Georges Essoh

Case Study: Create Optimal Hotel Recommendation

Date: 10/17/2021

Introduction

The goal here is to conduct exploratory data analysis, perform variable engineering and test several models to choose the most optimal model and even make the best recommendations

1) Data load

For the sake of performance, only the first 100,000 rows of the data set will be used. We can see that our train data set contains 24 variables.

```python
# Load the datasets
train =  pd.read_csv('train.csv', nrows = 100000)
test = pd.read_csv('test.csv', nrows = 100000)
destinations = pd.read_csv('destinations.csv', nrows = 100000)
#train = train.sample(frac=0.01, random_state=99)
#test = test.sample(frac=0.01, random_state=99)
#destinations = destinations.sample(frac=0.01, random_state=99)
train.shape
```

```
(100000, 24)
```

2) Exploratory Data Analysis

- Data information

```python
# data info
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 24 columns):
 #   Column                     Non-Null Count   Dtype  
---  ------                     --------------   -----  
 0   date_time                  100000 non-null  object 
 1   site_name                  100000 non-null  int64  
 2   posa_continent             100000 non-null  int64  
 3   user_location_country      100000 non-null  int64  
 4   user_location_region       100000 non-null  int64  
 5   user_location_city         100000 non-null  int64  
 6   orig_destination_distance  63078 non-null   float64
 7   user_id                    100000 non-null  int64  
 8   is_mobile                  100000 non-null  int64  
 9   is_package                 100000 non-null  int64  
 10  channel                    100000 non-null  int64  
 11  srch_ci                    99929 non-null   object 
 12  srch_co                    99929 non-null   object 
 13  srch_adults_cnt            100000 non-null  int64  
 14  srch_children_cnt          100000 non-null  int64  
 15  srch_rm_cnt                100000 non-null  int64  
 16  srch_destination_id        100000 non-null  int64  
 17  srch_destination_type_id   100000 non-null  int64  
 18  is_booking                 100000 non-null  int64  
 19  cnt                        100000 non-null  int64  
 20  hotel_continent            100000 non-null  int64  
 21  hotel_country              100000 non-null  int64  
 22  hotel_market               100000 non-null  int64  
 23  hotel_cluster              100000 non-null  int64  
dtypes: float64(1), int64(20), object(3)
memory usage: 18.3+ MB
```

Our train dataset mostly includes continuous variables (21/24 in total), and the rest of the columns are time data. Some of the variables in our dataset contain null or missing data. This will be addressed in the continuation of our explorative analysis.

- Data description

```python
#data description
train.describe(include='all')
```

	site_name	posa_continent	user_location_country	user_location_region	user_location_city	orig_destination_distance	user_id	is_mobile
count	100000.00000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000
mean	9.10014	2.637850	84.531040	311.630930	28465.223540	1897.609161	195700.878390	0.138030
std	12.09091	0.751001	54.320574	209.399151	16822.922817	1686.819919	110173.879786	0.344933
min	2.00000	0.000000	0.000000	0.000000	3.000000	0.005600	12.000000	0.000000
25%	2.00000	3.000000	66.000000	174.000000	13914.000000	725.696800	107548.000000	0.000000
50%	2.00000	3.000000	66.000000	311.000000	27733.000000	1897.609161	181983.000000	0.000000
75%	11.00000	3.000000	69.000000	385.000000	43113.000000	1897.609161	301357.000000	0.000000
max	53.00000	4.000000	239.000000	1025.000000	56495.000000	11641.224200	391007.000000	1.000000

8 rows × 27 columns

- Dealing with time data

```
# function to process time varaibles
from datetime import datetime
def get_year(x):
    if x is not None and type(x) is not float:
        try:
            return datetime.strptime(x, '%Y-%m-%d').year
        except ValueError:
            return datetime.strptime(x, '%Y-%m-%d %H:%M:%S').year
    else:
        return 2013
    pass
def get_month(x):
    if x is not None and type(x) is not float:
        try:
            return datetime.strptime(x, '%Y-%m-%d').month
        except:
            return datetime.strptime(x, '%Y-%m-%d %H:%M:%S').month
    else:
        return 1
    pass

def left_merge_dataset(left_dframe, right_dframe, merge_column):
    return pd.merge(left_dframe, right_dframe, on=merge_column, how='left')
```

```
# process date_time column
train['date_time_year'] = pd.Series(train.date_time, index = train.index)
train['date_time_month'] = pd.Series(train.date_time, index = train.index)
from datetime import datetime
train.date_time_year = train.date_time_year.apply(lambda x: get_year(x))
train.date_time_month = train.date_time_month.apply(lambda x: get_month(x))
del train['date_time']
```

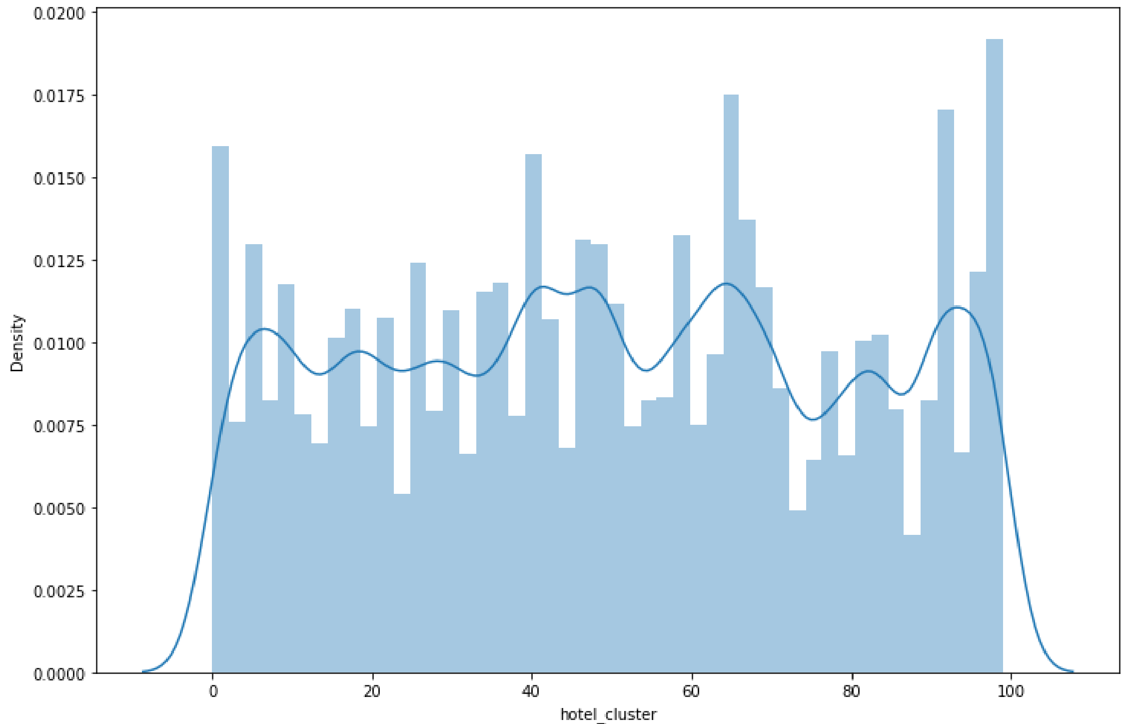
```
# process srch_ci column
train['srch_ci_year'] = pd.Series(train.srch_ci, index=train.index)
train['srch_ci_month'] = pd.Series(train.srch_ci, index=train.index)
# convert year & months to int
train.srch_ci_year = train.srch_ci_year.apply(lambda x: get_year(x))
train.srch_ci_month = train.srch_ci_month.apply(lambda x: get_month(x))
# remove the srch_ci column
del train['srch_ci']
```

```
# process srch-co column
train['srch_co_year'] = pd.Series(train.srch_co, index=train.index)
train['srch_co_month'] = pd.Series(train.srch_co, index=train.index)
# convert year & months to int
train.srch_co_year = train.srch_co_year.apply(lambda x: get_year(x))
train.srch_co_month = train.srch_co_month.apply(lambda x: get_month(x))
# remove the srch_co column
del train['srch_co']
```

- Target variable (hotel cluster) histogram

```python
# histogram of hotel cluster variable
plt.figure(figsize = (12,8))
sns.distplot(train['hotel_cluster'])
```

```
<AxesSubplot:xlabel='hotel_cluster', ylabel='Density'>
```



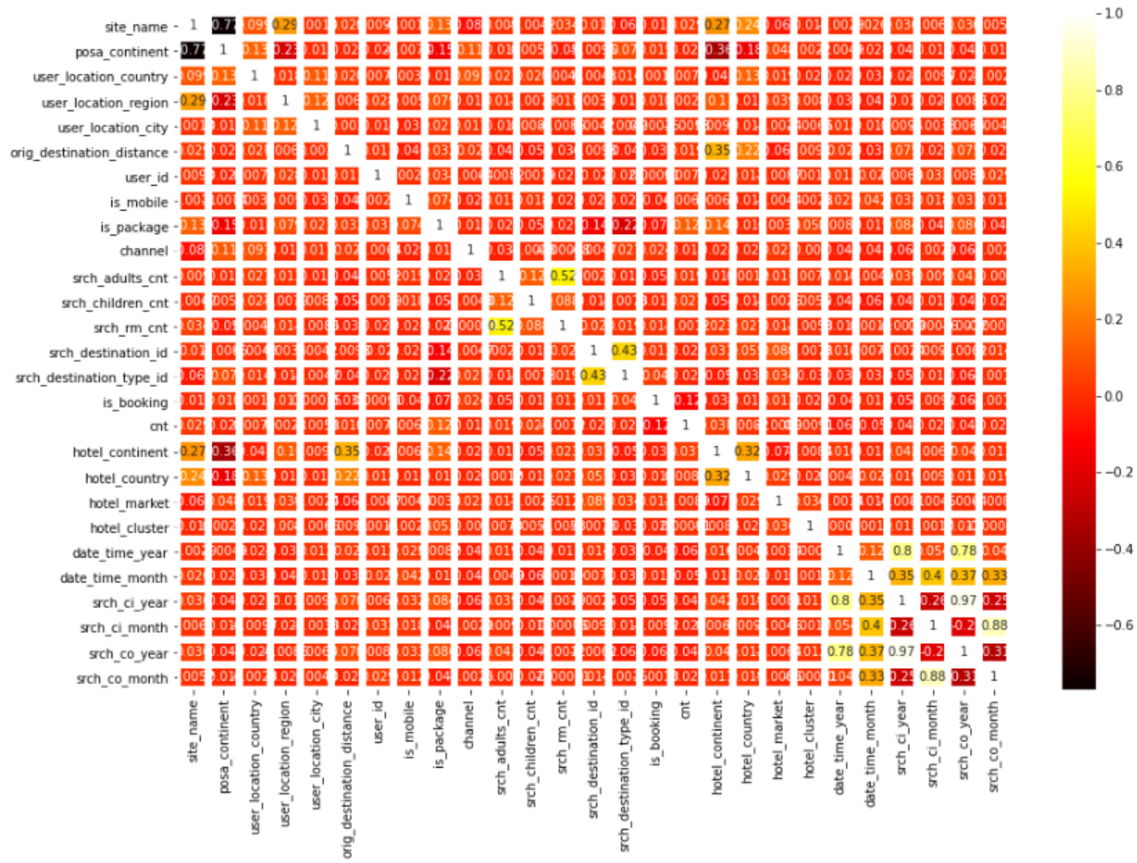
In the light of this histogram, it turns out that we, therefore, have a classification problem, more precisely a multi-class problem for which we have 100 classes as indicated in the histogram above.

3) Preliminary analysis

- Heat map

```python
#heat map
fig, ax = plt.subplots()
fig.set_size_inches(15,10)
sns.heatmap(train.corr(),cmap='hot',ax=ax,annot=True,linewidths=5)
```

<AxesSubplot:>



- Correlation coefficients

```python
#correlation coefficients
train.corr()['hotel_cluster'].sort_values()
```

```
srch_destination_type_id    -0.030064
is_booking                  -0.025380
user_location_country       -0.025170
hotel_country               -0.021170
site_name                   -0.019154
hotel_continent             -0.008441
srch_destination_id         -0.007258
user_location_city          -0.006580
channel                     -0.005956
srch_rm_cnt                 -0.005801
is_mobile                   -0.002803
srch_co_month               -0.000413
date_time_year               0.000696
cnt                          0.000914
date_time_month              0.001208
user_id                      0.001396
srch_ci_month                0.001682
posa_continent               0.002204
srch_children_cnt            0.005469
srch_adults_cnt              0.007322
user_location_region         0.008037
orig_destination_distance    0.009103
srch_ci_year                 0.010879
srch_co_year                 0.012014
hotel_market                 0.036107
is_package                   0.051955
hotel_cluster                1.000000
Name: hotel_cluster, dtype: float64
```

There is no strong correlation between the other variables and the hotel cluster variable. Therefore, there is no strong relationship between any variable in this dataset and the hotel cluster, suggesting that a linear regression model will be inconclusive. Because none of the variables stand out due to a strong correlation coefficient with the target variable, we will have to adopt a more intuitive approach to choose the variables of importance in this situation. The variables retained are therefore as follows:

- srch_destination_id - ID of the destination where the hotel search was performed
- hotel_country - Country where the hotel is located
- hotel_market - Hotel market
- hotel_cluster - ID of a hotel cluster
- is_package - Whether part of a package or not (1/0)
- is_booking - Booking (1) or Click (0)

In addition, we are collecting the data for the rows where the variable is_booking == 1 seems to be a good idea since we are only interested in reservations.

```
# Selecting rows for is8booking == 1
train1 = train.loc[train['is_booking'] == 1]
```

```
# Let's not merge data from train and destination where is_booking == 1
df = pd.merge(train1,destinations, on='srch_destination_id')
df.head()
```

	site_name	posa_continent	user_location_country	user_location_region	user_location_city	orig_destination_distance	user_id	is_mobile	is_package	channel
0	2	3	66	348	48862	2234.264100	12	0	1	9
1	13	1	46	171	15334	5655.315900	4539	0	0	9
2	2	3	66	356	4751	762.231500	6258	0	1	9
3	24	2	3	51	41641	1897.609161	14117	0	0	9
4	11	3	205	155	14703	996.381100	18999	0	1	9

5 rows × 176 columns

- Models

Split train into a training and test set

```
X = df[df.columns.difference(['user_id', 'hotel_cluster', 'is_booking'])]
y = df.hotel_cluster
X.shape, y.shape
```

```
((8235, 173), (8235,))
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(6176, 173) (6176,)
(2059, 173) (2059,)
```

Check if all the clusters are present in the training

```
y.nunique()
```

```
100
```

Support vector machine

```
classifier = make_pipeline(preprocessing.StandardScaler(), svm.SVC(decision_function_shape='ovo'))
np.mean(cross_val_score(classifier, X, y, cv=10))
```

```
0.09338747065554626
```

Naive Bayes

```
classifier = make_pipeline(preprocessing.StandardScaler(), GaussianNB(priors=None))
np.mean(cross_val_score(classifier, X, y, cv=10))
```

```
0.051365475586594156
```

Random Classifier

```
clf = make_pipeline(preprocessing.StandardScaler(), RandomForestClassifier(n_estimators=173,max_depth=10,random_state=0))
np.mean(cross_val_score(clf, X, y, cv=10))
```

```
0.09205192936096922
```

K Nearest Neighbor

```
classifier = make_pipeline(preprocessing.StandardScaler(), KNeighborsClassifier(n_neighbors=5))
np.mean(cross_val_score(classifier, X, y, cv=10, scoring='accuracy'))
```

```
0.05975268081492054
```

Conclusion
To conclude, none of the four models chosen based on the preliminary analysis results performed well. This is proof that we need to do more feature engineering to improve the result.

From the above algorithms, SVM performed the best. Yet, the cross-validation score is only 0.09, Which is poor.

Support vector machine (SVM) is a logical choice because this algorithm is capable of classification and regression. Here we have a multi-class (100) problem. SVM brings the benefit of being able to capture more complex relationships between data points.

Naive Bayes was chosen for its simplicity and speed of convergence. However, unfortunately, it is one of all models that perform the worst. Its simplistic approach is not appropriate in this situation.

A random forest is simply a collection of decision trees whose results are aggregated into one result. Their ability to limit overfitting without substantially increasing error due to bias is why they are such powerful models. With a score close to that of SVM, it comes in the second position.

K-Nearest Neighbor classifier KNN is a non-parametric, lazy learning algorithm. KNN performed very similar to Naive Bayes for the model in question.

SVM took very long to converge, but we achieved a much better result.