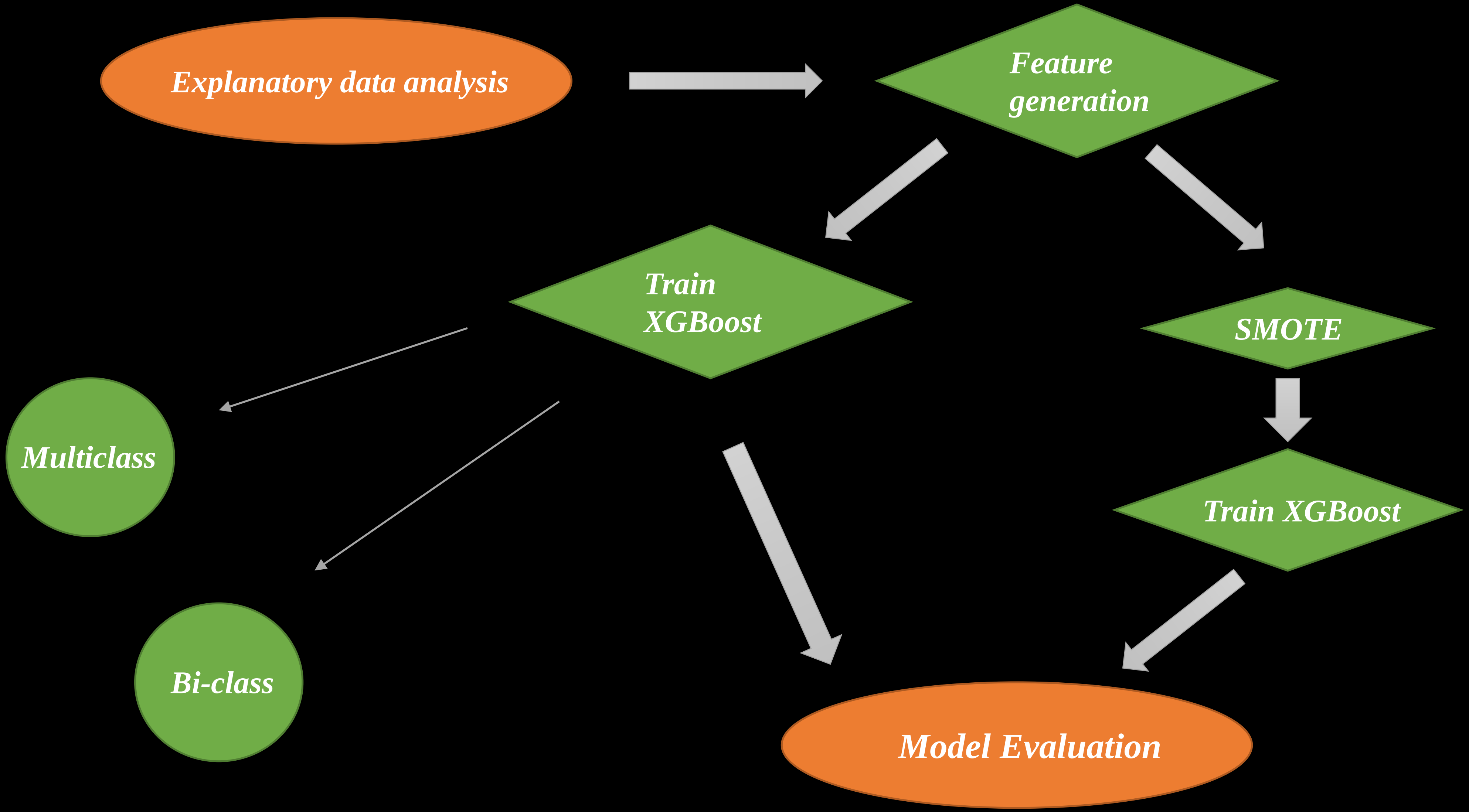
Information Extraction with *XGBoost*

Zijian Han (Harry)

Analysis Pipeline

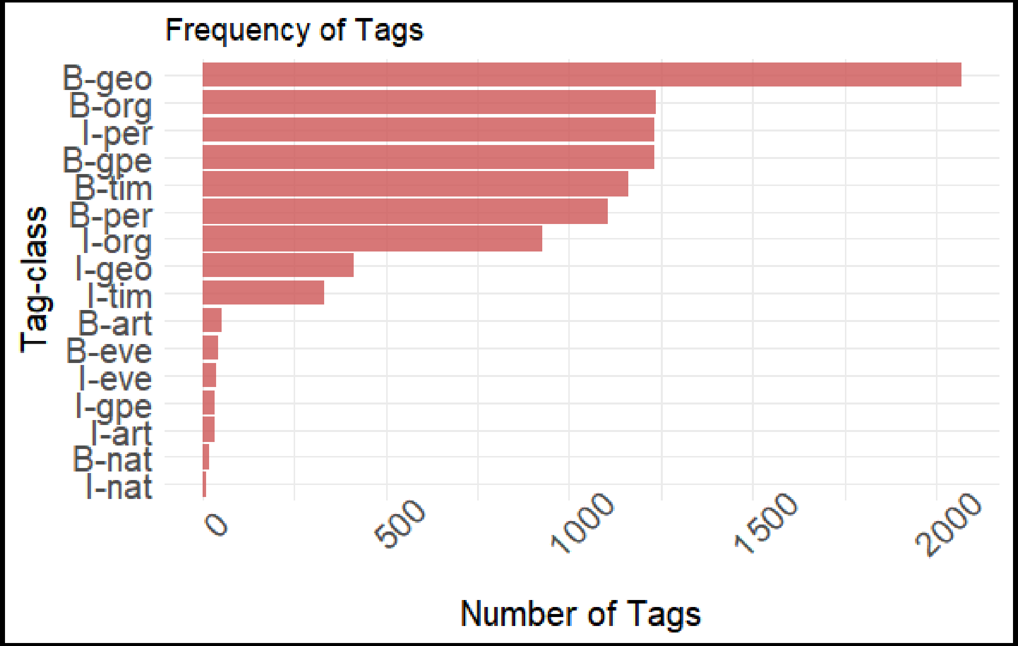


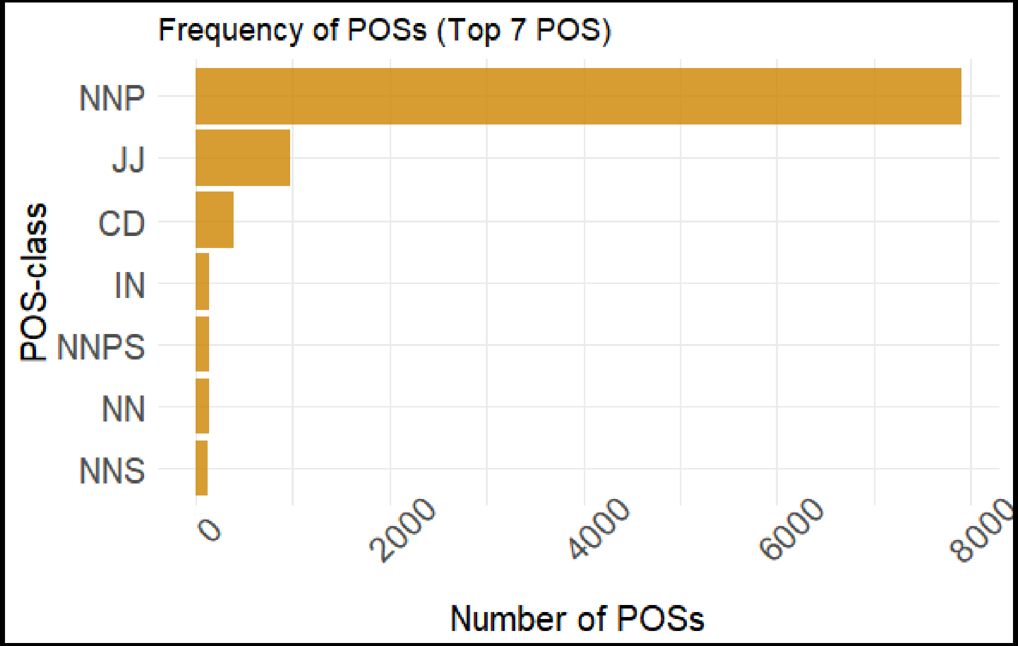
Data Exploration

- What we have for now:
  - 66161 tokens in 2999 sentences
  - POS(name entity) and Tag(to be classified)
- A first glance at these features
  - POS – factor with 41 levels (NN, NNP, JJ, VB, etc…)
  - Tag – factor with 17 levels (O, B-art, B-eve, B-geo,…, I-art, I-eve, I-geo,…)

	O	Art	Eve	Geo	Gpe	Nat	Org	Per	Tim
B-class	56217	53	45	2070	1230	20	1237	1107	1160
I-class		34	37	414	34	9	926	1234	334

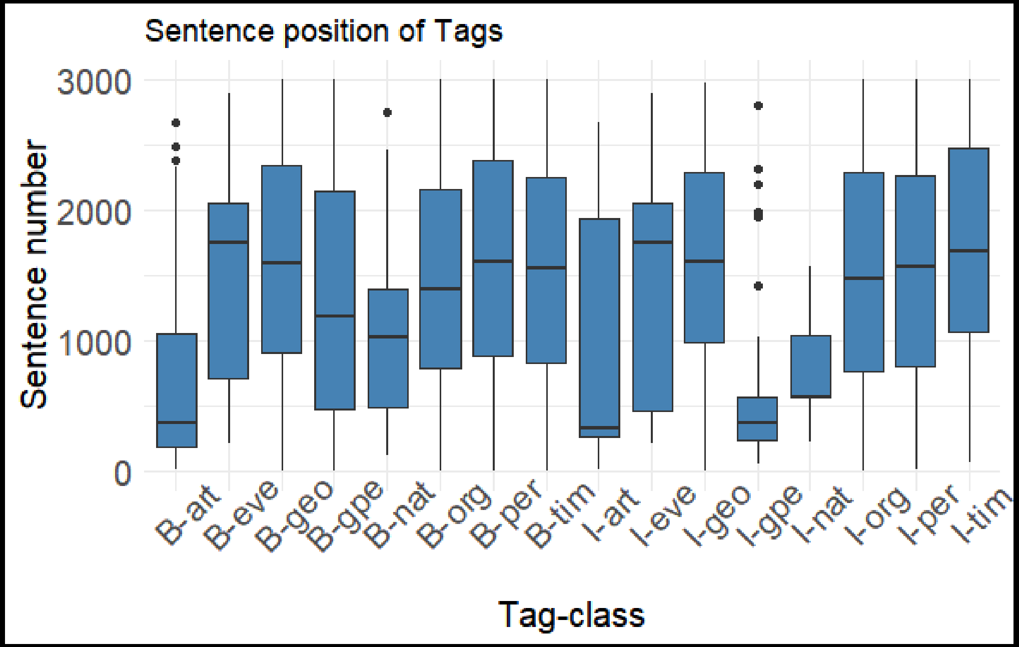
Data Exploration

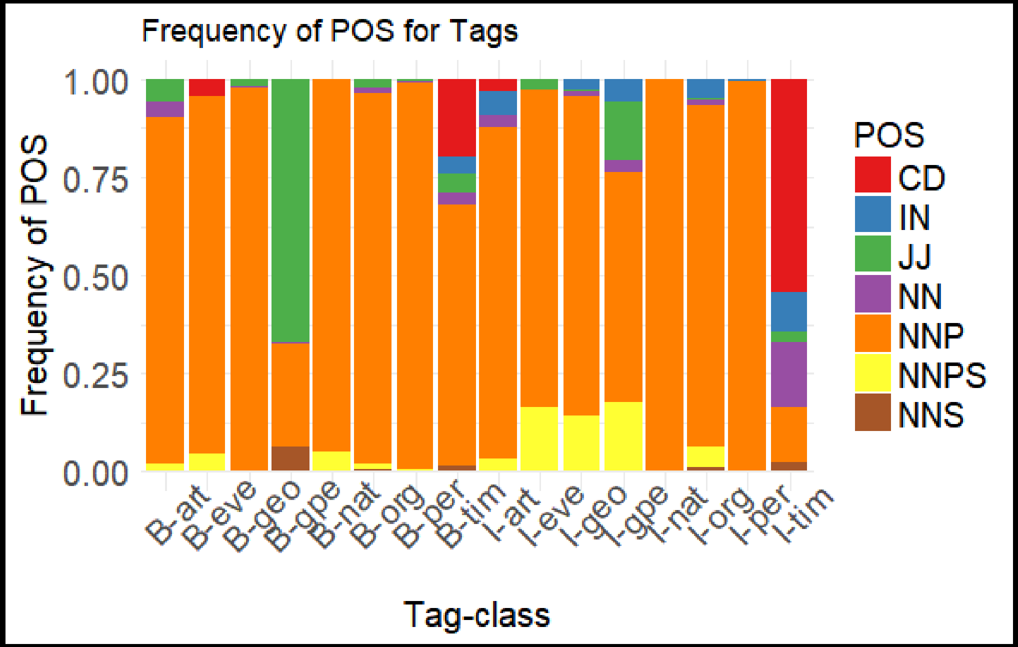




Both Tags and POS are imbalanced distributed.
The method to solve imbalance problem will be discussed later.

Data Exploration





Sentence number and POS are relevant to Tags, so we should use them as features in our model.

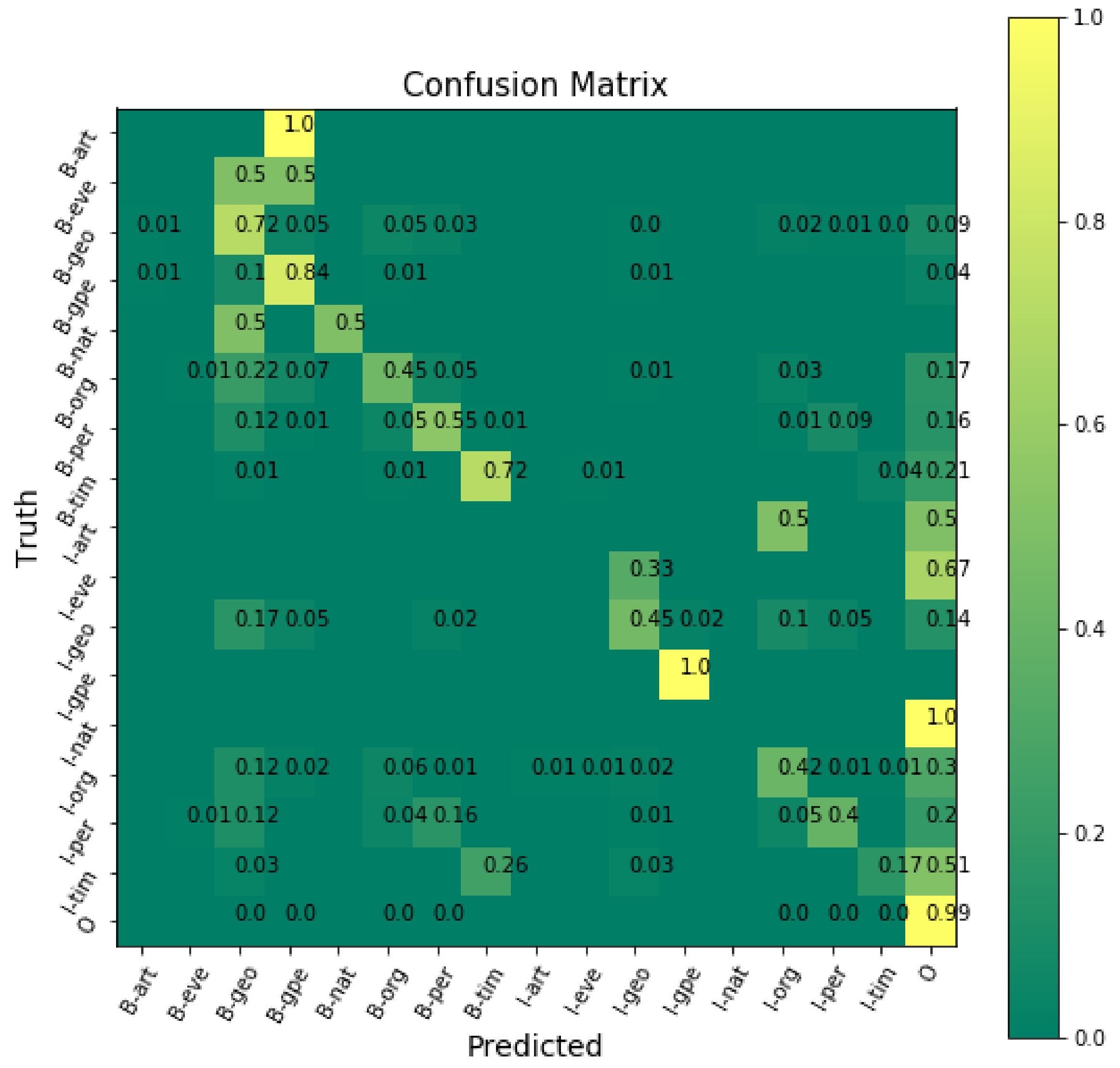
Feature generation

- Generate features for XGBoost model
  - Convert word to Integer
  - Get the neighbors
  - Sequence Models (Unigram, Bigram, Trigram, Hidden Markov, CRF, Perceptron, Affix models)
  - Other general features
    - Number of digits
    - Capital letter
    - Lower letter
    - Quotations?
    - Contain “ing”, “st”, “ed”, ”th”, etc…
  - Put all the features (around 130) together --- Becomes our input features for training XGBoost model

Model Training

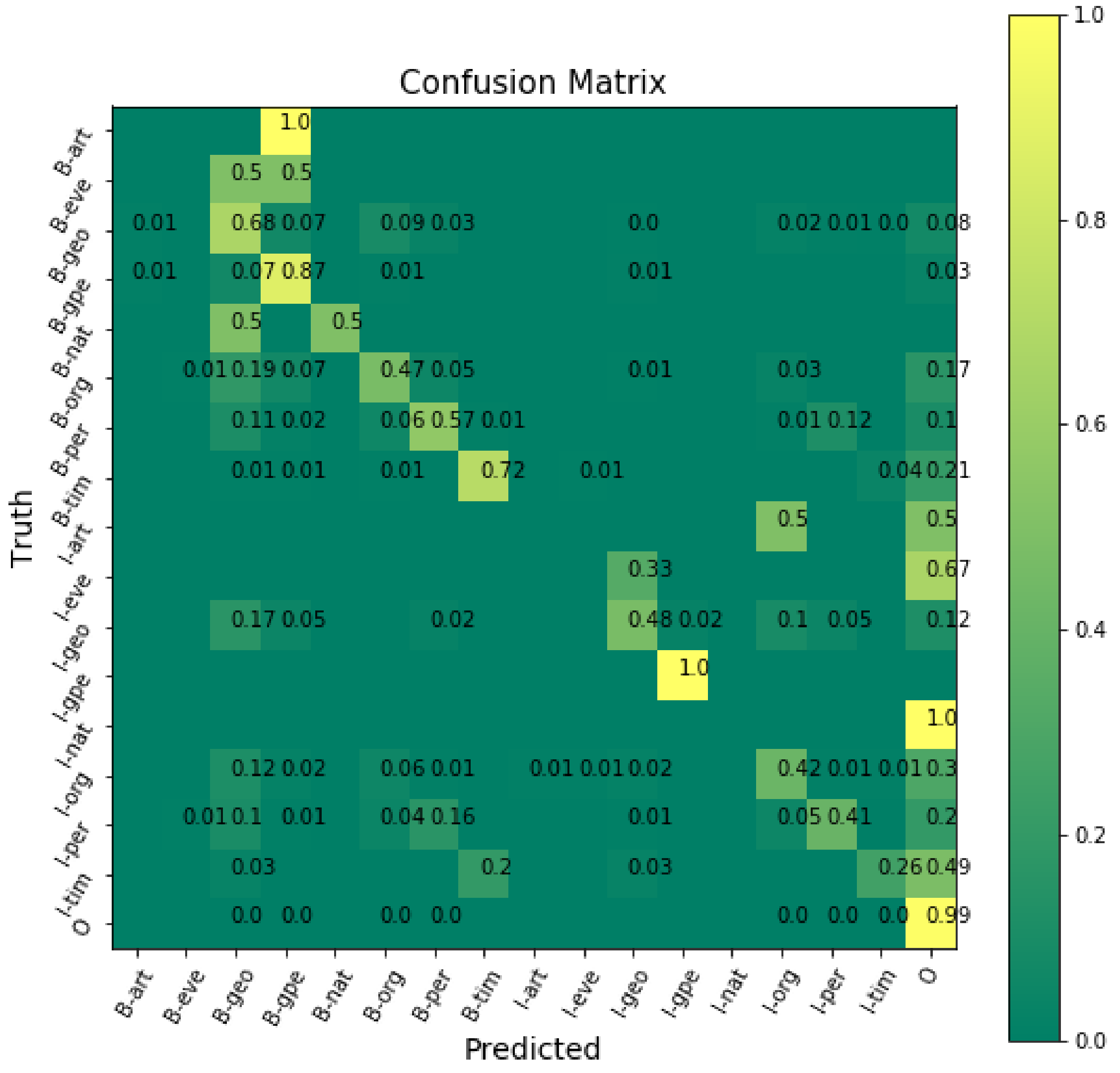
- Train, validation/test split
  - 65% Training data, 25% Validation data, 10% Testing data
- Model params:
  - learning_rate =0.1
  - n_estimators=100 (Should use larger number like 400, 500)
  - max_depth=5
  - min_child_weight=1
  - gamma=0
  - subsample=0.8
  - colsample_bytree=0.8
  - objective= multi:softmax/binary:logistic
- Model Calibration
  - Calibrate the probability

Model Evaluation Multi-class XGBoost

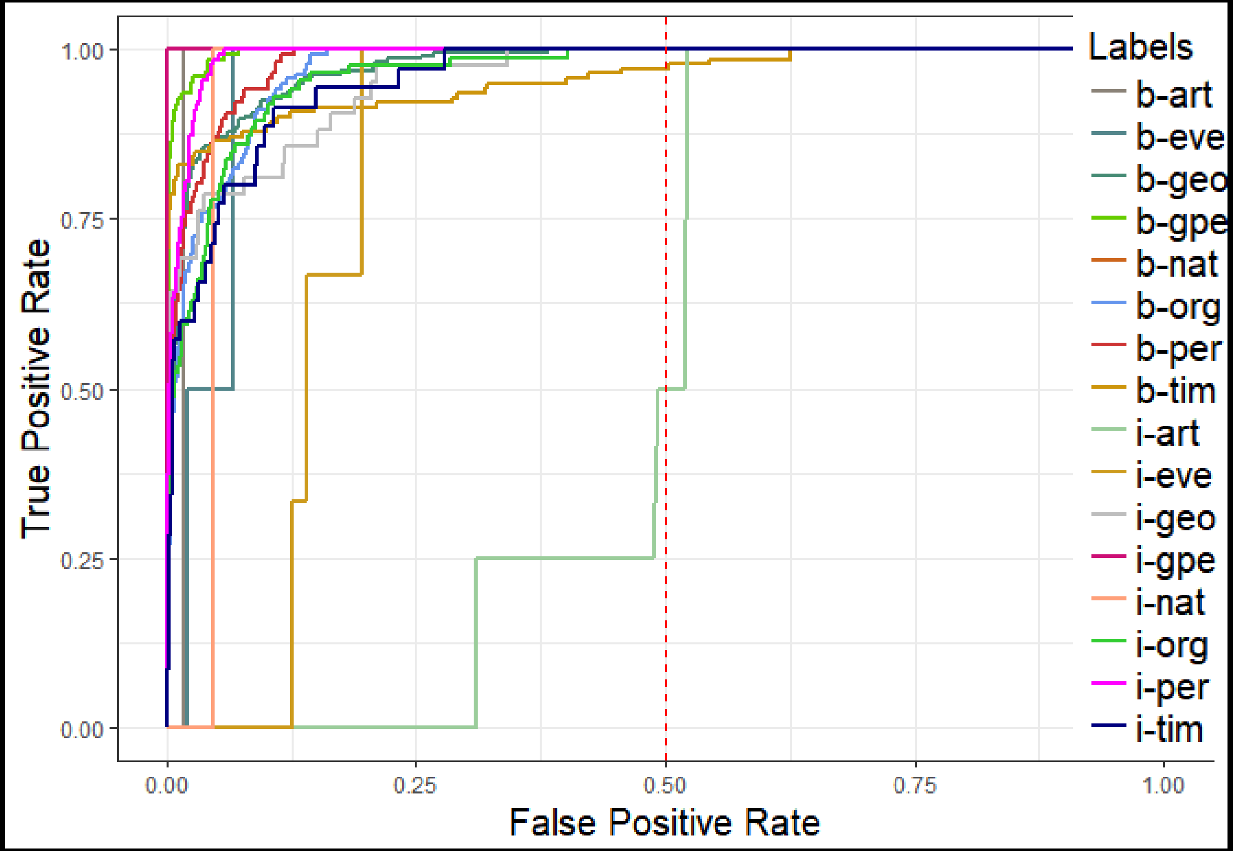


Model Evaluation Multi-class XGBoost with Calibration

In the multi-class model, some classes perform good (“B-gpe”,”I-gpe”,”B-tim”), some class even get an zero accuracy. This is because the sample size for those classes in both training and testing data are not enough, we could take a look at the binary classifier respectively.



Model Evaluation - Roc



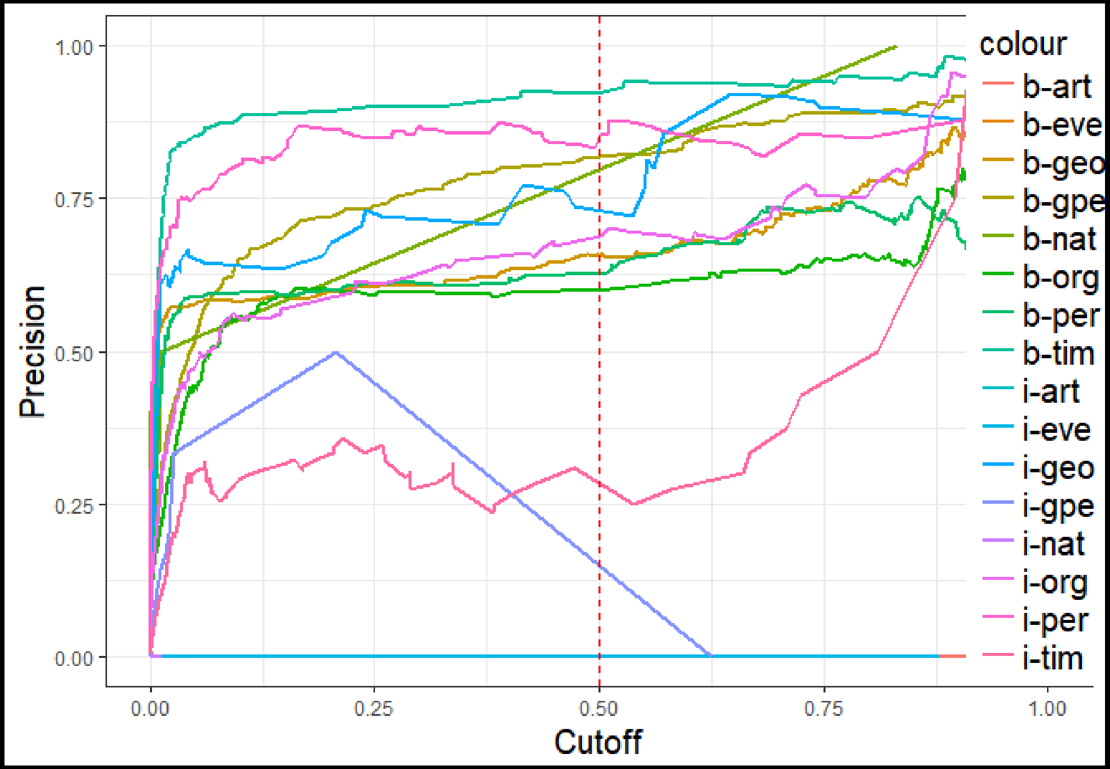
RoC curve for Binary-Classification XGBoost

RoC curve shows the overall performance of our model. We could see some classes (b-geo, B-gpe,B-org, I-org, etc..) are well classified. Some classes' RoC look strange(a straight line rather than a step function) because their sample size is extremely small (even 1 or 2) after training/testing split.

Model Evaluation - Precision

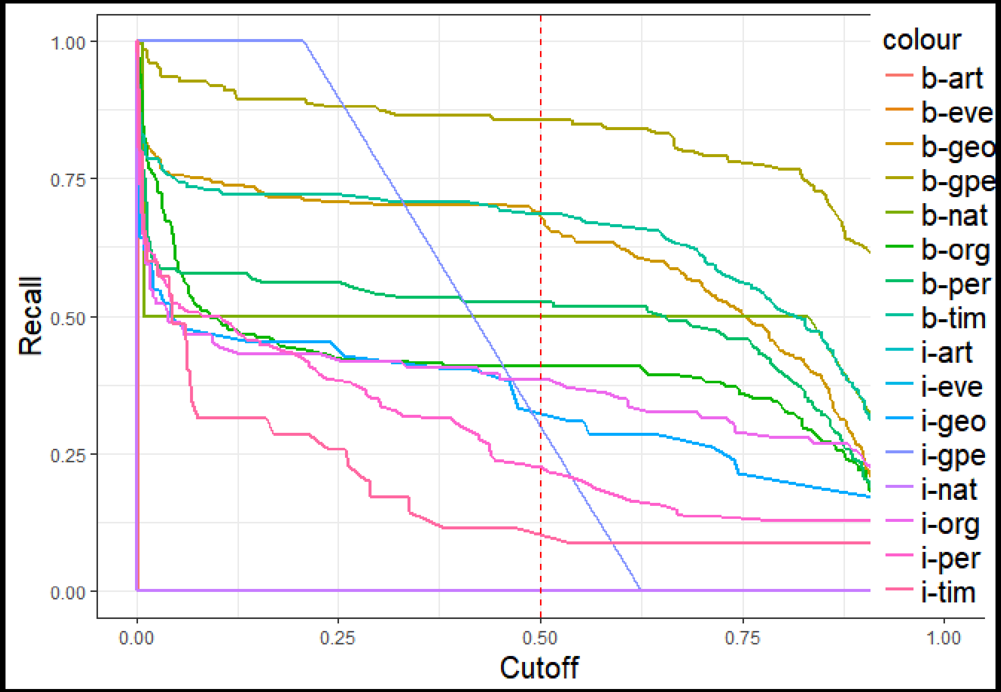
The precision curve shows us an insight of our model performance. In our case, for example, model for “b-art”, we have two classes, namely “b-art” and “O”. Precision is “How many predicted “b-art” is truly “b-art”.”

We can see here, the precision is controlled by the cutoff probability. Whenever the probability is higher than the cutoff value, we will predict the incident will occur, in our case “b-art”. So when the cutoff value is higher, it is more restrictive to give a positive prediction, all of the positive prediction are likely to be true.



Precision curve for Binary-Classification XGBoost

Model Evaluation - Recall



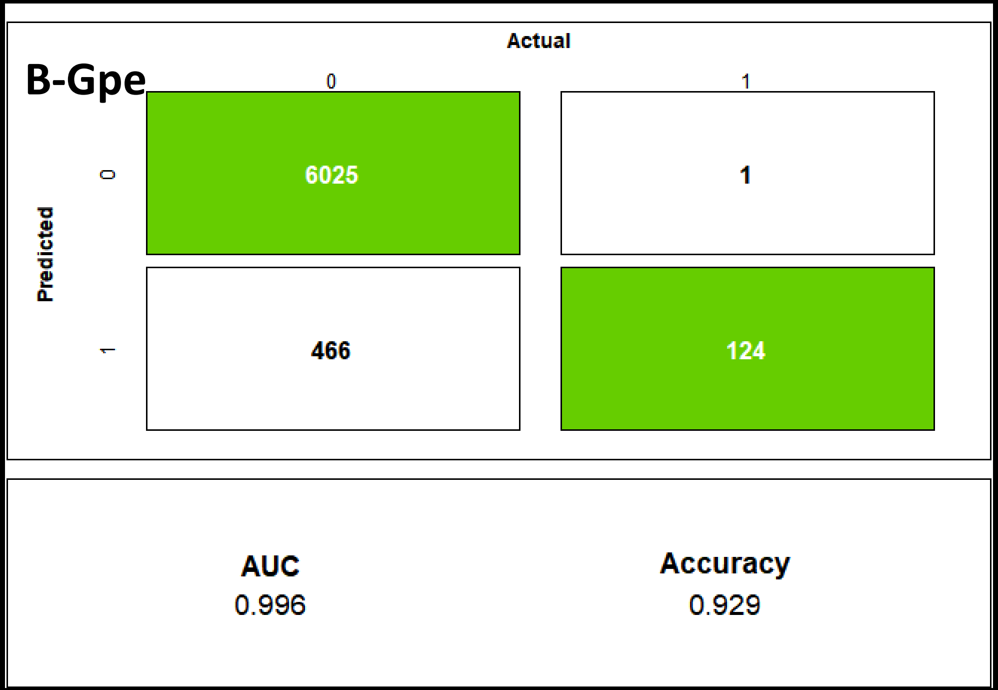
Recall curve for Binary-Classification XGBoost

In our case, the explanation of Recall is "What is the percentage of true "b-art" words are classified as "b-art".". It is also called sensitivity, recall is very critical in algorithm evaluation because it demonstrate the model's ability to detect the true positive value.

When the cut-off is small, we basically predict every words to ,e.g. "b-art", so we have a high recall but low precision. It is a trade-off between precision and recall depending on how we want our analysis to be.

Model Evaluation - Confusion matrix

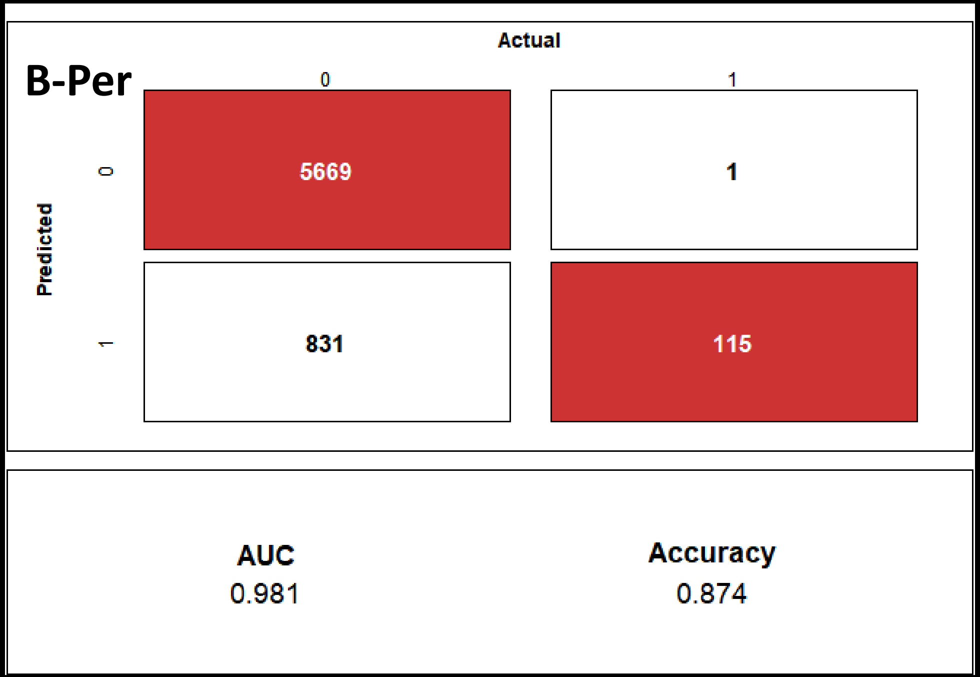
- Some good confusion matrices



B-Gpe

Predicted \ Actual	0	1
0	6025	1
1	466	124

AUC	Accuracy
0.996	0.929



B-Per

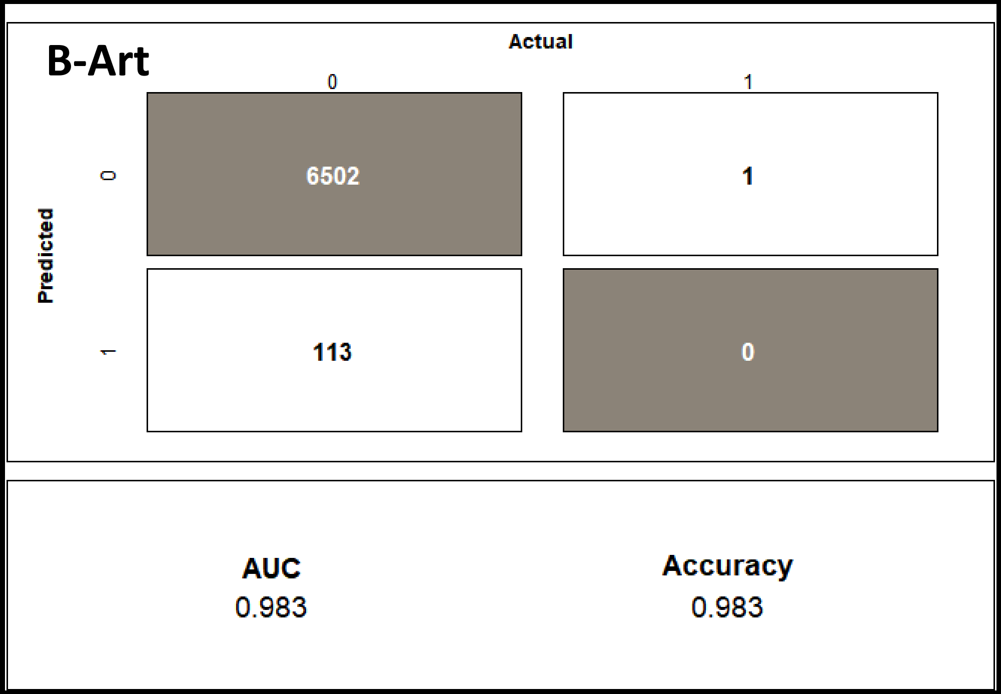
Predicted \ Actual	0	1
0	5669	1
1	831	115

AUC	Accuracy
0.981	0.874

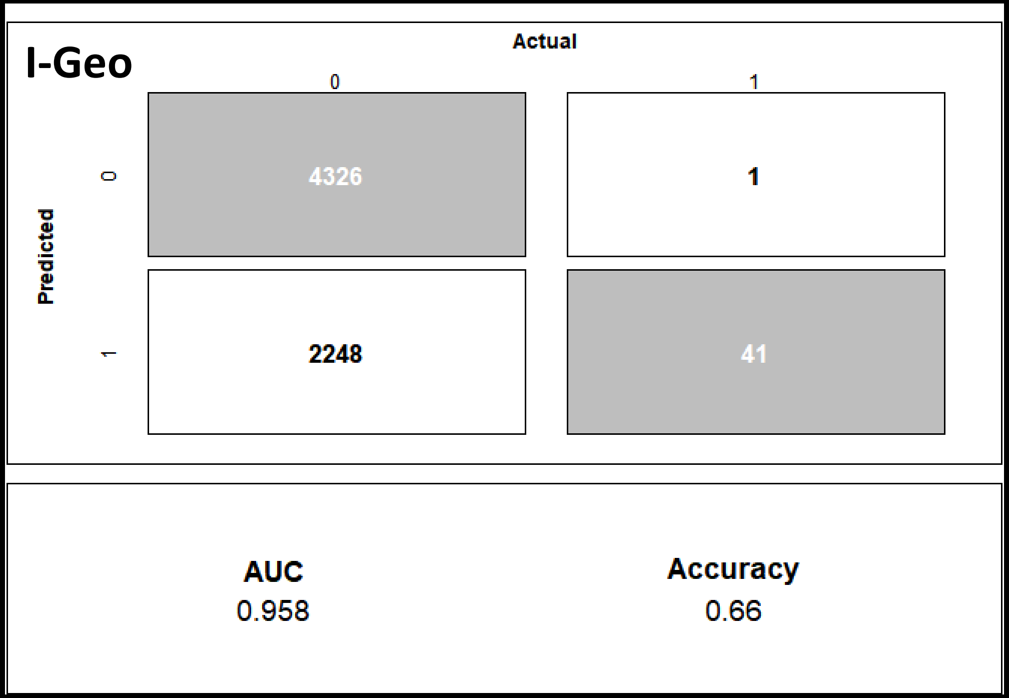
For both precision and recall, they perform good. We are very confident about this result because we have sufficient testing sample as well

Model Evaluation - Confusion matrix

- Some bad ones…



Even 0 recall using this cutoff.

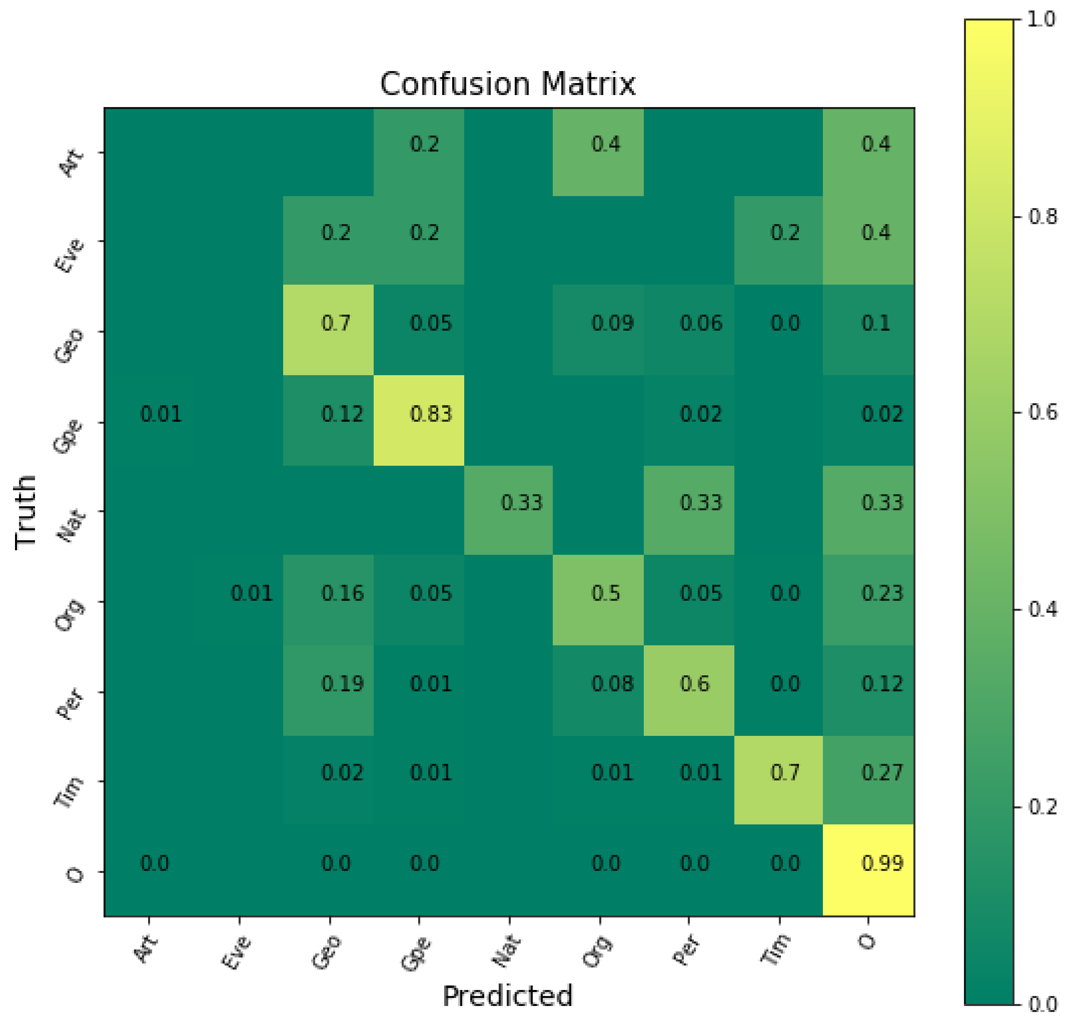


The “I-Geo” class have a pretty good recall, however, its precision rate is very poor…

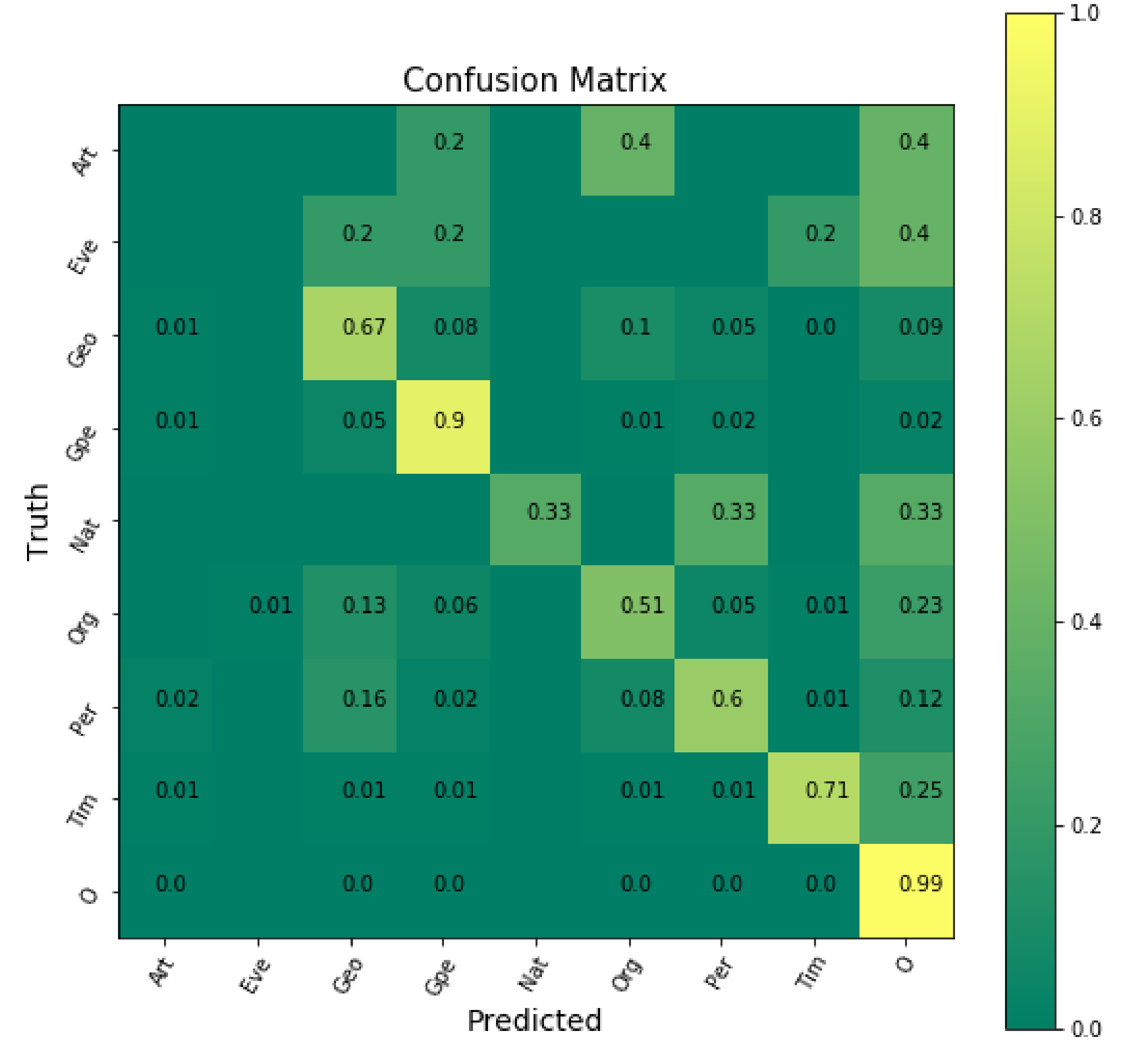
Imbalanced Sample

- We are suffered from imbalanced sample, from slide 3 Tag table, we have detected a serious imbalance for our tags group.
- Two possible ways to cure it
  - 1.Combine each sub-groups, regardless of “I” or “B”
    - Our new Tags will be “art, eve, geo, gpe, etc…”
  - 2. SMOTE(Synthetic Minority Over-sampling Technique)
    - We will oversample our training dataset using SMOTE
      - Advantage: Mitigates the problem of overfitting caused by random oversampling. No loss of useful information
      - Disadvantage: Not efficient for high-dimension data

Solution 1
– combine sub group
Multi-class XGBoost

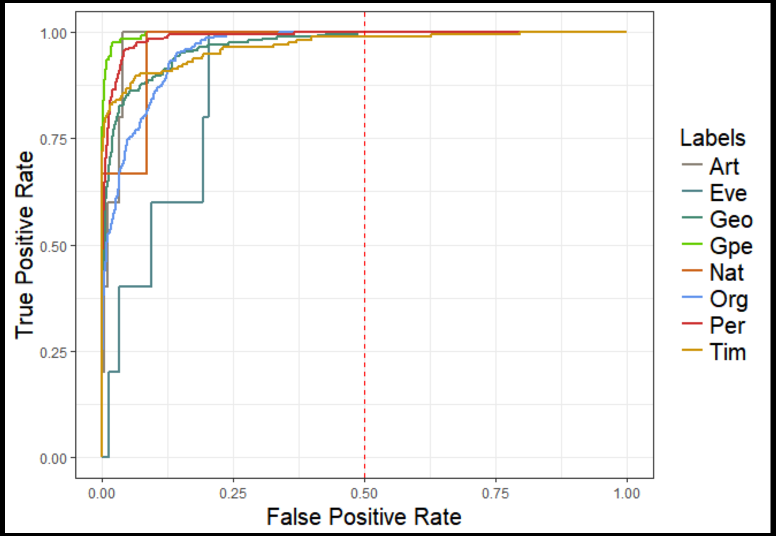


Solution 1 – combine sub group Multi-class XGBoost (after Calibration)

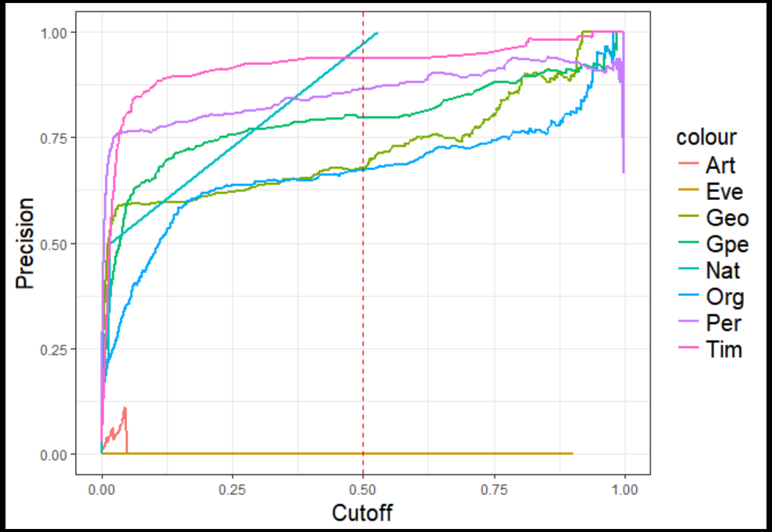


“Tim” and ”Gpe” performances are pretty good. We could use these classifier to predict labels for those two class efficiently. However, for “Art” and “Eve” groups, their performances are not improved.

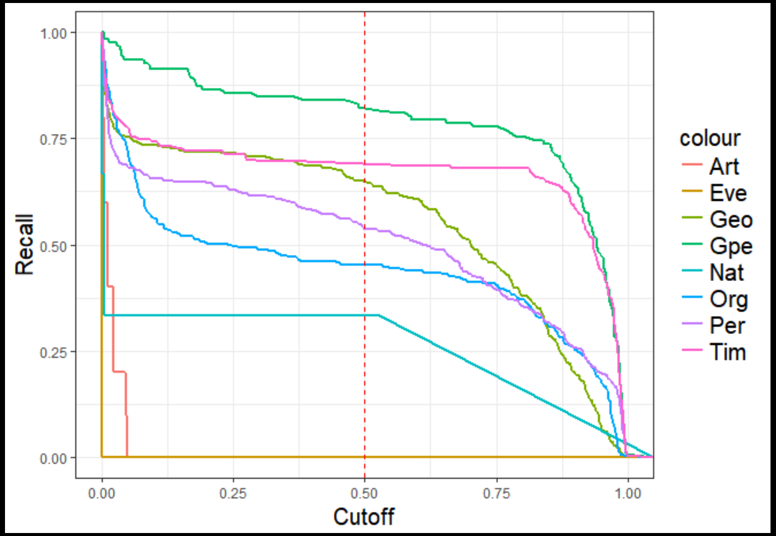
Solution 1 – combine sub group (RoC, Precision & Recall)



Roc curve from Combined sample



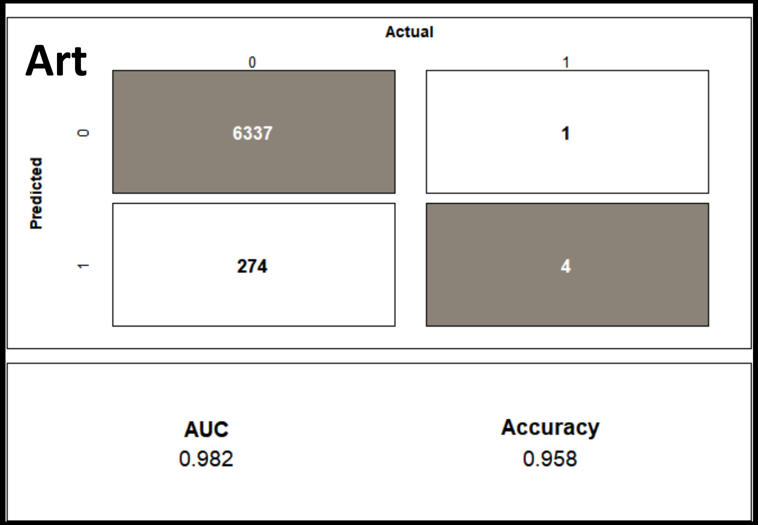
Precision curve from Combined sample



Recall curve from Combined sample

With 8 combined classes rather than 16 classes before, our model performs better because there are less extremity imbalance in our sample.
However, the "Nat" class still get pool performance. I think it is because it only has 29 words in our original dataset, and still only 3 words in testing dataset!

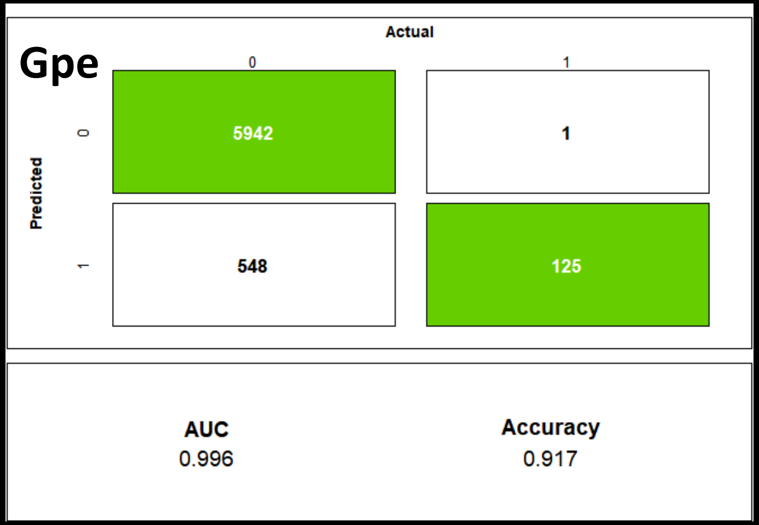
Solution 1 – combine sub group (Selected Binary Confusion Matrices)



Art

Predicted \ Actual	0	1
0	6337	1
1	274	4

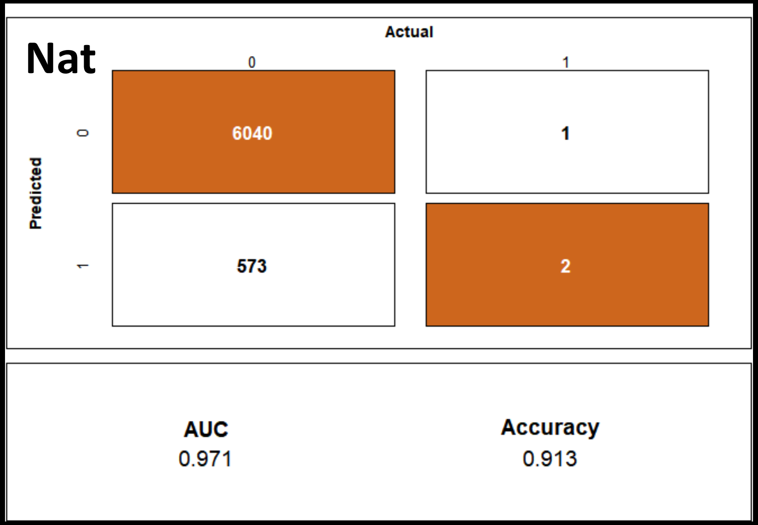
AUC	Accuracy
0.982	0.958



Gpe

Predicted \ Actual	0	1
0	5942	1
1	548	125

AUC	Accuracy
0.996	0.917



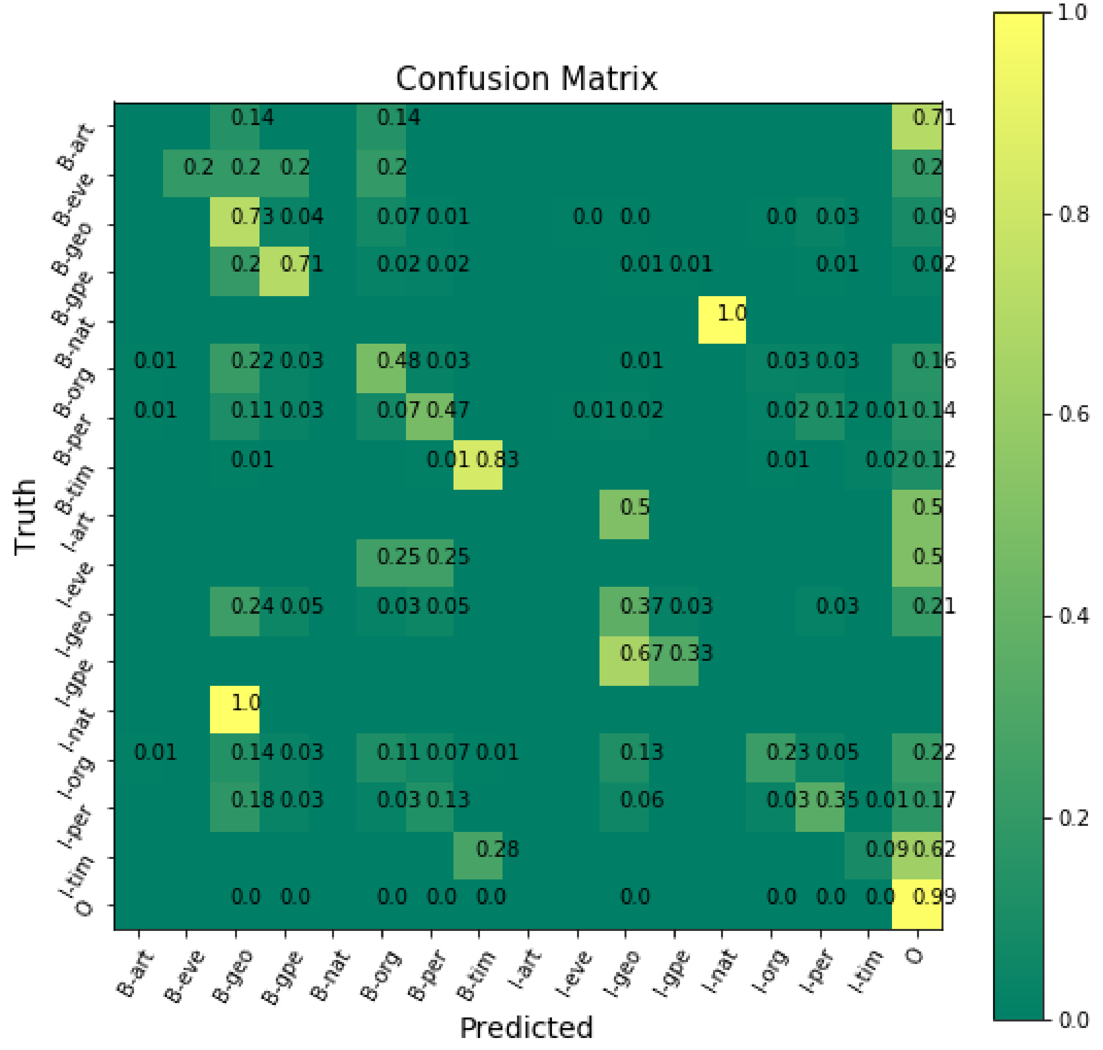
Nat

Predicted \ Actual	0	1
0	6040	1
1	573	2

AUC	Accuracy
0.971	0.913

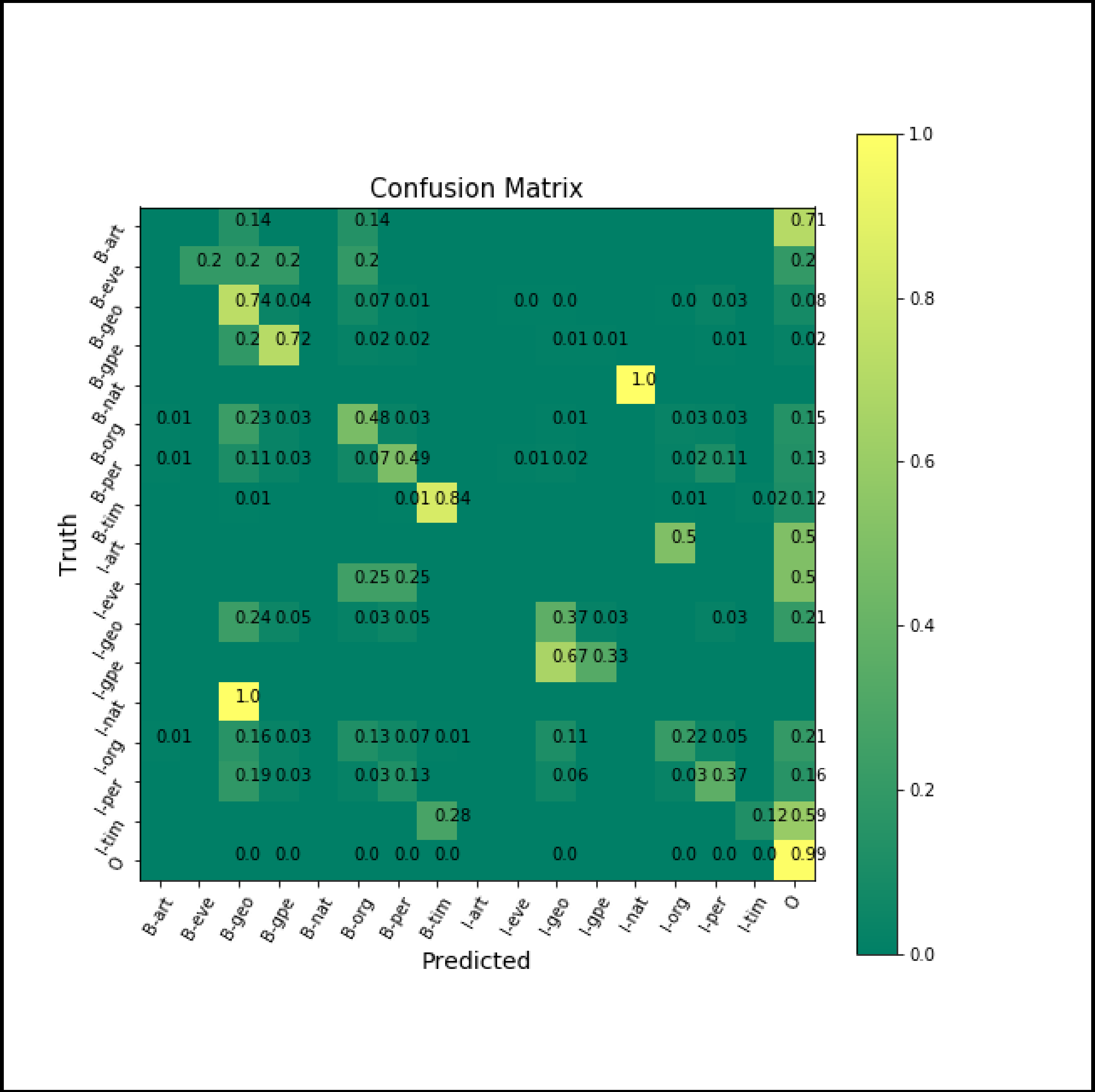
Again. I select some confusion matrices from the 8 combined classes just for illustration. The binary confusion matrices for “Art” and “Gpe” show us some good classification result.
Although the “Nat” class overall accuracy is very high (even closed to 1), it misclassifies 33% of the positive value to negative(recall = 66%), its precision is also very poor. This is actually bad result if our ultimate goal is to determine which word is “Nat”.

Solution 2 – SMOTE Multi-class XGBoost



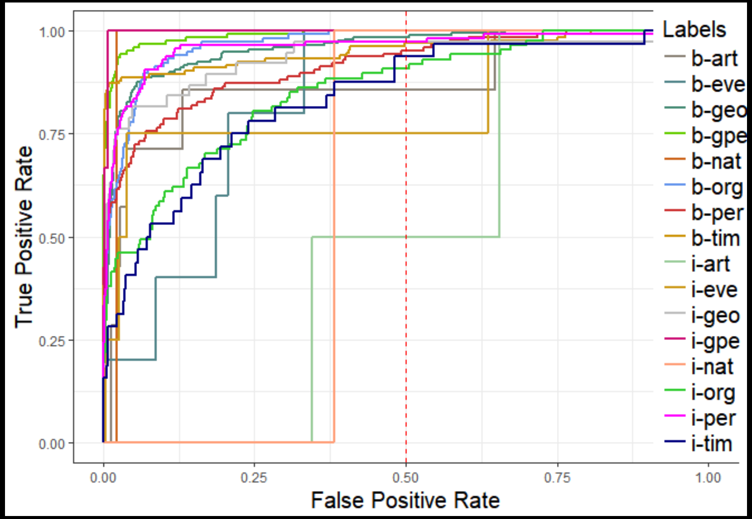
Truth \ Predicted	B-art	B-eve	B-geo	B-gpe	B-nat	B-org	B-per	B-tim	I-art	I-eve	I-geo	I-gpe	I-nat	I-org	I-per	I-tim	O
B-art			0.14			0.14											0.71
B-eve		0.2	0.2	0.2		0.2											0.2
B-geo			0.73	0.04		0.07	0.01			0.0	0.0			0.0	0.03		0.09
B-gpe			0.2	0.71		0.02	0.02				0.01	0.01			0.01		0.02
B-nat													1.0				
B-org	0.01		0.22	0.03		0.48	0.03				0.01			0.03	0.03		0.16
B-per	0.01		0.11	0.03		0.07	0.47			0.01	0.02			0.02	0.12	0.01	0.14
B-tim			0.01				0.01	0.83						0.01		0.02	0.12
I-art											0.5						0.5
I-eve						0.25	0.25										0.5
I-geo			0.24	0.05		0.03	0.05				0.37	0.03			0.03		0.21
I-gpe											0.67	0.33					
I-nat			1.0														
I-org	0.01		0.14	0.03		0.11	0.07	0.01			0.13			0.23	0.05		0.22
I-per			0.18	0.03		0.03	0.13				0.06			0.03	0.35	0.01	0.17
I-tim								0.28								0.09	0.62
O			0.0	0.0		0.0	0.0	0.0			0.0			0.0	0.0	0.0	0.99

Solution 2 – SMOTE Multi-class XGBoost (After Calibration)

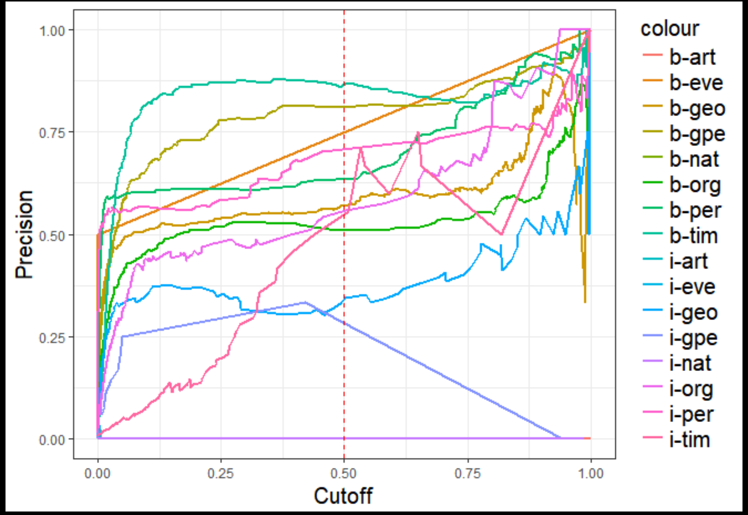


Some classes' performances are improved("B-geo","B-tim"). Performance of minority group is still bad. After all, we could not oversample testing dataset using SMOTE, the evaluation for those minority group is not robust.

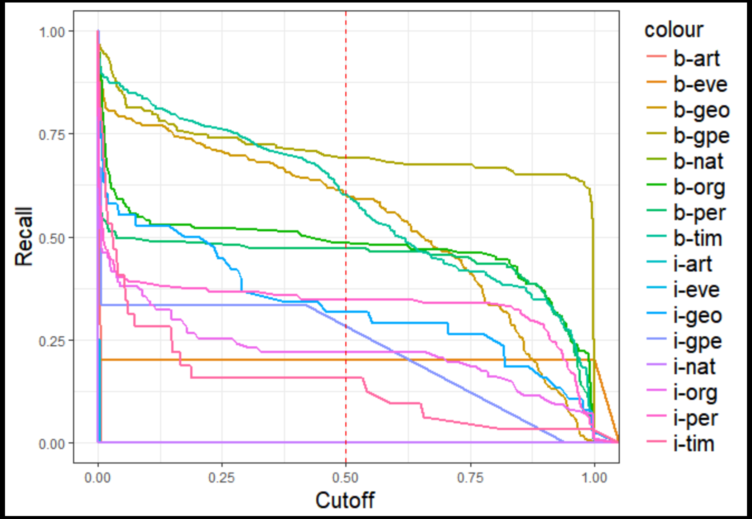
Solution2 – SMOTE (RoC, Precision & Recall)



Roc curve from SMOTE sample



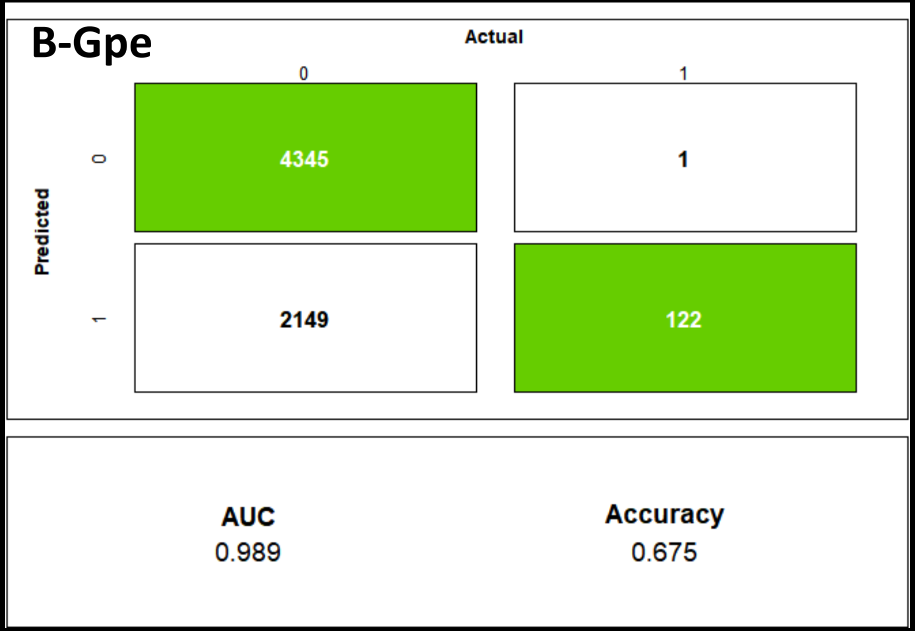
Precision curve from SMOTE sample



Recall curve from SMOTE sample

The performance is still not very good for some classes. Although we use SMOTE to oversample our training dataset, our testing dataset is still inefficient for us to do a robust evaluation. For majority classes with sufficient data, our model fits well.

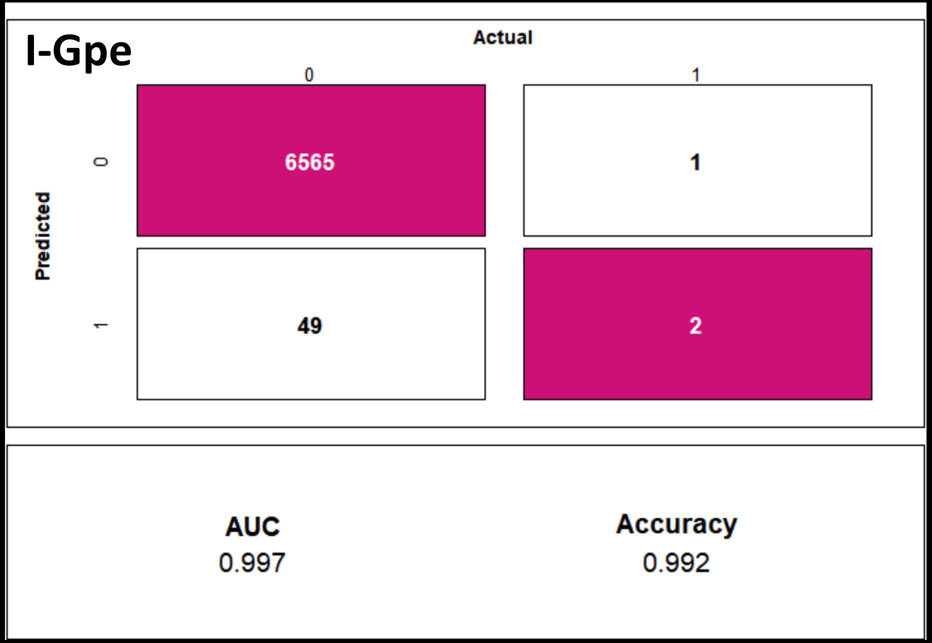
Solution2 – SMOTE (Selected Binary Confusion Matrices)


B-Gpe

Predicted \ Actual	0	1
0	4345	1
1	2149	122

AUC 0.989

Accuracy 0.675


I-Gpe

Predicted \ Actual	0	1
0	6565	1
1	49	2

AUC 0.997

Accuracy 0.992

Conclusion Note

- The plain sample model has already shown us we can predict correctly for the majority group with sufficient sample size, however, for those minority group, their prediction performance is very poor. Of course, our XGBoost model is not optimally tuned, so there could be an overfitting issue, in future projects with enough time and good computer condition, we should tune our XGBoost models' parameters to optimize our results.
- I use two methods trying to solve the imbalance sample problem. Actually, I think in a real project for clients, we could think it from two perspectives: (1) Enhance our algorithms (2) Enhance our sample. Here, these two methods are all belong to "enhance sample". I think the boosted ensemble model XGBoost has already done a good job to deal with minority groups. Other that that, I try to use SMOTE to oversample the minority groups in the training set. As I have discussed before, this oversample method is more robust than random sampling because it will not generate repeated data, so it could avoid overfitting. However, since the KNN approach is implemented in SMOTE, it is not very efficient to our high dimension feature space.

Conclusion Note

- The second I approach I used is very simple, I combine each sub-group tags into one group, for example, “I-geo” and “B-geo” into “Geo”. These will efficiently increase our sample size for each group and from the previous slides. Actually, maybe their performance is not bad, their performance evaluation is not robust because we only have a few observations for these two groups in the testing dataset.

Further Recommendations

- 1. Obtain enough data – solve the imbalance issue.
- 2. We could extract/delete more features. Actually, I should have deleted the redundant features(all 0; all with 1 same level, etc…). But I think XGBoost algorithm could do it by itself.
- 3. Optimally tune our model parameters with higher n_estimators. Additionally, we may assume a weight parameter in our XGBoost model.