Case Study II

Customer Response Prediction

Data Engineering

Data Engineering I -- Missing Data

1. Replace all “unknown” values with numpy.NaN
2. Calculate the percentage of NaN for each column
3. For numerical column (e.g. “custAge”), replace NaN with median value
4. For categorical column, randomly assigned values to NaN based on respective ratio in this column (e.g. for given column there are 40% ‘yes’ and 60% ‘no’. NaN in this column will be assigned ‘yes’ with probability of 40%, or ‘no’ with probability of 60%)

Data Engineering II -- Categorical Data

1. Replace NaN value if existing
2. Create dummy columns for each unique values in a given column
3. Remove the original categorical column

Data Engineering III -- Unbalanced Data

1. The label column (‘responded’) is composed of about 10% “yes” and 90% “no”, which is highly unbalanced
2. Two options: oversampling the minority class “yes” or undersampling the majority class “no”. I select the latter: undersampling
3. After undersampling, the ratio of “yes” : “no” = 4 : 6

Data Engineering IV -- Data Split

1. Split the re-balanced data to 80% training data and 20% validation data for final model selection

Modeling

Ensemble Models

1. Algorithms
    a. Random Forest
        i. each tree is a strong learning;
        ii. decrease variance by decorrelating trees and bagging
    b. Boosting (Regular Gradient Boosting, Adaboost, and XGBoost)
        i. a series of weak learner results in aggregated strong model
        ii. can achieve extremely low bias (even unbiased) model
2. Process
    a. Fixed several parameters based on experience
    b. Coarse grid searching other parameters using cross validation
    c. Finer grid searching

Metrics (from validation data)

	Accuracy	Precision	Recall	AUC (area under ROC)
Random Forest	0.7796	0.7364	0.6090	0.7928
Gradient Boost	0.7726	0.7458	0.5641	0.7957
AdaBoost	0.7726	0.7458	0.5641	0.7957
XGBoost	0.7703	0.7664	0.5256	0.7994

All 4 models perform very closely. I select XGBoost based on its largest AUC, since AUC is a more comprehensive metric over accuracy, particularly with unbalanced data and people care about the type I and type II errors. I also give rise to confusion matrix in the jupyter notebook for reference.

More thoughts about the model comparison

- In practice (e.g. Kaggle), Boosting overperform Random Forest since theoretically Random Forest is a biased classifier, while Boosting can be an unbiased classifier (at least very close to unbiased)
- Of all boosting algorithms, XGBoost is the most powerful because
  - Execution speed -- parallel processing in single tree building
    - parallelize node building at each level
    - parallelize split finding on each node
    - parallelize split finding at each level by features
  - Good model performance due to high flexibility
    - Allow customized loss objectives and evaluation criteria
  - Regularized boosting