CSCI 5901 - The Process of Data Science - Summer 2019

Assignment 1

Submitted by

Arjun Chandra Balaji Balaraman (B00803303) Anirudh Rayasam venkata (B00824329)

1a. Explain the dataset with your own words. Focus on the attributes' description.

The dataset, Zomato Bangalore Restaurants, is a large data set containing information about the restaurants in the city. Various attributes of restaurants, like their localities, the type of food they serve, the approximate cost of food, how popular is the restaurant etc., have been provided. Understanding these attributes and using these for predictions will help us gain valuable knowledge.

2 a,b. Distribution of attributes based on frequency and finding trends in the data.

The complete raw dataset was loaded into a Jupyter Notebook for exploring and understanding different attributes and their trends. The dataset contains many attributes out of which, few attributes seemed important and contained information that would be useful for predicting different things and gain insights.

Importing pandas library and reading the dataset

In [ ]:
```
import pandas as pd
```

In [0]:
```
df = pd.read_csv("zomato.csv")
```

Cleaning

We initially skimmed through the records (rows) and found that a lot of records only contained garbage value or are either completely empty. We filtered all such records using the 'url' attribute and deleted them using pandas commands. Now, we were left only with records which had valid data in them. This was an important step and this dataframe served as the base file for further cleaning.

In [0]:
```
# remove records that doesn't have a valid url
df1 = df[df['url'].astype(str).str.startswith('https')]
```

We removed all the duplicate records from the dataframe. Duplicate records were identified using a combination of the attributes – ‘Address’, ’Name’ and ’Location’ as these three attributes will sufficiently remove all duplicate restaurants from the dataset.

In [0]:
```
# drop duplicates based on location, name, address alone
ff = df1.drop_duplicates(subset=["location","name","address"], keep="first")
copy_ff = ff
```

Then, we are dropping all the columns that are not required for our prediction. This leaves us with five columns – location, rate, rest_type, cuisines and approx_cost(for two people). We’re also renaming the columns appropriately.

In [0]:
```
# taking only required columns
ff1 = ff[["name","location","rate","rest_type","cuisines","approx_cost(for two people)"]]
ff1["cost"] = ff1["approx_cost(for two people)"]
del ff1['approx_cost(for two people)']
print(ff1)
```

The Rate attribute is then cleaned. All the records which has either a ‘-‘, ‘NEW’ or empty value in ‘rate’ attribute are removed as these do not provide useful insights. At the same time, we are splitting the rating attribute on ‘/’ to get the actual rating value. A float value ranging between 1.0 and 5.0. Formatting ‘cost’ column is also performed.

Also, all the \r escape characters are removed.

In [334]:
```
# remove \r from all records
ff1.replace("\r","",regex=True,inplace=True)

# remove blank spaces in rate column
ff1["rate"].replace(" ","",regex=True,inplace=True)

# remove comma in cost column
ff1["cost"].replace(",","",regex=True,inplace=True)

# drop records that has empty records
ff2 = ff1.dropna()
ff3 = ff2

# split rate based on "/"
ff3[['rate','full_rate']] = ff3['rate'].str.split('/',expand=True)
del ff3["full_rate"]
dataset = ff3
del dataset["name"]
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:4042: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy
  method=method)
/usr/local/lib/python3.6/dist-packages/pandas/core/generic.py:6586: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy
  self._update_inplace(new_data)
/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:3391: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy
  self[k1] = value[k2]
```

2d. Selecting location with the highest attribute.

The following command considers the locations and the number of restaurants in each location, and based on that, the average rating of restaurants in that location. Then, we sort in descending order to get the highest average rating.

Lavelle Road was the winning location in this case, and here are few trends and characteristics we could find for this location.

In [335]:

```
# 2.d
# select location with highest average rating
# make location count for every unique values
ff4 = ff3.convert_objects(convert_numeric=True)
h_avg = ff4.groupby('location')['rate'].mean().reset_index(name='Avg_Rat
e')
h_avg = h_avg.sort_values(by='Avg_Rate',ascending=False)
print(h_avg)
```

```
                   location  Avg_Rate
50             Lavelle Road  4.073077
74              Sankey Road  4.041667
80           St. Marks Road  3.929412
42   Koramangala 3rd Block  3.926316
44   Koramangala 5th Block  3.905882
12            Church Street  3.898148
65         Race Course Road  3.862069
67     Rajarajeshwari Nagar  3.850000
70            Richmond Road  3.843939
43   Koramangala 4th Block  3.833010
69            Residency Road  3.829730
51                  MG Road  3.809412
25               Hosur Road  3.800000
71           Sadashiv Nagar  3.789286
27              Indiranagar  3.784186
31                Jayanagar  3.760656
46   Koramangala 7th Block  3.758915
28            Infantry Road  3.753846
35             Kalyan Nagar  3.740957
15           Cunningham Road  3.736170
45   Koramangala 6th Block  3.732824
85            Vasanth Nagar  3.730233
54             Malleshwaram  3.727228
76            Seshadripuram  3.723913
39              Koramangala  3.716667
82                   Ulsoor  3.713978
38                  Kengeri  3.700000
90                Yelahanka  3.700000
8              Brigade Road  3.675926
47   Koramangala 8th Block  3.675862
..                      ...       ...
16                   Domlur  3.521622
77             Shanti Nagar  3.514000
64                 RT Nagar  3.500000
10            CV Raman Nagar  3.500000
83              Uttarahalli  3.500000
58                 Nagawara  3.496471
6                 Bellandur  3.495720
53                 Majestic  3.495161
91              Yeshwantpur  3.490625
34            Kaggadasapura  3.490000
48        Kumaraswamy Layout  3.485937
30               Jalahalli  3.480000
3          Bannerghatta Road  3.479083
2                 Banaswadi  3.475177
19          Electronic City  3.475110
17           East Bangalore  3.475000
18                  Ejipura  3.473529
37           Kanakapura Road  3.470000
78             Shivajinagar  3.469697
79          South Bangalore  3.468421
24                   Hennur  3.467391
89            Wilson Garden  3.453125
52              Magadi Road  3.418750
87           West Bangalore  3.400000
57               Nagarbhavi  3.400000
```

```
60        North Bangalore  3.375000
62        Old Madras Road  3.355556
68        Rammurthy Nagar  3.353333
7            Bommanahalli  3.216364
63                 Peenya  3.200000

[92 rows x 2 columns]

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: FutureW
arning: convert_objects is deprecated.  To re-infer data dtypes for obj
ect columns, use DataFrame.infer_objects()
For all other conversions use the data-type specific converters pd.to_d
atetime, pd.to_timedelta and pd.to_numeric.
  """Entry point for launching an IPython kernel.
```

In [336]:

```
# charac of high rating location
h_location = h_avg['location'].iloc[0]
cuisine_charac = pd.DataFrame()
rest_charac = pd.DataFrame()

print("location with highest avg rating:",h_location)
neigh = ff3.loc[df['location'] == h_location]

for row in neigh["cuisines"]:
  cuisines = row.split(",")
  cuisines = [x.strip(' ') for x in cuisines]
  for i in cuisines:
    cuisine_charac = cuisine_charac.append(pd.Series(i),ignore_index=Tru
e)

for j in neigh["rest_type"]:
  rest = j.split(",")

  rest = [x.strip(' ') for x in rest]
  for k in rest:
    rest_charac = rest_charac.append(pd.Series(k),ignore_index=True)

h_cuisine = cuisine_charac[0].iloc[0]
h_rest = rest_charac[0].iloc[0]
print(h_location,"is famous for",h_cuisine,"cuisines")
print(h_location,"is famous for",h_rest,"restaurants")
print(h_location,"is famous for",h_rest,"restaurants")
```

```
location with highest avg rating: Lavelle Road
Lavelle Road is famous for North Indian cuisines
Lavelle Road is famous for Fine Dining restaurants
Lavelle Road is famous for Fine Dining restaurants
```

Importing important libraries for further processing

In [0]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Cleaning

Splitting columns like cuisine and rest_type into individual, proper records as a part of cleaning

We divide the multivalued columns like rest_type and cuisines, to individual records. This step is crucial in order to avoid duplicate values and gain proper 'One Hot Encoding'. The list attribute has attributes stored in a way that can lead to duplicates. For Ex: consider these two records in 'cuisines'

1. Chinese, Continental, North Indian
2. North Indian, Chinese, Fast Food, Biryani

When the system parses the above two list of attributes, it will parse 'Chinese' from the first list, but ' Chinese' (a space before C) from the second list. Now, the system may consider these two as two different categories and so, will encode them differently. This is incorrect and can cause incorrect encoding.

Therefore, we are splitting each list value as an individual record and trimming it to remove spaces. While performing 'One Hot Encoding'/'Dummy variables' to convert categorical values as continuous ones, without considering values redundantly as shown above.

In [0]:
```
# split list columns into rows
def splitColumnList(dataFrame, column):
    newDataFrame = pd.DataFrame()
    dataFrame[column] = dataFrame[column].str.split(',')
    for index, row in dataFrame.iterrows():
        for i in row[column]:
            newColumn = "new_"+column
            row[newColumn] = i
            newDataFrame = newDataFrame.append(row,ignore_index=True)
        print(index)
    newDataFrame[column] = newDataFrame["new_"+column]
    newDataFrame.drop("new_"+column, axis=1, inplace=True)
    return newDataFrame


temp_dataset = splitColumnList(dataset,"rest_type")
temp_dataset = splitColumnList(temp_dataset,"cuisines")
```

Removing spaces and dropping null values

Cleaning the unnecesary spaces from attributes and dropping rows with null values

In [0]:
```
# temp_dataset.to_csv("clean_data.csv")
# print(temp_dataset)
indexNames = temp_dataset[ temp_dataset['rate'] == '-' ].index
temp_dataset.drop(indexNames , inplace=True)
temp_dataset.dropna()
temp_data['location'] = temp_data['location'].str.replace(' ', '')
temp_data['rest_type'] = temp_data['rest_type'].str.replace(' ', '')
temp_data['cuisines'] = temp_data['cuisines'].str.replace(' ', '')
```

3a. Explain what is the task you're solving (e.g., supervised x unsupervised, classification x regression x clustering or similarity matching x etc.)?

The given problem statement requires the prediction of approximate_cost for two people, based on few features. Since, we already know the label attribute (attribute that needs to be predicted), and that we have the same attribute values for our training as well, we are performing a supervised prediction.

The cost is a continuous value and we consider Regression supervised modeling to predict continuous values.

3b. What models will you choose, and why?

We've considered the following models to perform our regression.

1. Decision Trees Regressor: Decision tree regressor is one of the most popular and efficient predictive modeling techniques for regressions. Few advantages of using this is, it doesn't require much preparation for performing, and any non-linear relationships between the features will not affect its predictions. We are using the weighted mean square error method to choose the nodes and making sure that the model doesn't go too deep and overfit.
2. Random Forest Regressor: Random Forest in an ensemble technique where it trains different decision trees, and combines the output to provide a more general prediction. Using Bagging method, different decision trees are created using different data samples. By combining the results in a meaningful manner, we can gain a model which does not overfit.
3. XGBoost: Extreme Gradient Boosting is a boosting type of ensemble technique which can be implemented on decision trees for better predictions. XGBoost is very fast, avoids overfitting and has features for tuning the model using parameters like 'learning_rate', 'n_estimators' etc. Therefore, XGBoost is a suitable model for this problem.

3c. Which metrics will you use to evaluate your model?

We are using the following three metrics to evaluate our model.

a. Mean Absolute Error – Calculates the average magnitude of absolute error. b. Mean Squared Error – The absolute error is squared, and its average's root is used as score. c. r2_score – Determines how close the data is fitted to the actual regression value. It gives the percentage of the score, giving us an estimate on our model's performance.

3d. How do you make sure to not overfit?

We are using Cross-validation method for training Decision Trees predictive model. We are using 5 folds to get a good model, considering the limitation of computational power, and the size of the cleaned dataset. This will help us in not overfitting the Decision Tree methodology.

Random Forest and XGBoost use Bagging and Boosting techniques, respectively, and therefore, do not overfit their models much. Hence, we aren't using an explicit cross-validation for these two models.

Loading Training and Testing Data | One Hot Encoding

We are loading the feature attributes and label attributes into variables, and performing One Hot Encoding on the categorical attributes

In [0]:
```
# load training and test data
X = temp_data[["location","rest_type","rate","cuisines"]]
Y = temp_data[["cost"]]

location = pd.get_dummies(X['location'],prefix='location')
X = X.drop('location',axis=1)
X = pd.concat([X,location],axis=1)

rest_type = pd.get_dummies(X['rest_type'],prefix='rest')
X = X.drop('rest_type',axis=1)
X = pd.concat([X,rest_type],axis=1)

cuisines = pd.get_dummies(X['cuisines'],prefix='cusine')
X = X.drop('cuisines',axis=1)
X = pd.concat([X,cuisines],axis=1)
```

Conversion

Converting String data types (if any) to float type.

In [344]:
```
X[list(X.columns)] = X[list(X.columns)].astype(float)
Y[list(Y.columns)] = Y[list(Y.columns)].astype(float)
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:3391: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy
  self[k1] = value[k2]
```

Decision Tree Regressor

We have trained the model using a Decision Tree Regressor using the data divided in the earlier steps.

After training and fitting the model, we evaluated it against the following metrics and each metric returned a value.

It was found that Decision Tree returned better performance than Linear Regression and SVR. As Linear Regression and SVR did not suit the problem well, we dropped using those models after a trial.

The model returned a r2_score of 0.73085.

In [329]:

```
# Decision Tree
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.2,ra
ndom_state=0)

from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_validate

from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error

regressor = DecisionTreeRegressor()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

score_r2 = r2_score(y_test, y_pred)
score_mean_squared_error = mean_squared_error(y_test, y_pred)
score_mean_absolute_error = mean_absolute_error(y_test, y_pred)

print(score_r2)
print(score_mean_squared_error)
print(score_mean_absolute_error)
```

```
0.7308575988920678
55420.51909804613
141.19893735822512
```

Random Forest Regressor

We have trained the model using a Random Forest Regressor using the data divided in the earlier steps.

After training and fitting the model, we evaluated it against the following metrics and each metric returned a value.

The model returned a r2_score of 0.76721.

In [330]:
```
# Random forest
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.2,ra
ndom_state=0)

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_validate

from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error

regressor = RandomForestRegressor()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

score_r2 = r2_score(y_test, y_pred)
score_mean_squared_error = mean_squared_error(y_test, y_pred)
score_mean_absolute_error = mean_absolute_error(y_test, y_pred)

print(score_r2)
print(score_mean_squared_error)
print(score_mean_absolute_error)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in
version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples,), for example using ra
vel().
  if sys.path[0] == '':

0.767214115444503
47934.15867457407
138.56744009082092
```

XGBoost Regressor

We have trained the model using a XGBoost Regressor using the data divided in the earlier steps.

After training and fitting the model, we evaluated it against the following metrics and each metric returned a value.

The model returned a r2_score of 0.73035.

In [328]:
```
# XGBoost
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.2,ra
ndom_state=0)

from xgboost import XGBRegressor
from sklearn.model_selection import cross_validate

from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error

regressor = XGBRegressor()
regressor.fit(X_test, y_test)
y_pred = regressor.predict(X_test)

score_r2 = r2_score(y_test, y_pred)
score_mean_squared_error = mean_squared_error(y_test, y_pred)
score_mean_absolute_error = mean_absolute_error(y_test, y_pred)

print(score_r2)
print(score_mean_squared_error)
print(score_mean_absolute_error)
```

```
[00:01:50] WARNING: /workspace/src/objective/regression_obj.cu:152: re
g:linear is now deprecated in favor of reg:squarederror.
0.7303594440475014
55523.09676683196
158.29385547803156
```

Random Forest with Cross Validation

We then used Cross Validation technique to train the Random Forest Regressor, and found performance of each fold. The r2_score of various models generated by different folds are listed below. We have evaluated and found that the r2_score using Cross Validation is better than that of the normal Random Forest Regressor.

In [346]:
```
# cross validation for Random forest
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.2,ra
ndom_state=0)

from sklearn.model_selection import cross_validate
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

from sklearn.metrics import make_scorer
from xgboost import XGBRegressor


regressor = XGBRegressor(learning_rate=0.3,n_estimators=300)
scorer = make_scorer(r2_score)

result = cross_validate(regressor,X_train,y_train,cv = 3,scoring=scorer)
print(result)
```

```
[01:49:41] WARNING: /workspace/src/objective/regression_obj.cu:152: re
g:linear is now deprecated in favor of reg:squarederror.
[01:50:06] WARNING: /workspace/src/objective/regression_obj.cu:152: re
g:linear is now deprecated in favor of reg:squarederror.
[01:50:29] WARNING: /workspace/src/objective/regression_obj.cu:152: re
g:linear is now deprecated in favor of reg:squarederror.
{'fit_time': array([24.38116956, 23.12699437, 21.97568893]), 'score_tim
e': array([0.07738805, 0.0711062 , 0.07295012]), 'test_score': array
([0.78220274, 0.78376558, 0.75871562])}
```

Tuning the XGBoost Model

The XGBoost model can be tuned to perform better, and we tried using GridSearch for it. XGBoost has parameters that can be used to tune the model's performance.

We're considering two major parameters, 'learning_rate' and 'n_estimator'

Learning_rate:

XGBoost models are fast, and can learn things quickly. This can sometimes lead to overfitting. In order to avoid this, we tune the learning_rate parameter. The default value is 0.1, but we can verify for various parameters to know which one gives the best result.

We have tried tuning the model between 6 learning rate values, and determined that the model performs best at learning_rate=0.3 as it has the best r2_score (0.741497).

N Estimator:

(We made a trial and error and got the below results) This parameter determines the number of trees used to estimate (estimators), and usually, the greater number of trees, the better is the performance. But at one time, it reaches saturation, and increasing the value doesn't increase the performance anymore.

We tried performing the prediction with three values, N= 100, N= 200 and N = 1000. There has been a significant increase in the performance as we increased the estimators.

N = 100 -> r2_score = 0.7627591 N = 200 -> r2_score = 0.7753006 N = 1000 -> r2_score = 0.7938902

In [345]:
```
#tuning model
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import LabelEncoder
import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot

label_encoded_y = LabelEncoder().fit_transform(Y)
model = XGBRegressor()
learning_rate = [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3]
param_grid = dict(learning_rate=learning_rate)
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=7)
grid_search = GridSearchCV(model, param_grid, n_jobs=-1, cv=kfold)
grid_result = grid_search.fit(X, label_encoded_y)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_p
arams_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
        print("%f (%f) with: %r" % (mean, stdev, param))
# plot
pyplot.errorbar(learning_rate, means, yerr=stds)
pyplot.title("XGBoost learning_rate vs Log Loss")
pyplot.xlabel('learning_rate')
pyplot.ylabel('Log Loss')
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/label.py:2
35: DataConversionWarning: A column-vector y was passed when a 1d array
was expected. Please change the shape of y to (n_samples, ), for exampl
e using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.p
y:657: Warning: The least populated class in y has only 1 members, whic
h is too few. The minimum number of members in any class cannot be less
than n_splits=10.
  % (min_groups, self.n_splits)), Warning)

[01:47:43] WARNING: /workspace/src/objective/regression_obj.cu:152: re
g:linear is now deprecated in favor of reg:squarederror.
Best: 0.741497 using {'learning_rate': 0.3}
-5.251274 (0.076278) with: {'learning_rate': 0.0001}
-4.312741 (0.061272) with: {'learning_rate': 0.001}
-0.293461 (0.018149) with: {'learning_rate': 0.01}
0.705897 (0.008327) with: {'learning_rate': 0.1}
0.730556 (0.008113) with: {'learning_rate': 0.2}
0.741497 (0.008865) with: {'learning_rate': 0.3}
```

Out[345]:
```
Text(0, 0.5, 'Log Loss')
```

Conclusion

Initially, we tried using the Decision Tree Regressor. And after using the ensemble method, Random Forest Regressor, we found that Random Forest gave better results than Decision Tree. We also tried XGBoost but it did not yield better results than Random Forest.

But XGBoost is highly robust to overfitting. Hence, we tried to tune its performance by scanning the hyper parameters using GridSearch. Using the parameters learning_rate and n_estimators, we tuned the performance successfully.

We also used Cross Validation on XGBoost and have provided the results above. Each fold had a decent r2_score (around 0.78) as seen above.

References

[1] J. Brownlee, "Tune Learning Rate for Gradient Boosting with XGBoost in Python", Machine Learning Mastery, 2019. [Online]. Available: https://machinelearningmastery.com/tune-learning-rate-for-gradient-boosting-with-xgboost-in-python/ (https://machinelearningmastery.com/tune-learning-rate-for-gradient-boosting-with-xgboost-in-python/). [Accessed: 03- Jul- 2019].

[2]"Understanding Random Forest", Towards Data Science, 2019. [Online]. Available: https://towardsdatascience.com/understanding-random-forest-58381e0602d2 (https://towardsdatascience.com/understanding-random-forest-58381e0602d2). [Accessed: 03- Jul- 2019].

[3]"API Reference — scikit-learn 0.21.2 documentation", Scikit-learn.org, 2019. [Online]. Available: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics). [Accessed: 03- Jul- 2019].

[4]"scikit-learn: machine learning in Python — scikit-learn 0.21.2 documentation", Scikit-learn.org, 2019. [Online]. Available: https://scikit-learn.org/stable/ (https://scikit-learn.org/stable/). [Accessed: 03- Jul- 2019].

[5]"3.2. Tuning the hyper-parameters of an estimator — scikit-learn 0.21.2 documentation", Scikit-learn.org, 2019. [Online]. Available: https://scikit-learn.org/stable/modules/grid_search.html#multimetric-grid-search (https://scikit-learn.org/stable/modules/grid_search.html#multimetric-grid-search). [Accessed: 03- Jul- 2019].

[6]Z. Z., "How to use pd.get_dummies() with the test set - FastML", Fastml.com, 2019. [Online]. Available: http://fastml.com/how-to-use-pd-dot-get-dummies-with-the-test-set/ (http://fastml.com/how-to-use-pd-dot-get-dummies-with-the-test-set/). [Accessed: 03- Jul- 2019].

[7]M. Editor, "How to Choose the Best Regression Model", Blog.minitab.com, 2019. [Online]. Available: https://blog.minitab.com/blog/how-to-choose-the-best-regression-model (https://blog.minitab.com/blog/how-to-choose-the-best-regression-model). [Accessed: 03- Jul- 2019].

[8]"Zomato Data Exploration and Visualization | Kaggle", Kaggle.com, 2019. [Online]. Available: https://www.kaggle.com/agnihotri/zomato-data-exploration-and-visualization (https://www.kaggle.com/agnihotri/zomato-data-exploration-and-visualization). [Accessed: 03- Jul- 2019].

[9]H. Poddar, "Zomato Bangalore Restaurants", Kaggle.com, 2019. [Online]. Available: https://www.kaggle.com/himanshupoddar/zomato-bangalore-restaurants (https://www.kaggle.com/himanshupoddar/zomato-bangalore-restaurants). [Accessed: 03- Jul- 2019].