Capstone Project

Bowen Liu

Machine Learning Engineer Nanodegree

Aug 7th, 2016

Definition

Project Overview

Like most companies, Red Hat is able to gather a great deal of information over time about the behavior of individuals who interact with them. They're in search of better methods of using this behavioral data to predict which individuals they should approach—and even when and how to approach them.

In this project, I created a classification algorithm that accurately identifies which customers have the most potential business value for Red Hat based on their characteristics and activities. With an improved prediction model in place, Red Hat will be able to more efficiently prioritize resources to generate more business and better serve their customers. The project was inspired by Kaggle competition.

Problem Statement

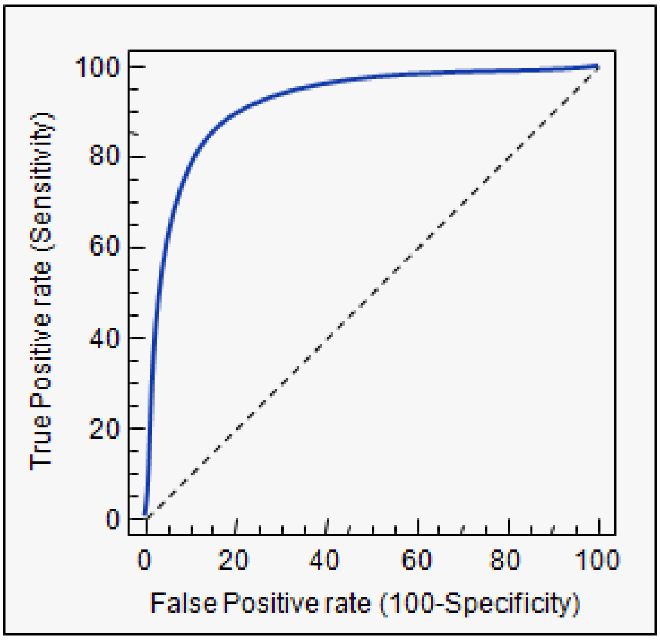
The problem is to predict the potential business value of a person who has performed a specific activity. The business value outcome is defined by a yes/no field attached to each unique activity. The outcome indicates whether or not each person has completed the outcome within a fixed window of time after each unique activity was performed. The objectives are built upon the followings pipeline:

1. Download the people and activity data and perform the exploratory data analysis
2. Data engineering and benchmark the classifier
3. Improve the performance using basic tree model
4. Improve the performance using ensemble method
5. Test the performance in Kaggle LeaderBoard

Metrics

In this project, receiver operating characteristic (ROC) is used to evaluate the model performance. According to Wikipedia definition, a receiver operating characteristic (ROC), or ROC curve, is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. The curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The true-positive rate is also known as sensitivity, recall or probability of detection in machine learning. The false-positive rate is also known as the fall-out or probability of false alarm and can be calculated as (1 - specificity). The ROC curve is thus the sensitivity as a function of fall-out. In general, if the probability distributions for both detection and false alarm are known, the ROC curve can be generated by plotting the cumulative distribution function of the detection probability in the y-axis versus the cumulative distribution function of the false-alarm probability in x-axis.

Fig. 1 A example plotting of ROC curve



Analysis

Data Exploration

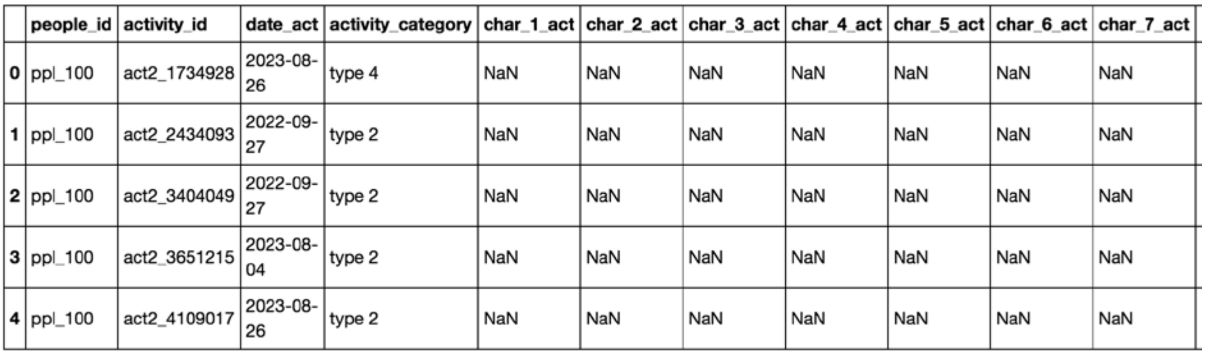
This data sets include two separate data files: a people file and an activity file.

The people file contains 189,118 unique people (and 41 corresponding characteristics) that have performed activities over time. Each row in the people file represents a unique person. Each person has a unique “people_id”.

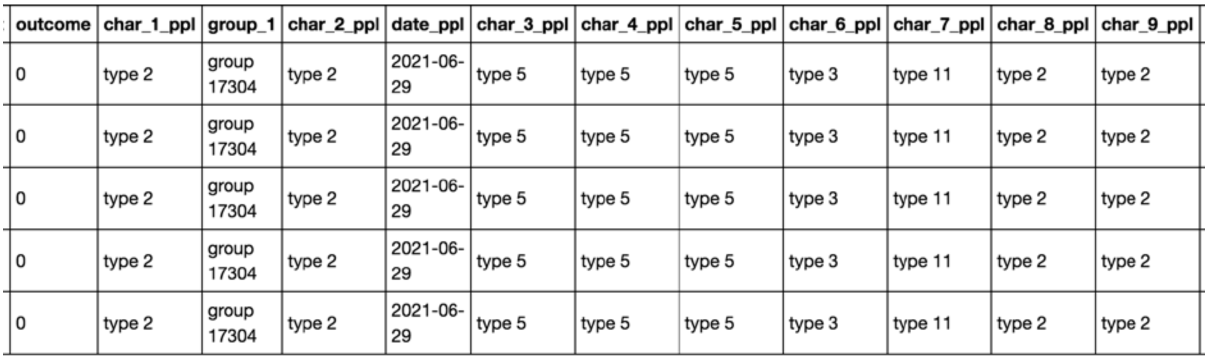
The activity file contains 2,197,291 unique activities (and 15 corresponding activity characteristics) that each person has performed over time. Each row in the activity file represents a unique activity performed by a person on a certain date. Each activity has a unique “activity_id”.

To develop a predictive model, both files are joined together on “people_id” as the common key into a single data set. All variables are categorical, with the exception of 'char_38' in the people file, which is a continuous numerical variable. (see EDA part)

Table 1. A sample data set, including the first 23 of total 55 columns after joining on “people_id”.



	people_id	activity_id	date_act	activity_category	char_1_act	char_2_act	char_3_act	char_4_act	char_5_act	char_6_act	char_7_act
0	ppl_100	act2_1734928	2023-08-26	type 4	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	ppl_100	act2_2434093	2022-09-27	type 2	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	ppl_100	act2_3404049	2022-09-27	type 2	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	ppl_100	act2_3651215	2023-08-04	type 2	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	ppl_100	act2_4109017	2023-08-26	type 2	NaN	NaN	NaN	NaN	NaN	NaN	NaN

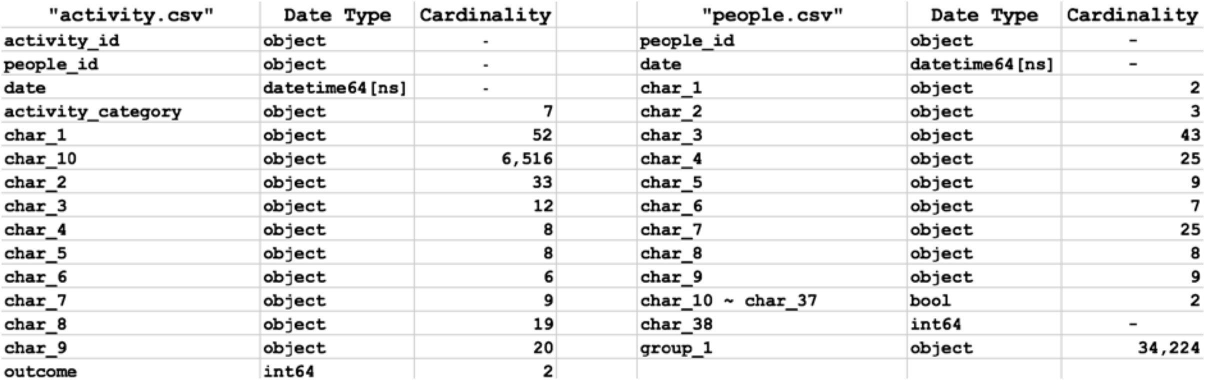


outcome	char_1_ppl	group_1	char_2_ppl	date_ppl	char_3_ppl	char_4_ppl	char_5_ppl	char_6_ppl	char_7_ppl	char_8_ppl	char_9_ppl
0	type 2	group 17304	type 2	2021-06-29	type 5	type 5	type 5	type 3	type 11	type 2	type 2
0	type 2	group 17304	type 2	2021-06-29	type 5	type 5	type 5	type 3	type 11	type 2	type 2
0	type 2	group 17304	type 2	2021-06-29	type 5	type 5	type 5	type 3	type 11	type 2	type 2
0	type 2	group 17304	type 2	2021-06-29	type 5	type 5	type 5	type 3	type 11	type 2	type 2
0	type 2	group 17304	type 2	2021-06-29	type 5	type 5	type 5	type 3	type 11	type 2	type 2

The data sets have four types of fields:

- Indexing fields: “people_id”, “activity_id”
- Datatime fields: “date” for both tables
- Object or bool fields: characteristics related to people or activity
- Label: “outcome”, the prediction target, either “0” or “1”
- “NaNs”: NaN existis in most categorical features, treat them as one type like “type 0”

Table. 2 Data Description. As shown in the data set, the majority of the fields are categorical with low cardinality. Hence the tree model would be more feasible than the linear model for the problem.



"activity.csv"	Date Type	Cardinality	"people.csv"	Date Type	Cardinality
activity_id	object	-	people_id	object	-
people_id	object	-	date	datetime64[ns]	-
date	datetime64[ns]	-	char_1	object	2
activity_category	object	7	char_2	object	3
char_1	object	52	char_3	object	43
char_10	object	6,516	char_4	object	25
char_2	object	33	char_5	object	9
char_3	object	12	char_6	object	7
char_4	object	8	char_7	object	25
char_5	object	8	char_8	object	8
char_6	object	6	char_9	object	9
char_7	object	9	char_10 ~ char_37	bool	2
char_8	object	19	char_38	int64	-
char_9	object	20	group_1	object	34,224
outcome	int64	2			

Exploratory Visualization

The data visualization will show how the binary label “outcome” distributes and how each predictor correlates with the label, which is helpful to understand each field and build the expectation of prediction. 2% of total data points is sampled to make the exploratory plotting. (see EDA part)

Fig. 2 A plot showing how the binary outcome distributes. The distribution is nearly balanced, which saves the work for weighting the data points or building a balanced data set.

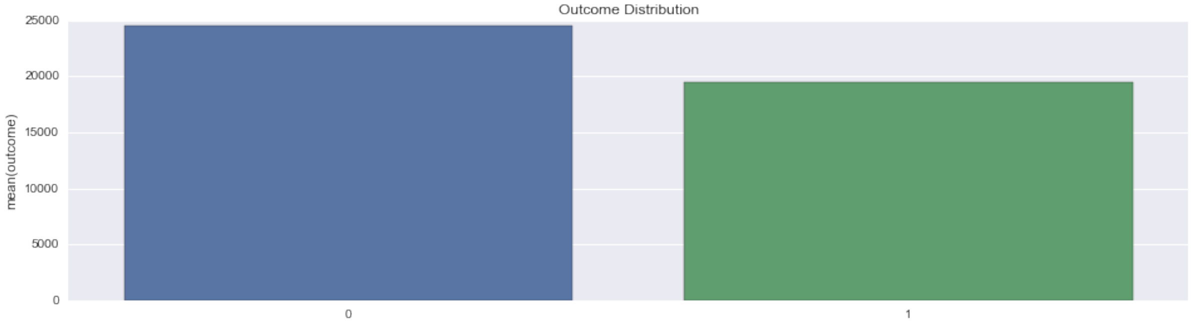


Fig. 3 A plotting showing the activity predictors with low cardinality including “char_1 ~char9” and “activity_category”. “char_1 ~char9” have the similar pattern with the majority outcome falling in the type “0” which actually NaN after imputing, while “activity_category” provides more variance for the outcome. Here only shows “char_9” for simplicity.

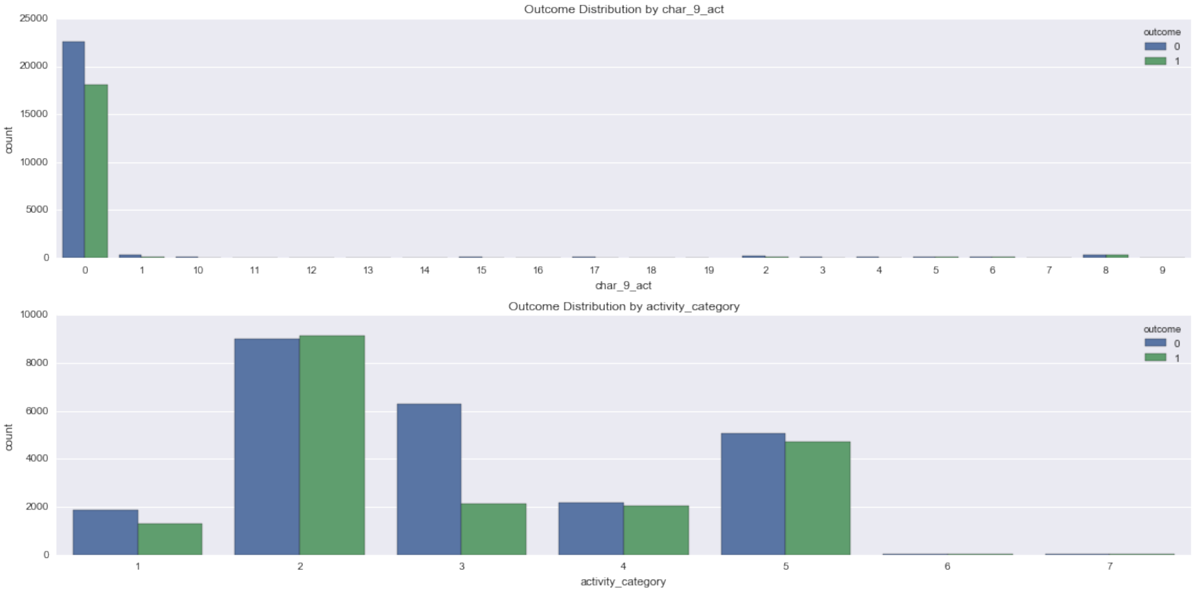


Fig. 4 A plotting showing the bool predictors from “char_10 ~char37”. They appear to have identical patterns. Here only shows “char_10_ppl and char_37” for simplicity.

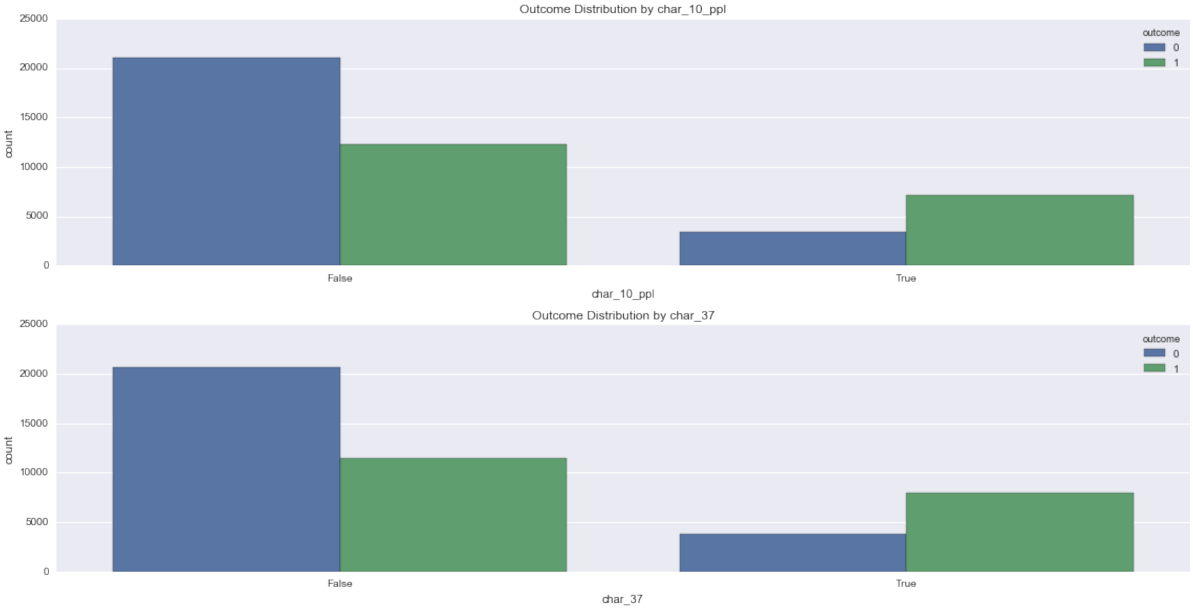


Fig. 5 A plotting showing the non-bool predictors of people. They show more variance compared with previous predictors, which might contribute more to the final classifier. Here only shows "char_4_ppl and char_8_ppl" for simplicity.

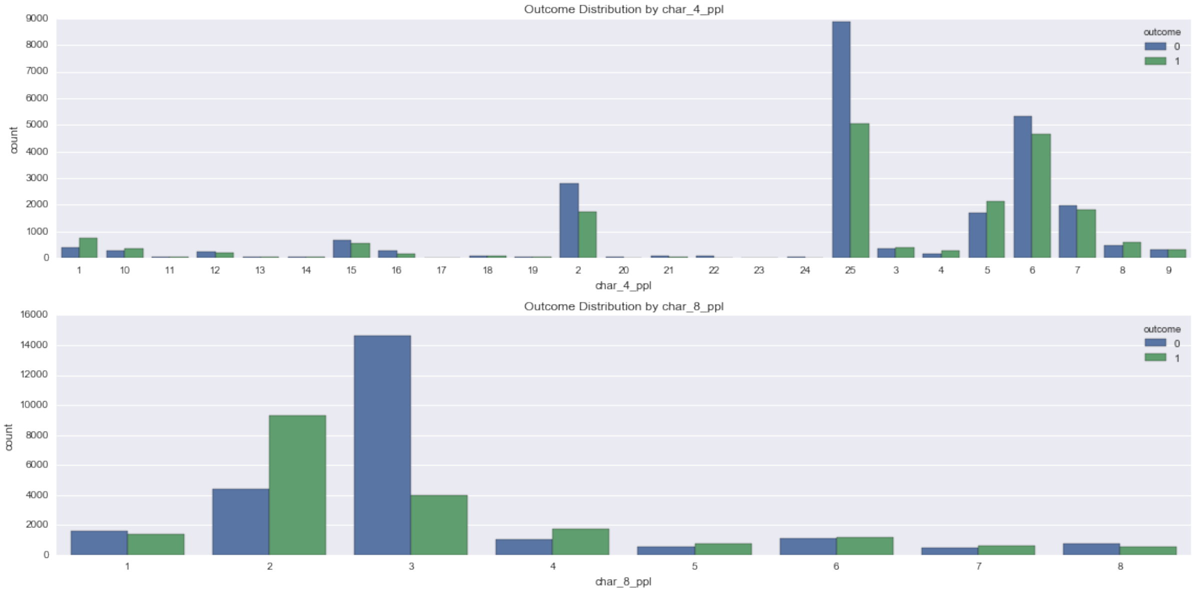
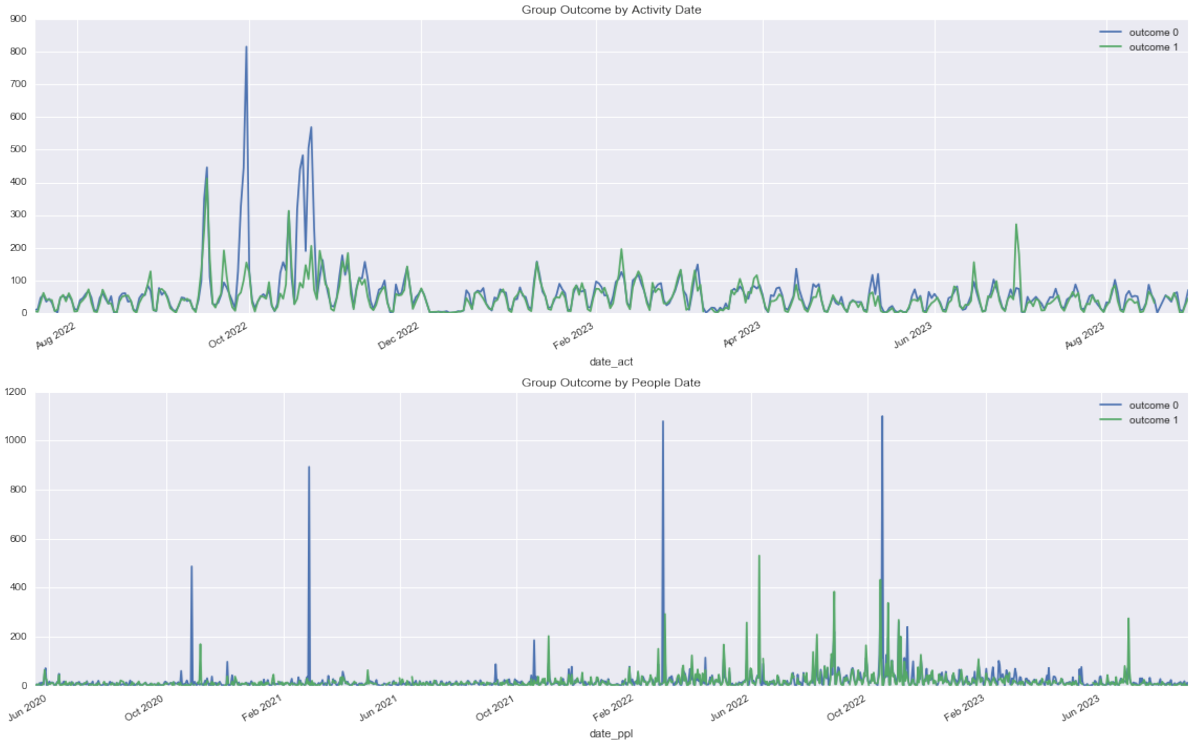


Fig. 6 A plotting showing the date predictors. Both vibrate weekly or month. Therefore, the date is decomposed into year, month, week, day, and day of week in the later step.



Algorithms and Techniques

Tree classifier is a better choice for this project, since the predictors are all but categorical features. Decision Tree is the basic tree model, and it requires little data preparation of normalization or dummy transformation as the linear models. It's easy to interpret and visualize. However, decision tree tends to go overfitting without proper tuning. Hence, Random Forest is introduced as the final classifier. A random forest is to ensemble a number of decision tree classifiers on various sub-samples of the dataset. The prediction results are averaged to improve the predictive performance and control over-fitting. The following parameters are the key to optimize the classifier:

- "max_features": the number of features chosen in each tree. Generally, increasing max_features improves the model performance, however, this is not necessarily true as too much more features decrease the diversity of model.
- "n_estimators": the number of trees chosen in the forest. More trees always output better prediction scores. But it takes more computation time.
- "min_sample_leaf": the threshold number of child leaf to split the node. Usually, a smaller leaf will improve training performance but tend to cause over-fitting.
- "criterion": gini or entropy, the function to measure the quality of split.

In the training stage, all the data are input into the Random Forest Classifier, then the classifier is applied on grid search to find the optimal classifier from the potential parameters pool. The evaluation is implemented in the cross validation with ROC score. (see random forest part)

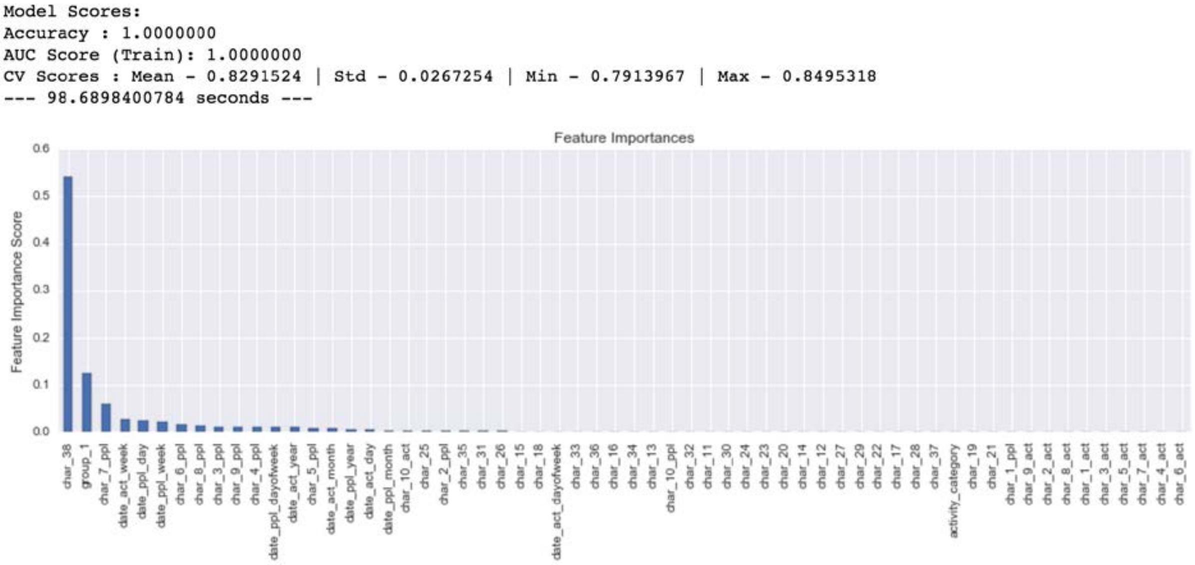
Benchmark

A dummy classifier was used to build the initial prediction. The classifier just predicted as the same outcome as the majority outcome in the train data. It output the accuracy of 0.56, basically the proportion of outcome 0 in the data visualization.

After then, a decision tree classifier was used to benchmark the tree performance. It gave us accuracy of 100% for training data (without pruning and stopping condition, decision tree always reaches 0 loss by memorizing the training data), and CV ROC score of 0.83.

Apparently, the model was overfitting. Rather than continuing with pruning or tuning parameters of decision tree, we went directly to the powerful random forest to control overfitting for the next stage. (see dummy and decision tree part)

Fig. 7 A plotting showing the score and feature importance of decision tree classifier



```
Model Scores:
Accuracy : 1.0000000
AUC Score (Train): 1.0000000
CV Scores : Mean - 0.8291524 | Std - 0.0267254 | Min - 0.7913967 | Max - 0.8495318
--- 98.6898400784 seconds ---
```

Methodology

Data Preprocessing

The data preprocessing consists of the following steps:

- Joined people and activity data sets on “people_id”.
- Stripped categorical value into numeric, like “type 1” to “1”.
- Imputed the “NaN” with “0” as one specific type.
- Transformed the bool values into binary 0 or 1
- Extracted the year, month, week, day, and day of week from date time field.

Besides, the sample data for exploratory visualization was dropped to control overfitting. (see EDA part)

Implementation

The modeling process was implemented in Jupyter Notebook. It can be further dived into the following steps:

1. Loaded people and activity data and preprocess them as the previous section.
2. Implemented a helper function of “model_fit”, which fits the data with the specific classifier to output the accuracy, roc score, running time, and a plotting of feature importance.
3. For tuning random forest, a dictionary containing candidate parameters was prepared and then fed into the grid search function (Sklearn package) to find the optimal parameters.
4. The optimal classifier was output from the grid search to predict the final results.

(see random forest part)

As for the complications of the coding process, it was efficient to create a wrapper function like “model_fit” to output all information for evaluation. Because, tuning process reused the same codes many times. Besides, in tuning random forest, a higher number of estimators was always chosen by the grid search. Therefore, it was necessary to search on the number of estimators at least. A fixed lower number of estimators took much less time in tuning other combinations of parameters.

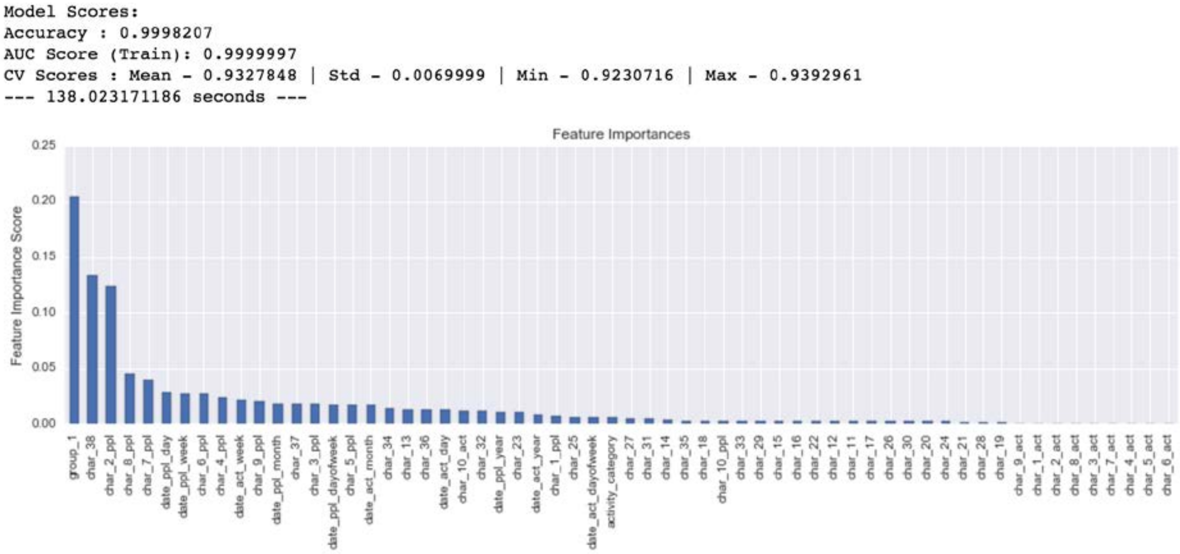
Refinement

As mentioned in the benchmark section, the default decision tree achieved the ROC scores of 0.829. To improve the prediction performance, we moved onto the random forest. The default settings reached 0.933. Then the ROC scores increase by using the following techniques:

1. By searching the combinations of “min_samples_leaf”, “max_features”, and “criterion”, the ROC score reached 0.945 with “min_samples_leaf” of 8, “max_features” of 10, and “criterion” of “entropy”.
2. Continued to tune “max_features” with higher features, since 10 is the upper boundary of the first search. The ROC score achieved 0.950 with 20 “max_features”.

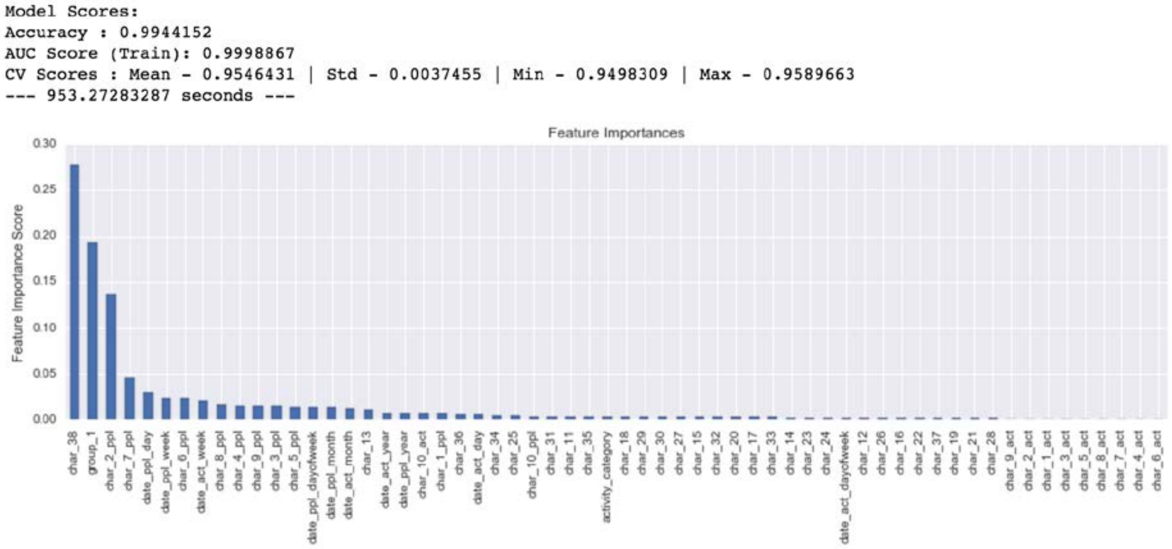
3. Based on the previous parameters, “n_estimators” was searched on and the ROC score ended up with 0.955 under 40 estimators. The performance can keep improving as higher estimators are used, but the running time becomes longer than before. In the final model, the running time reached 938 seconds, about 7 times of the 138 seconds under the default settings.

Fig. 8 A plotting showing the score and feature importance of random forest under the default settings.



```
Model Scores:
Accuracy : 0.9998207
AUC Score (Train): 0.9999997
CV Scores : Mean - 0.9327848 | Std - 0.0069999 | Min - 0.9230716 | Max - 0.9392961
--- 138.023171186 seconds ---
```

Fig. 9 A plotting showing the score and feature importance of random forest under the optimal settings.



```
Model Scores:
Accuracy : 0.9944152
AUC Score (Train): 0.9998867
CV Scores : Mean - 0.9546431 | Std - 0.0037455 | Min - 0.9498309 | Max - 0.9589663
--- 953.27283287 seconds ---
```

(see random forest part)

Results

Model Evaluation and Validation

During the modeling, a default cross validation was used in both grid search or the final model fitting. To make the work reproducible, a random state of 1234 was applied along the way. The final optimal model has the following description:

- The number of estimators is 40, it can be higher, but we stop here considering the running time.
- The max features are 20.
- The minimum samples leaf is 8.
- The entropy criterion is used to measure the quality of split.

To verify the generalization of the final model, the final model was used to predict the test data supplied by Red Hat on Kaggle. The prediction reached 0.964873 in the leaderboard, higher than the CV score of 0.954.

Justification

Based on the previous section, it can be seen that the prediction performance has improved significantly from the dummy classifier, decision tree, to the random forest under cross validation. It is also noticed that the test score is higher than the cv score, which means that the final model does has a better generalization performance. Therefore, the model is useful in predicting the activity outcome of a specific customer in spite of a loss of around 0.04.

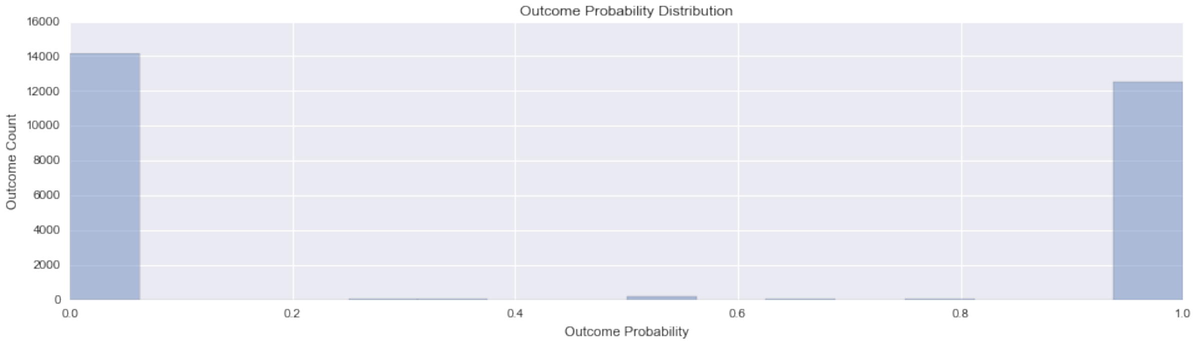
As for the feature importance (shown in Fig. 8), “char_38”, “group_1”, and “char_2_ppl” are top 3 important features. Date time types predictors also have some influence on the model. Bool types feature are relatively nominal and evenly in feature importance.

Conclusion

Free-Form Visualization

Other important plotting that not shown in previous sections (see EDA part):

Fig. 10 A plotting showing outcome probability distribution. The axis x represents the outcome probability. It is averaging all outcomes by one people_id. Since the outcome is either 0 or 1, so the probability represents the percentage of outcome 1 in a specific people. It seems some people always go into 0 outcome or 1 outcome. And Both almost share the whole outcome. The rest uncertain customers are much fewer in quantity, which partly explains why the final score can reach high close to 1.



Reflection

The modeling process for this project can be summarized as the following steps:

1. Described the problem and the objectives
2. Downloaded and preprocessed the data sets
3. Created the benchmark for the dummy classifier
4. Built the baseline decision tree classifier
5. Moved onto the ensemble random forest classifier
6. Tuned the key parameters to reach the optimal classifier
7. Tested the performance in the Kaggle Leaderboard

Understanding the data is the key to build and improve any model. I found step 2 difficult, especially as all predictors are anonymous in this completion. Except for necessary encoding for categorical features, some data time transformation was made after EDA. The possible balance issue for binary problem were ruled out. So was the time series split of train and test data. I also checked the possible data leakage. These work made it clear to decide on the feature and model selection.

As for the interesting part of this project, it was nice to see the model was built up from dummy classifier, basic decision tree, and the final random forest. Noticing how the random forest controlled the overfitting of decision tree and how the tuning process improved the whole performance step by step was the happy time.

Improvement

To achieve better performance, on one side, introducing new features from datatime was possible way. On another side, using other boosting methods like Gradient Boosting or XGBoosting can be an alternative, except for increasing the number of estimators in the random forest. Rather than simply averaging the predictions of each tree, the boosting builds a sequence of tree classifier, and the later tree focuses on the places where the previous tree makes mistakes.