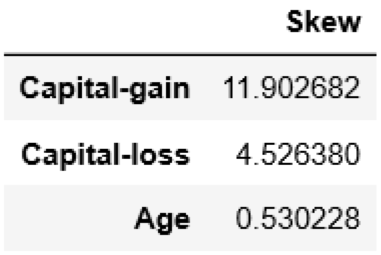
Report of the project: Income prediction

This report is a resume of all the activities that we do. To see the detail of each activity, please open the jupyter notebook which is correspond.

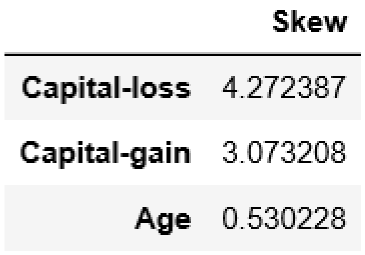
I – Data preprocessing (Datapreprocessing.ipynb):

In this session, we do the following task:

- We replace the missing value “?” by np.NaN and remove them
- Following of the boxplot of the column Hours-per-week, we group the working hours in 5 categories which we consider relevant:
    - less than 40 hours per week
    - between 40 and 45 hours per week
    - between 45 and 60 hours per week
    - between 60 and 80 hours per week, and
    - more than 80 hours per week
- We use logarithmic transformation to decrease the skewness of “Capital-gain” and “Capital-loss”



	Skew
Capital-gain	11.902682
Capital-loss	4.526380
Age	0.530228



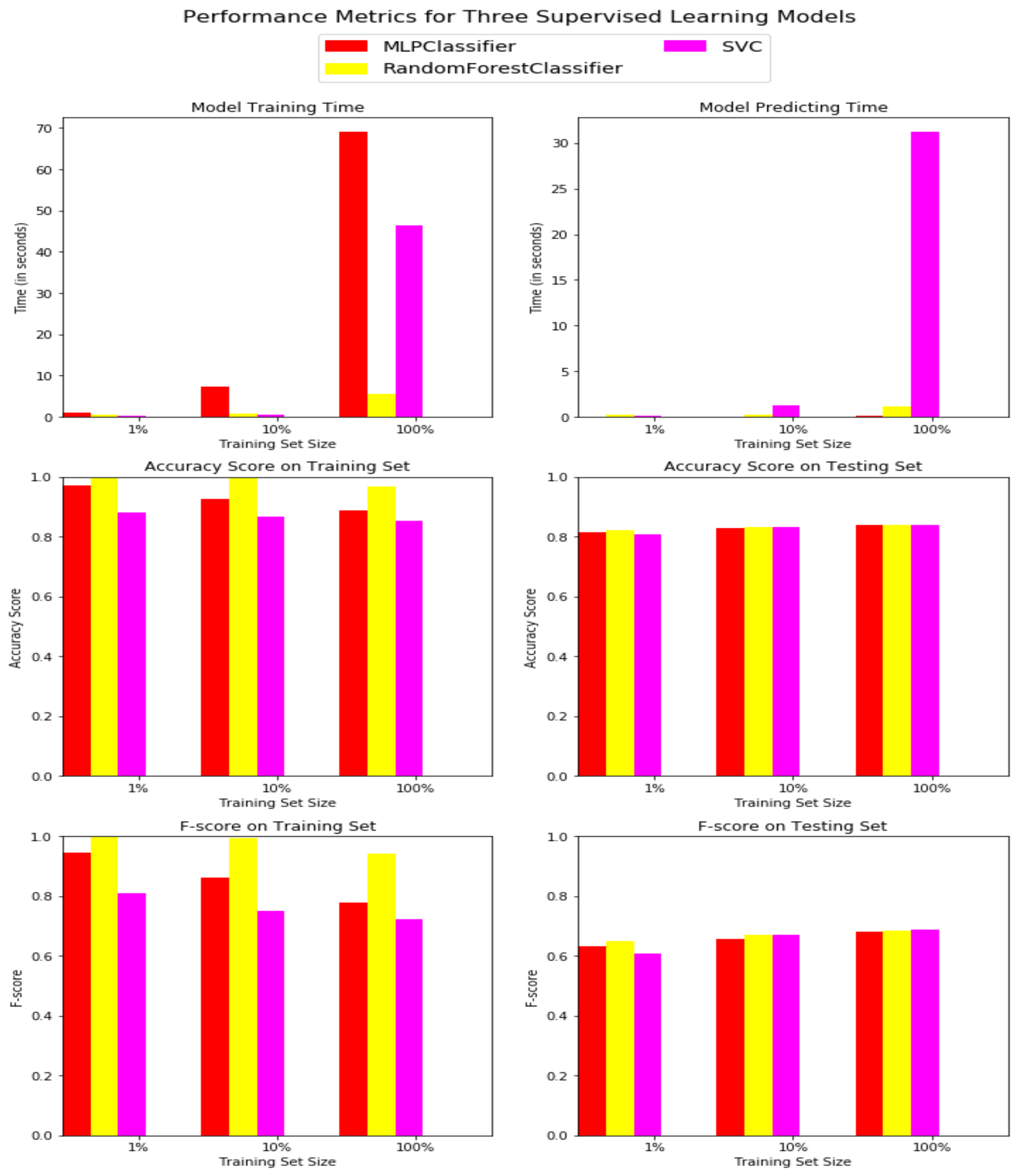
	Skew
Capital-loss	4.272387
Capital-gain	3.073208
Age	0.530228

- We use min max scaler to scale the attribute numeric.
- With the column Native country, we group each country in 5 categories which we consider relevant to decrease the complexity of dataset:
    - Asia
    - Europe
    - America_without_USA
    - United-States
- For categorical data, we use One Hot Encoding for converting to numeric data.

II – Evaluation model (Evaluation_model_data_without_oversampling.ipynb):

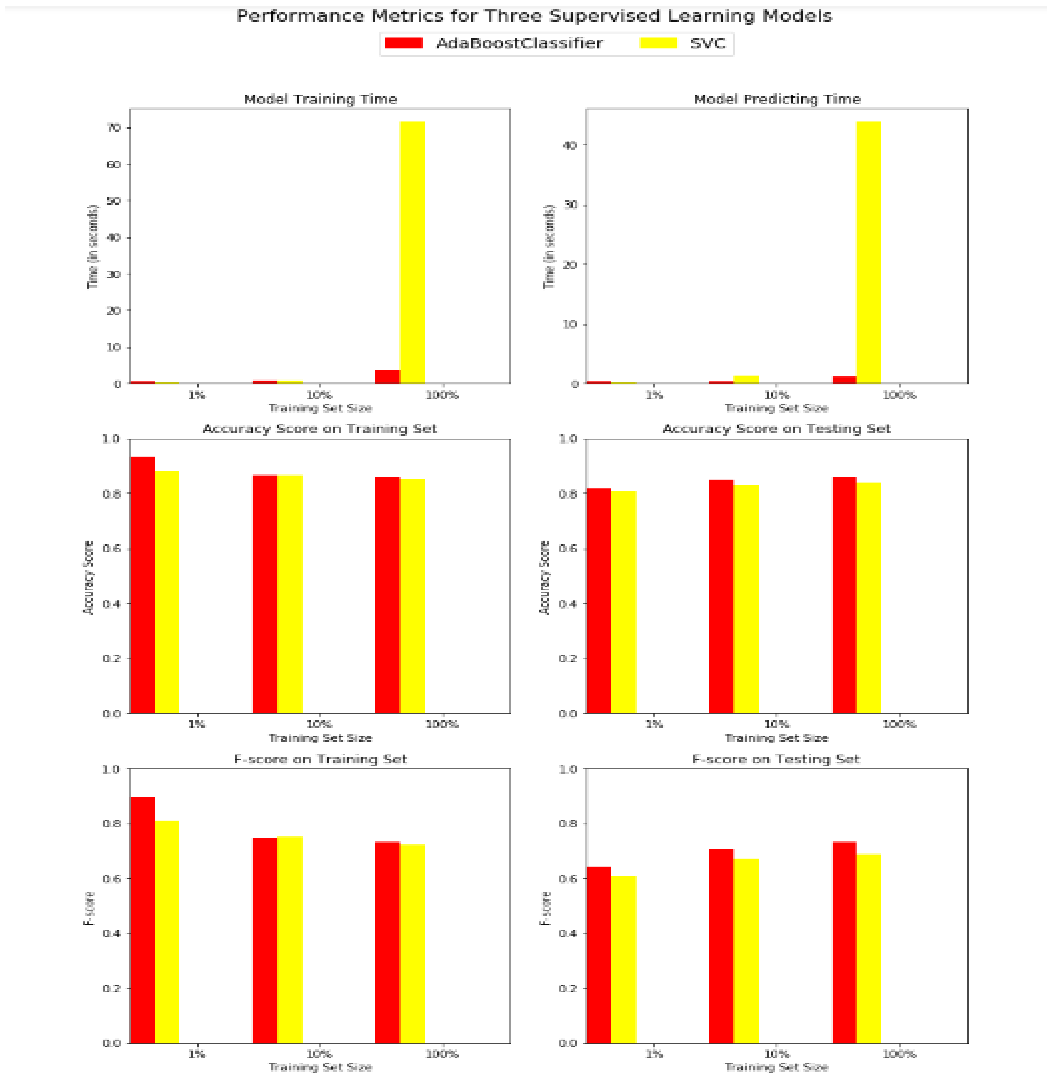
In this section, we will investigate three different algorithms, and determine which is best at modeling the data. 3 of these algorithms that we work on are: random forest, support vector machine, neural network.

To properly evaluate the performance of each model chosen above more efficiently, it's helpful to create a training and predicting pipeline that can quickly and effectively train models using various sizes of training data (we call validation data) and perform predictions on the testing data. So, we split the data set with 20% testing data, and 80% training data. We use stratified splitting to keep the distribution of data in each data.



According to the model performance graph above, we see that random forest and neural network seems to be overfitting in the training set with the accuracy and the f-score higher than in the testing set. We can see the big different between the result in the training set and test set. But with the support vector machine, the different is not too much and its result is the best in accuracy and f-score. We can say that the support vector machine does not overfit too much on the training set.

But according to the model performance graph above, Support Vector Classfier which takes dramatically more time to train and predict, but its results is reasonable by comparing the result between training set and test set (not having big difference). So, we try another algorithm to discover like Adaboost.



```
AdaBoostClassifier
```

	1%	10%	100%
train_time	0.547043	0.804059	3.555266
pred_time	0.365026	0.360027	1.123107
acc_train	0.933610	0.868159	0.859416
acc_test	0.819659	0.849826	0.858279
f_train	0.895522	0.744913	0.733685
f_test	0.640962	0.707945	0.731287

```
MLPClassifier
```

	1%	10%	100%
train_time	1.280295	9.430709	95.886196
pred_time	0.062001	0.064003	0.234016
acc_train	0.970954	0.927446	0.888221
acc_test	0.815515	0.828952	0.839549
f_train	0.945946	0.862725	0.779108
f_test	0.630366	0.654848	0.680358

```
RandomForestClassifier
```

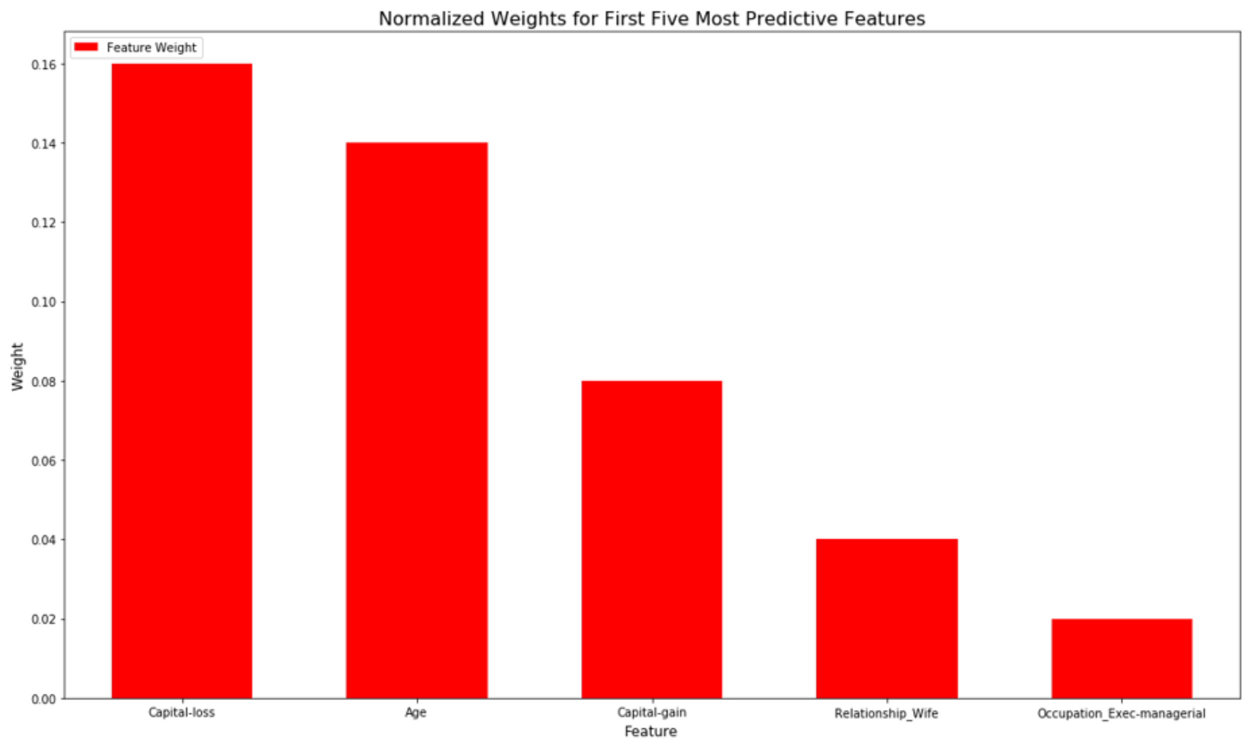
	1%	10%	100%
train_time	0.722090	1.154083	7.586568
pred_time	0.212983	0.370051	1.528115
acc_train	0.995851	0.994610	0.967134
acc_test	0.821979	0.833748	0.840046
f_train	0.996622	0.991611	0.941357
f_test	0.650057	0.669604	0.684218

```
SVC
```

	1%	10%	100%
train_time	0.324018	0.802056	70.614282
pred_time	0.128009	1.766119	46.354580
acc_train	0.879668	0.865672	0.853856
acc_test	0.807890	0.832090	0.839218
f_train	0.810185	0.749604	0.720449
f_test	0.606373	0.671654	0.686614

Compare with other classifiers, Adaboost classifier have the best result by its accuracy and f-score.

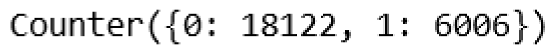
We would like to see which features provide the most predictive power when performing supervised learning on a dataset like data here. In this case, it means we wish to identify a small number of features that most strongly predict whether an individual makes at most or more than \$50,000.



We can see ‘capital-loss' is the most important feature. We think because the bigger capital loss means that the person has to have that volume of money to invest. The 'age' is the second one which may because the elder the people the more salary they will have to donor. The position of executive manager is the high position to have good salary too.

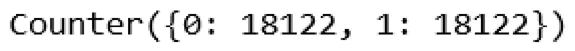
III – Extra (Evaluation_model_data_without_oversampling.ipynb):

We find that our dataset is imbalanced dataset. We see that We have an imbalanced dataset which have 18122 label "=<50k" and 6066 label ">50k" (0 and 1 after one hot encoding).



```
Counter({0: 18122, 1: 6006})
```

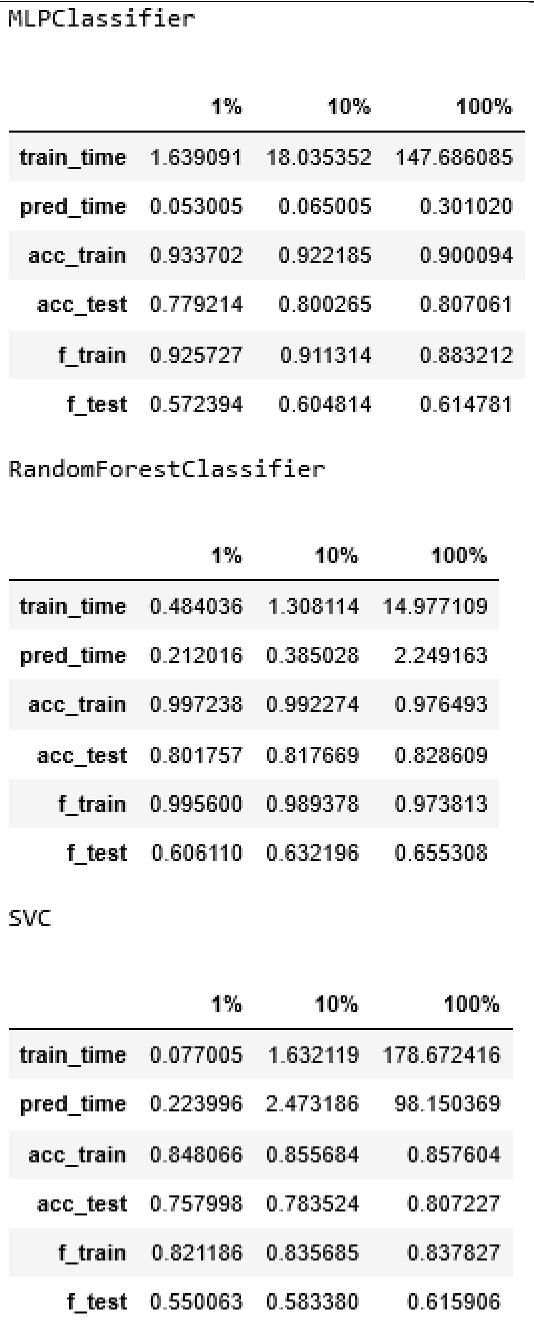
So we use algorithm SMOTE to make our data set balanced. In short, the idea of the method is to combine under-sampling of the majority class with a special form of over-sampling of the minority class. In the SMOTE algorithm the over-sampling of the minority class is carried out by creating new “synthetic” examples, rather than by over-sampling with replacement from the existing examples. The synthetic observations are generated with the help of the *k nearest neighbors* (KNN) algorithm. The majority class is under-sampled by randomly removing observations until the minority class becomes some pre-specified percentage of the majority class. And now we have a balanced dataset.



```
Counter({0: 18122, 1: 18122})
```

We do the same things with imbalanced dataset and we compare the result.

Balanced dataset



MLPClassifier

	1%	10%	100%
train_time	1.639091	18.035352	147.686085
pred_time	0.053005	0.065005	0.301020
acc_train	0.933702	0.922185	0.900094
acc_test	0.779214	0.800265	0.807061
f_train	0.925727	0.911314	0.883212
f_test	0.572394	0.604814	0.614781

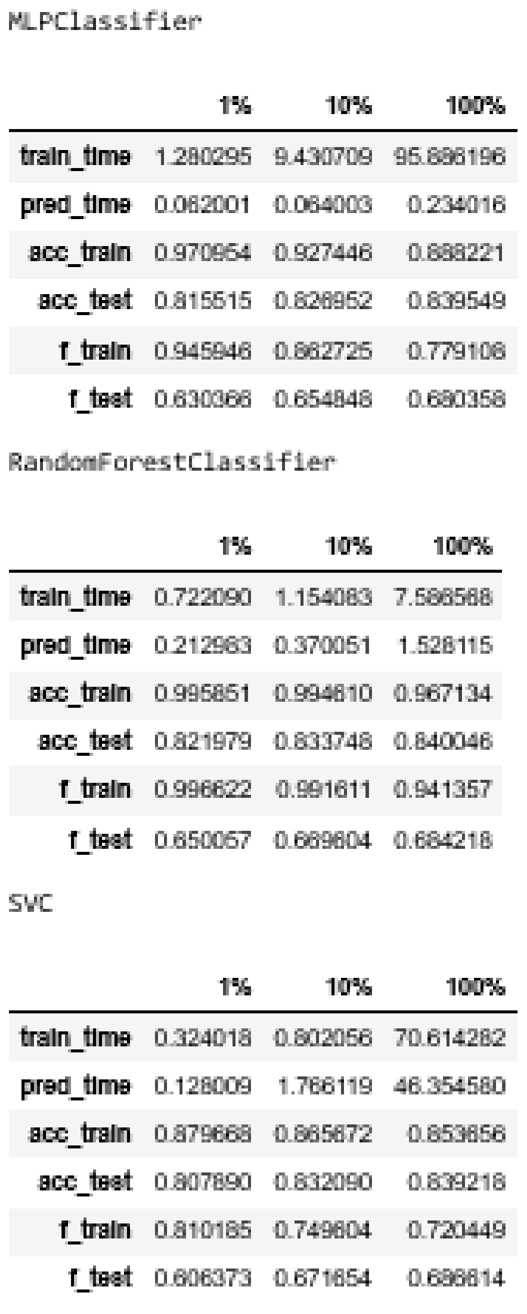
RandomForestClassifier

	1%	10%	100%
train_time	0.484036	1.308114	14.977109
pred_time	0.212016	0.385028	2.249163
acc_train	0.997238	0.992274	0.976493
acc_test	0.801757	0.817669	0.828609
f_train	0.995600	0.989378	0.973813
f_test	0.606110	0.632196	0.655308

SVC

	1%	10%	100%
train_time	0.077005	1.632119	178.672416
pred_time	0.223996	2.473186	98.150369
acc_train	0.848066	0.855684	0.857604
acc_test	0.757998	0.783524	0.807227
f_train	0.821186	0.835685	0.837827
f_test	0.550063	0.583380	0.615906

Imbalanced dataset



MLPClassifier

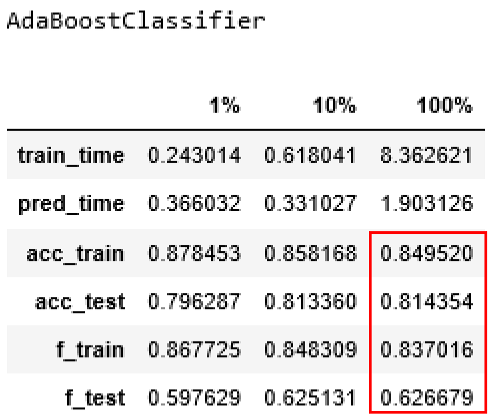
	1%	10%	100%
train_time	1.280295	9.430709	95.886196
pred_time	0.062001	0.064003	0.234016
acc_train	0.970954	0.927446	0.888221
acc_test	0.815515	0.828952	0.839549
f_train	0.945946	0.862725	0.779108
f_test	0.630366	0.654848	0.680358

RandomForestClassifier

	1%	10%	100%
train_time	0.722090	1.154083	7.586568
pred_time	0.212983	0.370051	1.528115
acc_train	0.995851	0.994810	0.967134
acc_test	0.821979	0.833748	0.840046
f_train	0.996622	0.991611	0.941357
f_test	0.650057	0.669804	0.684218

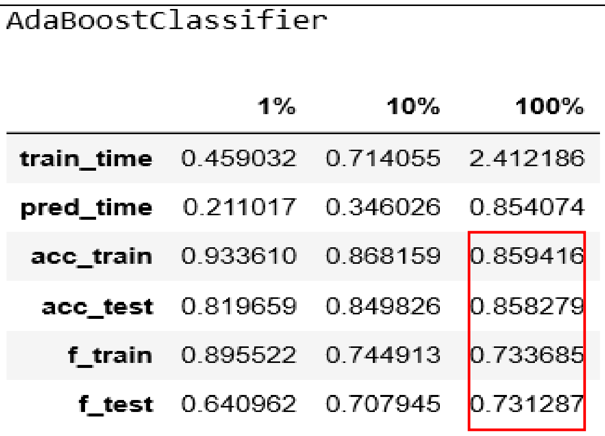
SVC

	1%	10%	100%
train_time	0.324018	0.802056	70.614282
pred_time	0.128009	1.766119	46.354580
acc_train	0.879668	0.865872	0.853856
acc_test	0.807890	0.832090	0.839218
f_train	0.810185	0.749804	0.720449
f_test	0.606373	0.671654	0.686814



AdaBoostClassifier

	1%	10%	100%
train_time	0.243014	0.618041	8.362621
pred_time	0.366032	0.331027	1.903126
acc_train	0.878453	0.858168	0.849520
acc_test	0.796287	0.813360	0.814354
f_train	0.867725	0.848309	0.837016
f_test	0.597629	0.625131	0.626679



AdaBoostClassifier

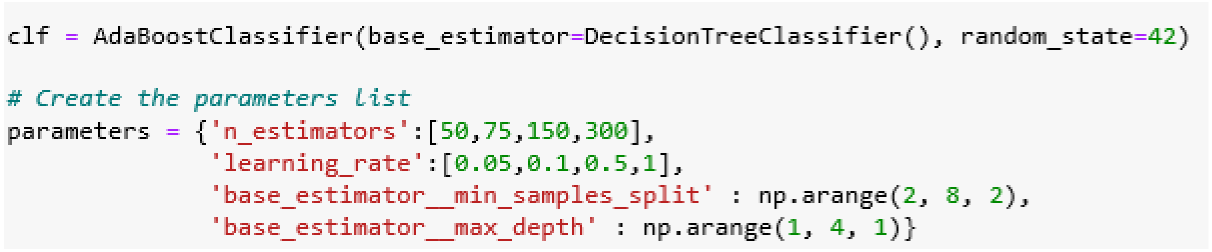
	1%	10%	100%
train_time	0.459032	0.714055	2.412186
pred_time	0.211017	0.346026	0.854074
acc_train	0.933610	0.868159	0.859416
acc_test	0.819659	0.849826	0.858279
f_train	0.895522	0.744913	0.733685
f_test	0.640962	0.707945	0.731287

But with the balanced dataset the result is similar and not better than old- imbalanced dataset. We can see that the best classifier is still AdaBoost. But the result f-score and accuracy is not better even worse (by looking the result in the test set). And more data which make the training time and prediction time of the classifiers becomes longer. So the way we apply SMOTE is not the efficient way.

IV – Perspective:

During the time of the project, we have some perspectives that can not be done by the lack of resource computer and time:

- We would like to tune the hyperparameters of AdaBoost classifier to have better result. This is some parameters that we would like to try with our Adaboost classifier with GridsearchCV.



```
clf = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(), random_state=42)

# Create the parameters list
parameters = {'n_estimators':[50,75,150,300],
              'learning_rate':[0.05,0.1,0.5,1],
              'base_estimator__min_samples_split' : np.arange(2, 8, 2),
              'base_estimator__max_depth' : np.arange(1, 4, 1)}
```

- We would like to use algorithm clustering to see how we can cluster the population into groups. And we can get insight from each cluster.

V – Library:

Sklearn

Numpy

Pandas

imbalanced-learn