Data Scientist Internship

Case Presentation
10 December, 2018

Thomas KUOCH
Final-year Student – Ecole Centrale de Lyon
M2 Data Science – Université Lyon 1

kuoch.thomas@gmail.com
thomas.kuoch@ecl15.ec-lyon.fr
+33 6 70 28 01 32

Summary

Summary

Objectives

The given dataset contains data that represent the current business of BlaBlaCar.

Based on this dataset, I will try to achieve these two objectives:

1. *Explain what factors influence whether a driver will travel alone or with at least one passenger (trip success)*
2. *Build a model that predicts the success of a trip*

In order to achieve these objectives, my approach will be the following:

- Understand correctly the data
- Clean up the data that need to be cleaned (type, etc.)
- Create new relevant features
- Build a first RandomForest prediction model and select important features
- Build other models and evaluate them

I will work on a Python 3 notebook with mainly *pandas* and *sklearn* libraries.

Summary

Presentation of the Dataset

The dataset is big: over 3 million rows and 20 columns.

```
In [2]: data = pd.read_csv('dataset_datascientist_case.csv', sep=',')
        data.shape

Out[2]: (3379778, 20)
```

Each row concerns a single trip. The notion of "trip" is defined in the instruction document. The 20 columns are the features of the trips.

The features can be grouped in 2 categories:

- Overall information about the trip (*offer_id, driver_id*, etc.). These features are often auto-incremented columns and can't be integrated in the prediction model
- Information about the nature of the trip: Distance, Price, Number of Seats, etc. These features will be useful for the prediction model

I will also focus on the second category of features. All the feature descriptions are available in the instruction document and I will not explain them again.

Summary

Data Preparation Process
Data Cleansing

Because of the dataset size, it will be time-saving to work on a sample of the dataset. Let's build also a sample of 10000 random-selected rows.

Then, some columns have to be cleaned in order to be taken into account for the prediction model, or to create new features, which are resulting of calculations between these columns.

The concerned features are:

- Dates and Times features: They have to be converted from a string to a datetime format. The concerned columns are: *trip_date, published_date, signup_date*
- Numeric features: They have to be converted from a string or an anormal numeric format (like comma delimitation) to a float or integer format. The concerned columns are: *driver_id, offer_id, trip_id, trip_distance_km, publication_site_id*

Data Preparation Process
Create new features

I have thought interesting to build these following features for my prediction models:

- Order Delay (*order_delay*): this is the offer duration and is also the difference between *trip_date* and *published_date*
- Trip day of week (*trip_day_of_week*): this is the day of week number, with 0 for Monday and 6 for Sunday
- Driver Seniority (*driver_seniority*) : this is the seniority of the driver and is also the difference between the *published_date* and *signup_date*

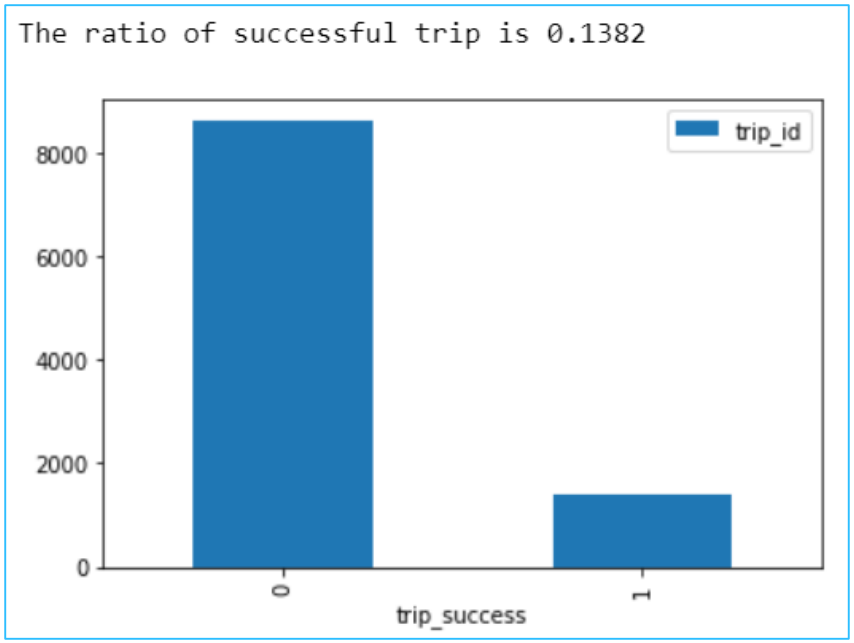
Then, I have built the class to predict, which is *success_trip*. The column *success_trip* results of the definition of a successful trip: ***a trip is successful if its confirmed_seat_count > 0.***

So, if a trip is successfull: trip_success = 1
Else, trip_success = 0

Summary

Data Visualization
Ratio of Successful Trip

In my sample of 10000 rows, the ratio of successful trip is lower than I could think (around 13%).



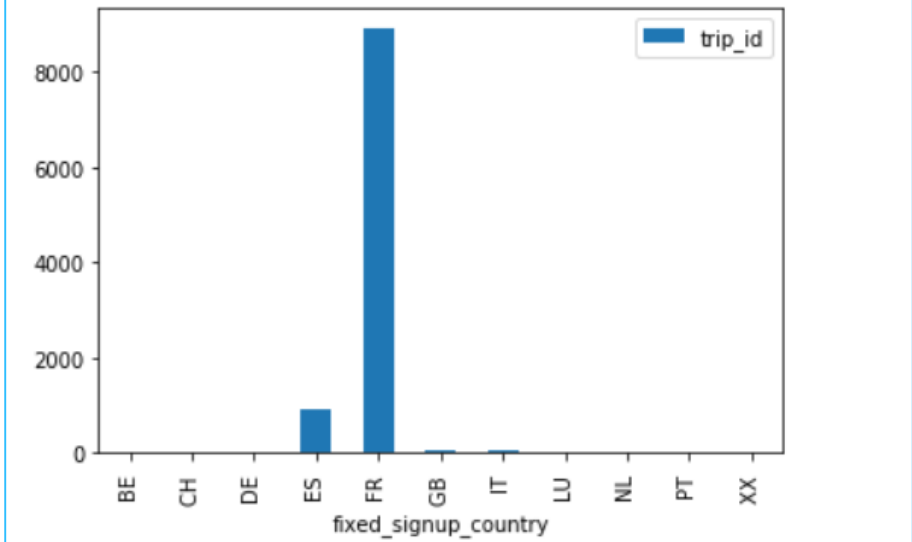
This low ratio can be explain by the definition of a trip. A main trip can give several trips, where stopovers have been taken into account. I don't know if this number of 13% is meaningful, because I'm not an expert in the ridesharing domain.

Data Visualization
Nationality Distribution

Most of our drivers are French (89.5 %).

```
The ratio of French drivers is 0.895

<matplotlib.axes._subplots.AxesSubplot at 0x2267b4df7b8>
```

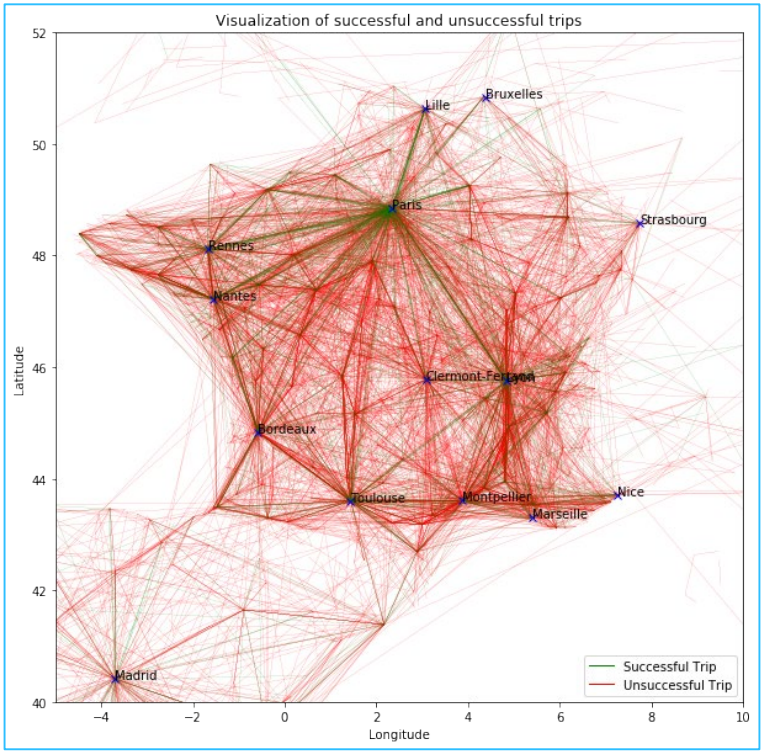


I will drop the nationality feature in my model. Most of the activity of BlaBlaCar take place in France.

Data Visualization
Trips

Thanks to location columns (*from_lon, to_lon, from_lat, to_lat*), I have plotted each trip with a line.

- If the trip is unsuccessful (*trip_success = 0*), the line is red
- If the trip is successful (*trip_success = 1*), the line is green



As expected, the map is quite red. This is because of the low ratio of successful trip.

Agglomerations are connected by these trips. These main roads concentrate successful trips. Paris is a very successful city.

I can deduce from this analysis that a trip has more chances to be successful if it connects two big agglomerations.

Nevertheless, I am not able to include these 4 location features in my predicting model.

Summary

Machine Learning models

Keep useful columns for train and test datasets

Only the following columns will be kept for the train and test datasets:

In [184]:

```
df = df[['is_main_trip',
         'unit_seat_price_eur',
         'seat_offered_count',
         'seat_left_count',
         'trip_distance_km',
         'is_comfort',
         'is_auto_accept_mode',
         'order_delay',
         'trip_day_of_week',
         'driver_seniority',
         'trip_success']]
```

Overall information features, location features, nationality, and *confirmed_seat_count* are dropped, as justified before. Notice that *confirmed_seat_count* cant'be taken into account because the class to predict (*trip_success*) is a calculation of this feature and is also totally correlated with it.

Train (*X_train, Y_train*) and test (*X_test, Y_test*) datasets will be built with a proportion of 25%. For our sample of 10000 rows, test datasets shape will be 2500.

Machine Learning models
First RandomForest Classifier

I have built a first RandomForest Classifier. Here are the results :

```
True Negative : 2101 / False Positive : 82 / False Negative : 212 / True Positive : 105
Confusion Matrix :
 [[2101   82]
 [ 212  105]]
Precision : 0.882
AUC : 0.647
Time Execution : 0.187 s
```

Precision is quite good: the model predicts correctly 88.2% of test instances.

To improve the precision metric, I will do the following steps:

1. Select only features which preserve 75% of the information
2. Optimize parameters of the RandomForest Classifier
3. Try other models

Machine Learning models
Selecting most important features

With RandomForest Classifer's method *feature_importance_,* the five most important features which preserves 75% of the information are:

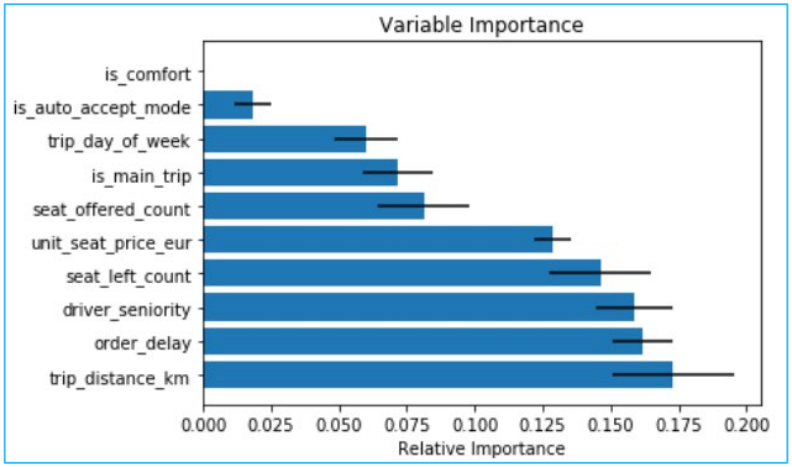
- *trip_distance_km*
- *order_delay*
- *driver_seniority*
- *seat_left_count*
- *unit_seat_price_eur*

	importance	cumulative importance
trip_distance_km	0.173110	0.173110
order_delay	0.161779	0.334889
driver_seniority	0.158863	0.493753
seat_left_count	0.146246	0.639999
unit_seat_price_eur	0.128473	0.768472
seat_offered_count	0.081238	0.849710
is_main_trip	0.071826	0.921536
trip_day_of_week	0.060180	0.981716
is_auto_accept_mode	0.018284	1.000000
is_comfort	0.000000	1.000000

On the opposite side, features are ordered by their relative importance.

It is interesting to notice that the day of week or a comfortable trip are not meaningful to determine if a trip is successful or not.

For the following parts, I will keep these five features only.



Machine Learning models
Optimize RandomForest Classifier parameters

RandomForest Classifier's parameters can be chosen to optimize the precision metric. This can be done thanks to the library *GridsearchCV*.

Two parameters will be optimized :

- The number of trees in the forest (*n_estimators*): *10, 50* or *100*
- The splitting criterion: *gini* or *entropy*

In our case, the best parameters are a number of tree of 100 and an entropy criterion.

In [189]:
```
params = {
    'n_estimators': [10, 50, 100],
    'criterion': ('gini', 'entropy')
}

clf = RandomForestClassifier(random_state=None)
clf = GridSearchCV(clf, params, scoring='precision', cv=5)
clf.fit(X_train, Y_train)

best_parameters = clf.best_params_
print('Best parameters:')
print(clf.best_params_)
```

```
Best parameters:
{'criterion': 'entropy', 'n_estimators': 100}
```

Machine Learning models
Second RandomForest Classifier

Running our RandomForest Classifier with these optimized parameters and reduced dataset (with only the five most important features) give a result of 100% precision.

```
True Negative : 2187 / False Positive : 0 / False Negative : 0 / True Positive : 313
Confusion Matrix :
 [[2187    0]
 [   0  313]]
Precision : 1.0
AUC : 1.0
Time Execution : 0.729 s
```

This can look like a too perfect model. Nevertheless, the classifier has learned on the train dataset and has been tested on the test dataset. If I run the script again with a new sample, the result of 100% Precision remains.

An hypothesis is that I have dropped noise by keeping only relevant features. If this is the case, it can explain this metric of 100% Precision.

Machine Learning models
Using a Naive Bayes Classifier

A Naive Bayes Classifier gives same results:

```
True Negative : 2187 / False Positive : 0 / False Negative : 0 / True Positive : 313
Confusion Matrix :
 [[2187    0]
 [   0  313]]
Precision : 1.0
AUC : 1.0
Time : 0.021 s
```

Good to notice that Naive Bayes Classifier has been much faster (34 times faster) than the RandomForest Classifier. For an overall deployment, this Naive Bayes Classifier must be chosen to save time.

An ID3 algorithm will give also same results (precision and time scores).

Summary

Conclusion

- This is a very interesting case. It has challenged me a lot, forced me to create new relevant features after cleaned some columns. Visualization of successful and unsuccessful trip was also exciting, even if I could not manage to include these location features into my prediction models.
- After running a RandomForest Classifier, I have optimized its parameters and kept only most relevant features.
- The five most important features which preserves 75% of the information are *trip_distance_km, order_delay, driver_seniority, seat_left_count, unit_seat_price_eur*.
- I have got 100% of prediction Precision. This seems a little bit weird that my solution performs so well.
- For a overall deployment using the 3 million rows, using a Naive Bayes Classifier will be time-saving.