Assignment 5.4 - Hotel Recommendation Modeling - Python

July 8, 2020

#
Assignment 5.3
Create Optimal Hotel Recommandation
Shani Kumar Bellevue University Week 5 – 5.3

0.0.1 Introduction:

All online travel agencies are scrambling to meet the Artificial Intelligence driven personalization standard set by Amazon and Netflix. In addition, the world of online travel has become a highly competitive space where brands try to capture our attention (and wallet) with recommending, comparing, matching, and sharing. For this assignment, we aim to create the optimal hotel recommendations for Expedia's users that are searching for a hotel to book. For this assignment, you need to predict which "hotel cluster" the user is likely to book, given his (or her) search details. In doing so, you should be able to demonstrate your ability to use four different algorithms (of your choice). The data set can be found at Kaggle: Expedia Hotel Recommendations. To get you started, I would suggest you use train.csv which captured the logs of user behavior and destinations.csv which contains information related to hotel reviews made by users. You are also required to write a one page summary of your approach in getting to your prediction methods. I expect you to use a combination of R and Python in your answer.

0.0.2 Source Data

https://www.kaggle.com/c/expedia-hotel-recommendations/data
train.csv - contains training set
test.csv - contains test set
destinations.csv - hotel search latent attributes

[43]:
```
import pandas as pd
import pandas_profiling as pp
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import warnings
from datetime import datetime
from sklearn import svm
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn import preprocessing
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestRegressor

# configure display of graph
%matplotlib inline

#stop unnecessary warnings from printing to the screen
warnings.simplefilter('ignore')
```

0.0.3 Load data into a dataframe

[11]:
```
#load training data
trn = pd.read_csv('data/train.csv', nrows = 100000)

# load test data
tst = pd.read_csv('./data/test.csv', nrows=100000)

# load destination data
dst = pd.read_csv('data/destinations.csv', nrows=100000)
```

0.0.4 Exploratory Data Analysis

[3]:
```
# eda of training data
pp.ProfileReport(trn)
```

[3]:
```
<pandas_profiling.ProfileReport at 0x7fde21efd518>
```

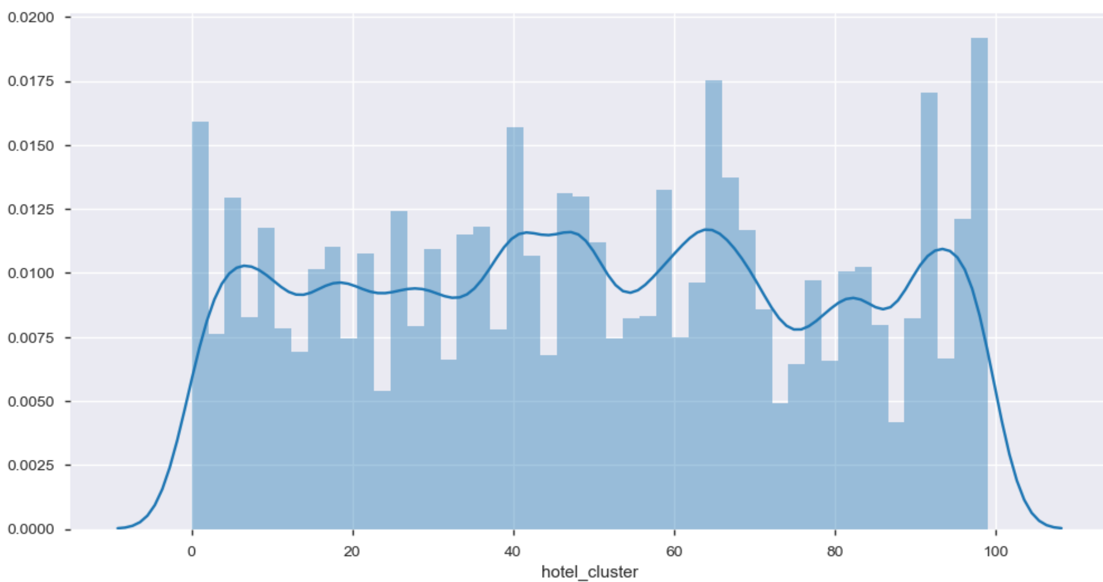
0.0.5 Observations based on exploratory analysis

- Channel count is increasing year over year.
- no features are strongly correlated with the target variable (hotel_cluster). This suggests that a linear method would perform poorly.
- There are 100 different clusters in target variable (hotel_cluster).
- There are two distinct values for **is_booking**, 0 and 1; where 1 if a booking, 0 if a click.
- **is_mobile** is 1 if booked through mobile phone, 0 if not.
- **is_package** is 1 if booked as part of a package, 0 if not.
- There are some date fields which will need some work to extract date, time, hour and weekday details.

0.0.6 See if there is any skewness in target data class

[6]:
```
# histogram of clusters
plt.figure(figsize=(12, 6))
sns.distplot(trn['hotel_cluster'])
```

[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fde0dc16f98>
```



The data is pretty much well distributed over all 100 clusters and no skewness in the data.

0.0.7 Extract Year and Montn details from date fields

The following date columns which needs some extra treatment

- **date_time** - Timestamp
- **srch_ci** - Checkin date
- **srch_co** - Checkout date

[12]:
```
# Extract year part from a date

def fetch_year(x):
    '''
    Args:
        datetime
    Returns:
        year as numeric
    '''
    if x is not None and type(x) is not float:
        try:
```

```python
            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

# Extract month part from a date

def fetch_month(x):
    '''
    Args:
        datetime
    Returns:
        month as numeric
    '''
    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
```

[13]:
```python
# extract year and month from date time column
trn['date_time_year'] = pd.Series(trn.date_time, index = trn.index)
trn['date_time_month'] = pd.Series(trn.date_time, index = trn.index)

trn.date_time_year = trn.date_time_year.apply(lambda x: fetch_year(x))
trn.date_time_month = trn.date_time_month.apply(lambda x: fetch_month(x))
del trn['date_time']
```

[14]:
```python
# extract year and month from check in date column
trn['srch_ci_year'] = pd.Series(trn.srch_ci, index = trn.index)
trn['srch_ci_month'] = pd.Series(trn.srch_ci, index = trn.index)

trn.srch_ci_year = trn.srch_ci_year.apply(lambda x: fetch_year(x))
trn.srch_ci_month = trn.srch_ci_month.apply(lambda x: fetch_month(x))
del trn['srch_ci']
```

[15]:
```python
# extract year and month from check out date column
trn['srch_co_year'] = pd.Series(trn.srch_co, index = trn.index)
trn['srch_co_month'] = pd.Series(trn.srch_co, index = trn.index)

trn.srch_co_year = trn.srch_co_year.apply(lambda x: fetch_year(x))
trn.srch_co_month = trn.srch_co_month.apply(lambda x: fetch_month(x))
del trn['srch_co']
```

[15]:
```
   site_name  posa_continent  user_location_country  user_location_region  \
0          2               3                     66                   348
1          2               3                     66                   348
2          2               3                     66                   348
3          2               3                     66                   442
4          2               3                     66                   442

   user_location_city  orig_destination_distance  user_id  is_mobile  \
0               48862                  2234.2641       12          0
1               48862                  2234.2641       12          0
2               48862                  2234.2641       12          0
3               35390                   913.1932       93          0
4               35390                   913.6259       93          0

   is_package  channel  ...  hotel_continent  hotel_country  hotel_market  \
0           1        9  ...                2             50           628
1           1        9  ...                2             50           628
2           0        9  ...                2             50           628
3           0        3  ...                2             50          1457
4           0        3  ...                2             50          1457

   hotel_cluster  date_time_year  date_time_month  srch_ci_year  \
0              1            2014                8          2014
1              1            2014                8          2014
2              1            2014                8          2014
3             80            2014                8          2014
4             21            2014                8          2014

   srch_ci_month  srch_co_year  srch_co_month
0              8          2014              8
1              8          2014              9
2              8          2014              9
3             11          2014             11
4             11          2014             11

[5 rows x 27 columns]
```

[16]:
```
# check the transformed data
trn.head()
```

[16]:
```
   site_name  posa_continent  user_location_country  user_location_region  \
0          2               3                     66                   348
1          2               3                     66                   348
2          2               3                     66                   348
3          2               3                     66                   442
4          2               3                     66                   442

   user_location_city  orig_destination_distance  user_id  is_mobile  \
```

```
0              48862                   2234.2641       12           0
1              48862                   2234.2641       12           0
2              48862                   2234.2641       12           0
3              35390                    913.1932       93           0
4              35390                    913.6259       93           0

   is_package  channel  ...  hotel_continent  hotel_country  hotel_market  \
0           1        9  ...                2             50           628
1           1        9  ...                2             50           628
2           0        9  ...                2             50           628
3           0        3  ...                2             50          1457
4           0        3  ...                2             50          1457

   hotel_cluster  date_time_year  date_time_month  srch_ci_year  \
0              1            2014                8          2014
1              1            2014                8          2014
2              1            2014                8          2014
3             80            2014                8          2014
4             21            2014                8          2014

   srch_ci_month  srch_co_year  srch_co_month
0              8          2014              8
1              8          2014              9
2              8          2014              9
3             11          2014             11
4             11          2014             11

[5 rows x 27 columns]
```

0.1 Modeling

For known combinations of user location cities, origin-destination distances and search destinations, will definitely help finding hotel cluster.

[19]:
```
pieces = [trn.
  ↪groupby(['srch_destination_id','hotel_country','hotel_market','hotel_cluster'])['is_booking
  ↪agg(['sum','count'])]
agg = pd.concat(pieces).groupby(level=[0,1,2,3]).sum()
agg.dropna(inplace=True)
```

[20]:
```
agg.head()
```

[20]:
```
                                                                   sum  count
srch_destination_id hotel_country hotel_market hotel_cluster
8                   50            416          32                    1      2
                                               60                    0      1
                                               77                    1      2
11                  50            824          94                    1      2
```

```
14                      27              1434            20                  1       3
```

[21]:
```
agg['sum_and_cnt'] = 0.85*agg['sum'] + 0.15*agg['count']
agg = agg.groupby(level=[0,1,2]).apply(lambda x: x.astype(float)/x.sum())
agg.reset_index(inplace=True)
```

[22]:
```
agg.head()
```

[22]:
```
   srch_destination_id  hotel_country  hotel_market  hotel_cluster  sum  \
0                    8             50           416             32  0.5
1                    8             50           416             60  0.0
2                    8             50           416             77  0.5
3                   11             50           824             94  1.0
4                   14             27          1434             20  1.0

   count  sum_and_cnt
0    0.4     0.469388
1    0.2     0.061224
2    0.4     0.469388
3    1.0     1.000000
4    0.6     0.812500
```

[23]:
```
agg_pivot = agg.
 ↪pivot_table(index=['srch_destination_id','hotel_country','hotel_market'],
                                    columns='hotel_cluster', values='sum_and_cnt').
 ↪reset_index()
```

[24]:
```
agg_pivot.head()
```

[24]:
```
hotel_cluster  srch_destination_id  hotel_country  hotel_market   0   1   2  \
0                                8             50           416 NaN NaN NaN
1                               11             50           824 NaN NaN NaN
2                               14             27          1434 NaN NaN NaN
3                               16             50           419 NaN NaN NaN
4                               19            102          1522 NaN NaN NaN

hotel_cluster   3   4   5   6  ...  90  91  92  93   94  95  96  97  98  99
0             NaN NaN NaN NaN  ... NaN NaN NaN NaN  NaN NaN NaN NaN NaN NaN
1             NaN NaN NaN NaN  ... NaN NaN NaN NaN  1.0 NaN NaN NaN NaN NaN
2             NaN NaN NaN NaN  ... NaN NaN NaN NaN  NaN NaN NaN NaN NaN NaN
3             NaN NaN NaN NaN  ... NaN NaN NaN NaN  NaN NaN NaN NaN NaN NaN
4             NaN NaN NaN NaN  ... NaN NaN NaN NaN  NaN NaN NaN NaN NaN NaN

[5 rows x 103 columns]
```

[27]:
```
trn = pd.merge(trn, dst, how='left', on='srch_destination_id')
trn = pd.merge(trn, agg_pivot, how='left',␣
 ↪on=['srch_destination_id','hotel_country','hotel_market'])
```

[18]:
```
trn_book = trn.loc[trn['is_booking'] == 1]
```

[31]:
```
trn.fillna(0, inplace=True)
trn.shape
```

[31]:
```
(100000, 276)
```

Create a pivot to map each cluster, and shape it accordingly so that it can be merged with the original data.

[ ]:
```
# step 1
factors = [train_book.
 ↪groupby(['srch_destination_id','hotel_country','hotel_market','is_package','hotel_cluster'])['is_booking'].
 ↪agg(['sum','count'])]
summ = pd.concat(factors).groupby(level=[0,1,2,3,4]).sum()
summ.dropna(inplace=True)
summ.head()
```

[ ]:
```
# step 2
summ['sum_and_cnt'] = 0.85*summ['sum'] + 0.15*summ['count']
summ = summ.groupby(level=[0,1,2,3]).apply(lambda x: x.astype(float)/x.sum())
summ.reset_index(inplace=True)
summ.head()
```

[ ]:
```
# step 3
summ_pivot = summ.
 ↪pivot_table(index=['srch_destination_id','hotel_country','hotel_market','is_package'],␣
 ↪columns='hotel_cluster', values='sum_and_cnt').reset_index()
summ_pivot.head()
```

Quickly check the destination data to determine the relationship with other data.

[ ]:
```
destination.head()
```

Merge the filtered booking data, pivotted data and destination data to form a single wide dataset.

[ ]:
```
train_book = pd.merge(train_book, destination, how='left',␣
 ↪on='srch_destination_id')
train_book = pd.merge(train_book, summ_pivot, how='left',␣
 ↪on=['srch_destination_id','hotel_country','hotel_market','is_package'])
train_book.fillna(0, inplace=True)
train_book.shape
```

[ ]:
```
train_book.head()
```

Since we are only interested in booking events, let us get rid of clicks.

[33]:
```
trn = trn.loc[trn['is_booking'] == 1]
```

[34]:
```
X = trn.drop(['user_id', 'hotel_cluster', 'is_booking'], axis=1)
y = trn.hotel_cluster
X.shape, y.shape
```

[34]:
```
((8270, 273), (8270,))
```

Check if all of the 100 clusters are present in the training data.

[35]:
```
y.nunique()
```

[35]: 100

0.1.1 1. Support Vector Machine (SVM)

[36]:
```
classifier = make_pipeline(preprocessing.StandardScaler(), svm.
 ↪SVC(decision_function_shape='ovo'))
np.mean(cross_val_score(classifier, X, y, cv=10))
```

[36]: 0.39911816734349126

0.1.2 2. Naive Bayes classifier

[37]:
```
classifier = make_pipeline(preprocessing.StandardScaler(),␣
 ↪GaussianNB(priors=None))
np.mean(cross_val_score(classifier, X, y, cv=10))
```

[37]: 0.12789974331138057

0.1.3 3. Logistic Regression

[38]:
```
classifier = make_pipeline(preprocessing.StandardScaler(),␣
 ↪LogisticRegression(multi_class='ovr'))
np.mean(cross_val_score(classifier, X, y, cv=10))
```

[38]: 0.35424495086296637

0.1.4 4. K-Nearest Neighbor classifier

[39]:
```
classifier = make_pipeline(preprocessing.StandardScaler(),␣
 ↪KNeighborsClassifier(n_neighbors=5))
np.mean(cross_val_score(classifier, X, y, cv=10, scoring='accuracy'))
```

[39]: 0.33520784873335174

[47]:
```
classifier = make_pipeline(preprocessing.StandardScaler(),
                           RandomForestRegressor(n_estimators=100))
np.mean(cross_val_score(classifier, X, y, cv=10))
```

[47]: 0.13409245219570218

[40]:
```
%%html
<style>
table {float:left}
</style>
```

<IPython.core.display.HTML object>

0.2 Result Summary

Model	Description	Results
Support Vector Machine (SVM)	Works well with large dataset and catagorical values	0.399
Naive Bayes Classifier	Simple and relatively Fast Model	0.127
Logistic Regression	Just to see performance with uncorrelated data	0.354
KNN Classifier	Simple and relatively fast Model	0.335
Random Forest	Easy to tune but not good for last cluster with huge data	0.134

From my analysis SVM provided best result and Naive Bayes classifier performed worst.

Final numbers are low but this is because I did not do exdensive analysis and data cleanup and feature extraction and exclusion. Permforming more analysis work should increase the final result.

Also I can see scores are much lower than those on Kaggle, which is expected because we tried to fit our models without exploiting the data leak. On Kaggle, people were able predict of the data 90% of the time just from using the data leak.

0.3 Summary

Here are the basis of the selection.

1. Support Vector Machine (SVM) Support Vector Machines (SVMs) is good at finding pairwise interactions in data, so I through it should be good at recommending a hotel cluster to a certain user. Though natively, they don't support multiclass classification, there are techniques we can use to it viably classify one out of 100 hotel clusters.

The benefit is that we can capture much more complex relationships between the datapoints without having to perform difficult transformations on our own. The downside is that the training time is much longer as it's much more computationally intensive.

It provided highest & best cross validation score.

2. Naive Bayes classifier Naive Bayes is a relatively simple classifier that can natively be run on multiclass data, so I thought at least try this type of classifier and see what kind of results we get initially.

But it has the worst performance of the four models. Therefore, this classifier is not recommended for the problem at hand.

3. Logistic Regression We chose to test Logistic Regression because it was a simple model we learned in class that can handle non-linear decision boundaries, which we were clearly dealing with.

Logistic Regression was close to the performance of SVM but slightly worse.

4. K-Nearest Neighbor classifier KNN was a good simple model to try because it 'trains' very quickly by offsetting most of the computation to the actual testing portion. Additionally it is relatively intuitive how the model works.

5. Random Forest classifier One of the largest weakness of Random Forest Classifiers is large class imbalances. In addition with many classes, the decision trees become very deep and complex which goes against the marginal differences that RF averaging is going for.

KNN performed very similar to Logistic Regression for the model in question but it was much faster than Logistic Regression

[ ]:
```

```