DSC 630: Predictive Analytics

Christopher M. Anderson

04/25/2020

Week 7

Assignment 7.3: Optimum Hotel Recommendations

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.

```
# Load the readr package:
library(readr)
```

Import Data

Let's import our data The training.csv file as exists on Kaggle is rather large, coming in at over 4GB uncompressed. Let's work with something smaller by randomly sampling just 1% of the records in that dataset.

```
import pandas as pd
import numpy as np
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import make_pipeline
from sklearn import svm
```

```
# Read the training data in from the csv file:
training = pd.read_csv('/Users/chris/Downloads/train.csv', sep=',').dropna()
dest = pd.read_csv('/Users/chris/Downloads/destinations.csv')
training = training.sample(frac=0.01, random_state=99)
```

Step 2: Dataframe Dimensions

Let's get a glimpse into the dataframe's dimensions and a peek at the information in it:

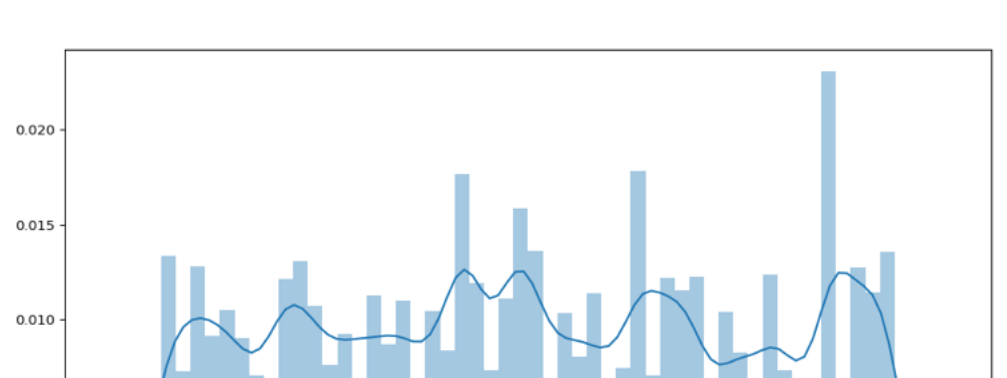
```
# Check the dimension of the table:
print("The dimension of the df is: ", training.shape)
## The dimension of the df is:  (241179, 24)
training.head()
##                     date_time  site_name  ...  hotel_market  hotel_cluster
## 32352134  2014-05-22 11:40:07          2  ...           177             44
## 29796021  2013-06-29 12:24:37          2  ...           659             59
## 15185156  2014-10-30 13:58:32          2  ...           642             22
## 3301948   2014-08-22 20:14:34          2  ...          1502             65
## 25429119  2014-03-25 18:47:43          2  ...           685              6
##
## [5 rows x 24 columns]
```

After getting a random sampling of the full dataset, it looks like there are 241,179 rows and 24 columns (variables) we'll be working with.

Exploratory EDA and Hotel Cluster

The objective is to predict a hotel reservation — the dependent variable `hotel_cluster` — that someone will book given the information — the other independent variables in the dataset — in their search. There are 100 clusters in total, and therefore we are dealing with a 100 class classification problem.

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



Functions and Data Wrangling

The `date_time`, `srch_ci` (checkin date), and `srch_co` (checkout date) columns are of no benefit as they exist right now in the dataset; let's make this information useable to us and extract the 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')
```

Wrangling date / time stamp data:

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

Wrangling the check in data:

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

Wrangling the check out data:

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

Now let's see our updated training dataframe after our wrangling maneuvers:

```
training.head()
##           site_name  posa_continent  ...  srch_co_year  srch_co_month
## 32352134          2               3  ...          2014              7
## 29796021          2               3  ...          2013              7
## 15185156          2               3  ...          2014             12
## 3301948           2               3  ...          2015              1
## 25429119          2               3  ...          2014              4
##
## [5 rows x 27 columns]
```

Preliminary Data 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.

```
training.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
## orig_destination_distance    0.006084
## user_location_region         0.006927
## srch_ci_year                 0.008562
## 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
```

Looking over the results here, no one independent variable has a linear correlation with our dependent variable, `hotel_cluster`, and therefore, using linear analysis might not be the best way to get our answers.

Aggregating Data and Pivot Table

Let's try to narrow things down by checking out the following search data: origin, destination, and distances. If we can group by these certain things, that may help to bring some clarity:

```
pieces =
[training.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)
agg.head()
##                                                             sum  count
## srch_destination_id hotel_country hotel_market hotel_cluster
## 4                   7             246          22             0      1
##                                                29             0      1
##                                                30             0      1
##                                                32             1      2
##                                                43             0      1
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  ...  sum  count  sum_and_cnt
## 0                    4              7           246  ...  0.0  0.125     0.073171
## 1                    4              7           246  ...  0.0  0.125     0.073171
## 2                    4              7           246  ...  0.0  0.125     0.073171
## 3                    4              7           246  ...  1.0  0.250     0.560976
## 4                    4              7           246  ...  0.0  0.125     0.073171
##
## [5 rows x 7 columns]
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()
## hotel_cluster  srch_destination_id  hotel_country  hotel_market  ...  97  98  99
## 0                                4              7           246  ... NaN NaN NaN
## 1                                8             50           416  ... NaN NaN NaN
## 2                               11             50           824  ... NaN NaN NaN
## 3                               14             27          1434  ... NaN NaN NaN
## 4                               16             50           419  ... NaN NaN NaN
##
## [5 rows x 103 columns]
```

Okay, now let's merge the destination table and our newly created aggregate pivot table and get a quick look at the shape of it:

```
training = pd.merge(training, dest, how='left', on='srch_destination_id')
training = pd.merge(training, agg_pivot, how='left', on=
['srch_destination_id','hotel_country','hotel_market'])
training.fillna(0, inplace=True)
training.shape
## (241179, 276)
```

Algorithms

Setting Up

Since we are only interested in the booking events data, let's run some algorithms with `is_booking`:

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

Let's get the labels and features:

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

X.shape, y.shape
## ((20032, 273), (20032,))
y.nunique()
## 100
```

Random Forest

```
clf = make_pipeline(preprocessing.StandardScaler(),
RandomForestClassifier(n_estimators=273,max_depth=10,random_state=0))
np.mean(cross_val_score(clf, X, y, cv=10))
## 0.24880132396216056
```

SVM

```
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))
## 0.324280146646298
```

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB

clf = make_pipeline(preprocessing.StandardScaler(), GaussianNB(priors=None))
np.mean(cross_val_score(clf, X, y, cv=10))
## 0.10388339646219294
```

Multi-Class Logistic Regression

```
from sklearn.linear_model import LogisticRegression

clf = make_pipeline(preprocessing.StandardScaler(), LogisticRegression(multi_class='ovr'))
np.mean(cross_val_score(clf, X, y, cv=10))
## 0.3009683578424778
```

KNN

```
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'))
## 0.25643954228338134
```