TEAM TK

Lee Young Ho
Park Jinah
Choi Wonjae

1 Problem Statement & Objectives
2 Data Analysis & Preprocessing
3 Baseline Model: S-Learner
4 Hyperparameter Tuning
5 Final Model: XGBoost Classifier
6 Evaluation
7 Conclusion

Problem Statement & Objectives

Objective: Maximise Incremental Activation Rate → P(Act = 1 | Rec = 1) – P(Act = 1 | Rec = 0)

Solution: In order to achieve the objective, the persuadables must be ranked above sleeping dogs.

Our Approaches

1. Direct Estimation Method
   a. S-learner (Baseline Model)

2. Two-Model Method
   a. T-learner
   b. X-learner

3. Class Transformation Method
   a. XGBoost Regression
   b. XGBoost Classifier (Final Model)

Training Process Illustration

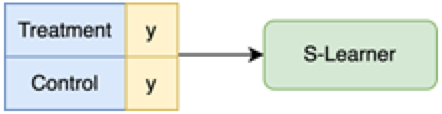


Fig 1. Direct estimation method training

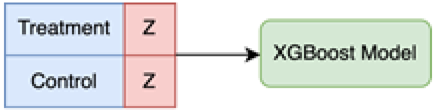
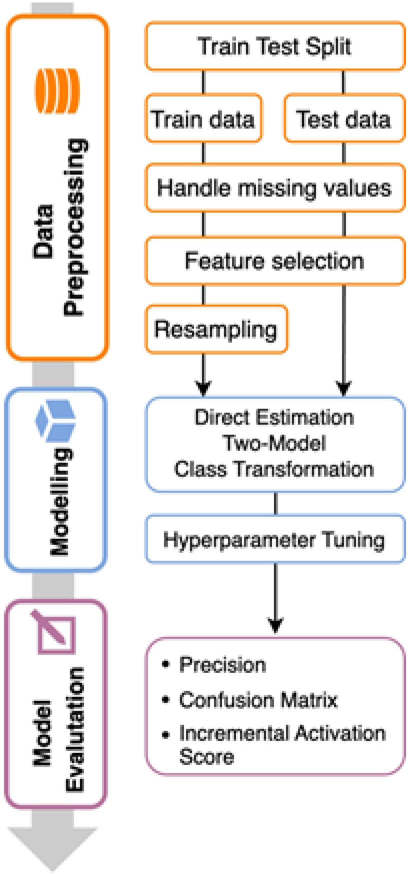


Fig 3. Class transformation training

Our Framework



* For the purpose of clarity and brevity, this presentation will focus exclusively on **baseline and final models.** Refer to notebooks for elaborations on alternative models.

Dataset Analysis & Preprocessing

Step 1) Train Test Split

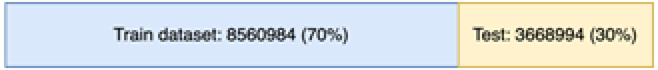


Fig 4. Train-test split ratio

Step 2) Handling Missing Values

The dataset has large data sparsity (around half the features with +50% missing values). To effectively handle missing values, we studied the density plot of the features in detail.

Step	Criteria	Fill NA Methods	Details
1	% of missing values > 70%	Boolean	1: If value exists 0: If value is missing
2	If the data distribution features several distinct and pronounced peaks	Mode	Fill NA with mode
3	Distance related features	Mean	Fill NA with mean of all values
4	Evenly distributed	Mean	Fill NA with mean of all values
4	Merchant Profile	Mean by Merchant	Fill NA with mean for each merchant id
5	Others	Zero	Fill NA with 0

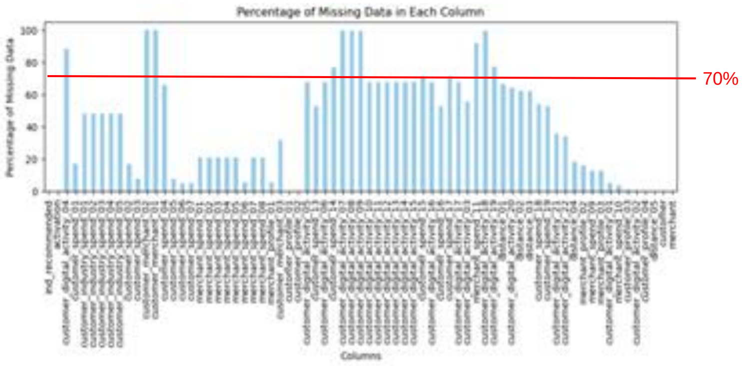


Fig 5. Percentage of missing data in each feature

Fig 6. Example of feature filled with mode

Fig 7. Example of feature filled with mean

Dataset Analysis & Preprocessing

Step 3) Feature Selection - dropping features x

3. 1) Handling categorical feature:

Out of all features, **“merchant_profile_01”** is the only categorical feature, with 67 unique class labels.

Reasons for dropping:

- With no clear knowledge of what merchant industry clear signifies, nominal categorical values may introduce **arbitrary or misleading ordinal relationships** between categories.
- Employing one-hot encoding to address this issue would result in the creation of an additional 67 features (**high dimensionality**).

3.2) Embedded approach:

Our team utilised XGBoost Classifier to drop features with 0 importance value.

Dropped features: customer_digital_activity_09, customer_digital_activity_18, customer_spend_17, customer_digital_activity_08, customer_merchant_01

Reasons for dropping:

- **Reduces dimensionality** of the dataset.
- By removing irrelevant features, we can improve the model performance and **reduce overfitting and noise**.

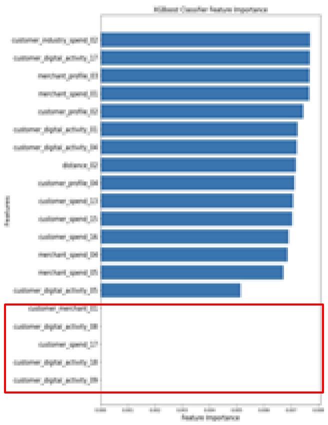


Fig 8. Feature importance extraction from XGBoost Classifier (Only bottom 20 shown)

Step 4) Resampling

The dataset is highly imbalanced with P(Act=1) = 0.00575.

Thereby, we decided to conduct random resampling on all the classes to match the size of (Act=1|Rec=0). The choice for the size was to minimise the need for oversampling, which may add unwanted bias to the model.

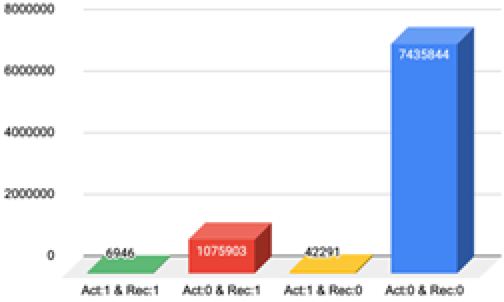


Fig 9. Original class ratio

Random Resampling

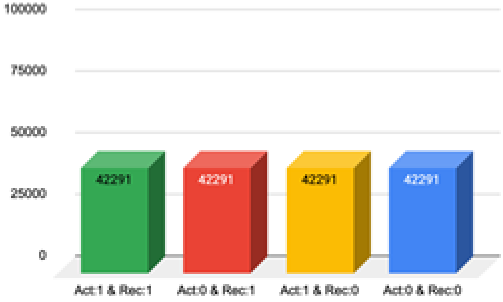


Fig 10. Resampled class ratio

Baseline Model: Uplift S-learner

Uplift S-learner

For our baseline model, it utilises the recommendation as a feature to conduct **binary classification** onto whether the given customer's activation will be 1 or 0. It serves its purpose as the baseline model as it can **easily be implemented** (Refer to Fig 1. for training framework)

For the estimator, our group employed Random Forest Classifier, which can **effectively handle complex non-linear relationships** through multiple decision trees. (Tuned parameters: n_estimator : 300, max_depth: 20)

Prediction Phase

Once the model is trained, S-learner calculates the uplift score for each entry following the below architecture.

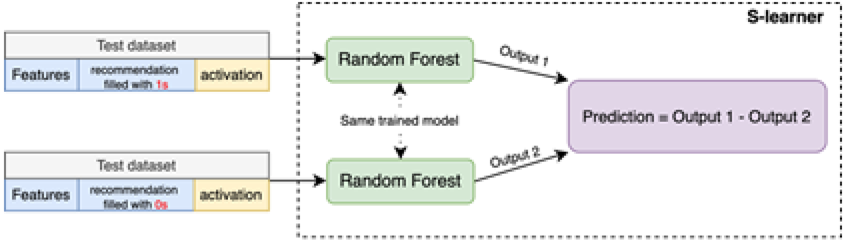


Fig 11. Baseline Model prediction architecture

This architecture is aligned with our objective to maximise incremental activation, following the $P(Act=1 \mid Rec=1) - P(Act=1 \mid Rec=0)$ formular.

Rewarding persuadables **Penalising sure things**

Evaluation: Qini Curve

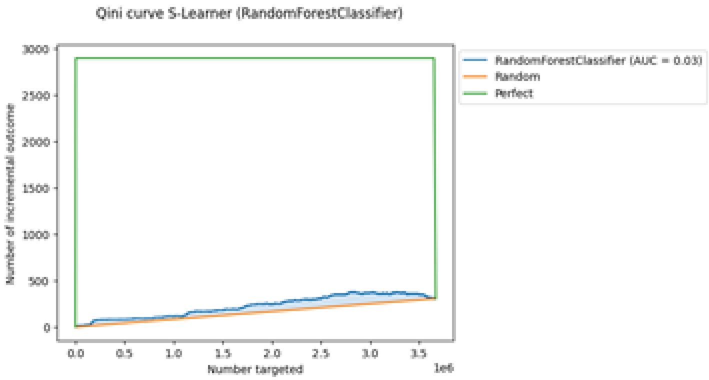


Fig 12. Baseline Mode Qini Curve

The AUC score of 0.03 indicates that the uplift model is performing **only slightly better** than random targeting, suggesting there is a lot of room for improvement.

Potential reasons:

1. RF model may not be complex enough to capture subtle effect of recommendation.
2. Overfitting could have occurred showing poor performance.
3. Prediction phase assumes linear relationship between model outputs and uplift score.

Data Preprocessing
Modelling
Model Evalutation

Final Model: XGBoost Classifier

Training Phase

- Multiclass Classification
- XGboost Classifier: An ensemble learner that outperforms other models through means of regularisation and sequential gradient boosting.

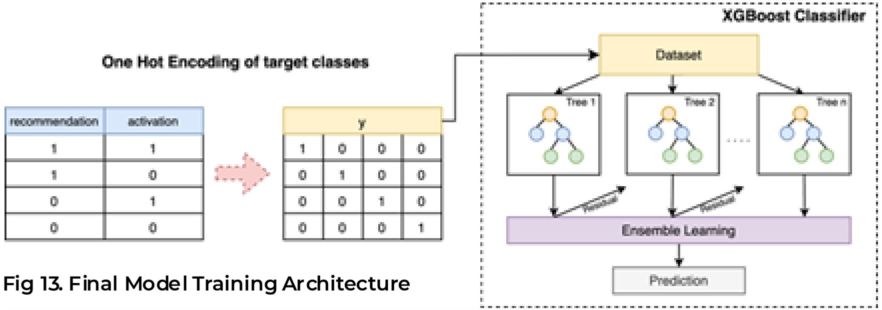


Fig 13. Final Model Training Architecture

Training Results

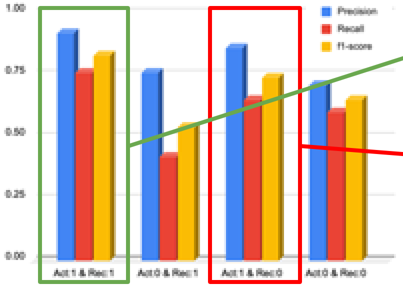


Fig 14. Precision, Recall, F1-score of Model

It is positive to note that the model has high precision on the two most influential classes in terms of uplift score.

Fig 15. Confusion Matrix of Model

Uplift Modelling

Persuadables	Sure Things
Lost Causes	Sleeping dogs

Prediction Phase

Class Transformation integrated in prediction phase

$$Z_i = Y_i \cdot W_i + (1 - Y_i)(1 - W_i)$$

$$\tau(X_i) = 2 \cdot P(Z_i = 1) - 1$$

Z_i: new target for customer i
Y_i: activation of customer i
W_i: treatment of customer i

Given the predicted probability of an entry:

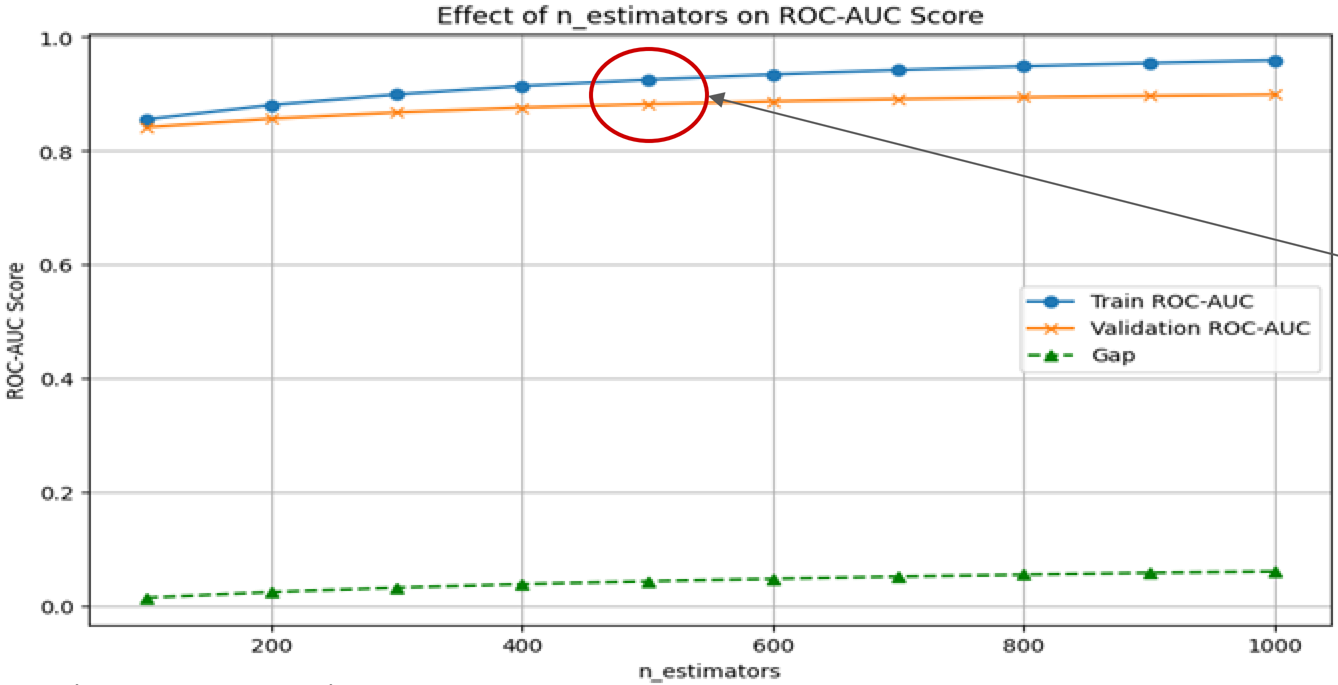
$$\begin{aligned} Predicted\ score &= P(Z = 1) \\ &= P(Act = 1 | Rec = 1) + P(Act = 0 | Rec = 0) \end{aligned}$$

- Incentivising the persuadables
- Penalising the sleeping dogs.

Hyperparameter Tuning

Methodology: We employed the **ROC AUC** score as the primary evaluation metric for **Cross Validation Grid Search**; fine-tuned the optimal parameters in a sequential manner.

Step 1) Number of Estimators

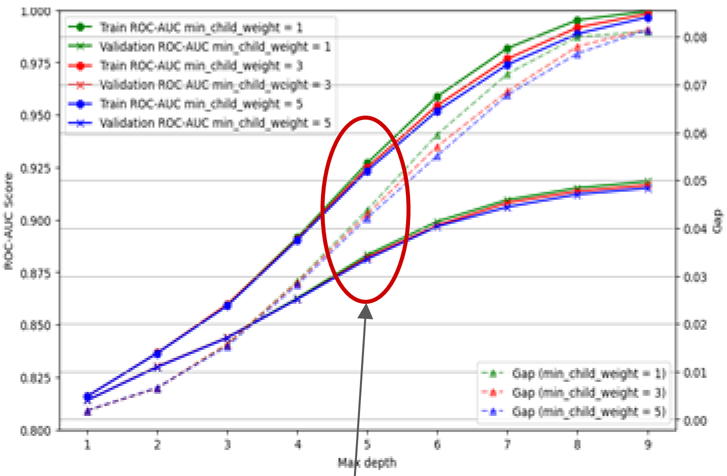


Chosen:
n_estimator = 500
Reason: increase in ROC-AUC score is not significant after 500.

Fig 16. Effect of n_estimator on ROC-AUC score

Hyperparameter Tuning

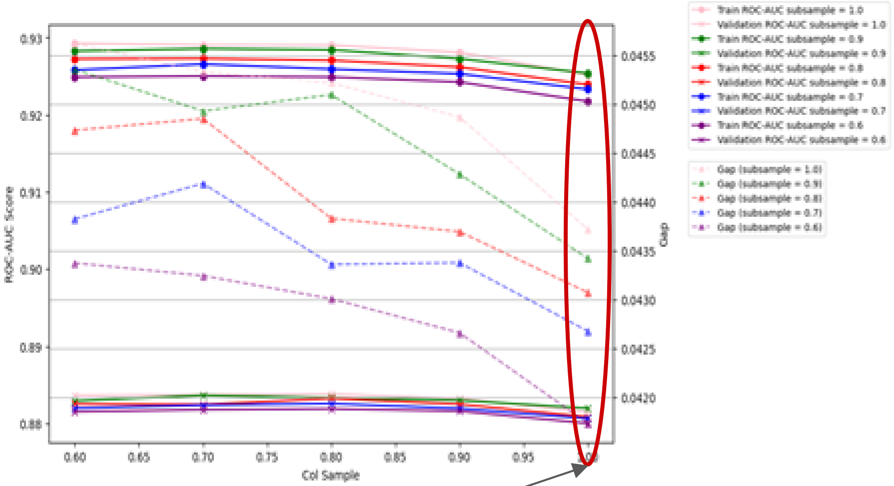
Step 2) Maximum depth and Minimum Child Weight



Chosen: max_depth = 5 & min_child_weight = 1
Reason: the gap increases significantly from max_depth = 6, and min_child_weight =1 has lowest gap.

Fig 17. Effect of max_depth and min_child_weight and on ROC-AUC score

Step 3) Sub-sample and Col-sample



Chosen: col_sample = 1.0 and sub_sample = 0.9
Reason: the gap decreases significantly at col_sample = 1.0, and sub_smaple 0.9 has the highest score

Fig 18. Effect of sub-sample and col-sample on ROC-AUC score

Hyperparameter Tuning

Step 4) Regularization Parameters -alpha

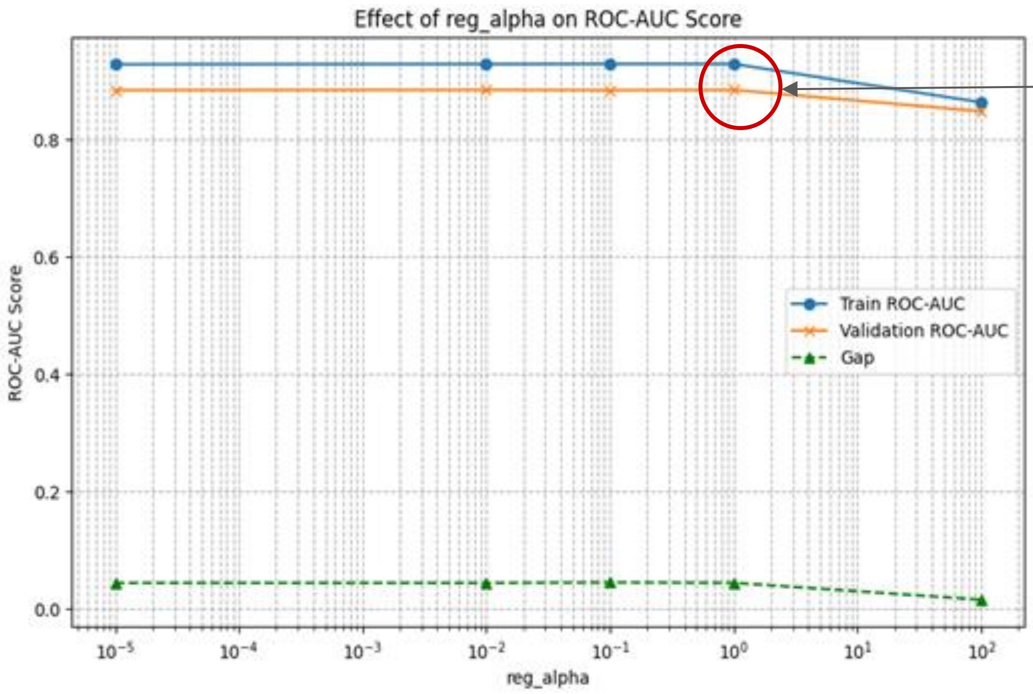


Fig 19. Effect of reg_alpha on ROC-AUC score

Chosen: reg_alpha = 1
Reason: has the highest score and difference in the gap is minimal.

Final Parameters

```
param_grid = {
  'n_estimators': 500,
  'learning_rate': 0.1,
  'max_depth': 5,
  'min_child_weight': 1,
  'gamma': 0,
  'subsample': 0.9,
  'colsample_bytree': 1.0,
  'reg_alpha':1.0,
  'tree_method' : "hist",
  'device' : "cuda"
}
```

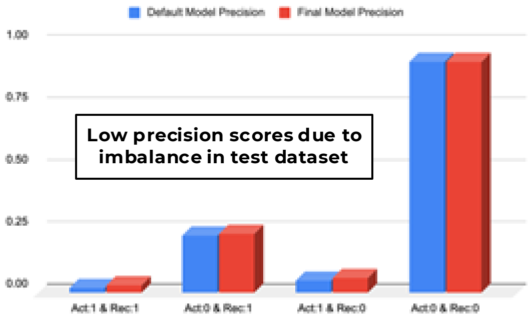
Data Preprocessing

Modelling

Model Evalutation

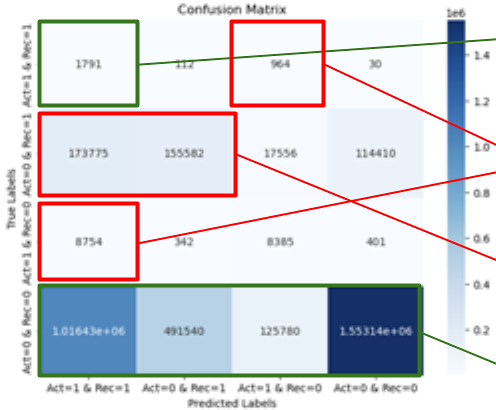
Evaluation

Evaluation on Test Dataset



Although the improvements might not seem significant (around 1~3% increase in precision), such **marginal gains can wield substantial impact**, particularly when dealing with massive datasets like the problem at hand. Note, once again positive improvements are shown in **True positives** and **False positives**.

Fig 20. Precision Score of Before & After Hyperparameter tuning



It is positive to note that the model has accuracy of **61.8%** in predicting the **true positives** correctly.

However, the limitation is that the model is confused between the two positive classes in numerous instances. → **confused between persuadables and sure things.**

Another room for improvement would be on distinguishing true negatives.

As both persuadables and lost causes take weight of 1 in our prediction phase, it is **acceptable** to be confused between the two classes

Fig 21. Confusion Matrix of Final Model

Incremental Activation Score

Methodology

1. Internal evaluation of different models using the test split.
2. To account for customers with less than 10 merchants, we calculated incremental activation score for top 3 to 10 merchants.
3. Finally, the models with highest scores were submitted to portal for actual evaluation

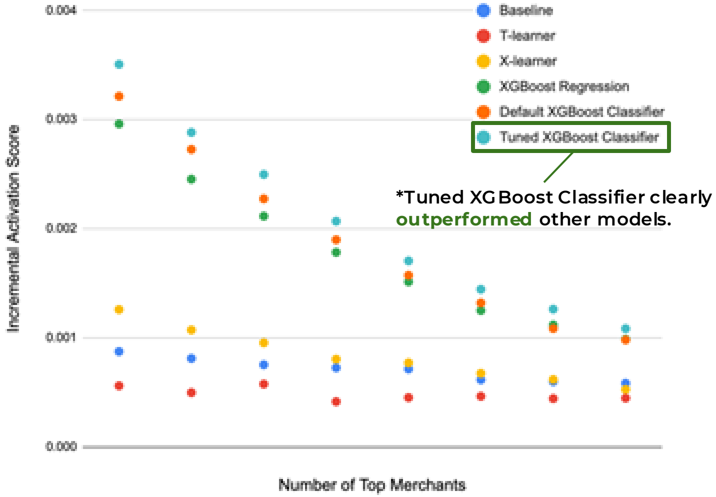


Fig 22. Incremental Activation Score on Customized testing function

*Significance of the results will be further discussed in the following conclusion.

Evaluation & Conclusion

Incremental Activation Score

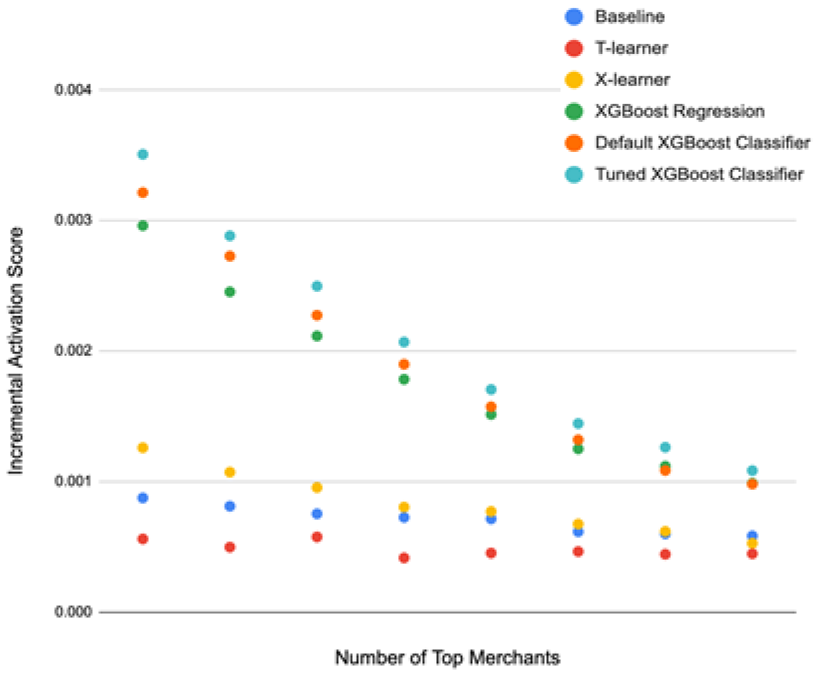


Fig 22. Incremental Activation Score on Customized testing function

Summing up, our team employed **three different approaches** to solve the uplift problem at hand:

1) Direct estimation
2) Two Model
3) Class Transformation

1) Baseline Model vs T-Learner

- Weakness of uplift signal within the dataset.
- P(Act=1 | Rec=1) = 8.11 x 10-4

2) Baseline Model vs X-Learner

- X-Learner is more powerful on imbalanced dataset.
- X-learner incorporates the treatment effect more effectively than the T-learner through cross learning meta-learners.

3) Final Model vs Other Approaches

- XGBoost: State-of-the Art Model
- Extreme gradient boosting techniques to learn complex relationships from the high dimensional dataset.

Code Explanation

Data Preprocessing

```python
def process(df):
    bool_cols = ['customer_digital_activity_04',
                 'customer_digital_activity_07',
                 'customer_digital_activity_08',
                 'customer_digital_activity_09',
                 'customer_digital_activity_10',
                 'customer_merchant_01',
                 'customer_merchant_02',
                 'merchant_spend_11']
    for col in bool_cols:
        df[col] = df[col].notnull().astype(int)
    #Replace with Mean values
    mean_cols = ['customer_digital_activity_01',
                 'customer_digital_activity_21',
                 'distance_01',
                 'distance_02',
                 'distance_03',
                 'distance_04',
                 'distance_05']
    for col in mean_cols:
        df[col].fillna(df[col].mean(), inplace=True)
    #Replace with mean of each merchants
    mean_merchants = ['merchant_profile_02',
                      'merchant_profile_03']
    merchant_avg = df.groupby('merchant')[mean_merchants].mean()
    for col in mean_merchants:
        df[col] = df[col].fillna(df['merchant'].map(merchant_avg[col]))
    for col in mean_merchants:
        df[col].fillna(df[col].mean(), inplace=True)

    #Replace with mode of each column
    mode_cols = ['customer_digital_activity_05',
                 'customer_digital_activity_06',
                 'customer_digital_activity_10',
                 'customer_digital_activity_11',
                 'customer_digital_activity_12',
                 'customer_digital_activity_13',
                 'customer_digital_activity_14',
                 'customer_digital_activity_15',
                 'customer_digital_activity_16',
                 'customer_digital_activity_17',
                 'customer_digital_activity_20']
    for col in mode_cols:
        df[col].fillna(df[col].mode()[0], inplace=True)
    #Replace NA with Zero
    df = df.fillna(0)
#Drop columns with unimportant features
df.drop(columns=['merchant_profile_01', 'customer_digital_activity_09',
         "customer_digital_activity_18", "customer_spend_17",
          "customer_digital_activity_08"], axis=1, inplace= True)

#Replace NA with Zero
df = df.fillna(0)

return df
```

Replace values with boolean

Fill na with columnwise mean values

Fill na with the mean value of each merchant

Fill na by feature mode

Drop unimportant features

Final Model: XGBClassifier

```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error

df = pd.read_csv('processed/train_dataset.csv')

conditions = [
    (df['ind_recommended'] == 1) & (df['activation'] == 1),
    (df['ind_recommended'] == 1) & (df['activation'] == 0),
    (df['ind_recommended'] == 0) & (df['activation'] == 1),
    (df['ind_recommended'] == 0) & (df['activation'] == 0)
]

classes = [0, 1, 2, 3]
df['classes'] = np.select(conditions, classes)

# Split y and drop unnecessary columns
df_train = df
y = df_train['classes']
df_train.drop(['activation', 'ind_recommended',
               'merchant', 'customer', 'classes'], axis=1, inplace=True)

# Apply OneHotEncoder for classification
encoder = OneHotEncoder()
y_encoded = encoder.fit_transform(np.array(y).reshape(-1, 1)).toarray()

# Initialize XGBClassifier with optimized parameters
model = XGBClassifier(n_estimators= 500,
    learning_rate= 0.1,
    max_depth= 5,
    min_child_weight= 1,
    gamma= 0,
    subsample= 0.9,
    colsample_bytree= 1.0,
    reg_alpha=1.0)

model.fit(df_train, y_encoded)
```

Load the processed train dataset

Set classes for each conditions

Split X and y for testing, drop unwanted columns

Use one hot encoder for encoded y

Optimized hyperparameters are used

Evaluation

```python
### Scoring function for participating teams :
def incr_act_top10(input_df: pd.DataFrame,
                   pred_col: str,
                   top: 10,
                   cm_key='customer',
                   treated_col='ind_recommended',
                   actual_col='activation'):
  #For correcting variable types
    input_df[[treated_col, actual_col, pred_col]] = input_df[[treated_col, actual_col, pred_col]].apply(pd.to_numeric, errors='coerce')
    input_df['rank_per_cm1'] = input_df.groupby(cm_key)[pred_col].rank(method='first', ascending=False)
    input_df = input_df.loc[input_df.rank_per_cm1 <= top,:]
    agg_df = input_df.groupby(treated_col,as_index=False).agg({actual_col:'mean'})
    agg_df.columns = [treated_col,'avg_30d_act']
    recommended_avg_30d_act = float(agg_df.loc[agg_df[treated_col]==1,'avg_30d_act'])
    not_recommended_avg_30d_act = float(agg_df.loc[agg_df[treated_col]==0,'avg_30d_act'])

    return (recommended_avg_30d_act-not_recommended_avg_30d_act)

output['predicted_score'] = model.predict_proba(X_test)

scores = []
for i in range(3, 11):
    scores.append(incr_act_top10(input_df = output, pred_col = 'predicted_score', top=i))
print(scores)
```

Calculate incremental activation score for top 3 merchants to top 10 merchants for each customer

```python
from sklearn.metrics import classification_report

print("Classification Report:")
print(classification_report(y_encoded, model.predict(X_train)
```

Classification Report Generation

```python
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Build confusion matrix
cm = confusion_matrix(y_train, output['predicted_score'])
print(cm)
# Visualize confusion matrix
plt.figure(figsize=(8, 6))
# sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', xticklabels=['Act=1 & Rec=1',
                                                                'Act=0 & Rec=1',
                                                                'Act=1 & Rec=0',
                                                                'Act=0 & Rec=0'],
            yticklabels=['Act=0 & Rec=0', 'Act=1 & Rec=0', 'Act=0 & Rec=1', 'Act=1 & Rec=1
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

Confusion Matrix Generation