source: Expedia

A Machine Learning Approach — Building a Hotel Recommendation Engine

AI driven personalization, travel recommendation

Susan Li Aug 13, 2018 · 5 min read

All online travel agencies are scrambling to meet the AI driven personalization standard set by Amazon and Netflix. In addition, the comparing, matching and sharing.

In this post, we aim to create the optimal hotel recommendations for Expedia's users that are searching for a hotel to book. We will model this problem as a multi-class classification problem and build SVM and decision tree in ensemble method to predict which "hotel cluster" the user is likely to book, given his (or her) search details.

The Data

The data is anonymized and almost all the fields are in numeric format. The data set can be found at Kaggle, we will use train.csv which captured the logs of user behavior, and destinations.csv which contains information related to hotel reviews made by users.

The Figure 1 below provides the schema of the train.csv:

Feature Name	Feature Description	Feature Data Type
date_time	Timestamp	string
site_name	ID of Expedia point of sale	int
posa_continent	ID of continent associate with site name	int
user_location_country	the ID of the country the user is located	int
user_location_region	the ID of the region the user is located	int
user_location_city	the ID of the city the user is located	int
orig_destination_distance	physical distance between a hotel and a customer at the time of search	double
user_id	ID of user	int
is_mobile	1 when a user connected from a mobile device, 0 otherwise	tinyint
is_package	1 if the click/booking was generated as part of a package, 0 otherwise	int
channel	ID of a marketing channel	int
srch_ci	Checkin date	string
srch_co	Checkout date	string
srch_adults_cnt	The number of adults specified in the hotel room	int

srch_destination_id	ID of the destination where the hotel search was performed	int
srch_destination_type_id	Type of destination	int
hotel_continent	Hotel continent	int
hotel_country	Hotel country	int
hotel_market	Hotel market	int
cnt	Number of similar events in the context of the same user session	int
is_booking	1 if a booking, 0 if a click	int
hotel_cluster	ID of a hotel cluster - ***This is what we are going to predict***	bigint

Figure 1

The Figure 2 below provides the schema of the destinations.csv:

Feature Name	Feature Description	Feature Data Type
srch_destination_id	ID of the destination where the hotel search was performed	int
d1-d149	latent description of search regions	double

Figure 2

```
import datetime
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import make_pipeline
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn import svm
```

Open in app

```
df = pd.read_csv('train.csv.gz', sep=',').dropna()
dest = pd.read_csv('destinations.csv.gz')
df = df.sample(frac=0.01, random_state=99)
df.shape
```

(241179, 24)

EDA

The objective is to predict which hotel_cluster a user will book given the information in his (or her) search. There are 100 clusters in total. In another word, we are dealing with a 100 class classification problem.

```
plt.figure(figsize=(12, 6))
sns.distplot(df['hotel_cluster'])
```

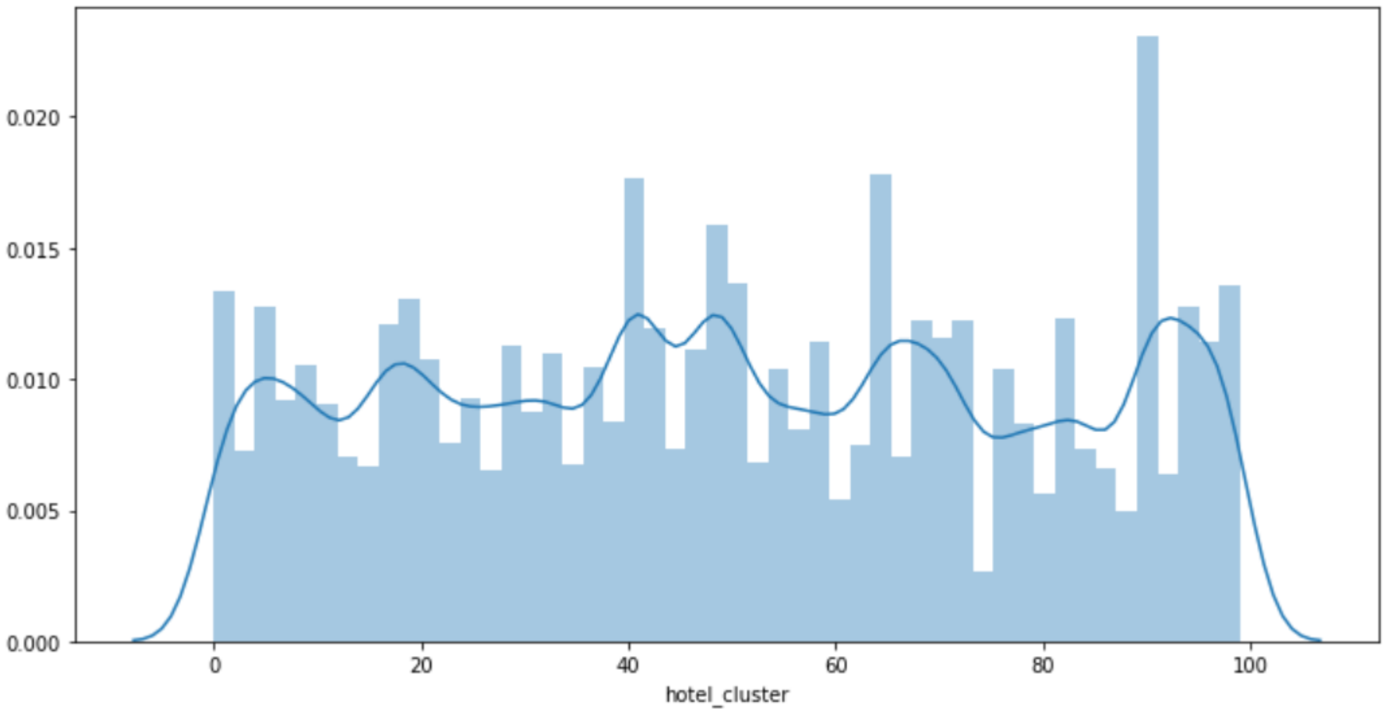

Figure 3

is skewness in the data.

Feature Engineering

The date time, checkin date and checkout date columns can not be used directly, we will extract year and month from them. First, we define a couple of functions to achieve that, and we also define a function to merge with destination.csv.

```
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')
```

Open in app

```
df['date_time_year'] = pd.Series(df.date_time, index = 
df.index)
df['date_time_month'] = pd.Series(df.date_time, index = 
df.index)

from datetime import datetime
df.date_time_year = df.date_time_year.apply(lambda x: 
get_year(x))
df.date_time_month = df.date_time_month.apply(lambda x: 
get_month(x))

del df['date_time']
```

Dealing with srch_ci column:

```
df['srch_ci_year'] = pd.Series(df.srch_ci, 
index=df.index)
df['srch_ci_month'] = pd.Series(df.srch_ci, 
index=df.index)

# convert year & months to int
df.srch_ci_year = df.srch_ci_year.apply(lambda x: 
get_year(x))
df.srch_ci_month = df.srch_ci_month.apply(lambda x: 
get_month(x))

# remove the srch_ci column
del df['srch_ci']
```

Dealing with srch_co column:

```
df['srch_co_year'] = pd.Series(df.srch_co, 
index=df.index)
df['srch_co_month'] = pd.Series(df.srch_co, 
index=df.index)
```

```
df.srch_co_month = df.srch_co_month.apply(lambda x:
get_month(x))

# remove the srch_co column
del df['srch_co']
```

Preliminary Analysis

After creating new features and removing the features that are not useful, we want to know if anything correlates well with hotel_cluster .This will tell us if we should pay more attention to any particular features.

```
df.corr()["hotel_cluster"].sort_values()
```

```
srch_destination_type_id    -0.036120
site_name                   -0.027497
hotel_country               -0.023837
is_booking                  -0.022898
user_location_country       -0.020239
srch_destination_id         -0.016736
srch_co_month               -0.005874
srch_rm_cnt                 -0.005570
srch_ci_month               -0.005015
date_time_month             -0.002142
channel                     -0.001386
date_time_year              -0.000435
cnt                          0.000378
hotel_continent              0.000422
user_location_city           0.001241
user_id                      0.003891
```

```
is_mobile                      0.008788
srch_co_year                   0.009287
posa_continent                 0.012180
srch_adults_cnt                0.012407
srch_children_cnt              0.014901
hotel_market                   0.022149
is_package                     0.047598
hotel_cluster                  1.000000
Name: hotel_cluster, dtype: float64
```

Figure 4

No column correlates linearly with hotel_cluster, this means that methods which model linear relationship between features might not be suitable for the problem.

Strategy

After a quick google search, it is not hard to learn that for known combinations of search destinations, hotel country, hotel market will definitely help finding the hotel cluster. Let's do that.

```
pieces =
[df.groupby(['srch_destination_id','hotel_country','hot
el_market','hotel_cluster'])
['is_booking'].agg(['sum','count'])]
agg = pd.concat(pieces).groupby(level=[0,1,2,3]).sum()
agg.dropna(inplace=True)
agg.head()
```

				sum	count
srch_destination_id	hotel_country	hotel_market	hotel_cluster		

Open in app

30	0	1
32	1	2
43	0	1

Figure 5

```
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)
agg.head()
```

	srch_destination_id	hotel_country	hotel_market	hotel_cluster	sum	count	sum_and_cnt
0	4	7	246	22	0.0	0.125	0.073171
1	4	7	246	29	0.0	0.125	0.073171
2	4	7	246	30	0.0	0.125	0.073171
3	4	7	246	32	1.0	0.250	0.560976
4	4	7	246	43	0.0	0.125	0.073171

Figure 6

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

2	11	50	824	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	14	27	1434	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	16	50	419	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	0.344828	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 103 columns

Figure 7

Merge with the destination table and our newly created aggregate pivot table.

```
df = pd.merge(df, dest, how='left',
on='srch_destination_id')
df = pd.merge(df, agg_pivot, how='left', on=
['srch_destination_id','hotel_country','hotel_market'])
df.fillna(0, inplace=True)
df.shape
```

(241179, 276)

Implementing Algorithms

We are only interested in booking events.

```
df = df.loc[df['is_booking'] == 1]
```

Get features and labels.

```
X = df.drop(['user_id', 'hotel_cluster', 'is_booking'],
axis=1)
y = df.hotel_cluster
```

Naive Bayes

Open in app

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

0.10347912437041926

K-Nearest Neighbors Classifier

```
from sklearn.neighbors import KNeighborsClassifier

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

0.25631461834732266

Random Forest Classifier

We report performance measurement by [*k*-fold cross-validation](), and [Pipeline]() makes it easier to compose estimators.

```
clf = make_pipeline(preprocessing.StandardScaler(),
RandomForestClassifier(n_estimators=273,max_depth=10,ra
ndom_state=0))
np.mean(cross_val_score(clf, X, y, cv=10))
```

0.24865023372782996

Multi-class Logistic Regression

```
clf = make_pipeline(preprocessing.StandardScaler(),
LogisticRegression(multi_class='ovr'))
np.mean(cross_val_score(clf, X, y, cv=10))
```

0.30445543572367767

SVM Classifier

SVM is very time consuming. However, we achieved a much better result.

```
from sklearn import svm

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

0.32287271373150050

Seems we need to do more feature engineering in order to improve the result. Stay tuned for further improvement!

Source code can be found on Github. Have a great week!

Data Science Machine Learning Recommendation System Python Programming