**MOOC Dropout**

**Prediction Report**

**Group Members:** Woochan Lee, Ishan Jain, Rhea Chugh

**Table of Contents**

1. [**Introduction**](https://docs.google.com/document/d/1v_Xi-nQHZRA9-j59FnRYXi-saFF51dcjh2mPYBaE3sU/edit#heading=h.30j0zll) **3**
2. **Division of Labor 3**
3. **Understanding the Data 3**
4. **Data-Based Hypotheses 4**
   1. Hypothesis on Event Counts 4
   2. Hypothesis on Day Counts 4
5. **Pre-Processing 5**
   1. Building Feature Vector 1: Event-Counts 5
   2. Building Feature Vector 2: Day-Counts 5
      1. *Handling the Corrupt Data 5*
      2. *Feature Vector 2 6*
   3. Testing the Two Feature Vectors 7
   4. Building the Final Feature Vector 8
6. **Classifiers 8**
   1. Handling Imbalanced Data 8
   2. Choice of Classifiers 10
   3. Optimal Hyper Parameters 11
7. **Classification Results 12**
8. **Ensemble-Based Method: Hard Voting 13**
9. **Conclusion 14**
10. **Introduction**

Students’ high dropout rate on MOOC platforms has been heavily criticized, and predicting their likelihood of dropout would be useful for maintaining and encouraging students’ learning activities. Therefore, in this project, we will predict dropout on XuetangX, one of the largest MOOC platforms in China.

The competition participants need to predict whether a user will drop a course within the next 10 days based on his or her prior activities. If a user C leaves no records for course C in the log during the next 10 days, we define it as a dropout from course C.

Deep analysis of the provided dataset and understanding the data, we were able to draw several useful features for this project. In order to do that, we brainstormed and came up with some assumptions which would be the foundation of our project and that would help us shape the Feature Vector Construction.

1. **Division of Labor**

|  |  |  |
| --- | --- | --- |
| **Woochan Lee [20142294]** | **Ishan Jain [20311273]** | **Rhea Chugh [20316053]** |
| * Generate Feature 1 * Test Undersampling * Test Logistic Regression * Hard Voting to Ensemble Results | * Generate Feature 2 * Test Oversampling * Integrate Results for Presentation and Report | * Pre-Process Log Data * Balanced Resampling   - Test Adaboost   * RF, XGBoost and MLP Classification |

1. **Understanding the Data**

We were given a number of files to kickstart our analysis. The structure is as follows :

* Date.csv: Gives us more information about the timespan of each course
* Object.csv: Gives us more information about each module in a course
* Enrollment\_(train/test).csv : User enrollment records
* Log\_(train/test).csv : Behavior Records
* True\_train.csv: Ground Truth about a dropout

Looking at these datasets, we understood that an enrollment means the enrollment of a student for a particular class and that each enrollment may contain multiple events and actions that tell us what the student did at a point in time. For the sake of this project, after exploring the contents of the data, we realised the object.csv file was not providing us with any valuable insight so we decided to drop it.

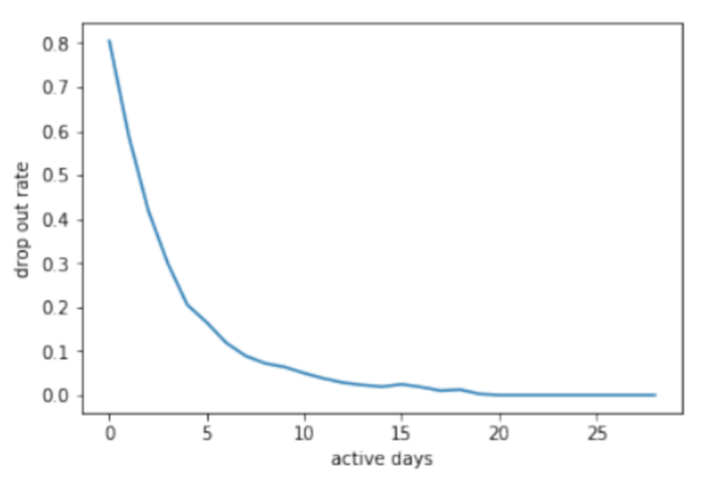
1. **Data-Based Hypotheses** 
   1. **Hypothesis on Event Counts**

Our first assumption was that any student who completes a course would have more counts for each event i.e. watching more course videos, solving problems, participating in discussions etc.

Besides events, there is also the information of whether the student accessed the course in a server or a browser. We also believed that this information was useful in terms of how a browser would be mean more videos and problem events as compared to a server. Thus, we hypothesized that a student who spends more time in a browser would have a lower chance of dropping out of a course as well.

* 1. **Hypothesis on Day Counts**

Our second assumption was a time based one. We hypothesized that the greater the number of active days (days where the student had events on a particular enrollment), the smaller the chance of a dropout, and vice versa. This is illustrated in the simple chart below.



1. **Pre-Processing** 
   1. **Building Feature Vector 1: Event-Counts**

Based on Hypothesis A, we were able to build our first Feature Vector. In order to build it, we had to cluster all the log information by enrollment\_id, and then simply for each enrollment\_id, count the number of events in each category.

|  |
| --- |
| # enrollement\_id |

|  |  |
| --- | --- |
| **Index** | **Features** |
| 1 | # problem |
| 2 | # video |
| 3 | # access |
| 4 | # wiki |
| 5 | # discussion |
| 6 | # navigate |
| 7 | # page close |
| 8 | # server |
| 9 | # browser |

*Table 1. Event Features*

* 1. **Building Feature Vector 2: Day-Counts**
     1. **Handling Corrupt Data**

Based on Hypothesis B, we reviewed the log records and timestamps to better understand the vectorization of active days. During this pre-processing session, we noticed that the Log shows access to some modules before they were even posted i.e. log time stamp was before the *start-date* of a module in object.csv.

In order to solve this, we deleted those log instances so as to ensure that it did not affect our calculation of active days. We chose to handle this corrupt data only for Feature Vector 2 since Feature Vector 1 is not particularly dependent on dates and time. Regardless of inconsistent timestamps, Feature 1 revolves solely around the occurrence of the event itself.

Some modules did not have a start date in object.csv so we decided not to delete log instances with those modules. After the handling of this corrupt data, we found that ***2% of the Enrollment ID’s (1570 out of 72395) had such corrupt log instances.*** These findings further proved that our feature extraction on each enrollment ID would benefit from this data preprocessing in terms of efficiency and more correctness.

* + 1. **Feature Vector 2**

To extract this feature vector, we clustered the dataset by enrollment\_id, then converted the time element into Dates after which it was easy to find the first and last date of the event for a specific user. Then we would be able to find *last log time - first log time*, count the effective study days, by normalizing all events to midnight and finding the unique days when people were active.

Then we divided the course duration by 3, beginning, middle, end, and based on that allocate to which time period of the course, the event was in, and then we can keep incrementing those indexes based on the number of events. Using this information of beginning, middle and end, we were able to find out that users were more likely to have event logs in the beginning and middle periods of a course compared to the end of a course. Also, we figured that users that had some log record during the end period of a course were more likely to complete the course without a dropout.

|  |
| --- |
| # enrollement\_id |

|  |  |
| --- | --- |
| **Index** | **Features** |
| 1 | # last log time - first log time |
| 2 | # effective study days |
| 3 | # events in course start |
| 4 | # events in course mid |
| 5 | # events in course end |

*Table 2. Day Count Features*

* 1. **Testing the Two Feature Vectors**

In order to see decide which of these two feature vectors would be our final

vector of choice, we decided to run one of our classifiers on each. Upon running Random Forest with cross-validation between n\_estimators = [50, 80, 100, 120, 150], we got the following results.

====== RF Classification Report for Testing Events Feature Vector: ======

precision recall f1-score support

0 0.63 0.58 0.61 4902  
 1 0.90 0.91 0.90 19111  
  
 micro avg 0.85 0.85 0.85 24013  
 macro avg 0.76 0.75 0.76 24013

weighted avg 0.84 0.85 0.84 24013  
  
Prediction ROC/AUC Score: 0.8067593035520692  
  
Confusion Test Matrix:   
[[ 2854 2048]  
 [ 1643 17468]]

====== RF Classification Report for Testing Day Count Feature Vector: ======

precision recall f1-score support

0 0.55 0.68 0.61 4902  
 1 0.91 0.86 0.89 19111  
  
 micro avg 0.82 0.82 0.82 24013  
 macro avg 0.73 0.77 0.75 24013  
weighted avg 0.84 0.82 0.83 24013  
  
Prediction ROC/AUC Score: 0.821301000205781  
  
Confusion Test Matrix:   
[[ 3344 1558]  
 [ 2690 16421]]

* 1. **Building Final Feature Vector**

After comparing the random forest classification results for the two feature vectors, we believed that a better classification result could be obtained after merging the two feature vectors. Thus, our final feature vector contained both the event counts and the date-related features obtained from the two feature vectors.

|  |
| --- |
| # enrollement\_id |

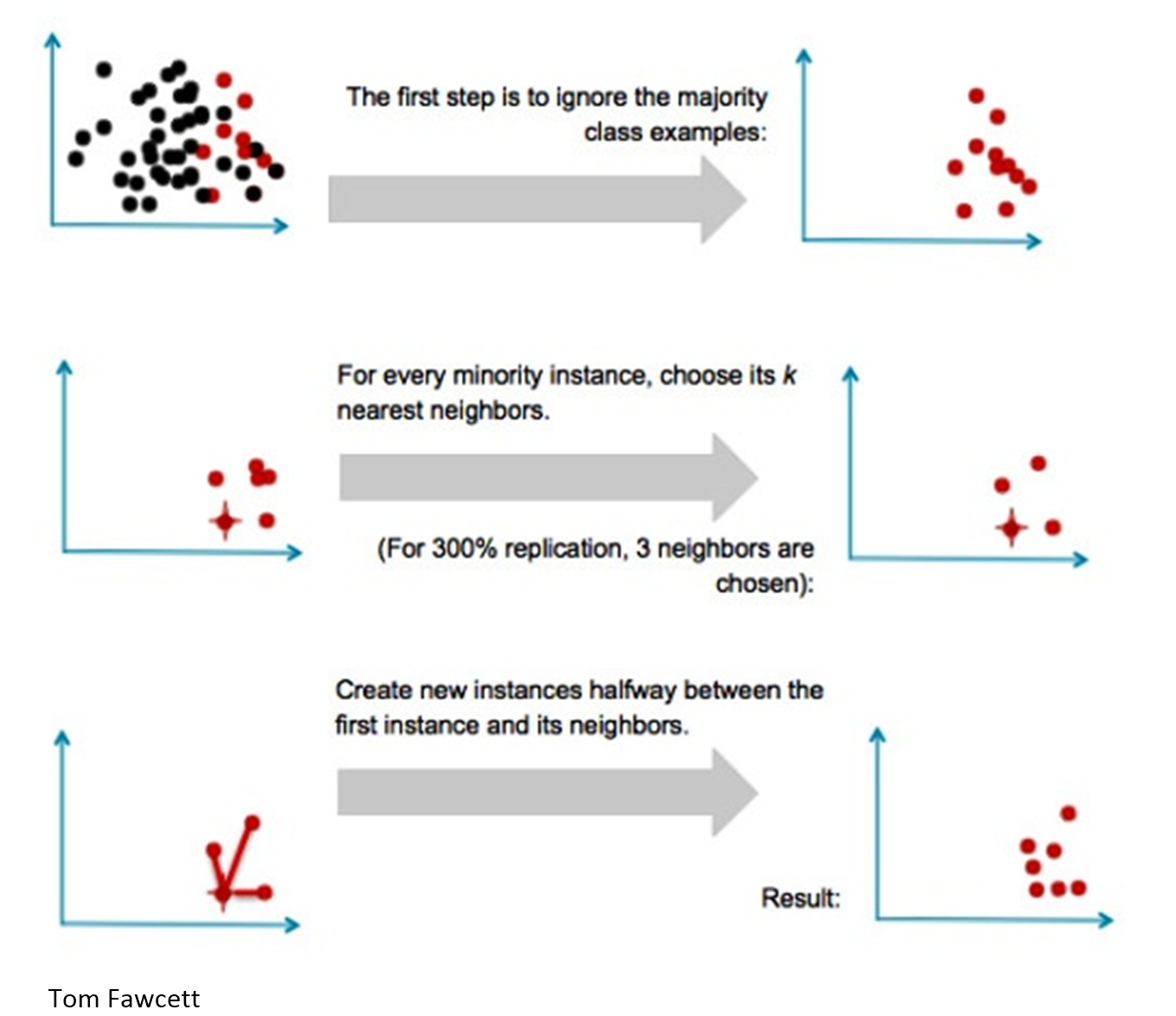
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Index** | **Features** | **Index** | **Features** | **Index** | **Features** |
| 1 | # problem | 6 | # navigate | 11 | # effective study days |
| 2 | # video | 7 | # page close | 12 | # events in course start |
| 3 | # access | 8 | # server | 13 | # events in course mid |
| 4 | # wiki | 9 | # browser | 14 | # events in course end |
| 5 | # discussion | 10 | # last log-first log |  | |

*Table 3. Final Merged Features*

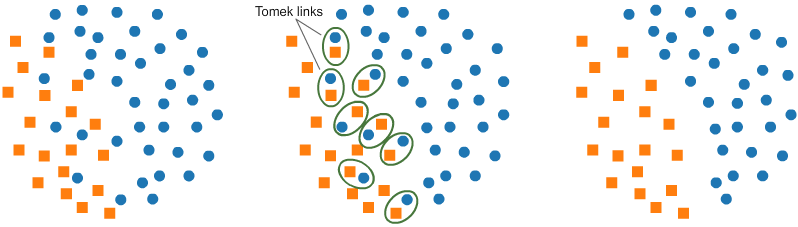
1. **Classifiers**
   1. **Handling Imbalanced Data**

Before running the classifiers on this final feature vector, we realised that we need to perform resampling. Out of the 72,395 enrollments, around 80% turned out to be dropouts, and around 20% completed the course, implying that there was an imbalance in data. There are a variety of resampling techniques for addressing this imbalance. Although there are simple random resampling methods defined in sklearn, we decided to use a combination of SMOTE(Synthetic Minority Oversampling TEchnique) and Tomek Links to tackle this issue.

SMOTE oversamples the minority class by creating new elements using K-nearest neighbours. As you can see in the diagram below, out of all the classes, we first begin by ignoring the majority class, leaving us only with the minority class. Then, for each minority class instances, we choose its k-nearest neighbours (which in our case is the default value of 5). Then, it creates new instances halfway between the first instance and its neighbours as seen in the diagram below.



Tomek link is an undersampling technique for the majority class. This works by removing instances of the majority class in each of the Tomek Links. Tomek links are defined to be pairs of very close instances, but of opposite classes. The Tomek resampler involves removing the instances of the majority class of each pair of Tomek Links as seen in the diagram below.



Although random undersampling is easy to implement, we thought that it had the possibility to remove important instances or features from the majority class. And so, we used Tomek Links to minimize these “important” losses. After some research, we also found out that Tomek helps to concentrate on removing “noisy” samples, thus minimizing our loss of important instances.

Moreover, simple random oversampling can potentially lead to overfitting. Through some research, we found out that SMOTE can alleviate some of these problems of overfitting as well as cause decision boundaries for the minority class to spread out. This led us to believe that a combination of SMOTE and Tomek would be the ideal method for solving the problem of imbalance.

* 1. **Choice of Classifiers**

We initially were inclining towards SVM but decided not to use it because it was too inefficient to train the dataset. After testing out Logistic Regression, the results were not too impressive and so we finally narrowed down our list of classifiers to Neural Network, Random Forest and XGBoost for the following reasons.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **Neural Network** | **Random Forest** | **XGBoost** |
| Extracts complex patterns from logs  Hidden layer efficiently identifies important information and shaves off | Less overfitting of decision trees  Less variance of classifier not performing well on test data due to the use of multiple trees | An adaptive boosting tree model  Relatively fast to construct the model compared to Adaboost  AdaBoost is mainly for *exponential loss functions* but XGBoost is for *generic loss functions*, thus more flexible |

* 1. **Optimal Hyper Parameters**

In order to find the optimal hyper parameters, we used the method suggested in class, 5-Fold Cross-Fold Validation. This allowed us to find the following optimal parameters:

|  |  |  |
| --- | --- | --- |
| **Neural Network** | **Random Forest** | **XGBoost** |
| parameters= [{**'hidden\_layer\_sizes'**: [50, 100, 150, 200],  **'alpha'**: [0.0001, 0.00001, 0.000001, 0.0000001]}] | parameters =  [{**'n\_estimators'**: [50, 80, 100, 120, 150]}] | parameters =  [{**'n\_estimators'**: [100, 200, 300, 400, 500],  **'learning\_rate'**: [0.0001, 0.001, 0.01, 0.1, 0.2]}] |
| Hidden Units: 200  Alpha: 0.0001 | N-estimators:150 | N-estimators: 500  Learning rate: 0.2 |

The reason we used a larger range for candidate n\_estimators for xgboost

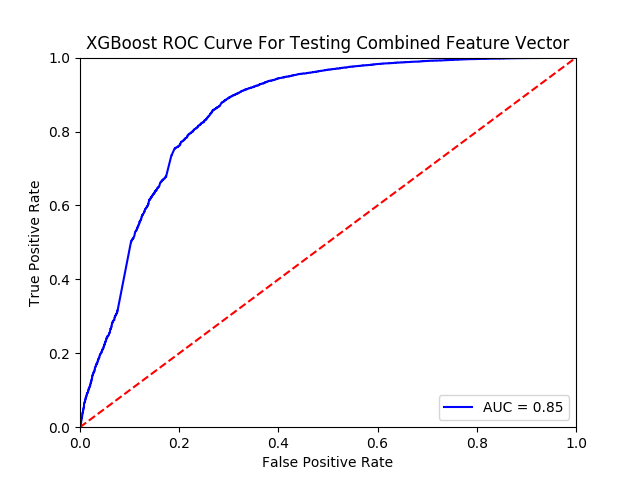
than random forest is because with [50, 80, 100, 120, 150], xgboost’s best

learning\_rate was coming up to 0.3 which seemed too big a value. But with a larger number of n\_estimators, the performance of learning\_rates 0.1 and 0.2 came up to 0.882. The program by picked 0.2 as the tie breaker. At 0.1 and 0.2, we stopped making the candidate n\_estimators bigger as we did not want the trees to overfit. Thus making xgboost’s candidate n\_estimators larger than random forest’s.

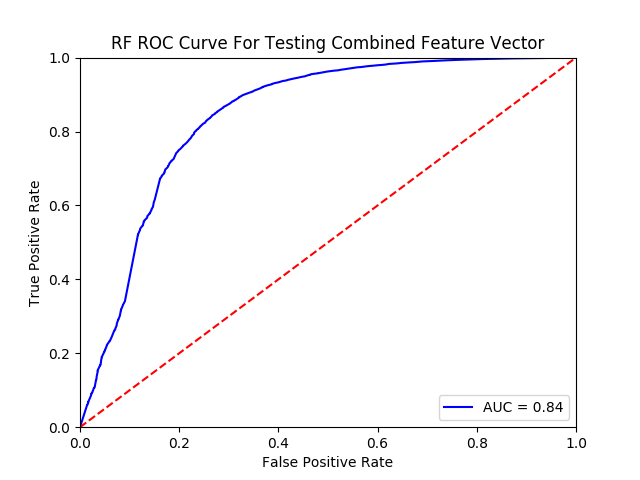
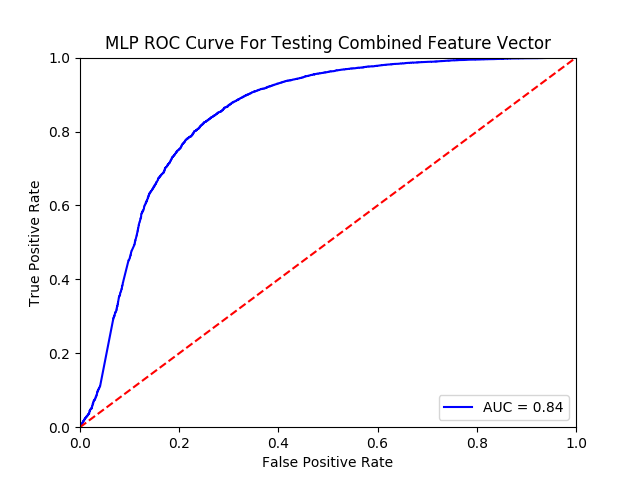
1. **Classification Results**

The following is a summary table of the classification results on the combined feature vector.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Random Forest** | | **MLP** | | **XGBoost** | |
| Precision | Recall | Precision | Recall | Precision | Recall |
| Micro Avg | 0.86 | 0.86 | 0.83 | 0.83 | 0.86 | 0.86 |
| Macro Avg | 0.79 | 0.77 | 0.74 | 0.79 | 0.79 | 0.78 |

For the purpose of our evaluation, we chose to compare the results using the micro averages of precision and recall. This is because Micro-average is preferable when there is an issue of class imbalance. In the Micro-average method, the individual TPs, FPs, and FNs of the system for different sets are summed up to get the statistics. Whereas Macro-Averaging involves taking the average precision and recall of the system on different sets. Even though we performed balanced sampling on our training data, we recognise that the test data might still have class imbalance and thus we chose to evaluate the Micro averages.

It can be seen that the overall performance of XGBoost and Random Forest is the same. Whereas MLP’s performance is a bit lower even though it performed equally well in terms of the entire operating range. Which is why the team also chose to evaluate the results using **ROC curves.** Using the ROC curves, we observed that MLP’s performance matched up to that of XGBoost and Random Forest; and that **XGBoost** performed the best with an **AUC of 0.85**.



1. **Ensemble-Based Method : Hard Voting**

We decided to use hard voting to ensemble the three classifiers we implemented. While soft voting takes into account each classifier’s uncertainty in their final decision, hard voting only takes into account each classifier’s final decision and does a simple majority vote. Since we were primarily concerned about whether an enrollment would drop out or not, it made sense to use the hard voting method. Thus with hard-voting, our **final results** are as follows:

Classification Report for Voting (Training)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 0 | 0.96 | 0.88 | 0.92 | 57026 |
| 1 | 0.89 | 0.96 | 0.92 | 57026 |
| Micro avg | 0.92 | 0.92 | 0.92 | 114052 |

Confusion Matrix for Voting (Training)

[[50181 6845]  
 [ 2132 54894]]

Classification Report for Voting (Test)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 0 | 0.71 | 0.62 | 0.66 | 4902 |
| 1 | 0.91 | 0.93 | 0.92 | 19111 |
| Micro avg | 0.87 | 0.87 | 0.87 | 24013 |

Confusion Matrix For Voting (Test)

[[ 3029 1873]  
 [ 1254 17857]]

**Train Accuracy: 0.9212902886402694  
Test Accuracy: 0.8697788697788698**

1. **Conclusion**

To sum up, we have executed several different ways of predicting the drop out rate of MOOC according to XueTang data, including but not limited to experimentation around feature selection, data cleaning, feature extraction, classifier selection, and finally optical hyperparameter tuning. We not only resampled the data to tackle class imbalance but also tried to use a powerful classifier outside of the scikit-learn library, XGBoost. We believe we have reached a good result, notably when we compared to other groups during the presentations.