Comp4331 Project Report

Group member: Li Chung Yat, Yip Ho Ming
Student ID: 20422979, 20421092

Background

There are many massive open online courses running on the internet these days. For example, course to use certain software or lectures to explain theoretical and computational subjects. However, there are extremely high dropout rate on these

Objectives

Due to high dropout rate on massive open online course, this project aims to study causes and pattern of dropouts by modelling prediction classifier on one of the MOOC in mainland China, XuetangX.

Introduction

We have tried multiple classification tools: Logistic Regression, Random Forest and Neural Network are the three being chosen.

Logistic regression is being choose because the target variable is a binary variable, either drop or stay. Logistic regression has good performance in binary classification with a balanced dataset. Besides, it can show the impacts of each variables on the result based on the coefficient of estimators. If the coefficient of attribute is low, it is not impactful to predicting dropout. This gives us direction and insights to adjust data in preprocessing.

Random forest is chosen because it is one of the most accurate and simple learning algorithms available. It can give us a brief estimation of the result as a example. For many data sets, it produces a highly accurate classifier. In this case, the data sets are a very large amount, so random forest should also give an accurate and reliable result. In addition, it runs efficiently on large databases, so the running time would not be so long as the other two classifying methods. It also gives estimates of what variables are important in the classification, so that we can drop some non-useful variables later on to speed up the overall running time. The data is observed that not balanced, so random forest is a great method as it can balancing the data with the "RandomForestClassifier" function in sklearn and find the error in class population unbalanced data sets. However, random forests have been observed to overfit for some datasets with noisy classification task, especially predicting the training result with a training model. For data including categorical variables with different number of levels, random forests are biased in favor of those attributes with more levels. Hence, the variable importance scores from random forest are not reliable for this type of data.

We choose Neural Network is because we want to try a classifier which has good performance in solving complex pattern because logistic regression and random forest already chosen are both linear solver (Logistic regression in fact exist convergence behavioral but Neural Network will perform much better). The dataset provided is large which provide great learning condition for neural network.

Data Preprocessing

I. Import Data
   Both training and testing 's enrollment, log, truth csv and date csv are imported using read_csv function of pandas. Our project did not use object csv because it only contains around 20000 unique participants' information of MOOC. There are only two ways to use object csv, either shrink down the size of dataset and use those participants who have information in object csv or concatenate but left sparse in those do not have information. Method 1 is not feasible because the model precision and recall will drop due to loss of information in large scale. Attribute added by object csv has very low contribution to prediction results using Method 2 as shown by the logistic regression. Thus, object csv's predictors are not being included for our classification model because they are not impactful.

II. Indicator variable

Using pandas' get_dummies function, we convert all course_id into indicator variables. Each enrollment id can only have 1 course id attached, if an enrolled student participated in course A, indicator variable 'course A' will have value 1 for that enrollment id.

'Skip10' is another dummy variable we have. This column is decided by calculating the duration between each event of each enrollment_id from starting date of the course to ending date of the course. Hence, log.csv, date.csv and enrollment_train.csv are also used for determining. From the enrollment_train.csv, I extract the course_id that each enrollment_id taken and find the starting date and ending date finding the same course_id in date.csv. Then, we transfer all columns about the date to datetime format, which will be easier for calculation. Finally, we can calculate the duration of activity of each enrollment_id from the starting date of the course to the ending date of the course. If any duration is larger than 10 days, the enrollment_id will be determined as dropped the course and the 'Skip10' of that enrollment_id will be 1. Otherwise, it will keep as 0.

	enrollment_id	1pvLqtotBsKv7QSOsLicJDQMHx3lui6d	3VkHkmOtom3jM2wCu94xgzzu1d6Dn7or	3cnZpv6ReApmCaZyaQwi2izDZxVRdC01	skip10
0	1	0	0	0	0
1	4	0	0	0	0
2	5	0	0	0	0
3	7	0	0	0	0
4	13	0	0	0	1
5	14	0	0	0	0
					1

III. Numerical Variable

By grouping enrollment_id in log csv, we can obtain total count of events encountered and source of these events for each enrolled student. After carrying out logistic regression, we found that some of these numerical variables are exist strong multicollinearity. Still, most of them are critical in predicting student dropout rate.

event_access	event_discussion	event_navigate	event_page_close	event_problem	event_video	event_wiki	source_browser	source_server
107.0	0.0	25.0	66.0	87.0	29.0	0.0	195.0	119.0
64.0	0.0	15.0	10.0	6.0	4.0	0.0	19.0	80.0
226.0	34.0	30.0	87.0	170.0	86.0	0.0	431.0	202.0
203.0	33.0	20.0	60.0	94.0	69.0	0.0	325.0	154.0
200.0	4.0	15.0	25.0	150.0	69.0	0.0	314.0	149.0
60.0	0.0	20.0	5.0	12.0	5.0	0.0	19.0	83.0
90.0	3.0	14.0	18.0	17.0	10.0	1.0	47.0	106.0

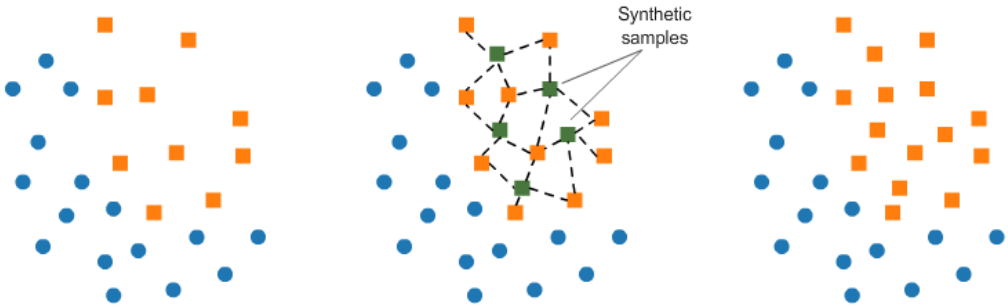
IV. Standardization

All numerical variables are standardized using Min-Max scaling. For a binary classification, min-max scaling is the best scale to use as it avoids magnitude bias raised due to different original scale of variables.

event_access	event_discussion	event_navigate	event_page_close	event_problem	event_wiki	source_browser	source_server
0.049468	0.000000	0.038521	0.169666	0.085799	0.000000	0.085265	0.023179
0.029589	0.000000	0.023112	0.025707	0.005917	0.000000	0.008308	0.015582
0.104485	0.007798	0.046225	0.223650	0.167653	0.000000	0.188456	0.039346
0.093851	0.007569	0.030817	0.154242	0.092702	0.000000	0.142108	0.029996
0.092464	0.000917	0.023112	0.064267	0.147929	0.000000	0.137298	0.029022
0.027739	0.000000	0.030817	0.012853	0.011834	0.000000	0.008308	0.016167
0.041609	0.000688	0.021572	0.046272	0.016765	0.002959	0.020551	0.020647

V. SMOTE

As the given dataset is not a balanced dataset, we decided to use the method SMOTE to balance the dataset so as to give a more precise result on both prediction result. The naïve model that predict all result as 1 can have accuracy with around 80 percent. Thus, it is a must to perform balancing of dataset in this case. The technique we used is SMOTE (Synthetic Minority Over-Sampling Technique), clearly stated in its name, it is a over-sampling technique to balance a dataset. Over-sampling increase the weighting of minority observations whereas under-sampling cut majority group observation to reach balance. We both agreed that over-sampling is a more suitable technique in this case. As the size is so imbalance that we will lose so many information if we do the under-sampling approach. SMOTE carries out balancing by choosing similar observations and altering that observation one column at a time by a random value limited by a threshold in difference between neighboring records.



Raw Performing SMOTE Processed data

Classification methods

I. Logistic Regression

a. Parameters tuning

Before tuning any parameters, we first fit preprocessed data into logistic regression model with default setting. The classification summary of the first-fit result is printed on the next page. We can see the magnitude of coefficient is quite variate. Some of them only have zero point something whereas there are some variables have coefficient over 200. Most of the variables that have small coefficient are different courses that student have while the large one come from total count of certain event. However, both of these results actual make a lot of sense. In logistic regression, when the model carries prediction on dependent variable, all predictors X will be plugged in into the equation generated by the training model and output the prediction with cutoff value 0.5 in a balanced dataset in which range between 0 to 1. Thus, attributes having decimal value coefficient are actually reasonable. Some coefficients of courses have positive value and some are negative, this observation can be interpreted as some courses quality or material covers do not favor students but some are appealing that provide the motivation to continue the course. For those variables with large coefficient, we can see a pattern that they are those numerical data we obtained from log csv. We suspect there are strong multicollinearity problem between predictors or the range of these attributes are too large. Extreme value in raw dataset leading to standardization of data off-scaled and having a very small value. Logistic regression model compensates these small value impact on prediction by assigning extremely large estimation coefficient (Beta head) in the model.

First-fit classification summary:

	coef	std err	z	P>|z|	[0.025	0.975]
const	1.0473	nan	nan	nan	nan	nan
1pvLqtotBsKv7QSOsLicJDQMHx3lui6d	-0.3011	nan	nan	nan	nan	nan
3VkHkmOtom3jM2wCu94xgzzu1d6Dn7or	0.2311	nan	nan	nan	nan	nan
3cnZpv6ReApmCaZyaQwi2izDZxVRdC01	-0.2895	nan	nan	nan	nan	nan
5Gyp41oLVo7Gg7vF4vpmggWP5MU70QO6	0.1831	nan	nan	nan	nan	nan
5X6FeZozNMgE2VRi3MJYjkkFK8SETtu2	-0.0038	nan	nan	nan	nan	nan
7GRhBDsirIGkRZBtSMEzNTyDr2JQm4xx	0.2915	nan	nan	nan	nan	nan
81UZtt1JJwBFYMj5u38WNKCSVA4IJSDv	-0.3436	nan	nan	nan	nan	nan
9Bd26pfDLvkPINwLnpaGcf0LrLUvY1Mz	-0.4468	nan	nan	nan	nan	nan
9Mq1P5hrrLw6Bh9X4W4ZjisQJDdxjz9x	-0.1345	nan	nan	nan	nan	nan
9zpXzW9zCfU8KGBWkhIsGH8B8czISH4J	0.0827	nan	nan	nan	nan	nan

A3fsA9Zfv1X2fVEQhTw51IKENdNrEqT3	-0.1152	nan	nan	nan	nan	nan
AXUJZGmZ0xaYSWazu8RQ1G5c76ECT1Kd	0.1161	nan	nan	nan	nan	nan
DABrJ6O4AotFwuAbfo1fuMj40VmMpPGX	0.1206	nan	nan	nan	nan	nan
DPnLzkJJqOOPRJfBxIHbQEERiYHu5ila	-0.1004	nan	nan	nan	nan	nan
Er0RFawC4sHagDmmQZcBGBrzamLQcblZ	0.2188	nan	nan	nan	nan	nan
G8EPVSXsOYB5YQWZGiz1aVq5Pgr2GrQu	-0.2077	nan	nan	nan	nan	nan
H2IDW05SyKnwntZ6Fora76aPAEswcMa5	-0.4729	nan	nan	nan	nan	nan
HbeAZjZFFQUe90oTP0RRO0PEtRAqU3kK	0.1835	nan	nan	nan	nan	nan
I7Go4XwWgpjRJM8EZGEnBpkfSmBNOlsO	0.1110	nan	nan	nan	nan	nan
KHPw0gmg1Ad3V07TqRpyBzA8mRjj7mkt	-0.0747	nan	nan	nan	nan	nan
NmbZ3BmS8V4pMg6oxXHWpqqMZCE1jvYt	-0.0871	nan	nan	nan	nan	nan
RXDvfPUBYFIVdlueBFbLW0mhhAyGEqpt	0.1495	nan	nan	nan	nan	nan
SpATywNh6bZuzm8s1ceuBUnMUAeoAHHw	-0.5811	nan	nan	nan	nan	nan
TAYxxh39I2LZnftBpL0LfF2NxzrCKpkx	-0.1486	nan	nan	nan	nan	nan
V4tXq15GxHo2gaMpaJLZ3IGEkP949IbE	-0.1558	nan	nan	nan	nan	nan
WM572q68zD5VW8pcvVTc1RhhFUq3iRFN	0.0725	nan	nan	nan	nan	nan
Wm3dddHSynJ76EJV6hyLYKGGRL0JF3YK	0.3453	nan	nan	nan	nan	nan
X78EhlW2JxwO1l6S3U4yZVwkEQpKXLOj	-0.1868	nan	nan	nan	nan	nan
a2E7NQC7nZB7WHEhKGhKnKvUWtsLAQzh	0.2356	nan	nan	nan	nan	nan
bWdj2GDclj5ofokWjzoa5jAwMkxCykd6	-0.0967	nan	nan	nan	nan	nan
fbPkOYLVPtPglt0MxizjfFJov3JbHyAi	0.7437	nan	nan	nan	nan	nan
gvEwgd64UX4t3K7ftZwXiMkFuxFUAqQE	0.6008	nan	nan	nan	nan	nan
mTmmr5zd8l4wXhwiULwjSmSbi9ktcFmV	0.3025	nan	nan	nan	nan	nan
nSfGxfEtzw5G72fVbfaowxsV46Pg1xlc	0.2086	nan	nan	nan	nan	nan
q6A6QG7qMpyNcznyT2XalxnfNGkZRxXl	0.1926	nan	nan	nan	nan	nan
shM3Yy9vxHn2aqjSYfQXOcwGo0hWh3Ml	-0.1212	nan	nan	nan	nan	nan
tXbz2ZYaRyb2ZsWUBPoYzAmisOhHQrYl	0.7151	nan	nan	nan	nan	nan
xMd9DzNyUCTLRPVbwWVzf4vq06oqrTT1	-0.3426	nan	nan	nan	nan	nan
ykoe1cCWK134BJmfbNoPEenJOlWdtQOZ	0.1523	nan	nan	nan	nan	nan
event_access	43.1753	5.732	7.533	0.000	31.941	54.409
event_discussion	221.6382	12.281	18.047	0.000	197.568	245.709
event_navigate	6.1315	2.115	2.899	0.004	1.987	10.276
event_page_close	-1.1540	1.133	-1.019	0.308	-3.374	1.066
event_problem	8.7619	1.893	4.629	0.000	5.052	12.472
event_wiki	23.0275	1.519	15.155	0.000	20.049	26.006
source_browser	-22.5943	4.126	-5.476	0.000	-30.681	-14.508
source_server	-268.6443	14.147	-18.990	0.000	-296.371	-240.918
skip10	0.8684	0.016	54.375	0.000	0.837	0.900

VIF testing:
Based on what we suspected, we decided to perform VIF test on the model to check for multicollinearity. VIF test is Variance Inflation Factor test, the test is conducted by the ratio of variance in logistic regression model with multiple terms, divided by the variance of a model with one term alone. VIF of each attribute is calculated by:

$$VIF_i = \frac{1}{1 - R_i^2} \quad ; where\ R^2\ is\ the\ goodness\ of\ fit\ measure\ of\ model$$

	VIF Factor	features
0	1.1	1pvLqtotBsKv7QSOsLicJDQMHx3lui6d
1	1.1	3VkHkmOtom3jM2wCu94xgzzu1d6Dn7or
2	1.1	3cnZpv6ReApmCaZyaQwi2izDZxVRdC01
3	1.1	5Gyp41oLVo7Gg7vF4vpmggWP5MU70QO6
4	1.0	5X6FeZozNMgE2VRi3MJYjkkFK8SETtu2
5	1.0	7GRhBDsirIGkRZBtSMEzNTyDr2JQm4xx
6	1.3	81UZtt1JJwBFYMj5u38WNKCSVA4IJSDv
7	1.2	9Bd26pfDLvkPINwLnpaGcf0LrLUvY1Mz
8	1.0	9Mq1P5hrrLw6Bh9X4W4ZjisQJDdxjz9x
9	1.0	9zpXzW9zCfU8KGBWkhlsGH8B8czISH4J
10	1.1	A3fsA9Zfv1X2fVEQhTw51lKENdNrEqT3
11	1.2	AXUJZGmZ0xaYSWazu8RQ1G5c76ECT1Kd
12	1.0	DABrJ6O4AotFwuAbfo1fuMj40VmMpPGX
13	1.4	DPnLzkJJqOOPRJfBxIHbQEERiYHu5ila
14	1.1	Er0RFawC4sHagDmmQZcBGBrzamLQcblZ
15	1.0	G8EPVSXsOYB5YQWZGiz1aVq5Pgr2GrQu
16	1.3	H2IDW05SyKnwntZ6Fora76aPAEswcMa5
17	1.1	HbeAZjZFFQUe90oTP0RRO0PEtRAqU3kK
18	1.5	I7Go4XwWgpjRJM8EZGEnBpkfSmBNOlsO
19	1.0	KHPw0gmg1Ad3V07TqRpyBzA8mRjj7mkt
20	1.1	NmbZ3BmS8V4pMg6oxXHWpqqMZCE1jvYt
21	1.0	RXDvfPUBYFlVdlueBFbLW0mhhAyGEqpt
22	1.2	SpATywNh6bZuzm8s1ceuBUnMUAeoAHHw
23	1.0	TAYxxh39I2LZnftBpL0LfF2NxzrCKpkx
24	1.1	V4tXq15GxHo2gaMpaJLZ3IGEkP949lbE
25	1.0	WM572q68zD5VW8pcvVTc1RhhFUq3iRFN
26	1.0	Wm3dddHSynJ76EJV6hyLYKGGRL0JF3YK
27	1.0	X78EhlW2JxwO1l6S3U4yZVwkEQpKXLOj
28	1.1	a2E7NQC7nZB7WHEhKGhKnKvUWtsLAQzh
29	1.0	bWdj2GDclj5ofokWjzoa5jAwMkxCykd6

	VIF Factor	features
30	1.0	fbPkOYLVPtPgIt0MxizjfFJov3JbHyAi
31	1.0	gvEwgd64UX4t3K7ftZwXiMkFuxFUAqQE
32	1.0	mTmmr5zd8l4wXhwiULwjSmSbi9ktcFmV
33	1.1	nSfGxfEtzw5G72fVbfaowxsV46Pg1xlc
34	1.0	q6A6QG7qMpyNcznyT2XaIxnfNGkZRxXI
35	1.2	shM3Yy9vxHn2aqjSYfQXOcwGo0hWh3MI
36	1.0	tXbz2ZYaRyb2ZsWUBPoYzAmisOhHQrYl
37	1.1	xMd9DzNyUCTLRPVbwWVzf4vq06oqrTT1
38	1.1	ykoe1cCWK134BJmfbNoPEenJOlWdtQOZ
39	147.0	event_access
40	67.1	event_discussion
41	14.5	event_navigate
42	29.0	event_page_close
43	37.2	event_problem
44	1.9	event_wiki
45	159.2	source_browser
46	318.4	source_server
47	1.2	skip10

From the results, we can conclude that our dataset indeed exists multicollinearity problem. According to the common practice of data scientists, attributes which have VIF Factor larger than 10 will be deemed as strong multicollinearity. However, we should not drop all attributes that is larger than threshold 10 at once. It is because after we drop one of the attributes, the VIF Factor of the model will be changed and probably some attributes can be reserved. ‘source_browser’ and ‘source_server’ are both dropped by a function we defined in which drop the predictor has maximum VIF Factor and recalculate the VIF Factor of new model again. Clearly, dropping predictor one by one in VIF testing is crucial, we only drop 2 predictors. If we did not follow this approach, we have to drop 7 attributes in total. To explain, we can take a look back to our raw dataset provided. Certain event will always come along with certain source. For instance, event_access can only happen if source is server. Attribute ‘event’ and ‘source’ exist some relationship in the raw data source log csv.

After VIF test we fit the model and print out the summary of classification again.

Re-fit classification summary:

	coef	std err	z	P>|z|	[0.025	0.97
const	1.0376	2.92e+05	3.56e-06	1.000	-5.72e+05	5.72e+(
1pvLqtotBsKv7QSOsLicJDQMHx3lui6d	-0.0690	2.92e+05	-2.36e-07	1.000	-5.72e+05	5.72e+(
3VkHkmOtom3jM2wCu94xgzzu1d6Dn7or	0.3321	2.92e+05	1.14e-06	1.000	-5.72e+05	5.72e+(
3cnZpv6ReApmCaZyaQwi2izDZxVRdC01	-0.3629	2.92e+05	-1.24e-06	1.000	-5.72e+05	5.72e+(
5Gyp41oLVo7Gg7vF4vpmggWP5MU70QO6	0.1792	2.92e+05	6.14e-07	1.000	-5.72e+05	5.72e+(
5X6FeZozNMgE2VRi3MJYjkkFK8SETtu2	0.0390	2.92e+05	1.34e-07	1.000	-5.72e+05	5.72e+(
7GRhBDsirIGkRZBtSMEzNTyDr2JQm4xx	0.3687	2.92e+05	1.26e-06	1.000	-5.72e+05	5.72e+(
81UZtt1JJwBFYMj5u38WNKCSVA4IJSDv	-0.3262	2.92e+05	-1.12e-06	1.000	-5.72e+05	5.72e+(
9Bd26pfDLvkPINwLnpaGcf0LrLUvY1Mz	-0.3577	2.92e+05	-1.23e-06	1.000	-5.72e+05	5.72e+(
9Mq1P5hrrLw6Bh9X4W4ZjisQJDdxjz9x	-0.1417	2.92e+05	-4.86e-07	1.000	-5.72e+05	5.72e+(
9zpXzW9zCfU8KGBWkhlsGH8B8czISH4J	0.1027	2.92e+05	3.52e-07	1.000	-5.72e+05	5.72e+(
A3fsA9Zfv1X2fVEQhTw51IKENdNrEqT3	-0.2492	2.92e+05	-8.54e-07	1.000	-5.72e+05	5.72e+(
AXUJZGmZ0xaYSWazu8RQ1G5c76ECT1Kd	-0.0175	2.92e+05	-6.01e-08	1.000	-5.72e+05	5.72e+(
DABrJ6O4AotFwuAbfo1fuMj40VmMpPGX	0.0465	2.92e+05	1.6e-07	1.000	-5.72e+05	5.72e+(
DPnLzkJJqOOPRJfBxlHbQEERiYHu5ila	-0.2338	2.92e+05	-8.01e-07	1.000	-5.72e+05	5.72e+(
Er0RFawC4sHagDmmQZcBGBrzamLQcblZ	0.3530	2.92e+05	1.21e-06	1.000	-5.72e+05	5.72e+(
G8EPVSXsOYB5YQWZGiz1aVq5Pgr2GrQu	-0.3090	2.92e+05	-1.06e-06	1.000	-5.72e+05	5.72e+(
H2lDW05SyKnwntZ6Fora76aPAEswcMa5	-0.6103	2.92e+05	-2.09e-06	1.000	-5.72e+05	5.72e+(
HbeAZjZFFQUe90oTP0RRO0PEtRAqU3kK	0.1994	2.92e+05	6.83e-07	1.000	-5.72e+05	5.72e+(
I7Go4XwWgpjRJM8EZGEnBpkfSmBNOlsO	0.0193	2.92e+05	6.62e-08	1.000	-5.72e+05	5.72e+(
KHPw0gmg1Ad3V07TqRpyBzA8mRjj7mkt	-0.0748	2.92e+05	-2.56e-07	1.000	-5.72e+05	5.72e+(
NmbZ3BmS8V4pMg6oxXHWpqqMZCE1jvYt	-0.0804	2.92e+05	-2.75e-07	1.000	-5.72e+05	5.72e+(
RXDvfPUBYFlVdlueBFbLW0mhhAyGEqpt	0.0733	2.92e+05	2.51e-07	1.000	-5.72e+05	5.72e+(
SpATywNh6bZuzm8s1ceuBUnMUAeoAHHw	-0.6046	2.92e+05	-2.07e-06	1.000	-5.72e+05	5.72e+(
TAYxxh39I2LZnftBpL0LfF2NxzrCKpkx	-0.0469	2.92e+05	-1.61e-07	1.000	-5.72e+05	5.72e+(
V4tXq15GxHo2gaMpaJLZ3lGEkP949lbE	-0.0530	2.92e+05	-1.82e-07	1.000	-5.72e+05	5.72e+(
WM572q68zD5VW8pcvVTc1RhhFUq3iRFN	0.1518	2.92e+05	5.2e-07	1.000	-5.72e+05	5.72e+(
Wm3dddHSynJ76EJV6hyLYKGGRL0JF3YK	0.4433	2.92e+05	1.52e-06	1.000	-5.72e+05	5.72e+(
X78EhlW2JxwO1I6S3U4yZVwkEQpKXLOj	-0.3151	2.92e+05	-1.08e-06	1.000	-5.72e+05	5.72e+(
a2E7NQC7nZB7WHEhKGhKnKvUWtsLAQzh	0.1318	2.92e+05	4.51e-07	1.000	-5.72e+05	5.72e+(
bWdj2GDclj5ofokWjzoa5jAwMkxCykd6	-0.1472	2.92e+05	-5.04e-07	1.000	-5.72e+05	5.72e+(
fbPkOYLVPtPglt0MxizjfFJov3JbHyAi	0.7509	2.92e+05	2.57e-06	1.000	-5.72e+05	5.72e+(
gvEwgd64UX4t3K7ftZwXiMkFuxFUAqQE	0.5374	2.92e+05	1.84e-06	1.000	-5.72e+05	5.72e+(
mTmmr5zd8l4wXhwiULwjSmSbi9ktcFmV	0.3755	2.92e+05	1.29e-06	1.000	-5.72e+05	5.72e+(
nSfGxfEtzw5G72fVbfaowxsV46Pg1xlc	0.3151	2.92e+05	1.08e-06	1.000	-5.72e+05	5.72e+(
q6A6QG7qMpyNcznyT2XalxnfNGkZRxXl	0.2636	2.92e+05	9.03e-07	1.000	-5.72e+05	5.72e+(
shM3Yy9vxHn2aqjSYfQXOcwGo0hWh3Ml	-0.0029	2.92e+05	-9.78e-09	1.000	-5.72e+05	5.72e+(
tXbz2ZYaRyb2ZsWUBPoYzAmisOhHQrYl	0.7880	2.92e+05	2.7e-06	1.000	-5.72e+05	5.72e+(
xMd9DzNyUCTLRPVbwWVzf4vq06oqrTT1	-0.4712	2.92e+05	-1.61e-06	1.000	-5.72e+05	5.72e+(
ykoe1cCWK134BJmfbNoPEenJOlWdtQOZ	0.0405	2.92e+05	1.39e-07	1.000	-5.72e+05	5.72e+(
event_access	-39.0012	1.279	-30.491	0.000	-41.508	-36.4
event_discussion	-1.3358	3.458	-0.386	0.699	-8.113	5.4
event_navigate	-33.5053	1.061	-31.594	0.000	-35.584	-31.4
event_page_close	-10.8638	0.414	-26.259	0.000	-11.675	-10.0
event_problem	-0.9954	0.510	-1.952	0.051	-1.995	0.0
event_wiki	5.5039	1.154	4.768	0.000	3.241	7.7
skip10	0.8919	0.016	56.212	0.000	0.861	0.9

From the new report, the courses indicator variables behave similarly with the first fit result. However, there are obvious changes on those 'event' numerical variables. The coefficient is much smaller and these variables are capable to generate some interesting findings. 'access', 'discussion', 'navigate', 'page close' and 'problem' are resulted in an increase of likelihood that a student is going to stay in the course on MOOC. Only 'wiki' will increase the tendency that a student will be predicted as a dropout. All these traits show the number of times students access the platform is positively related to dropout probability. In other word, student who are more active in using the MOOC platform is less likely to drop the course. People might doubt why 'problem' will have positive effect on a student staying and completing the course. The best explanation will be students have to actively use the platform so that they will encounter problems. Still, the magnitude of coefficient is relatively small compare to the others. Showing that negative impacts bring along in encountering troubles in these online platforms will leads to dropout but is compensated by impact for being a frequent user. 'Skip10' is actually a variable which plays as a important role in this model. Although coefficient of those events is significant, the value of data is not due to the extreme value effect on min-max scaling as we have mentioned before. However, 'skip10' has the same nature as our target variable, a binary variable, with coefficient 0.8919. Under the condition that a student did not access the platform for 10 days, the probability of that student being a dropout is 0.8919 which is critical and huge.

Hyperparameter tuning: C
C is the regularization variables in logistic regression, it penalizes increasing magnitude of parameter values so as to prevent model is overfitted by training data.

```
Accuracy for C = 0.01: 0.7345378795802392
Accuracy for C = 0.1: 0.7914531255447478
Accuracy for C = 1: 0.8033068368022871
Accuracy for C = 10.0: 0.8043091726806819
Accuracy for C = 100.0: 0.8045445037130007
Accuracy for C = 1000.0: 0.804500923892201
Accuracy for C = 10000.0: 0.804500923892201
Accuracy for C = 100000.0: 0.8045183558205209
Accuracy for C = 1000000.0: 0.8045270717846809
Accuracy for C = 10000000.0: 0.8045183558205209
Accuracy for C = 100000000.0: 0.8045270717846809
Accuracy for C = 1000000000.0: 0.804500923892201
Accuracy for C = 10000000000.0: 0.8045880835338005
Accuracy for C = 100000000000.0: 0.804500923892201
Accuracy for C = 1000000000000.0: 0.804500923892201
```

When C = 10000000000, the accuracy will be the best, keeping all other parameters as default.

Hyperparameter tuning: tol
Tol is the tolerance for the stopping criteria. This is deemed as a threshold for the model stop searching or the optimal when the threshold is achieved. In logistic regression case, it is the maximum of the gradient for the i-th component.

```
Accuracy for tol = 10: 0.5
Accuracy for tol = 1: 0.7441864519053097
Accuracy for tol = 0.1: 0.799009866471429
Accuracy for tol = 0.01: 0.8040476937558833
Accuracy for tol = 0.001: 0.8043614684656417
Accuracy for tol = 0.0001: 0.804500923892201
Accuracy for tol = 1e-05: 0.8045270717846809
Accuracy for tol = 1e-06: 0.8045270717846809
```

When tol = 1e-5, the accuracy will not further improve. The model will not stop too early or over tolerated.

Hyperparameter tuning: solver
There are few solvers we can choose from in logistic regression. Newtons' method (newton-cg), Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm (lbfgs), Library for Large Linear Classification (liblinear), Stochastic Average Gradient (sag), SAGA (saga). Different solver has different advantages. Most of them are related to memory saving and finding optimal point.

```
Accuracy for solver = newton-cg: 0.8045270717846809
Accuracy for solver = lbfgs: 0.8041697172541227
Accuracy for solver = liblinear: 0.8045270717846809
Accuracy for solver = sag: 0.8045183558205209
Accuracy for solver = saga: 0.8045270717846809
```

We can see Newtons' method, Library for Large Linear Classification and SAGA all have the same accuracy. Based on what sklearn mention in the classifier library, liblinear is suitable for non-multiclass problems whereas saga is good for large dataset. We decided to use liblinear as it is recommended for high dimension dataset according to the winner of ICML 2008 who use liblinear in SVM in the competition.

b. Classification Result

For training dataset:
Confusion matrix:

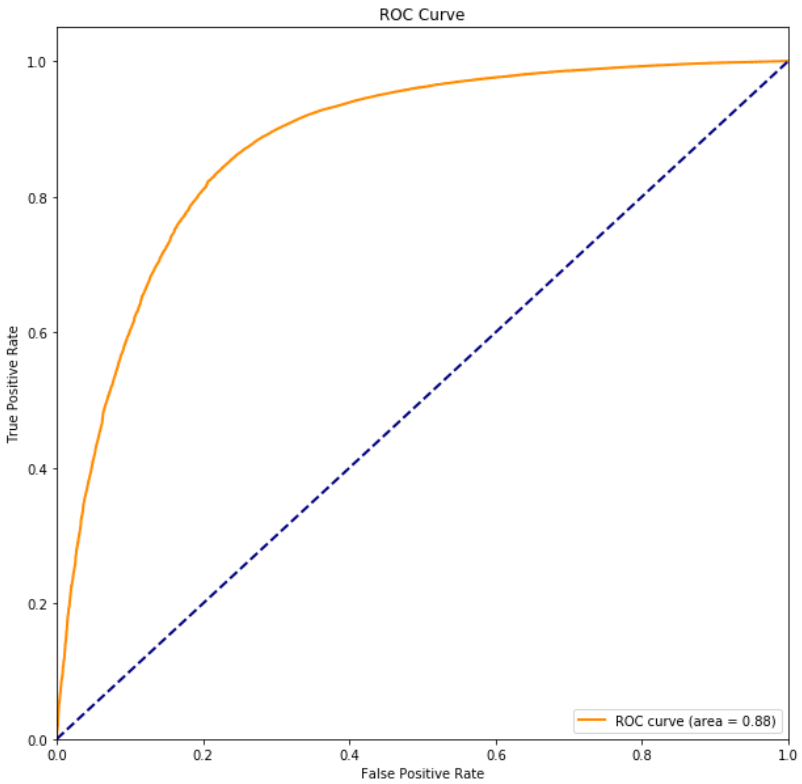
```
[[41624 15742]
[ 6685 50681]]
```

Classification report:

	precision	recall	f1-score	support
0	0.86	0.73	0.79	57366
1	0.76	0.88	0.82	57366
avg / total	0.81	0.80	0.80	114732

```
Accuracy: 0.8045270717846809
```

ROC Curve:



For testing dataset:
Confusion matrix:

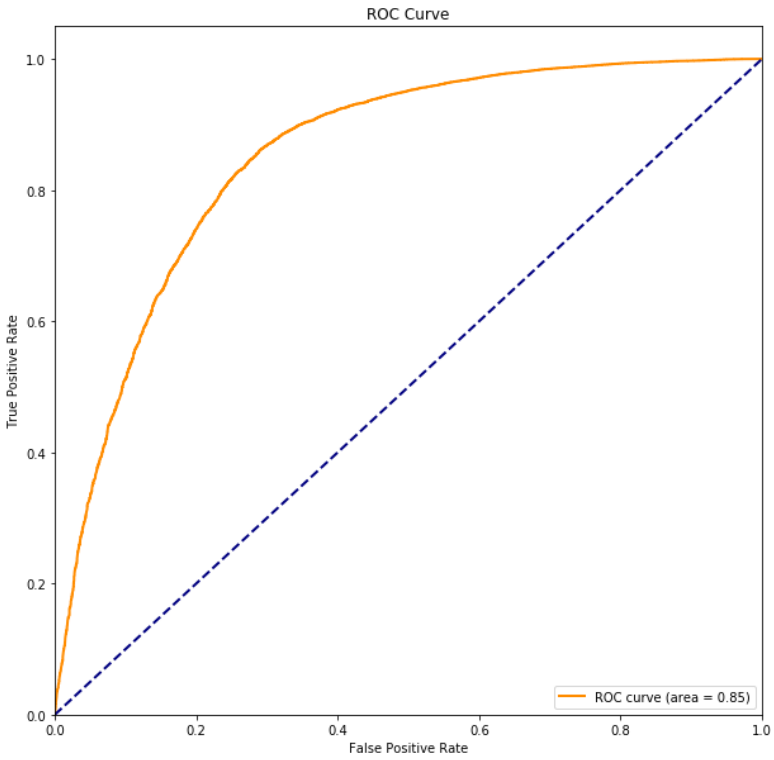
```
[[ 3521  1381]
 [ 2858 16253]]
```

Classification report:

	precision	recall	f1-score	support
0	0.55	0.72	0.62	4902
1	0.92	0.85	0.88	19111
avg / total	0.85	0.82	0.83	24013

```
Accuracy: 0.8234706200807895
```

ROC curve:



Just to emphasis, it is not precise if we just evaluate the accuracy for this case because the dataset is too imbalance that blind guessing with all dropping can achieve accuracy 0.8 in a unbalanced dataset. After balancing, we manage to get a 0.804 in training and even 0.83 in testing dataset which have not undergo any balancing technique. Average precision, recall. F1-score are all over 80 percent in our model. From the ROC Curve we plotted, training dataset has area under the ROC with 0.88, which is quite high and implies that the type I error and type II error in our model is very little.

II. Random Forest

a. Parameters tuning

The parameters that were tuned are “n_estimators”: the number of trees in the forest, “criterion”: the function to measure the quality of a split, “max_depth”: the maximum depth of the tree, “min_samples_split”: the minimum number of samples required to split an internal node, “min_samples_leaf”: the minimum number of samples required to be at a leaf node and “bootstrap”: whether bootstrap samples are used when building trees.

The reason that we chose the parameters above to tune is due to our understanding on these parameters and it effect to the accuracy. In addition, we made a assumption before tuning the parameters, the assumption is that all the parameters are independent to each other, so we may we a better accuracy by tuning them one by one with keeping the previous tuned parameter fixed.

On the other hand, before the tuning process, we have fixed some parameters, they are “random_state”: the seed used by the random number generator, “max_features”: the number of features to consider when looking for the best split and “class_weight”: for balancing the data. “max_features” and “random_state” are fixed is because they can increase the learn result for more learning trial in random forest, in simple, to increase the accuracy. The “class_weight” is fixed is because as we known, the train data sets given are not balanced, so we use this function to balance the data and train the random forest model.

The following is the tuning process:
The first parameter I tuned is “n_estimators”.

```
training dataset accuracy for number of trees used 25: 0.8571607046183317
training dataset accuracy for number of trees used 50: 0.8574938574938575
training dataset accuracy for number of trees used 100: 0.8589514013242827
training dataset accuracy for number of trees used 150: 0.8588264689959605
```

The first four indicators I used are 25, 50, 100 and 150. I found that accuracy near 100 trees is better. Hence, I try other values near 100.

```
training dataset accuracy for number of trees used 80: 0.8592845541998084
training dataset accuracy for number of trees used 90: 0.8589514013242827
training dataset accuracy for number of trees used 100: 0.8589514013242827
training dataset accuracy for number of trees used 110: 0.8585766043393162
training dataset accuracy for number of trees used 120: 0.8586182484487569
training dataset accuracy for number of trees used 130: 0.8590763336526048

training dataset accuracy for number of trees used 60: 0.8590346895431641
training dataset accuracy for number of trees used 70: 0.8586598925581976
training dataset accuracy for number of trees used 80: 0.8592845541998084
training dataset accuracy for number of trees used 130: 0.8590763336526048
training dataset accuracy for number of trees used 140: 0.8588264689959605
```

We can see that at the number of trees used is 80, the model has the highest accuracy.

After that, I keep “n_estimators”=80 and tune other parameter, the second one is “max_depth”.

```
training dataset accuracy for maximun depth of the tree used 50: 0.8587848248865197
training dataset accuracy for maximun depth of the tree used 100: 0.8592845541998084
training dataset accuracy for maximun depth of the tree used 250: 0.8592845541998084
training dataset accuracy for maximun depth of the tree used 500: 0.8592845541998084
```

It shows that when the maximum depth is over 100, the accuracy would not change anymore. Hence, I try deeply in the range between 50 to 100. The reason that I try is because I would like to know the depth of the tree and with a smaller depth, the model can run faster.

```
training dataset accuracy for maximun depth of the tree used 60: 0.8591179777620456
training dataset accuracy for maximun depth of the tree used 70: 0.8592845541998084
training dataset accuracy for maximun depth of the tree used 80: 0.8592845541998084
training dataset accuracy for maximun depth of the tree used 90: 0.8592845541998084
```

```
training dataset accuracy for maximun depth of the tree used 61: 0.8592012659809269
training dataset accuracy for maximun depth of the tree used 62: 0.8592012659809269
training dataset accuracy for maximun depth of the tree used 63: 0.8592845541998084
training dataset accuracy for maximun depth of the tree used 64: 0.8592845541998084
training dataset accuracy for maximun depth of the tree used 65: 0.8592845541998084
training dataset accuracy for maximun depth of the tree used 66: 0.8592845541998084
training dataset accuracy for maximun depth of the tree used 67: 0.8592845541998084
training dataset accuracy for maximun depth of the tree used 68: 0.8592845541998084
training dataset accuracy for maximun depth of the tree used 69: 0.8592845541998084
```

Thus, the depth of the tree is around 63 and I will keep this as our parameter in the model as it is the smallest depth with fastest running time.

Then, keeping the above parameters fixed, I tune the “min_samples_leaf”.

```
training dataset accuracy for the minimum number of samples required to be at a leaf node used 50: 0.816432765585308
training dataset accuracy for the minimum number of samples required to be at a leaf node used 100: 0.809436555199267
training dataset accuracy for the minimum number of samples required to be at a leaf node used 250: 0.8073127056177903
training dataset accuracy for the minimum number of samples required to be at a leaf node used 500: 0.8052305001457544
```

Same as usual, I try with 4 indicators first to see the tendency of the accuracy. I find that with smaller value, the accuracy will be higher.

```
training dataset accuracy for the minimum number of samples required to be at a leaf node used 10: 0.8463748802731853
training dataset accuracy for the minimum number of samples required to be at a leaf node used 20: 0.8325906800483072
training dataset accuracy for the minimum number of samples required to be at a leaf node used 30: 0.8232207554241453
training dataset accuracy for the minimum number of samples required to be at a leaf node used 40: 0.8175155124307667
```

I try some smaller number. We can see that even 10, 20, 30 and 40 are not small enough to show the accuracy in the previous tuning process. So, I try some even smaller number to see how the result would be.

```
training dataset accuracy for the minimum number of samples required to be at a leaf node used 1: 0.8592845541998084
training dataset accuracy for the minimum number of samples required to be at a leaf node used 2: 0.8625327947361846
training dataset accuracy for the minimum number of samples required to be at a leaf node used 3: 0.8614084037812851
training dataset accuracy for the minimum number of samples required to be at a leaf node used 4: 0.8577853662599425
training dataset accuracy for the minimum number of samples required to be at a leaf node used 5: 0.8556615166784658
```

The surprising thing in the result is that the accuracy is not the best at “min_samples_leaf”=1, but 2. Hence, I choose 2 as the parameter for the model.

The next parameter I tune is “min_samples_split”.

```
training dataset accuracy for the minimum number of samples required to split an internal node used 50: 0.8447091158955565
training dataset accuracy for the minimum number of samples required to split an internal node used 100: 0.829509015949694
training dataset accuracy for the minimum number of samples required to split an internal node used 250: 0.8171823595552409
training dataset accuracy for the minimum number of samples required to split an internal node used 500: 0.8098529962936742

training dataset accuracy for the minimum number of samples required to split an internal node used 10: 0.8607420980302336
training dataset accuracy for the minimum number of samples required to split an internal node used 20: 0.855036855036855
training dataset accuracy for the minimum number of samples required to split an internal node used 30: 0.8500395619039687
training dataset accuracy for the minimum number of samples required to split an internal node used 40: 0.8465831008203889

training dataset accuracy for the minimum number of samples required to split an internal node used 2: 0.8625327947361846
training dataset accuracy for the minimum number of samples required to split an internal node used 3: 0.8625327947361846
training dataset accuracy for the minimum number of samples required to split an internal node used 4: 0.8625327947361846
training dataset accuracy for the minimum number of samples required to split an internal node used 5: 0.8628243035022696

training dataset accuracy for the minimum number of samples required to split an internal node used 6: 0.8626577270645067
training dataset accuracy for the minimum number of samples required to split an internal node used 7: 0.8624495065173031
training dataset accuracy for the minimum number of samples required to split an internal node used 8: 0.8621996418606588
training dataset accuracy for the minimum number of samples required to split an internal node used 9: 0.8619497772040144
```

By repeating a similar process in the above, this parameter also has a tendency that the smaller it is, the higher the accuracy. At the time some, just like before, it does not have best accuracy at 1 but 5, which also surprised me during the turning process.

The “bootstrap” will be tuned. In some dataset, whether using bootstrap would bring a significantly different result.

```
training dataset accuracy for not using bootstrap: 0.7538000249864657

training dataset accuracy for using bootstrap: 0.8628243035022696
```

We can see that with using bootstrap, the accuracy is much higher than not using, which is a great example for the phenomenon that I mentioned before. Hence, in this model, I choose to use bootstrap.

The last parameter that we are going to tune is the “criterion”. There are two types of criterion in the Random Forest Classifier in Sklearn. They are “entropy” and “gini”.

```
training dataset accuracy for using gini as criterion: 0.8628243035022696
```

```
training dataset accuracy for using entropy as criterion: 0.862616082955066
```

From the above results, we can see that using “gini” has a slightly higher accuracy then using “entropy” as criterion. However, I would not determine using which one as our parameter based on this result only but using further result about precision and recall.

```
For test dataset
training dataset accuracy: 0.862616082955066


Classification report for classifier RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='entropy', max_depth=63, max_features=None,
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=2,
            min_samples_split=5, min_weight_fraction_leaf=0.0,
            n_estimators=80, n_jobs=None, oob_score=False, random_state=0,
            verbose=0, warm_start=False):
              precision    recall  f1-score   support

           0       0.71      0.55      0.62      4902
           1       0.89      0.94      0.92     19111

   micro avg       0.86      0.86      0.86     24013
   macro avg       0.80      0.75      0.77     24013
weighted avg       0.85      0.86      0.86     24013


Confusion matrix:
[[ 2684  2218]
 [ 1081 18030]]
```

Above is the report for “entropy”.

```
For test dataset
training dataset accuracy: 0.8628243035022696


Classification report for classifier RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=63, max_features=None,
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=2,
            min_samples_split=5, min_weight_fraction_leaf=0.0,
            n_estimators=80, n_jobs=None, oob_score=False, random_state=0,
            verbose=0, warm_start=False):
              precision    recall  f1-score   support

           0       0.72      0.53      0.61      4902
           1       0.89      0.95      0.92     19111

   micro avg       0.86      0.86      0.86     24013
   macro avg       0.81      0.74      0.76     24013
weighted avg       0.85      0.86      0.85     24013


Confusion matrix:
[[ 2609  2293]
 [ 1001 18110]]
```

Above is the report for “gini”.
We can see that although “entropy” has a lower accuracy, however, the balance in precision and recall is better than that in using “gini”. From our point of view in statistic and analysis, we believe that a balancing result in precision and recall is much more important than accuracy, because if the result has a non-balanced precision and recall, then the accuracy is meaningless and non-useful. Hence, we choose to use “entropy” as “criterion” is the model.

Finally, after the parameter tuning, the final parameters for the model is [random_state=0, max_features=None, class_weight="balanced", n_estimators=80, max_depth=63, min_samples_leaf=2, min_samples_split=5, bootstrap=True, criterion="entropy"]

b. Classification Result

The result for training dataset is meaningless because in random forest, it would be overfitting and the accuracy is very high or even close to 1. Thus, the result for training dataset would not shown here.

Classification result for testing dataset:

Confusion matrix:

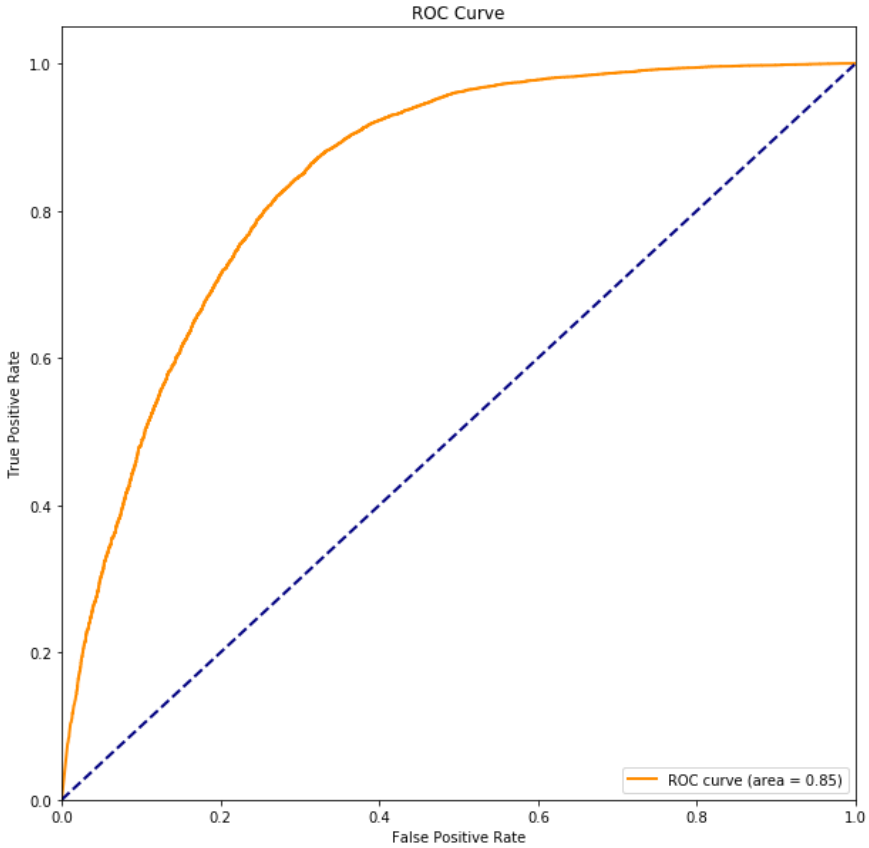
```
[[ 2684  2218]
 [ 1081 18030]]
```

Classification report:

	precision	recall	f1-score	support
0	0.71	0.55	0.62	4902
1	0.89	0.94	0.92	19111
micro avg	0.86	0.86	0.86	24013
macro avg	0.80	0.75	0.77	24013
weighted avg	0.85	0.86	0.86	24013

```
Accuracy: 0.862616082955066
```

ROC curve:



From the result, we can see that the precision and recall are not still balanced even we used the function of balancing the datasets. It is concluded that it is the problem of random forest classification model. From the ROC Curve we plotted, training dataset has area under the ROC with 0.85, which is quite high and implies that the type I error and type II error in our model is very little. Hence, we think that the classification result is satisfied.

III. Neural Network

a. Parameters tuning

Hyperparameter tuning: hidden layer size

There are so many combination and possibilities are available for the depth and size of hidden layer size. Without a grasp on parameter tuning of neural network, we have to tuning our parameters blindly and those beyond our grasps to tune are leave behind. When one day we are experience enough we can possibly make a better result.

We tried up to at most 3 layers and size with some multiples of 5 units.

For 1 layer:

```
training dataset accuracy for hidden layer sizes 5 : 0.8142279398947112
testing dataset accuracy for hidden layer sizes 5 : 0.8068129763045018
training dataset accuracy for hidden layer sizes 10 : 0.817592302060454
testing dataset accuracy for hidden layer sizes 10 : 0.8143089160038313
training dataset accuracy for hidden layer sizes 25 : 0.8207823449429976
testing dataset accuracy for hidden layer sizes 25 : 0.8213884146087536
training dataset accuracy for hidden layer sizes 50 : 0.8219677160687515
testing dataset accuracy for hidden layer sizes 50 : 0.8104360138258443
training dataset accuracy for hidden layer sizes 75 : 0.8245476414600983
testing dataset accuracy for hidden layer sizes 75 : 0.8084787406821305
training dataset accuracy for hidden layer sizes 100 : 0.8240944113237806
testing dataset accuracy for hidden layer sizes 100 : 0.8110190313580145
training dataset accuracy for hidden layer sizes 150 : 0.8222291949935502
testing dataset accuracy for hidden layer sizes 150 : 0.8116436929996252
training dataset accuracy for hidden layer sizes 200 : 0.826770212320887
testing dataset accuracy for hidden layer sizes 200 : 0.8060633823345688
training dataset accuracy for hidden layer sizes 250 : 0.8267004846076073
testing dataset accuracy for hidden layer sizes 250 : 0.8259692666472328
```

For 2 layers:

```
training dataset accuracy for hidden layer sizes 50 , 10 : 0.8259509116898511
testing dataset accuracy for hidden layer sizes 50 , 10 : 0.8150585099737642
training dataset accuracy for hidden layer sizes 75 , 10 : 0.8252884984136946
testing dataset accuracy for hidden layer sizes 75 , 10 : 0.8263440636321993
training dataset accuracy for hidden layer sizes 100 , 10 : 0.8267789282850468
testing dataset accuracy for hidden layer sizes 100 , 10 : 0.8106858784824886
training dataset accuracy for hidden layer sizes 150 , 10 : 0.8305093609455078
testing dataset accuracy for hidden layer sizes 150 , 10 : 0.8373381085245492
training dataset accuracy for hidden layer sizes 200 , 10 : 0.8328713872328557
testing dataset accuracy for hidden layer sizes 200 , 10 : 0.8234706200807895
training dataset accuracy for hidden layer sizes 250 , 10 : 0.8318516194261409
testing dataset accuracy for hidden layer sizes 250 , 10 : 0.8305501186857119
training dataset accuracy for hidden layer sizes 5 , 25 : 0.817958372555172
testing dataset accuracy for hidden layer sizes 5 , 25 : 0.8141006954566277
training dataset accuracy for hidden layer sizes 10 , 25 : 0.8207997768713176
testing dataset accuracy for hidden layer sizes 10 , 25 : 0.8240536376129597
training dataset accuracy for hidden layer sizes 25 , 25 : 0.8243384583202594
testing dataset accuracy for hidden layer sizes 25 , 25 : 0.8201807354349727
training dataset accuracy for hidden layer sizes 50 , 25 : 0.8269183837116062
```

For 3 layers:

```
training dataset accuracy for hidden layer sizes 250 , 250 , 10 : 0.862096015061186
testing dataset accuracy for hidden layer sizes 250 , 250 , 10 : 0.8128513721734061
training dataset accuracy for hidden layer sizes 5 , 5 , 25 : 0.8156137781961441
testing dataset accuracy for hidden layer sizes 5 , 5 , 25 : 0.8116020488901845
training dataset accuracy for hidden layer sizes 10 , 5 , 25 : 0.821087403688596
testing dataset accuracy for hidden layer sizes 10 , 5 , 25 : 0.8191812768083955
training dataset accuracy for hidden layer sizes 25 , 5 , 25 : 0.8198846006345222
testing dataset accuracy for hidden layer sizes 25 , 5 , 25 : 0.82696872527381
training dataset accuracy for hidden layer sizes 50 , 5 , 25 : 0.8226039814524283
testing dataset accuracy for hidden layer sizes 50 , 5 , 25 : 0.8142256277849498
training dataset accuracy for hidden layer sizes 75 , 5 , 25 : 0.8309364431893456
testing dataset accuracy for hidden layer sizes 75 , 5 , 25 : 0.8251363844584183
training dataset accuracy for hidden layer sizes 100 , 5 , 25 : 0.8299689711675906
testing dataset accuracy for hidden layer sizes 100 , 5 , 25 : 0.8267605047266064
training dataset accuracy for hidden layer sizes 150 , 5 , 25 : 0.8333071854408535
```

We believe if we keep on trying more layers and more different sizes, we can probably find a much better solution for this. However, as the limited time we have and for the sake of our sanity, we decided to pick 200,200,25 as our hidden layer sizes. The reason why we print our accuracy of testing dataset as well is for tracking over-fitting effect of neural network.

Hyperparameter tuning: alpha

Alpha is similar the C in logistic regression which is a penalizing factor for overfitting. It penalizes weights with large magnitudes.

```
training dataset accuracy for aplha 1e-06: 0.8609019279712722

C:\Users\Lcyy\Anaconda3\lib\site-packages\sklearn\neural_networ
Optimizer: Maximum iterations (200) reached and the optimizatio
  % self.max_iter, ConvergenceWarning)

training dataset accuracy for aplha 1e-05: 0.8635341491475788
training dataset accuracy for aplha 0.0001: 0.8507478297249241
training dataset accuracy for aplha 0.001: 0.8340393264302898
training dataset accuracy for aplha 0.01: 0.8210699717602761
training dataset accuracy for aplha 0.1: 0.8164069309347
training dataset accuracy for aplha 1: 0.8072464526025869
training dataset accuracy for aplha 10: 0.5
training dataset accuracy for aplha 100: 0.5
```

When alpha = 0.0001, the accuracy will be the best.

Hyperparameter tuning: solver

The default solver is ‘adam’, it has the best performance in handling large dataset. Adam is derived from adaptive moment estimation, it combines both strength from Adaptive Gradient Algorithm and Root Mean Square Propagation. ‘lbfgs’ is the same as the one in logistic regression whereas ‘sgd’ is the gradient descent algorithm in MLPClassifier.

```
training dataset accuracy for solver sgd: 0.812101244639682
training dataset accuracy for solver lbfgs: 0.8128159537007984

C:\Users\Lcyy\Anaconda3\lib\site-packages\sklearn\neural_networ
Optimizer: Maximum iterations (200) reached and the optimizatio
  % self.max_iter, ConvergenceWarning)

training dataset accuracy for solver adam: 0.8635341491475788
```

‘adam’ in fact gives the best result.

Hyperparameter tuning: activation

It is the crucial part for our neural network model as activation function is the mathematical expression, we used for calculating the weighted sum of inputs adding to a bias. Then, the classifier will decide whether or not to remove the neuron created. There are multiple activation functions in sklearn. We are not going to introduce all these one by one but we want to highlight ‘logistic’ and ‘relu’ function we have covered in class. ‘logistic’ is actually the sigmoid function and ‘relu’ is the rectified linear unit function.

```
training dataset accuracy for activation identity: 0.804099989540843
training dataset accuracy for activation logistic: 0.8121535404246418
training dataset accuracy for activation tanh: 0.8404804239444967

C:\Users\Lcyy\Anaconda3\lib\site-packages\sklearn\neural_network\multila
Optimizer: Maximum iterations (200) reached and the optimization hasn't
  % self.max_iter, ConvergenceWarning)

training dataset accuracy for activation relu: 0.8635341491475788
```

‘relu’ seems to be a more suitable parameter in this case.

Hyperparameter tuning: initial learning rate
It is the learning rate for tuning the weighting for each neuron if a misclassified is occurred. We use default setting, constant, in parameter learning rate. Initial learning rate will not change in fitting.

```
training dataset accuracy for learning rate 0.001: 0.8635341491475788
training dataset accuracy for learning rate 0.01: 0.8290973747515951
training dataset accuracy for learning rate 0.1: 0.8007792071959
training dataset accuracy for learning rate 0.2: 0.7942160861834536
training dataset accuracy for learning rate 0.3: 0.5
training dataset accuracy for learning rate 0.4: 0.5
training dataset accuracy for learning rate 0.5: 0.5
training dataset accuracy for learning rate 0.6: 0.5
training dataset accuracy for learning rate 0.7: 0.5
```

Initial learning rate at 0.001 which is the default setting gives the accuracy.

b. Classification Result

Under the condition,

```
MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(200, 200, 25), learning_rate='constant',
       learning_rate_init=0.001, max_iter=400, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=1, shuffle=True,
       solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)
```

For training dataset:
Confusion matrix:

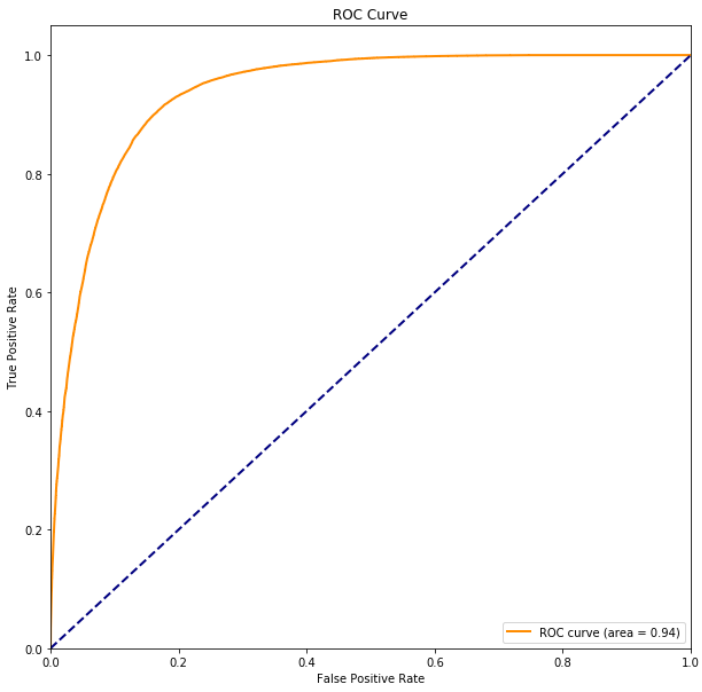
```
[[48203  9163]
 [ 5849 51517]]
```

Classification report:

	precision	recall	f1-score	support
0	0.89	0.84	0.87	57366
1	0.85	0.90	0.87	57366
avg / total	0.87	0.87	0.87	114732

```
Accuracy: 0.86915594603075
```

ROC curve:



For training dataset:
Confusion matrix:

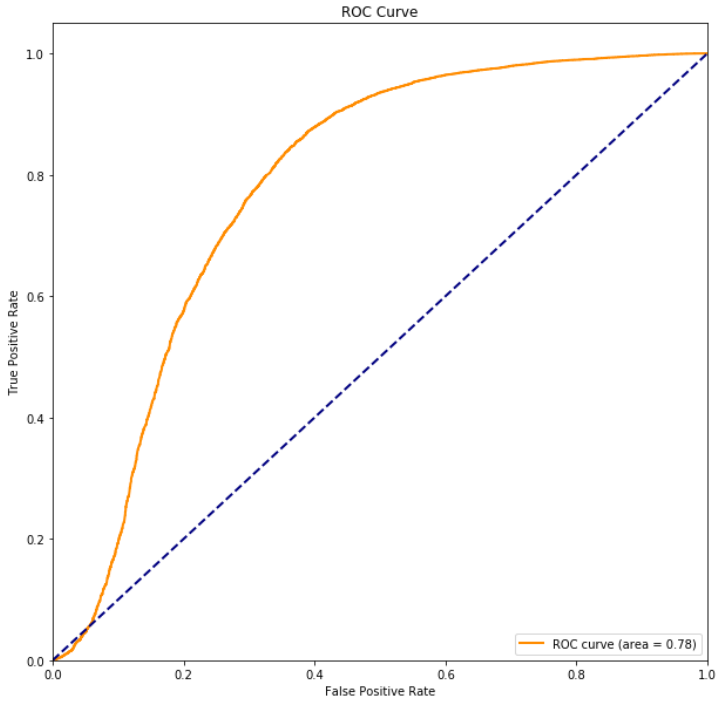
```
[[ 3005  1897]
 [ 2551 16560]]
```

Classification report:

	precision	recall	f1-score	support
0	0.54	0.61	0.57	4902
1	0.90	0.87	0.88	19111
avg / total	0.82	0.81	0.82	24013

```
Accuracy: 0.8147670012076792
```

ROC curve:



As we can see from the classification summary of the training dataset itself have very good performance, from precision, recall to accuracy all area is classified with over 0.85 percent. The ROC curve has 0.94-unit square area under it. Implies that Type I and type II error seldom occur in our model. However, if we fit the testing dataset in the trained model, the indicators from our classification shows that is still decent, over 0.8 in average. However, the ROC shows the sensitivity of our model is not as good as the training dataset. If we take a deeper look in our classification report, the indictors for class ‘0’ is not that good. The statistics shows that records belong to class ‘0’ have a considerable probability of being misclassified as class ‘1’ but the high-quality performance in class ‘1’ compensate for that and leverage out the poor performance in class ‘0’. Resulted in a decent performance in average score.

Reference

1. Avinash Sharma V, “Understanding Activation Functions in Neural Networks” https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

2. Jason Brownlee, “Gentle Introduction to the Adam Optimization Algorithm for Deep Learning”, https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

3. Rafael Alencar, “Resampling strategies for imbalanced datasets”, https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets

4. Michael Galarnyk, “Logistic Regression using Python (scikit-learn)”, https://towardsdatascience.com/logistic-regression-using-python-sklearn-numpy-mnist-handwriting-recognition-matplotlib-a6b31e2b166a