# Machine Learning Engineer Nanodegree

## Capstone Project

Kei Fukutani
July 14, 2018

## I. Definition

### Project Overview

This project is based on the Kaggle competition, "Expedia Hotel Recommendations"[1]. This competition is about a hotel search application that provides personalized hotel recommendations for its users. The customer data will be used to predict which types of hotels people are going to book.

I chose this competition, as there are a lot of situations in the real world that are similar to this kind of recommendation problems. I also believe that I can explore various machine learning techniques that are able to predict much better than human ability to solve this problem.

One of the early researches on this problem is the research paper by Gourav G. Shenoy et al[2]. They suggest that ensemble learning methods contribute to better performance.

### Problem Statement

My goal is to predict hotel groups for a user event based on the search event logs, such as hotel location, check-in/out date, number of adults, and other attributes associated with that event, including user location, distance between the user and the searched hotel, whether or not the hotel is booked, etc.

The company has in-house algorithms to form the hotel groups ("hotel cluster"), where similar hotels for a search are grouped together. Using provided logs of customer's search behaviors as described above, we predict the likelihood of 100 different hotel clusters that a user will book.

### Metrics

I use precision in micro averaging as a metric because false positives for each class are ineffective recommendations for users. Precision in micro averaging is defined as,

$$PRECISION_{micro} = \frac{TP_1 + \cdots + TP_k}{TP_1 + \cdots + TP_k + FP_1 + \cdots + FP_k}$$

where $TP_i$ is the counts of True Positives for class i, $FP_i$ is the counts of False Positives for class i, and k is the number of classes. This metric is used when evaluating our classifier. My classifier models trained using machine-learning techniques output one predicted hotel cluster for each data. For example, if true hotel clusters and predicted hotel clusters for 6 data are the following,

- True hotel clusters: [1, 81, 40, 52, 81, 3]
- Predicted clusters: [3, 81, 33, 52, 70, 3]

then the precision in micro averaging is 0.50.

Additionally, I track prediction time as well as training time to see if it is adequate to use the model on a web application for hotel search.

## II. Analysis

## Data Exploration

The datasets are provided by the competition organizer[3]. The training dataset has 37,670,293 rows, and the test dataset has 2,528,293 rows. In this project, I use the first 1,000,000 rows of the training dataset to analyze and preprocess the data for computational resource reasons.

The training dataset has user's search event logs, and there are 15 categorical features and 5 numerical features in the data (**Fig. 1**). Some of the numerical features are the distance between a user and a hotel, the number of adults/children. Most of the other columns are categorical. Some of the features have NaN values.

**Fig. 1** Sample data of training dataset (For simplicity, some columns are not displayed)

| | date_time | user_location_city | orig_destination_distance | user_id | srch_ci | srch_co | ... |
|---|---|---|---|---|---|---|---|
| **0** | 8/11/14 7:46 | 48862 | 2234.2641 | 12 | 8/27/14 | 8/31/14 | ... |
| **1** | 8/11/14 8:22 | 48862 | 2234.2641 | 12 | 8/29/14 | 9/2/14 | ... |
| **2** | 8/11/14 8:24 | 48862 | 2234.2641 | 12 | 8/29/14 | 9/2/14 | ... |
| **3** | 8/9/14 18:05 | 35390 | 913.1932 | 93 | 11/23/14 | 11/28/14 | ... |
| **4** | 8/9/14 18:08 | 35390 | 913.6259 | 93 | 11/23/14 | 11/28/14 | ... |
| **5** | 8/9/14 18:13 | 35390 | 911.5142 | 93 | 11/23/14 | 11/28/14 | ... |
| **6** | 7/16/14 9:42 | 10067 | NaN | 501 | 8/1/14 | 8/2/14 | ... |
| **7** | 7/16/14 9:45 | 10067 | NaN | 501 | 8/1/14 | 8/2/14 | ... |
| **8** | 7/16/14 9:52 | 10067 | NaN | 501 | 8/1/14 | 8/2/14 | ... |
| **9** | 7/16/14 9:55 | 10067 | NaN | 501 | 8/1/14 | 8/2/14 | ... |

|  | ... | srch_adults_cnt | srch_children_cnt | srch_destination_id | is_booking | hotel_country | hotel_cluster |
|---|---|---|---|---|---|---|---|
| **0** | ... | 2 | 0 | 8250 | 0 | 50 | 1 |
| **1** | ... | 2 | 0 | 8250 | 1 | 50 | 1 |
| **2** | ... | 2 | 0 | 8250 | 0 | 50 | 1 |
| **3** | ... | 2 | 0 | 14984 | 0 | 50 | 80 |
| **4** | ... | 2 | 0 | 14984 | 0 | 50 | 21 |
| **5** | ... | 2 | 0 | 14984 | 0 | 50 | 92 |
| **6** | ... | 2 | 0 | 8267 | 0 | 50 | 41 |
| **7** | ... | 2 | 0 | 8267 | 0 | 50 | 41 |
| **8** | ... | 2 | 0 | 8267 | 0 | 50 | 69 |
| **9** | ... | 2 | 0 | 8267 | 0 | 50 | 70 |

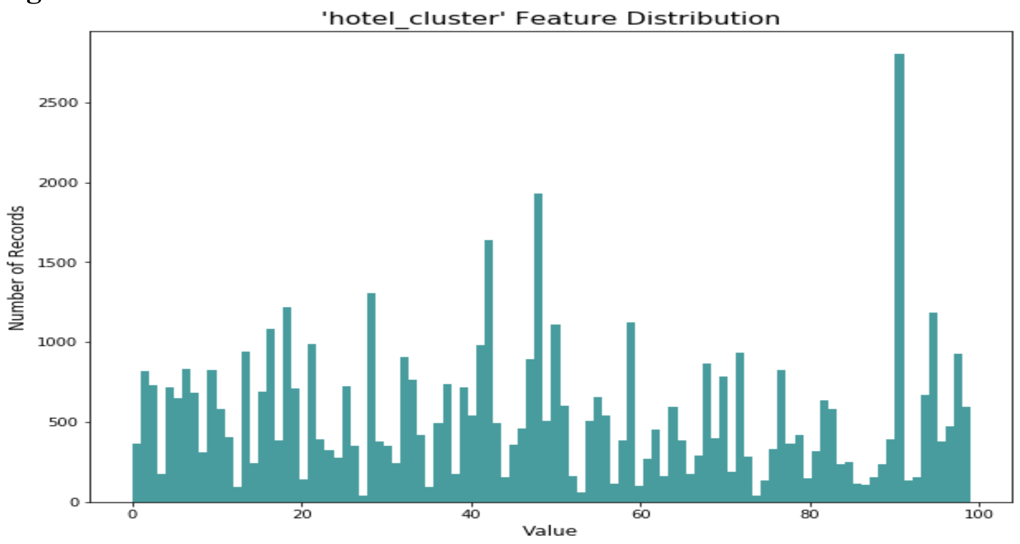
| Column name | Description |
|---|---|
| date_time | Timestamp |
| site_name | ID of the Expedia point of sale (i.e. Expedia.com, Expedia.co.uk, Expedia.co.jp, ...) |
| posa_continent | ID of continent associated with site_name |
| user_location_country | The ID of the country the customer is located |
| user_location_region | The ID of the region the customer is located |
| user_location_city | The ID of the city the customer is located |
| orig_destination_distance | Physical distance between a hotel and a customer at the time of search. A null means the distance could not be calculated. |
| user_id | ID of user |
| is_mobile | 1 when a user connected from a mobile device, 0 otherwise |
| is_package | 1 if the click/booking was generated as a part of a package (i.e. combined with a flight), 0 otherwise. |
| channel | ID of a marketing channel |
| srch_ci | Checkin date |
| srch_co | Checkout date |
| srch_adults_cnt | The number of adults specified in the hotel room |
| srch_children_cnt | The number of (extra occupancy) children specified in the hotel room |
| srch_rm_cnt | The number of hotel rooms specified in the search |
| srch_destination_id | ID of the destination where the hotel search was performed |
| srch_destination_type_id | Type of destination |
| hotel_continent | Hotel continent |
| hotel_country | Hotel country |
| hotel_market | Hotel market |
| is_booking | 1 if a booking, 0 if a click |
| cnt | Numer of similar events in the context of the same user session |
| hotel_cluster | ID of a hotel cluster |

To train our model, I only use booking event logs ('is_booking' is 1), and create a new feature "length_of_stay" using "srch_ci" and "srch_co" because I consider that the length of stay can be related to which hotel a user will book. This feature is a numerical value. The 'date_time' and 'user_id' might be useful, but I drop these columns in this project.

## Exploratory Visualization

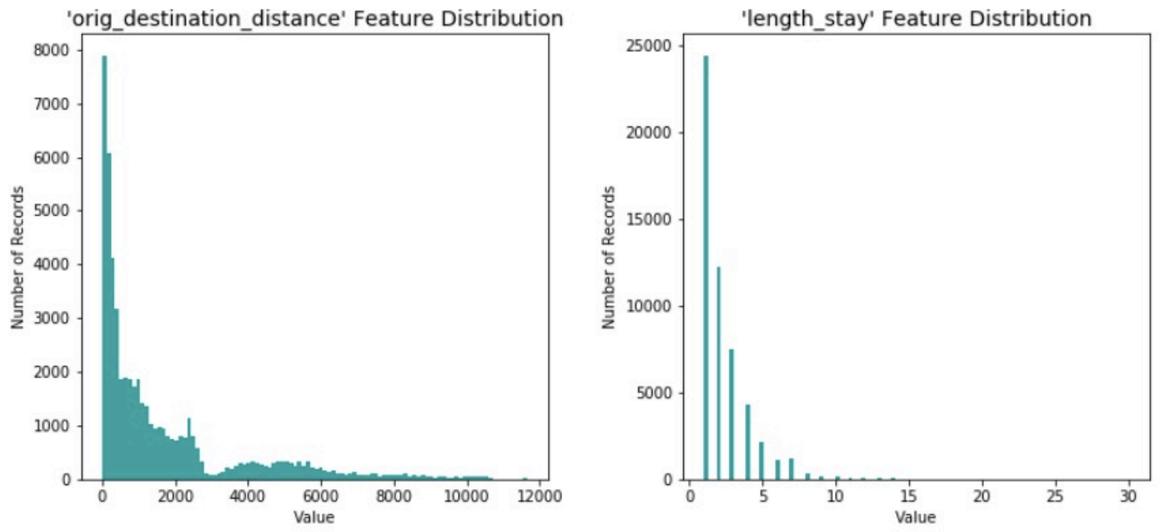
The "hotel_cluster" label that we attempt to predict has 100 different values. The most frequent cluster has 2,806 records, and the least frequent cluster has 36 records (**Fig. 2**) after dropping rows that have a NaN value in any feature and that have the value of 0 in the feature 'is_booking'. The hotel clusters are not evenly distributed, but considering a total of 54,004 records, they are distributed over all the 100 clusters well enough that a model without any intelligence would not perform very well.

**Fig. 2**



As **Fig. 3** shows, some features such as 'orig_destination_distance' and 'length_stay' appear to have some outliers.

**Fig. 3**



## Algorithms and Techniques

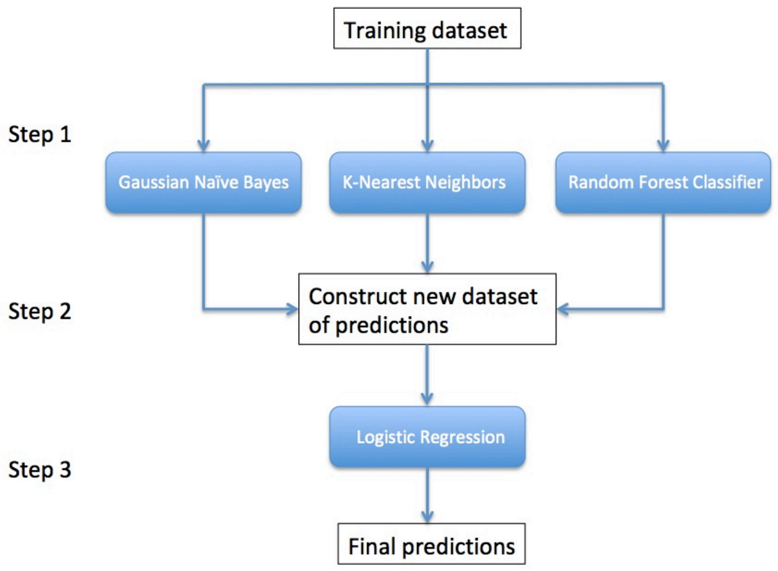
I use supervised machine learning algorithms to predict the hotel clusters. Multi-class classification models from scikit-learn, such as Gaussian Naive Bayes (GNB), K-Nearest Neighbors (KNN), and Random Forest Classifier (RFC) are trained on the training dataset that has features of a user search event and a hotel cluster label. Then I make predictions about the hotel clusters based on the trained model. Additionally, I put those three trained models together and use a technique "Stacking" from mlxtend (https://rasbt.github.io/mlxtend/) with Logistic Regression as a combiner algorithm to make predictions. I call it "Stacking Classifier" in this project. Stacking is one of ensemble methods that help improve machine learning results by combining several models.

The performance of those four models (GNB, KNN, RFC, Stacking Classifier) is compared to select a final model.

- Gaussian Naive Bayes - are based on Bayes theorem but assumes independence between every pair of features. I use default parameters.
- K-Nearest Neighbors - uses k nearest neighbors to make predictions. I use default parameters except that 'n_neighbors' is 1.
- Random Forest Classifier - creates a decision tree drawn on subsets of the features with replacement. I use default parameters.
- Stacking Classifier (https://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/) - an ensemble method where the predictions from several base learners are combined and used as inputs to a combiner algorithm, which makes a final prediction (**Fig. 4**). I use default parameters except that GNB, KNN, RFC are used as base learners and Logistic Regression is used as a combiner algorithm.

**Fig. 4** Stacking Classifier Algorithm

```
Algorithm   Stacking
1: Input: training data $D = \{x_i, y_i\}_{i=1}^{m}$
2: Ouput: ensemble classifier $H$
3: Step 1: learn base-level classifiers
4: for $t = 1$ to $T$ do
5:     learn $h_t$ based on $D$
6: end for
7: Step 2: construct new data set of predictions
8: for $i = 1$ to $m$ do
9:     $D_h = \{x'_i, y_i\}$, where $x'_i = \{h_1(x_i), ..., h_T(x_i)\}$
10: end for
11: Step 3: learn a meta-classifier
12: learn $H$ based on $D_h$
13: return $H$
```



## Benchmark

I use a random model as a benchmark that picks one hotel cluster out of 100 randomly. The final model is compared with the benchmark model. Since the hotel clusters are relatively evenly distributed over all the dataset, the accuracy of the random model should be around 0.01.

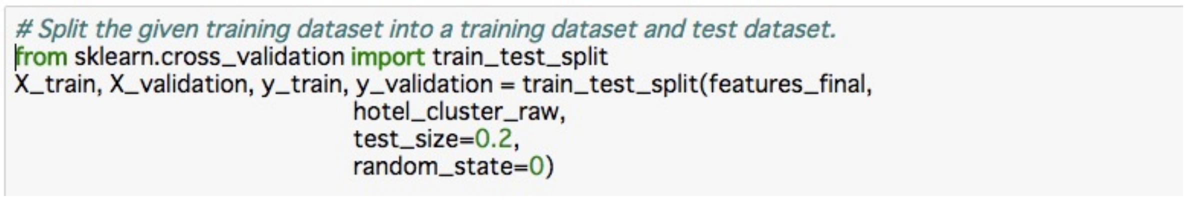
## III. Methodology

## Data Processing

The rows that have a NaN value in any feature and that have the value of 0 in the feature 'is_booking' are removed. Then, I calculate length of stay using 'srch_ci' (check-in date) and 'srch_co' (check-out date) and add it to the training data. After that, I drop 'date_time', 'user_id', 'srch_ci', 'srch_co', and 'hotel_cluster' columns to make features data. A logarithmic transformation is applied on the skewed features so that the outliers do not negatively affect the performance of a learning algorithm. Next, the numerical features are normalized using MinMaxScaler from sklearn. The categorical features are converted to numerical entries using one-hot encoding. To reduce the number of features, all rare categories are encoded to the same feature 'RARE_VALUE' before the one-hot encoding. The final number of features is 1729.

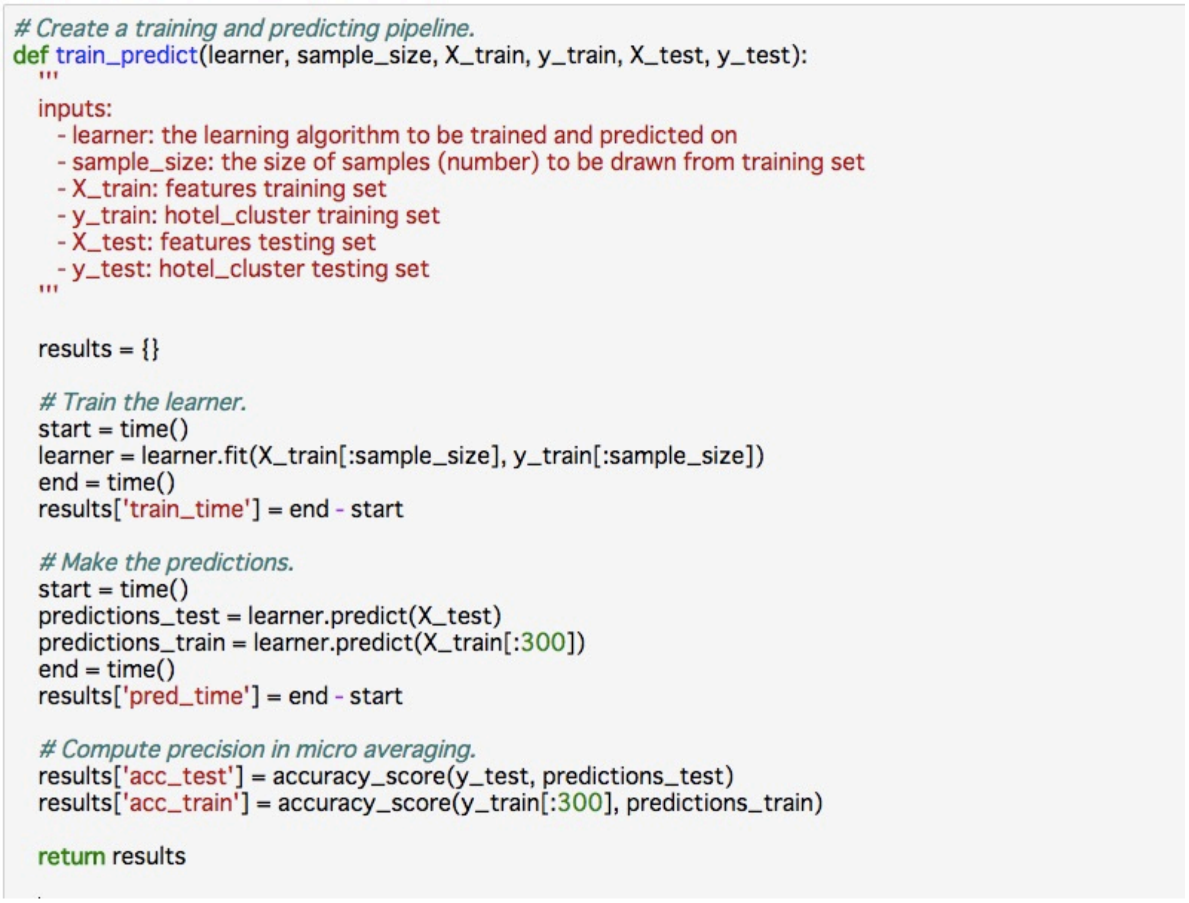
## Implementation

The provided training dataset are split into a training dataset and a validation dataset. The validation dataset is used to validate the four models and is about 20% of the provided training dataset in size. At this point, the total number of samples is 54,004, and the training dataset has 43,203 samples, and the validation dataset has 10,801 samples.



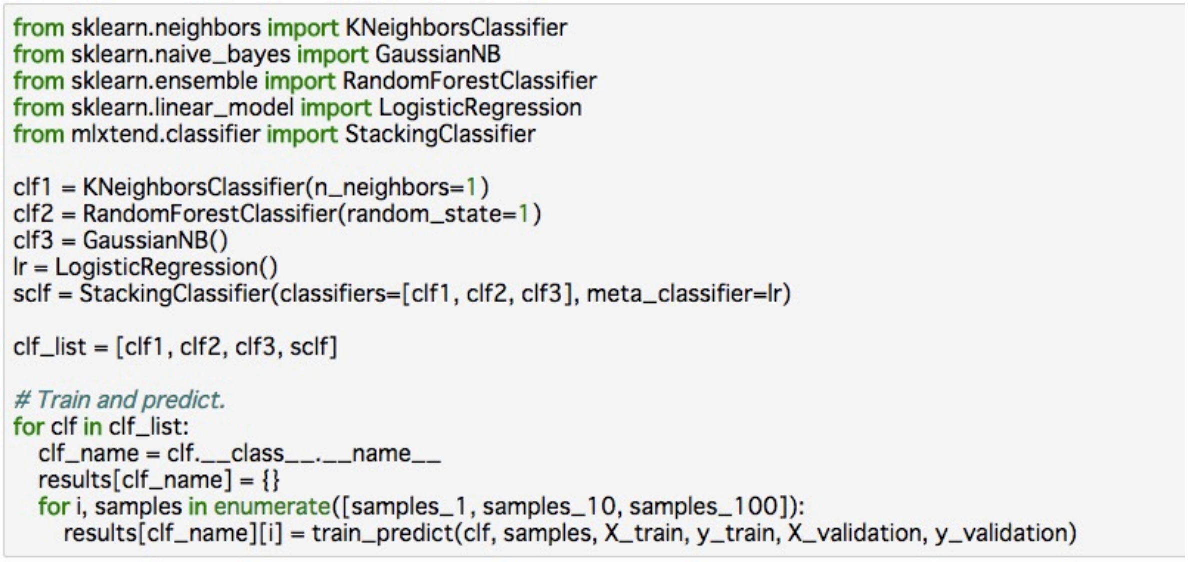
```
# Split the given training dataset into a training dataset and test dataset.
from sklearn.cross_validation import train_test_split
X_train, X_validation, y_train, y_validation = train_test_split(features_final,
                                    hotel_cluster_raw,
                                    test_size=0.2,
                                    random_state=0)
```

A training and predicting pipeline is created like the following.



```
# Create a training and predicting pipeline.
def train_predict(learner, sample_size, X_train, y_train, X_test, y_test):
    '''
    inputs:
       - learner: the learning algorithm to be trained and predicted on
       - sample_size: the size of samples (number) to be drawn from training set
       - X_train: features training set
       - y_train: hotel_cluster training set
       - X_test: features testing set
       - y_test: hotel_cluster testing set
    '''

    results = {}

    # Train the learner.
    start = time()
    learner = learner.fit(X_train[:sample_size], y_train[:sample_size])
    end = time()
    results['train_time'] = end - start

    # Make the predictions.
    start = time()
    predictions_test = learner.predict(X_test)
    predictions_train = learner.predict(X_train[:300])
    end = time()
    results['pred_time'] = end - start

    # Compute precision in micro averaging.
    results['acc_test'] = accuracy_score(y_test, predictions_test)
    results['acc_train'] = accuracy_score(y_train[:300], predictions_train)

    return results
```

KNN, GNB, RFC, and Stacking Classifier are implemented as shown below. Those learners are trained and make predictions using the training and predicting pipeline. 'StackingClassifier' from mlxtend is used to build the Stacking Classifier model.



```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from mlxtend.classifier import StackingClassifier

clf1 = KNeighborsClassifier(n_neighbors=1)
clf2 = RandomForestClassifier(random_state=1)
clf3 = GaussianNB()
lr = LogisticRegression()
sclf = StackingClassifier(classifiers=[clf1, clf2, clf3], meta_classifier=lr)

clf_list = [clf1, clf2, clf3, sclf]

# Train and predict.
for clf in clf_list:
    clf_name = clf.__class__.__name__
    results[clf_name] = {}
    for i, samples in enumerate([samples_1, samples_10, samples_100]):
        results[clf_name][i] = train_predict(clf, samples, X_train, y_train, X_validation, y_validation)
```

## Refinement

The RFC model was fine-tuned using grid search with some parameters. Those parameters are:

- n_estimators: [16, 32, 64]
- max_depth: [64, 128, 256]
- max_features: [160, 320, 640]

## IV. Results

## Model Evaluation and Validation

Of the four models, the Random Forest Classifier model performed the best. The accuracy scores (precisions in micro averaging) of the four models were:

| Model | Accuracy Score |
|---|---|
| Gaussian Naive Bayes | 0.0538 |
| Random Forest Classifier | 0.1817 |
| K-Nearest Neighbors | 0.1752 |
| Stacking Classifier | 0.0502 |

Then I fine-tuned the Random Forest Classifier model using grid search. The best parameters were n_estimators: 64, max_depth: 256, and max_features: 640. The accuracy score was 0.2093.

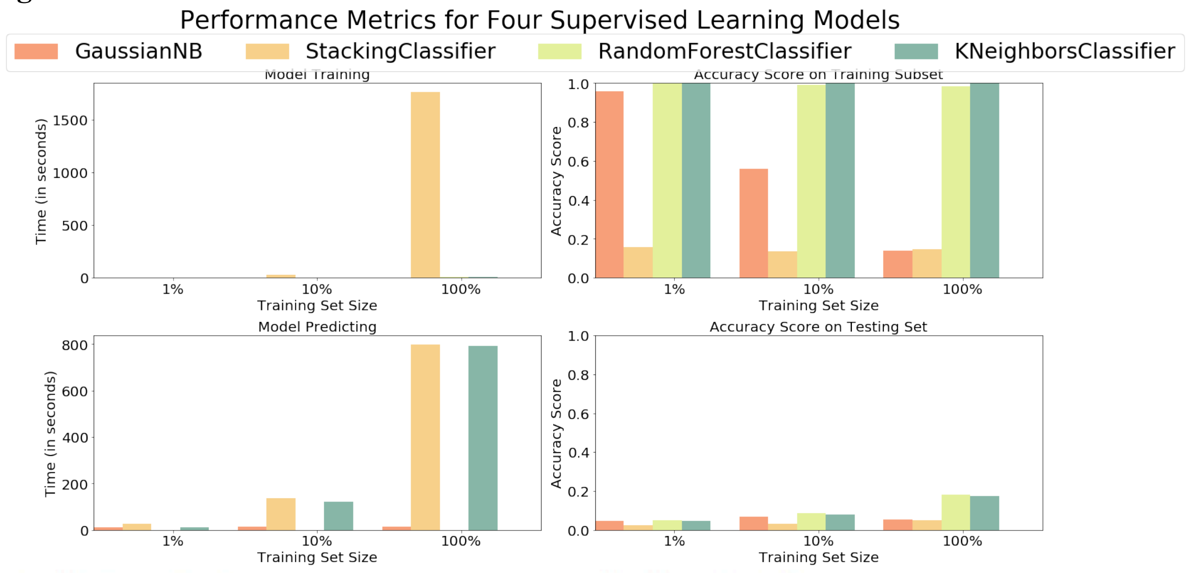
## Justification

The accuracy score of the random model was 0.00981, which is around 0.01 as expected. The accuracy score of the optimized final model was 0.2093. This is at least better than the random model, but is not as high as I thought it would be.

## V. Conclusion

### Free-Form Visualization

**Fig. 5**



RandomForestClassifier

| | 1% | 10% | 100% |
|---|---|---|---|
| acc_test | 0.0496 | 0.0875 | 0.1817 |
| acc_train | 0.9967 | 0.99 | 0.9833 |
| pred_time | 0.2317 | 0.2876 | 0.3459 |
| train_time | 0.0411 | 0.4829 | 11.4098 |

KNeighborsClassifier

| | 1% | 10% | 100% |
|---|---|---|---|
| acc_test | 0.0469 | 0.0788 | 0.1752 |
| acc_train | 1 | 1 | 1 |
| pred_time | 12.2942 | 129.0846 | 838.9432 |
| train_time | 0.0134 | 0.3593 | 9.3555 |

GaussianNB

| | 1% | 10% | 100% |
|---|---|---|---|
| acc_test | 0.0471 | 0.0689 | 0.0538 |
| acc_train | 0.9567 | 0.56 | 0.14 |
| pred_time | 16.1094 | 18.4002 | 18.5483 |
| train_time | 0.0175 | 0.1983 | 2.4204 |

StackingClassifier

| | 1% | 10% | 100% |
|---|---|---|---|
| acc_test | 0.024 | 0.0328 | 0.0502 |
| acc_train | 0.1567 | 0.1367 | 0.1467 |
| pred_time | 28.7841 | 148.3524 | 857.5905 |
| train_time | 0.5515 | 28.0266 | 1890.043 |

The predicting time of the Random Forest Classifier was the fastest of the four models, and this model might be useful for a hotel search web application when responding to user search events, but the predicting time of the other models were slow and almost useless for the real time recommendation.

### Reflection

I tried the Stacking model for the first time. Stacking does not always seem to contribute to a better performance. It might help generalize to new data, but if one of the models of the Stacking model performs very poorly, then the Stacking model also seems to perform poorly.

Another thing that I struggled with in this project was to deal with the big dataset. At first, I used all of the provided data to explore, preprocess, and train the models. The problem was that hours and hours after executing my program, I just found out there was an easy mistake or out of memory error in my program. I learned from this project that I should input small data first to see if the program does not result in an error, and then feed big data into the same program.

## Improvement

The accuracy scores were not as good as I expected. There might be a room for improvement in feature engineering using domain knowledge. In this project, I dropped the user_id, but user_id could be useful for restricting the range of clusters because every user should have its own preference. Additionally, the rows that have the value of 0 in the feature 'is_booking' (click events) and their time data might be helpful for predicting the clusters because the path to the booking events might have some patterns. In terms of ensemble methods including the Stacking Classifier, it seems like base learners have to be as accurate as possible and as diverse as possible. Needless to say, using the full data should improve the accuracy scores based on machine learning.

## References

[1] https://www.kaggle.com/c/expedia-hotel-recommendations
[2] Gourav G. Shenoy, Mangirish A. Wagle, Anwar Shaikh. "Kaggle Competition: Expedia Hotel Recommendations." Indiana University. arXiv:1703.02915 [cs.IR] 6 Mar 2017
[3] https://www.kaggle.com/c/expedia-hotel-recommendations/data
[4] https://www.kaggle.com/c/expedia-hotel-recommendations#evaluation