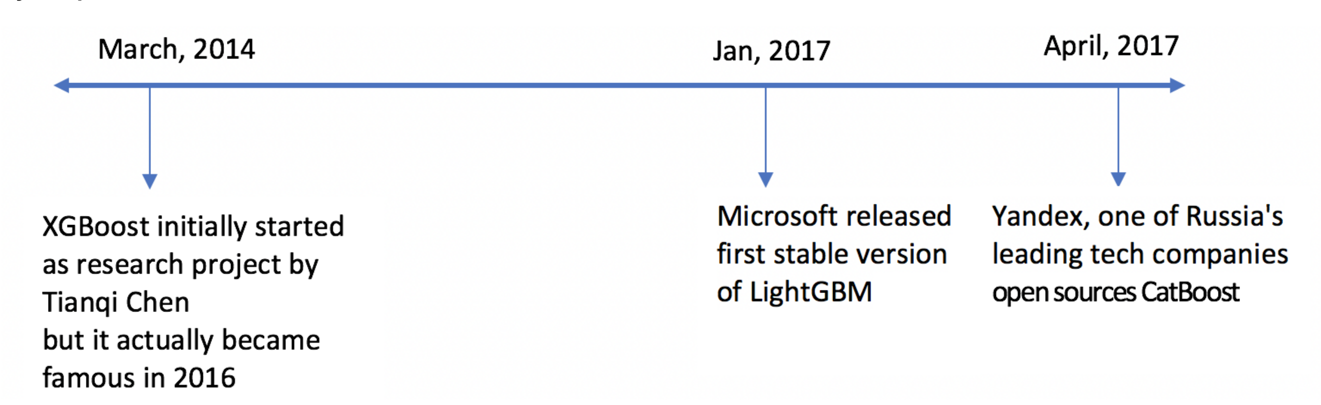
```
plt.figure(figsize=(20, 8))
all_train_dataset.parent_category_name.value_counts(dropna=False).plot(kind='bar', rot=0)
plt.title('Figure 3')
plt.xlabel('Parent Category Name')
plt.ylabel('Count')
plt.xticks(rotation=90)
sns.despine()

# plt.figure(figsize=(20, 8))
# all_train_dataset.city.value_counts(dropna=False).plot(kind='bar', rot=0)
# plt.xlabel('city')
# plt.xticks(rotation=90)
# sns.despine()
```



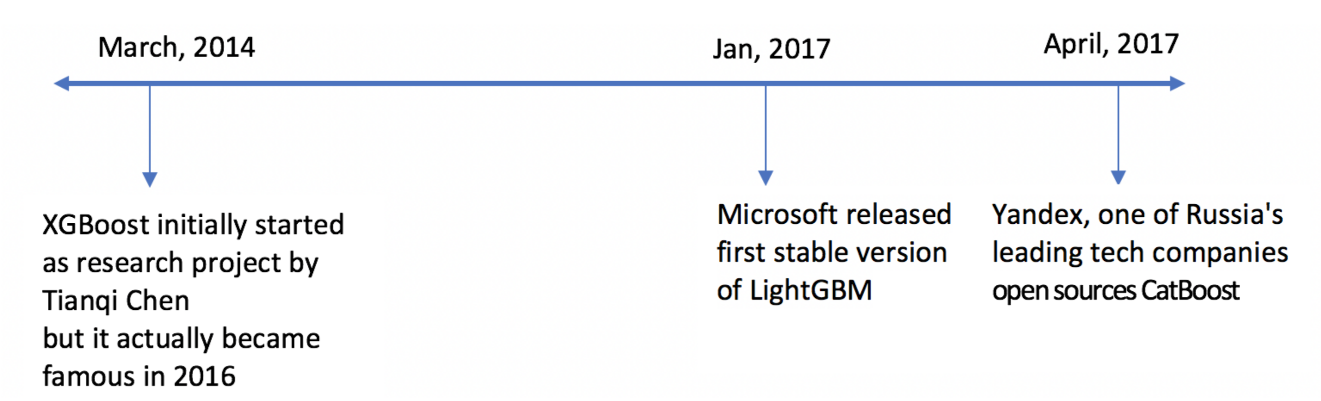
Algorithms and Techniques

As this is a supervised learning problem with a regression solution with range [0,1]; I want to use different ensemble methods with tuning of hyper-parameters for the best model later for improvements. I tried with following algorithms:

Boosting: Training a bunch of individual models in a sequential way. Each individual model learns from mistakes made by the previous model.

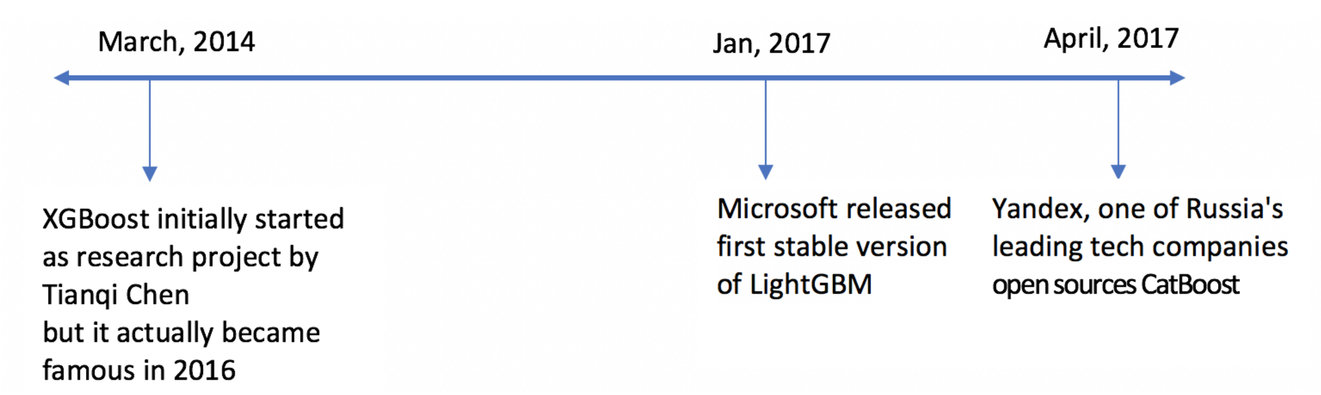

March, 2014
Jan, 2017
April, 2017
XGBoost initially started as research project by Tianqi Chen but it actually became famous in 2016
Microsoft released first stable version of LightGBM
Yandex, one of Russia's leading tech companies open sources CatBoost

Bagging: Training a bunch of individual models in a parallel way. Each model is trained by a random subset of the data.


March, 2014
Jan, 2017
April, 2017
XGBoost initially started as research project by Tianqi Chen but it actually became famous in 2016
Microsoft released first stable version of LightGBM
Yandex, one of Russia's leading tech companies open sources CatBoost

- The `XGBoostRegressor` algorithm was choosen after research into Kaggle Competitions and it was found out that it proves extremely effective in such arenas. XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It produces an ensemble of weak decision tree learners via additive training (boosting). XGBoost is short for Extreme Gradient Boosting. This is based on Gradient boosted trees. Boosted trees are basically an ensemble of decision trees which are fit sequentially so that each new tree makes up for errors in the previously existing set of trees. The model is ”boosted” by focusing new additions on correcting the residual errors of the last version of the model. Then you take an approximate step in the gradient direction by training a model to predict the gradient given the data. XGBoost algorithm tuning is a tricky process. Heavy computation power is required for such level of tuning.
- `LightGBM` from Microsoft and published in 2017 also introduces two techniques to improve performance. Gradient-based One-Side Sampling which inspects the most informative samples while skipping the less informative samples. *LightGBM improves on XGBoost. The LightGBM paper uses XGBoost as a baseline and outperforms it in training speed and the dataset sizes it can handle. The accuracies are comparable. LightGBM in some cases reaches it’s top accuracy in under a minute and while only reading a fraction of the whole dataset. This goes to show the power of approximation algorithms and intelligently sampling a dataset to extract the most information as fast as possible.*
- `Catboost` improves over LightGBM by handling categorical features better. Traditionally categorical features are one-hot-encoded, this incurs the curse of dimensionality if the feature has many distinct values. Previous algorithms tackled this issue of sparse features as we saw with EFB above. Catboost deals with categorical features by, “generating random permutations of the dataset and for each sample computing the average label value for the sample with the same category value placed before the given one in the permutation”. They also process the data with GPU acceleration, and do feature discretization into a fixed number of bins (128 and 32).

Read More 1 Read More 2


March, 2014
Jan, 2017
April, 2017
XGBoost initially started as research project by Tianqi Chen but it actually became famous in 2016
Microsoft released first stable version of LightGBM
Yandex, one of Russia's leading tech companies open sources CatBoost

More similar algorithms I will tried are :

- `AdaBoostRegressor` is a boosting ensemble model and works especially well with the decision tree. Boosting model's key is learning from the previous mistakes, e.g. misclassification data points.(learns from the mistakes by increasing the weight of misclassified data points.)
- `GradientBoostingRegressor` learns from the mistake—residual error directly, rather than update the weights of data points.

Read More 3

After getting intuition, I used GridSearchCV for further analysis.

Benchmark

As this is like a recommended system, I thought of using Decision tree as with GridSearchCV as hyperparameter tuning to save its best model, but later used `RandomForestRegressor` for comparision to other models.

`RandomForestRegressor` is an ensemble model using bagging as the ensemble method and decision tree as the individual model.

Random Forest Classifier fits number of decision trees on subsamples of a dataset and averages the results.

A random forest is a collection of random decision trees. In which at each node you will randomly draw a subset of features and the decision tree will predict the classification. Then the same is done with several trees and bagged. This ensemble method will reduce overfitting and provide good classification. The bias-variance trade-off for this algorithm is good and therefore the possibility of overfitting is drastically reduced.

RandomForestRegressor have RMSE error of 0.10 in training data but have 0.24 in validation data,meaning its overfit the data in training set.

III. Methodology

Data Preprocessing

- Data preprocessing/cleaning
  - Text data Label encoded for the required text columns (except for item_id, user_id, title, description as they have too many unique values or i can say every value is unique only ;)).
  - filled the missing values with some constant. Columns `param_1` , `param_2` , `param_3` , `description` , `image` contain NA values as I said earlier. In this pre-processing step I filled them with default values depending on their values values/types e.g.
    - string values have empty string assigned and float values got 0.
    - image column converted to bool type if it exist/non-exist condition. I didn't work on image processing for sentiments check.
    - price range changed with log transform.
  - Split data into training & validation set

Note : ***I didn't used word vector algorithms (as I am not familiar for those topics right now much.)***

Preprocess steps functions

---

1. `processData` function is used to process the NaN values in data an also to update different data types as required by different boosting algorithms.

```
 One e.g. I got somthing like **ValueError: DataFrame.dtypes for data must be int, float
or bool. Did not expect the data types in fields item_id, user_id, activation_date, user_
type, title_description.**

As These columns have text data with varying lengths, I need to drop them for my algorith
m training by boosting algorithm methods.
```

---

1. `uniqueLabelsExtract` function is using LabelEncoder to find different category types for different data in train & test datasets. Read more about Label Encoders.

---

1. `encode2Labels` function will encode our text data to their respective category for specific columns.

   I used log for price column to adjust range of variation here. (*Using a logarithmic transformation significantly reduces the range of values caused by outliers.*)

   Also, for image column, I didn't done processing on those files of zip as per in dataset, But I want to make use of this column to make algorithm check if an image in description effect the outcome or not.

Implementation

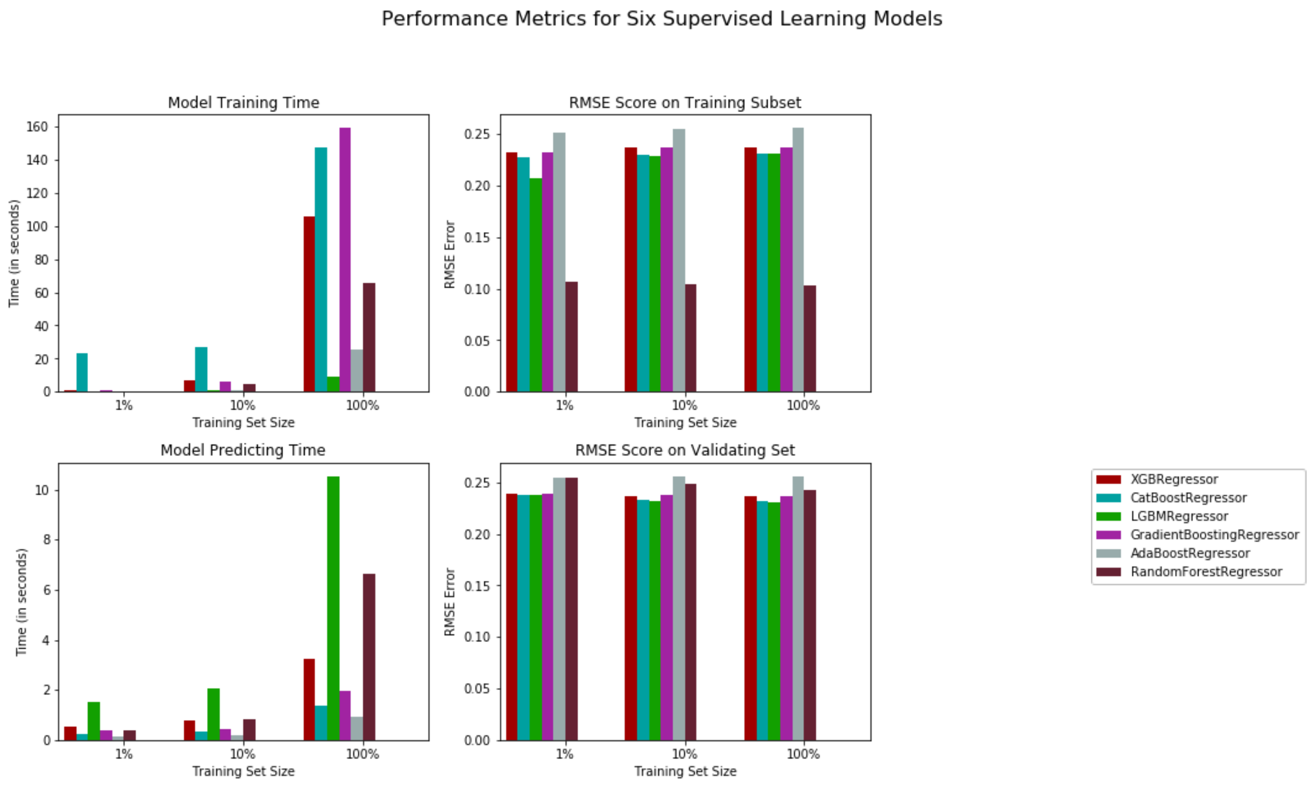
- Evaluate Algorithm

1. Build Models (try in between AdaBoost, GradientBoost, Lightgbm, Xgboost, Catboost)

Training step for evaluating algorithms

`train_predict` function used to save different algorithms *train time*, *validation time*, *training error* and validation error*.

Result of different algorithms without hyper-paramter tuning, i.e. with only default paramters:



As We can check: XGBRegressor, CatBootRegressor, LGBMRegressor outperforms GradientBoostingRegressor, AdaBoostRegressor. Let us view the charts in more details:

```
- RandomForestRegressor have RMSE error of 0.10 in training data but have 0.24 in validation dat
a,meaning its overfit the data in training set.
- AdaBoost speed is fine, but as per RMSE error, it lags behind other algorithms.
- GradientBoostingRegressor is fine also but I want to explore first 3 in more details, i didn't
use it further :)
- Best one in all for now is LGBMRegressor & CatBoostRegressor as similar range of errors help m
e build my intuition.
```

Default Hyper-paramters - Results:

Model	**RMSE train**	**RMSE Validation**
XGBRegressor	0.23682830033249136	0.2369794766776338

Model	**RMSE train**	**RMSE Validation**
CatBoostRegressor	0.23143335687294486	0.23165537332953787
LGBMRegressor	0.23050971081118116	0.2308279440054516
GradientBoostingRegressor	0.23667756154847222	0.23682248919971033
AdaBoostRegressor	0.2560140460201985	0.25613432272469766
RandomForestRegressor	0.10365176756308926	0.24306912479686782

Refinement

- Model tuning to improve results
  - Selected best model with tuning hyper-parameters. First I will choose a range of parameters in some step increments & also some randomized values for covering large parameter space also.

2. Select best model

Training step with hyperparamters

`useStaticParamters` is used to set static paramters for (algorithms I deicded above) GridSearchCV.

`useDynamicParamters` is used to set dynamic paramters generation for (algorithms I deicded above) GridSearchCV.

3. Make predictions on validation set

`modelEvaluator` is training each model/learner with GridSearchCV for finding better paramters based on **RMSE metric**

Hyper-parameters tuned - Results:

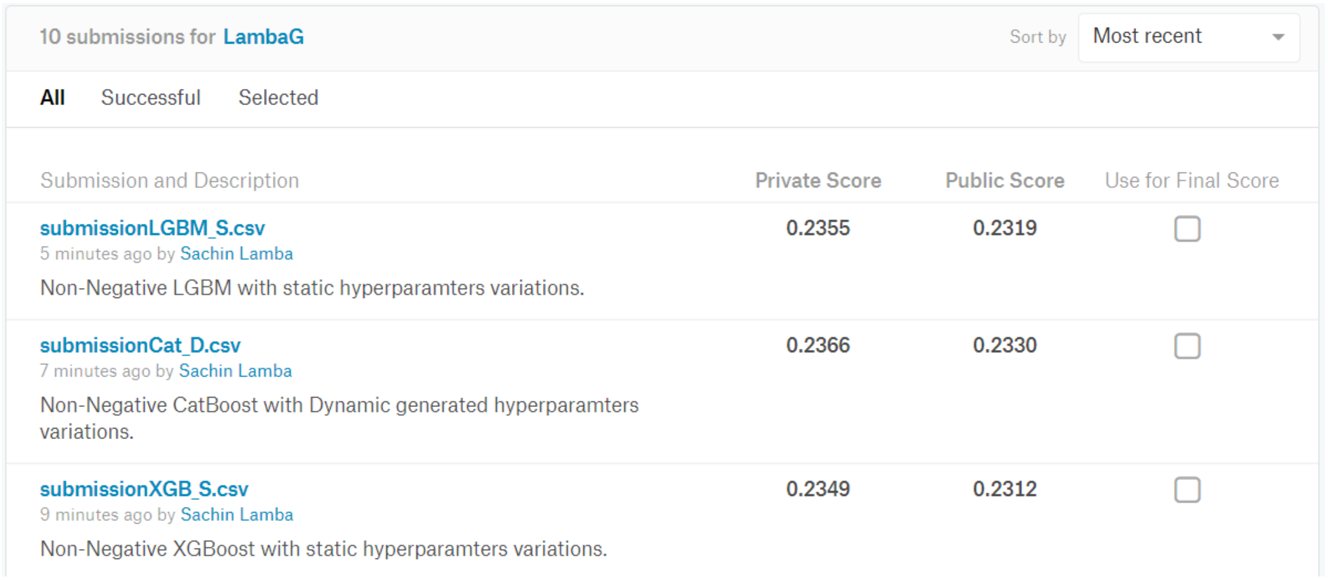
Model	**RMSE train**	**RMSE Validation**	**Hyper-paramters tuned**
XGBRegressor	0.22380145181210112	0.22688456523621184	n_estimators, num_leaves, learning_rate
CatBoostRegressor	0.2314429399431053	0.23164672658468144	n_estimators, learning_rate
LGBMRegressor	0.224661456645794	0.2275443533988149	n_estimators, max_depth, learning_rate

Hyper-Paramters are tuned both statically and dynamically for coverage of large set

IV. Results

Model Evaluation and Validation

Submission @Kaggle of csv file.

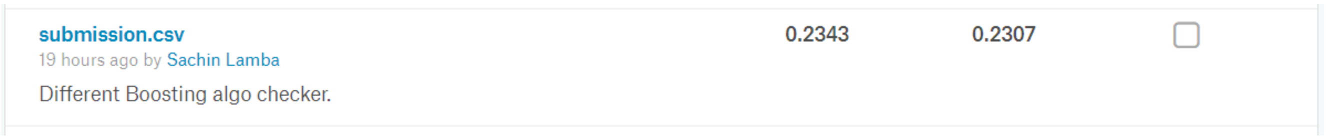


10 submissions for LambaG

Sort by Most recent

All Successful Selected

Submission and Description	Private Score	Public Score	Use for Final Score
submissionLGBM_S.csv 5 minutes ago by Sachin Lamba Non-Negative LGBM with static hyperparamters variations.	0.2355	0.2319	☐
submissionCat_D.csv 7 minutes ago by Sachin Lamba Non-Negative CatBoost with Dynamic generated hyperparamters variations.	0.2366	0.2330	☐
submissionXGB_S.csv 9 minutes ago by Sachin Lamba Non-Negative XGBoost with static hyperparamters variations.	0.2349	0.2312	☐


submission.csv 0.2343 0.2307
19 hours ago by Sachin Lamba
Different Boosting algo checker.

As I can compare, XGB outperform other by small margin. But I need to do more traininig for much better results. Top score in this competition is at 0.21 which I am not even close. I will be reading more about word embedding so that i can use them in my training set and not to drop them.

With these many dataset, I am getting intuition from Result graph titled : "Performance metrics for six supervised learning models" that i choose models which are not overfitting and also have better timing & accuracy score(error) too.

Justification

Unseen data can tell us about a model scalability, how much good it is. My benchmark model failed on this step only. As I checked my model submission on kaggle private dataset, I am getting accuracy to what I expect with the limited dataset I used here. Because of reduction of dataset, Image processing & text to word vector conversion got skiped which made the model fast to predict to comparable best models exist for this problems at kaggle. Hence, These models seems to be robust. Alongside this, the model seems to be trustable and aligns well with expected solutions outcomes.

Model	RMSE train	RMSE Validation
(Tuned) XGBRegressor	0.22380145181210112	0.22688456523621184
(Benchmark) RandomForestRegressor	0.10365176756308926	0.24306912479686782

As I can compare from above table, XGBRegressor validation error is 0.227 is better than RandomForestRegressor's validation error of 0.243 and also, We can see that RandomForestRegressor overfitted the model which can need some more work compare to XGBRegressor.

V. Conclusion

Free-Form Visualization

In [10]:

```python
print(np.sum(train_output > 0.5))
print(np.sum(train_output <= 0.5))
```

```
177754
1325670
```

In [10]:

```python
def processData(dataset):
    print("*******Filling missing/NA values...********")
    dataset['param_1'].fillna(value='', inplace=True)
    dataset['param_2'].fillna(value='', inplace=True)
    dataset['param_3'].fillna(value='', inplace=True)
    dataset['description'].fillna(value='', inplace=True)
    dataset['image_top_1'].fillna(value=0, inplace=True)
    dataset['price'].fillna(value=1, inplace=True)

    print("*******Data type adjustments...********")
    # https://stackoverflow.com/questions/21018654/strings-in-a-dataframe-but-dtype-is-object
    # https://stackoverflow.com/questions/33957720/how-to-convert-column-with-dtype-as-object-to-string
-in-pandas-dataframe
    # dataset['param_1'] = dataset['param_1'].astype(str)
    # dataset['param_2'] = dataset['param_2'].astype(str)
    # dataset['param_3'] = dataset['param_3'].astype(str)
    # dataset['price'] = dataset['price'].astype('float32')
    # dataset['item_seq_number'] = dataset['item_seq_number'].astype('uint32')
    data_type = {
        'param_1': str,
        'param_2': str,
        'param_3': str,
        'price': 'float64',
        'price': 'float64',
        'item_seq_number': 'uint32',
        'image_top_1': 'uint32',
    }
    dataset = dataset.astype(data_type)
    dataset['param123'] = (dataset['param_1']+'_'+dataset['param_2']+'_'+dataset['param_3']).astype(str
)
    dataset['title_description']= (dataset['title']+' '+dataset['description']).astype(str)
    dataset['image_exists'] = dataset['image'].isnull().astype(int)

    del dataset['param_2'], dataset['param_3'], dataset['description'], dataset['title'], dataset['imag
e']
    gc.collect()

    return dataset

def uniqueLabelsExtract(train_data, test_data):
    DF = pd.concat([train_data, test_data], axis = 0)
    print("*******Starting Label Encoding process...********")
    # https://medium.com/@contactsunny/label-encoder-vs-one-hot-encoder-in-machine-learning-3fc27336562
1
    le_region = LabelEncoder()
    le_region.fit(DF.region)

    le_city = LabelEncoder()
    le_city.fit(DF.city)

    le_user_type = LabelEncoder()
    le_user_type.fit(DF.user_type)

    le_category_name = LabelEncoder()
    le_category_name.fit(DF.category_name)

    le_parent_category_name = LabelEncoder()
    le_parent_category_name.fit(DF.parent_category_name)

    le_param_1 = LabelEncoder()
    le_param_1.fit(DF.param_1)

    le_param123 = LabelEncoder()
    le_param123.fit(DF.param123)

    print("Finished Label Encoding process.")
    del DF
    gc.collect()

    return le_region, le_city, le_user_type, le_category_name, le_parent_category_name, le_param_1, le_
param123

def encode2Labels(dataset):
    print("*******Encoding dataset via LabelEncoder...*********")
    dataset['region'] = le_region.transform(dataset['region'])
    dataset['city'] = le_city.transform(dataset['city'])
    dataset['user_type'] = le_user_type.transform(dataset['user_type'])
    dataset['category_name'] = le_category_name.transform(dataset['category_name'])
    dataset['parent_category_name'] = le_parent_category_name.transform(dataset['parent_category_name']
)
    dataset['param_1'] = le_param_1.transform(dataset['param_1'])
    dataset['param123'] = le_param123.transform(dataset['param123'])
    dataset['price'] = np.log1p(dataset['price'])
    dataset['item_seq_number'] = np.log(dataset['item_seq_number'])
    print("*******Finished Encoding dataset.*********")
    return dataset

def rmse(y, y_pred):
    Rsum = np.sum((y - y_pred)**2)
    n = y.shape[0]
    RMSE = np.sqrt(Rsum/n)
    return RMSE
import gc
from sklearn.preprocessing import LabelEncoder

train_output = all_train_dataset['deal_probability'].astype('float32')
train_input = all_train_dataset.drop(['deal_probability'], axis=1)
# del all_train_dataset['deal_probability'] ###### can be used
test_input = all_test_dataset.copy()
```

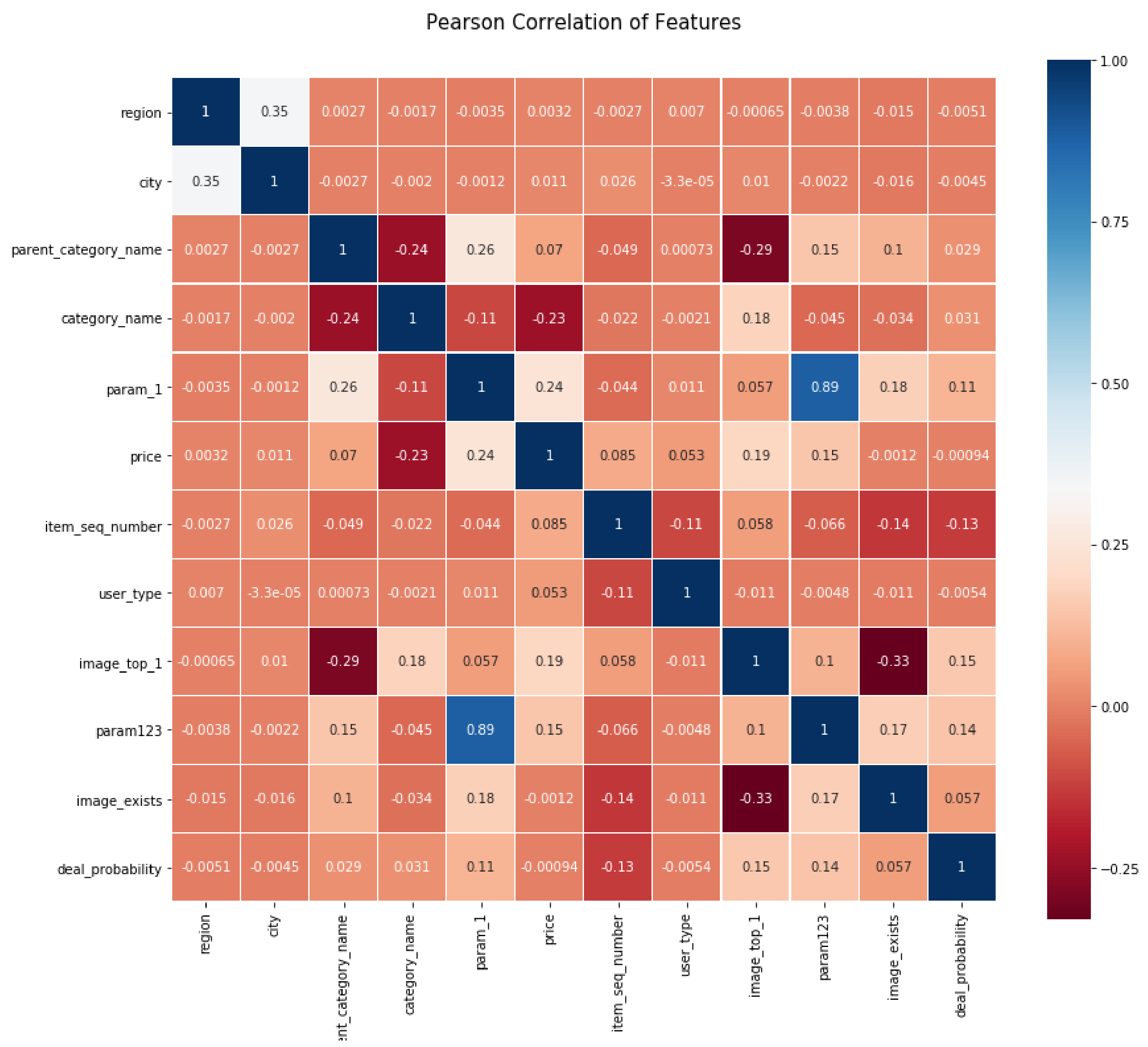
```
print("Training data processing...")
train_input = processData(train_input)
print("Testing data processing...")
test_input = processData(test_input)
print("Label Encoder generator...")
le_region, le_city, le_user_type, le_category_name, le_parent_category_name, le_param_1, le_param123 = 
\
                                    uniqueLabelsExtract(train_input, test_input)

train_input = encode2Labels(train_input)
test_input = encode2Labels(test_input)

col_to_delete_for_fit = ['item_id', 'user_id', 'activation_date', 'title_description']

train_plot = train_input.drop(col_to_delete_for_fit, axis=1)
train_plot['deal_probability'] = train_output
# all_train_dataset.info()
# train_plot.info()
```

In [21]:

```
colormap = plt.cm.RdBu
plt.figure(figsize=(14,12))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sns.heatmap(train_plot.astype(float).corr(),linewidths=0.1,vmax=1.0,
            square=True, cmap=colormap, linecolor='white', annot=True)
```

Out[21]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ca192dfe80>
```



Takeaway from the Plots

The Pearson Correlation plot can tell us is that there are not many features strongly correlated with one another. This is good from a point of view of feeding these features into our learning model because this means that there isn't much redundant or superfluous data in our training set and we are happy that each feature carries with it some unique information. Here are two most correlated features are that of param_1 and param123 as lateral is generated by earlier one in pre-processing step only.

Reflection

As I find that I need to learn fasttext for text conversion, I tried to do that with some research. I was able to produce word vectors for the text columns too. But later I found that these columns as per word vector will consist of nested list of numeric data(`experiments/Capstone Solution.ipynb`) for each column which any boost algo didn't expect in its dataset. So, After this searching, I had to drop those columns for this projects to complete it and later learn more about them so that I can do better in NLP part.

Improvement

Their can be various improvements try:

- One can be to use fasttext for text conversion to word vector. I need to read about more about NLP for that so that those text columns can be utilized by boosting algorithms.
- Use Computer vision to have images tell about product sentiments. It can help more in deducing the probability of buyer interest to some extent.