Aggregated Machine Learning Model to Reduce Loss Function in Federated Learning Approach

Team Members : Mitodru Ghosh | Ipsita Sannyashi | Tanmoy Das | Souradeep De
Guided By: Prof. Dr. Souvik Pal

Definition: In machine learning, a loss function, also known as a cost function, is a measure of how well a model is performing with respect to its training data. The goal of a machine learning algorithm is typically to minimize this loss function, which essentially means making the predictions of the model as close as possible to the actual values. The choice of loss function depends on the task at hand and the nature of the data.

Overview of Loss Functions:

1. **Mean Squared Error (MSE)**:
   - MSE is one of the most common loss functions, especially in regression problems.
   - It calculates the average of the squares of the differences between the predicted values and the actual values.
   - Mathematically, MSE is defined as: \[\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \]
   where \(y_i\) are the actual values and \(\hat{y}_i\) are the predicted values.

2. **Binary Cross-Entropy Loss (Log Loss)**:
   - Binary cross-entropy loss is commonly used in binary classification problems.
   - It measures the dissimilarity between the true labels and the predicted probability distributions.
   - Mathematically, it is defined as: \[\text{Binary Cross-Entropy} = -\frac{1}{n} \sum_{i=1}^{n} \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)\right] \]
   where \(y_i\) are the actual binary labels (0 or 1) and \(\hat{y}_i\) are the predicted probabilities.

3. **Categorical Cross-Entropy Loss**:
   - Categorical cross-entropy loss is used in multi-class classification problems.
   - It measures the dissimilarity between the true class distributions and the predicted class distributions.
   - Mathematically, it is defined as: \[\text{Categorical Cross-Entropy} = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij}) \]
   where \(y_{ij}\) is the indicator function (1 if sample \(i\) is in class \(j\), 0 otherwise) and \(\hat{y}_{ij}\) is the predicted probability of sample \(i\) being in class \(j\).

4. **Hinge Loss**:
   - Hinge loss is commonly used in binary classification problems, especially with support vector machines (SVMs).
   - It penalizes misclassified examples proportional to the distance from the decision boundary.

- Mathematically, it is defined as: \[\text{Hinge Loss} = \frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y_i \cdot \hat{y}_i) \]
  where \(y_i\) are the actual binary labels (-1 or 1) and \(\hat{y}_i\) are the predicted scores.

5. **Huber Loss**:
   - Huber loss is a combination of MSE and MAE (Mean Absolute Error).
   - It is less sensitive to outliers compared to MSE.
   - Mathematically, it is defined as:

\[\text{Huber Loss} = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2, & \text{if } |y_i - \hat{y}_i| \leq \delta \\ \delta |y_i - \hat{y}_i| - \frac{1}{2}\delta^2, & \text{otherwise} \end{cases} \]
where \(\delta\) is a hyperparameter that controls the threshold between MSE and MAE.

Factors to Consider:

1. **Nature of the Problem**: Determine whether your problem is a binary classification, multi-class classification, or regression problem.

2. **Model Interpretability**: Some loss functions might be more suitable if interpretability is important. For example, cross-entropy loss is commonly used when dealing with probabilities and classification problems.

3. **Model Robustness**: Consider the robustness of your model to outliers. If your dataset contains outliers, a loss function like Huber loss might be more appropriate as it's less sensitive to outliers compared to MSE.

4. **Algorithm Requirements**: Certain algorithms might have inherent requirements or preferences for specific loss functions. For instance, SVMs often use hinge loss.

5. **Evaluation Metric Alignment**: Ensure that the loss function aligns with the evaluation metric you plan to use to assess the performance of your models. It's important to optimize the loss function that directly relates to the metric you care about.

In conclusion, the choice of loss function depends on various factors including the nature of the problem, model interpretability, robustness, algorithm requirements, and evaluation metric alignment. Consider these factors carefully while selecting the appropriate loss function for your strategic marketing project.

Loss Functions ideal for our Case Study:

Based on the conclusion of your project, the best loss function for your imbalanced classification task is not directly related to the specific mathematical formula used in the loss function itself.

Here's why:

- Standard loss functions like cross-entropy are not ideal for imbalanced data because they prioritize overall accuracy, neglecting the minority class (subscribers) with low representation.
- The key here is addressing the class imbalance. You achieved this through oversampling, a technique that increases the minority class samples in your dataset.

Therefore, the best approach you implemented involves a combination of:

1. Oversampling: This technique (specifically SMOTE) helped improve the model's performance on the minority class (subscribers).
2. Model Selection: AdaBoost with oversampling achieved the highest recall (78.3%) for the minority class, indicating better identification of subscribers compared to other models.

Alternative Loss Functions for Imbalanced Data:

While oversampling is a good approach, there are loss functions specifically designed for imbalanced data. Here are a couple of options to consider for future exploration:

- Weighted Cross-Entropy Loss: Assigns higher weights to the minority class, making the model pay more attention to it during training.
- Focal Loss: Downweights easy-to-classify examples, focusing the model on learning from harder examples, which can be beneficial in imbalanced settings.

Conclusion:

For your specific case, oversampling combined with AdaBoost selection proved most effective. In future projects with imbalanced data, consider exploring alternative loss functions alongside oversampling for potentially even better results.

1. Binary Cross Entropy Loss: Given the problem statement and the conclusions from your project, where the focus is on predicting whether a client will subscribe to a term deposit (a binary classification problem with imbalanced classes), the most suitable loss function to choose would be the binary cross-entropy loss.

This choice aligns well with the objective of maximizing recall for the minority class (subscribed clients) while still considering accuracy as an evaluation metric. Binary cross-entropy loss penalizes the model based on the predicted probabilities compared to the true binary labels, making it suitable for classification tasks where class imbalance exists.

Now, let's calculate the binary cross-entropy loss using the provided information. We'll use AdaBoost classifier as the selected model, as it gives the best recall score for the subscribe class when oversampled.

Assuming the AdaBoost classifier has been trained and provides predicted probabilities (\\(\\hat{y}_i\\)) for each instance:

- True labels (actual): \\(y_i\\)
- Predicted probabilities: \\(\\hat{y}_i\\)

We will then calculate the binary cross-entropy loss using the formula mentioned earlier:

$$ \text{Binary Cross-Entropy} = -\frac{1}{n} \sum_{i=1}^{n} \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)\right] $$

Here, n represents the total number of instances.

Given the actual and predicted probabilities, we would compute this loss over the entire dataset.

2. Weighted Binary Cross-Entropy Loss: Given that the focus is on predicting whether a client will subscribe to a term deposit, and that recall for the minority class (subscribed clients) is more important than precision, and accuracy is still considered as an evaluation metric, the choice of loss function should indeed align with these objectives.

Considering this, the appropriate loss function would be the weighted binary cross-entropy loss. In imbalanced classification problems like this, where one class is significantly smaller than the other, it's common to use weighted loss functions to give more importance to the minority class.

In weighted binary cross-entropy, you assign different weights to each class based on their proportion in the dataset. The weight for the minority class (class 1, subscribed clients) would typically be higher to reflect its importance.

The formula for weighted binary cross-entropy loss is similar to binary cross-entropy, but it includes weights for each class:

$$ \text{Weighted Binary Cross-Entropy} = -\frac{1}{n} \sum_{i=1}^{n} \left[w_1 y_i \log(\hat{y}_i) + w_0 (1 - y_i) \log(1 - \hat{y}_i)\right] $$

where w_1 is the weight for class 1 (subscribed clients) and w_0 is the weight for class 0 (not subscribed clients), and n represents the total number of instances.

Given the class proportions (11.7% subscribed, 88.3% not subscribed), you can assign weights accordingly. Typically, the weight for the minority class would be higher than the weight for the majority class.

Now, we can calculate the weighted binary cross-entropy loss using the provided information and the predicted probabilities from the AdaBoost classifier.

Ensemble Techniques

Data Description: The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to assess if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

Domain: Banking

Context: Leveraging customer information is paramount for most businesses. In the case of a bank, attributes of customers like the ones mentioned below can be crucial in strategizing a marketing campaign when launching a new product.

Attribute Information

- **age**: age at the time of call
- **job**: type of job
- **marital**: marital status
- **education**: education background at the time of call
- **default**: has credit in default?
- **balance**: average yearly balance, in euros (numeric)
- **housing**: has a housing loan?
- **loan**: has a personal loan?
- **contact**: contact communication type
- **day**: last contact day of the month (1 -31)
- **month**: last contact month of year ('jan', 'feb', 'mar', ..., 'nov', 'dec')
- **duration**: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration = 0 then Target = 'no'). Yet, the duration is not known before a call is performed. Also, after the end of the call Target is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.
- **campaign**: number of contacts performed during this campaign and for this client (includes last contact)
- **days**: number of days that passed by after the client was last contacted from a previous campaign
- **previous**: number of contacts performed before this campaign and for this client
- **outcome**: outcome of the previous marketing campaign
- **target**: has the client subscribed a term deposit? ('yes', 'no')

Learning Outcomes

- Exploratory Data Analysis
- Preparing the data to train a model
- Training and making predictions using an Ensemble Model
- Tuning an Ensemble model

……..

Table of Contents:-

        - Checking the distribution of target variable
            - Observation 4 - Distribution of target variable
        - Univariate and Bivariate Visualization
            - Observation 5 - Comments from categorical columns
        - Check outlier and distribution for numerical columns and also plot it's relation with target variable
            - Observation 6 - Comments from numerical columns
        - Print categorical and numerical columns list, remove outliers with upper and lower percentile values being 99 and 1, respectively and get dummies
        - Use MICE imputer to handle outliers that were filled with in the earlier step
            - Observation 7 - Observation after MICE
        - Checking whether count of 0 in previous column is equal to count of -1 in days column
            - Observation 8 - days and previous
        - Multivariate visualization
            - Check scatter matrix
            - Correlation matrix
                - Observation 9 - Correlation Matrix
        - Creating age groups and check relation with balance and target; also with campaign and target
            - Observation 10 - Comments
- Modeling
    - Dummy Classifier -- Baseline Model
    - Standard Models
        - Logistic Regression
        - k-Nearest Neighbor Classifier
        - Naive Bayes Classifier
        - Oversampling and Naive Bayes
        - Oversampling and Logistic Regression
    - Ensemble Techniques
        - Decision Tree Classifier
        - Bagging, AdaBoost, Gradient Boosting Classifier
        - Oversampling and AdaBoost Classifier
        - Random Forest Classifier
        - Oversampling and Random Forest Classifier
- Comparing model results
- Conclusion

Import Packages:-

This code cell imports the drive module from the google.colab library, which is used in Google Colab notebooks for accessing and interacting with Google Drive.

The drive.mount('/content/gdrive') line mounts your Google Drive to the Colab notebook environment. This allows you to access files stored in your Google Drive from within the notebook.

When you run this cell, it will prompt you to authorize access to your Google Drive. After authorization, your Google Drive will be mounted at the specified location (/content/gdrive), and you'll be able to navigate and interact with your files using the notebook.

This code cell imports several Python packages and sets up the environment for data analysis and modeling tasks. Let's break it down:

1. **Basic Packages Installation**:
    - !pip install impyute: Installs the impyute package, which is used for imputation (filling in missing values) in datasets.
2. **Import Statements**:
    - import pandas as pd, numpy as np, matplotlib.pyplot as plt, seaborn as sns: Imports the Pandas library for data manipulation, NumPy for numerical computations, Matplotlib and Seaborn for data visualization.
    - from scipy import stats; from scipy.stats import zscore, norm, randint: Imports statistical functions from SciPy for data analysis.
    - import matplotlib.style as style; style.use('fivethirtyeight'): Sets the style of Matplotlib plots to 'fivethirtyeight'.
    - import plotly.express as px: Imports Plotly Express for interactive visualizations.
    - %matplotlib inline: Configures Matplotlib to display plots inline in the notebook.
3. **Impute and Encode**:
    - from sklearn.preprocessing import LabelEncoder: Imports the LabelEncoder class from scikit-learn, which is used for encoding categorical variables.
    - from impyute.imputation.cs import mice: Imports the mice function from the impyute.imputation.cs module, which is used for Multiple Imputation by Chained Equations (MICE) imputation.
4. **Modeling - Algorithms and Metrics**:
    - Imports various machine learning models and evaluation metrics from scikit-learn for classification tasks:
        - from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, f1_score, recall_score, precision_score: Metrics for evaluating classification models.
        - Model classes:
            - from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier, BaggingClassifier, VotingClassifier: Ensemble models like Gradient Boosting, Random Forest, Bagging, and Voting classifiers.
            - from sklearn.linear_model import LogisticRegression: Logistic Regression model.
            - from sklearn.neighbors import KNeighborsClassifier: k-Nearest Neighbors (KNN) classifier.
            - from sklearn.tree import DecisionTreeClassifier: Decision Tree classifier.
            - from sklearn.naive_bayes import GaussianNB: Gaussian Naive Bayes classifier.
            - from sklearn.dummy import DummyClassifier: Dummy classifier for baseline comparison.
        - Utility functions and classes:
            - from sklearn.metrics import make_scorer: Function for creating custom scoring metrics.
        - from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold, cross_val_score: Classes and functions for model evaluation and hyperparameter tuning.
5. **Oversampling**:

    - from imblearn.over_sampling import SMOTE: Imports the Synthetic Minority Over-sampling Technique (SMOTE) from the imblearn library for dealing with class imbalance.
6. **Suppress Warnings**:
    - import warnings; warnings.filterwarnings('ignore'): Suppresses warnings to avoid cluttering the output.
7. **Visualize Tree**:
    - from sklearn.tree import export_graphviz: Imports the export_graphviz function from scikit-learn for visualizing decision trees.
    - from IPython.display import Image: Imports the Image class from IPython.display for displaying images.
    - from os import system: Imports the system function from the os module for executing system commands.
8. **Display Settings**:
    - pd.options.display.max_rows = 10000: Sets the maximum number of rows displayed in Pandas DataFrame to 10000.
    - pd.options.display.max_columns = 10000: Sets the maximum number of columns displayed in Pandas DataFrame to 10000.
9. **Random Seed Initialization**:
    - random_state = 42: Sets the random seed to ensure reproducibility of results.
    - np.random.seed(random_state): Initializes the random number generator of NumPy with the specified random seed.

Overall, this code cell sets up the necessary environment for data analysis, preprocessing, modeling, and evaluation tasks in a machine learning project.

Observation 1 - Dataset shape

Dataset has 45211 rows and 17 columns, with no missing values.

Exploratory Data Analysis

Performing exploratory data analysis on the bank dataset. Below are some of the steps performed:

- Get descriptive statistics including five point summary
- Check unique values in object columns
- Check distribution of Target column
- Count plot by Target for categorical columns (job, marital, education, default, housing, loan, contact, day, month, outcome)
- Check outliers in each numerical columna (age, balance, duration, campaign, pdays, previous)
- Distribution of numerical columns (with and without outliers)
- Distribution of numerical columns for Target values (subscribed and didn`t subscribed to term deposit)
- Fill outliers with upper and lower percentile values being 99 and 1, respectively by nan values
- Frequency encoding for categorical columns with string values
- Convert type of categorical columns (job, marital, education, default, housing, loan, contact, day, month, outcome, Target) to float for MICE training. Creating multiple imputations, as opposed to single imputations to complete datasets, accounts for the statistical uncertainty in the imputations.

MICE algorithms work by running multiple regression models and each missing value is modeled conditionally depending on the observed (non-missing) values.
- Get correlation matrix and check absolute correlation of independent variables with Target. Drop columns based on these.
- Create age groups and check if there is relation with balance and target, campaign and target.

Observation 2 - information on the type of variable and min-max values

Client info

- **Categorical**
  - **job**: Nominal. Includes type of job. 'blue-collar' is the most frequently occurring in the data.
  - **marital**: Nominal. Most of the clients are married in the dataset we have.
  - **education**: Ordinal. Most of the clients have secondary level education.
  - **default**: Binary. Most clients don't have credit in default.
  - **housing**: Binary. Most of the clients have housing loans.
  - **loan**: Binary. Most of the clients don't have personal loans.
- **Numerical**
  - **age**: Continuous ratio (has true zero, technically). Whether it's discrete or continuous depends on whether they are measured to the nearest year or not. At present, it seems it's discrete. Min age in the dataset being 18 and max being 95.
  - **balance**: Continuous, ratio. Range of average yearly balance is very wide from -8019 euros to 102127 euros.

Last contact info

- **Categorical**
  - **contact**: Nominal. Includes communication type with the client, most frequently used communication mode is cellular.
  - **day**: Ordinal. Includes the last contact day of the month.
  - **month**: Ordinal. Includes the last contact month of the year.
- **Numerical**
  - **duration**: Continuous, interval. Includes last contact duration in seconds. Min value being 0 and max value being 4918. It would be important to check if a higher duration of call leads to more subscriptions.

This campaign info

- **Numerical**
  - **campaign**: Discrete, interval. Min number of contacts performed during this campaign being 1 and it also represents about 25% of the value and max being 63.

Previous campaign info

- **Categorical**
  - **outcome**: Nominal. Includes outcome of the previous marketing campaign. Most occuring value being 'unknown'.
- **Numerical**
  - **pdays**: Continuous, interval. Min number of days that passed by after the client was last contacted from a previous campaign being -1 which may be dummy value for the cases where client wasn't contacted and max days being 63.

    - **previous**: Discrete, ratio. Min number of contacts performed before this campaign is 0 and max being 275.

Target

- **Categorical**
    - **Target**: Binary. Most occurring value being 'no' i.e. cases where the client didn't subscribe to the term deposit.

Observation 3 - Descriptive statistics for the numerical variables

Descriptive statistics for the numerical variables (age, balance, duration, campaign, pdays, previous)

- **age**: Range of Q1 to Q3 is between 33 to 48. Since mean is slightly greater than median, we can say that age is right (positively) skewed.
- **balance**: Range of Q1 to Q3 is between 72 to 1428. Since mean is greater than median, we can say that balance is skewed towards right (positively).
- **duration**: Range of Q1 to Q3 is between 103 to 319. Since mean is greater than median, we can say that duration is right (positively) skewed.
- **campaign**: Range of Q1 to Q3 is between 1 to 3. Since the mean is greater than median, we can say that the campaign is right (positively) skewed.
- **pdays**: 75% of data values are around -1 which is a dummy value. It needs further checking without considering the -1 value.
- **previous**: 75% of data values are around 0 which is a dummy value, maybe cases where the client wasn't contacted. It needs further checks.

Observation 4 - Distribution of target variable

Out of 45211 cases, only 5289 (=11.69%) are the cases where the client has subscribed to the term deposit.

Univariate and Bivariate Visualization

Looking at one feature at a time to understand how the values are distributed, checking outliers, checking relation of the column with Target column (bi).

Observation 5 - Comments from categorical columns

- Management has a subscription rate of ~25 percent followed by technicians.
- Married and single clients are more likely to subscribe then divorced clients
- Clients with education of secondary followed by tertiary are more likely to subscribe to term deposits
- Most of the clients don't have credit in default and their subscription rate is higher then people with default
- Cellular communication type have higher subscription rate
- Most of the subscription were made in May and August

Check outlier and distribution for numerical columns and also plot it's relation with target variable

Observation 6 - Comments from numerical columns

- Used quantile method to check outliers in numerical columns. It appears that there are outliers in each of the numerical columns.
- It appears that removing outliers below 25% percentile and above 75% percentile will bring the age and pdays columns to almost normal distribution.

Print categorical and numerical columns list, remove outliers with upper and lower percentile values being 99 and 1, respectively and get dummies

Use MICE imputer to handle outliers that were filled with np.nan in the earlier step

Observation 7 - Observation after MICE

Column	Before MICE	After MICE
age	Range of Q1 to Q3 is 33-48. Mean > Median, right (positively) skewed	Range of Q1 to Q3 is unchanged, because of change in min and max values there's a slight reduction is mean, right skewed
balance	Range of Q1 to Q3 is 72-1428. Mean > Median, skewed towards right (positively)	Range of Q1 to Q3 is 81 to 1402, reduction in mean, right skewed
duration	Range of Q1 to Q3 is 103-319. Mean > Median, right (positively) skewed	Range of Q1 to Q3 is 106-316, right skewed
campaign	Range of Q1 to Q3 is 1-3. Mean > Median, right (positively) skewed	Unchanged range and skewness
days	75% of data values are around -1	Unchanged

previous	75% of data values are around 0	Unchanged

Checking whether count of 0 in previous column is equal to count of -1 in days column

Observation 8 - days and previous

Count of 0 in previous is equal to count of -1 in the pdays column, we might replace -1 in pdays with 0 to account for cases where the client wasn't contacted previously. Checking correlation between variables and target next…

Multivariate visualization

Checking relationships between two or more variables. Includes correlation and scatterplot matrix, checking relation between two variables and Target.

Observation 9 - Correlation Matrix

- poutcome_unknown and pdays; contact_unknown and contact_cellular; poutcome_unknown and previous; marital_married and marital_single; poutcome_unknown and poutcome_failure; pdays and poutcome_failure; previous and pdays; poutcome_failure and previous columns are correlated with each other.
- duration, poutcome_success, poutcome_unknown and previous are a few columns which have a relatively strong correlation with the Target column.

Observation 10 - Comments

- Created age_group and checked its relation with balance and target and it appears that higher the balance range more are the chances that the client would subscribe to the term deposit irrespective of age group. It also appears that clients within age group 50 have the highest range of balance.
- Then checked the relation between campaign, age group and target and it appears that campaigns for clients with age group 20 and 60 are less.

Modeling

- Create a baseline model
- Use different classification models (Logistic, K-NN and Naïve Bayes) to predict whether the client subscribes to term deposit.
- Training and making predictions using an Ensemble Model.

Logistic Regression, kNN and Naive Bayes

Oversampling the one with better accuracy and recall score for subscribe

Ensemble Techniques Used: Decision Tree Classifier, Bagging Classifier, AdaBoost Classifier, Gradient Boosting Classifier and Random Forest Classifier. Oversampling the ones with higher accuracy and better recall for subscribers.

Conclusion

The classification goal is to predict if the client will subscribe (yes/no) to a term deposit.

Most of the ML models work best when the number of classes are in equal proportion since they are designed to maximize accuracy and reduce error. Thus, they do not take into account the class distribution / proportion or balance of classes. In our dataset, the clients subscribing to term deposit (class 'yes' i.e. 1) is 11.7% whereas those about 88.3% of the clients didn't subscribe (class 'no' i.e. 0) to the term deposit.

Building a DummyClassifier, baseline model, in our case gave an accuracy of 88.2% with zero recall and precision for predicting minority class i.e. where the client subscribed to term deposits. In this case, important performance measures such as precision, recall, and f1-score would be helpful. We can also calculate these metrics for the minority, positive, class.

- **Precision**: When it predicts the positive result, how often is it correct? i.e. limit the number of false positives.
- **Recall**: When it is actually the positive result, how often does it predict correctly? i.e. limit the number of false negatives.
- **f1-score**: Harmonic mean of precision and recall.

The confusion matrix for class 1 (Subscribed) would look like:

	Predicted: 0 (Not Subscribed)	**Predicted: 1 (Subscribed)**
Actual: 0 (Not Subscribed)	True Negatives	False Positives
Actual: 1 (Subscribed)	False Negatives	True Positives

- **Precision would tell us cases where actually the client hadn't subscribed to the term deposit but we predicted it as subscribed.**
- **Recall would tell us cases where actually the client had subscribed to the term deposit but we predicted it as didn't subscribe.**

In our case, it would be recalled that would hold more importance than precision. So choosing recall particularly for class 1 and accuracy as as evaluation metric. Also important would be how is model behaving over the training and test scores across the cross validation sets.

Modeling was subdivided in two phases, in the first phase we applied standard models (with and without the hyperparameter tuning wherever applicable) such as Logistic Regression, k-Nearest Neighbor and Naive Bayes classifiers. In the second phase apply ensemble techniques such as Decision Tree, Bagging, AdaBoost, Gradient Boosting and Random Forest classifiers. Oversampling the ones with higher accuracy and better recall for subscribers.

Oversampling, which is one of the common ways to tackle the issue of imbalanced data. Over-sampling refers to various methods that aim to increase the number of instances from the underrepresented class in the data set. Out of the various methods, we chose Synthetic Minority Over-Sampling Technique (SMOTE). SMOTE's main advantage compared to traditional random naive over-sampling is that by creating synthetic observations instead of reusing existing observations, the classifier is less likely to overfit.

In the first phase (Standard machine learning models vs baseline model),

- Best recall for minority class in Naive Bayes of 52.1%
- However better accuracy of 89.9% was observed in logistic regression with hyperparameter tuning
- Oversampling both, recall of Naive Bayes increases but accuracy drops significantly whereas oversampled logistic regression with hyperparameter tuned model out of which we found a significant improvement of 78.4% in the recall for subscribe against the baseline model, however the accuracy dropped by 5.4%. But again, it's recall that's more important here.

In the second phase (Ensemble models vs baseline model),

- Decision tree classifier gives the best recall score for subscribe class but is also prone to overfitting across the cross validation set. Reducing the max depth did solve the problem of overfitting, but resulted in lower recall subscribe.
- AdaBoost classifier gave the best recall when compared with decision tree reduced max depth, bagging and gradient boosting classifier. Thus, oversampled that which gave a recall of 78.3% for the subscribe (yes) class. Best till now among ensembles.
- Tried different methods with random forest i.e. hyperparameter tuning, oversampling and even reducing max depth.. but recall score didn't improve when compared with AdaBoost classifier when oversampled.
- Thus two better models from this phase are AdaBoost classifier when oversampled with a recall of 78.3%, accuracy of 81.1% and Random Forest classifier when hyperparameter tuned and oversampled, max depth reduced with recall of 78% and accuracy of 81.2%
- **AdaBoost's recall for subscribe is an improvement of 78.3% from the baseline model while the accuracy did reduce. It's a tradeoff we have to deal with.**

Important Glossary

A. Helper Function in Machine Learning: In the context of machine learning, a helper function refers to a small, reusable piece of code designed to perform a specific task or assist in a particular aspect of the machine learning workflow. Helper functions are used to encapsulate commonly performed tasks, reduce code redundancy, improve code readability, and facilitate code reuse across different parts of a machine learning project.

Here are some common tasks for which helper functions are often created in machine learning projects:

1. Data Preprocessing: Helper functions can be created to handle tasks such as data cleaning, feature scaling, feature extraction, encoding categorical variables, handling missing values, and splitting datasets into training and testing sets.

2. Model Training and Evaluation: Helper functions can be used to streamline the process of training machine learning models, including tasks such as fitting models to training data, making predictions on test data, evaluating model performance using metrics such as accuracy, precision, recall, F1-score, ROC-AUC, etc., and visualizing evaluation results using plots such as confusion matrices, ROC curves, or precision-recall curves.

3. Hyperparameter Tuning: Helper functions can assist in performing hyperparameter tuning tasks, such as defining hyperparameter search spaces, conducting grid search or randomized search, selecting the best hyperparameter configuration based on cross-validation performance, and visualizing hyperparameter tuning results.

4. Model Interpretability: Helper functions can aid in interpreting machine learning models by providing insights into feature importance, generating partial dependence plots, SHAP (SHapley Additive exPlanations) values, or LIME (Local Interpretable Model-agnostic Explanations) explanations.

5. Ensembling and Stacking: Helper functions can be useful for implementing ensemble learning techniques such as bagging, boosting, or stacking, by combining multiple base models to improve predictive performance.

6. Deployment: In some cases, helper functions may also assist in deploying machine learning models into production environments, including tasks such as model serialization, building APIs for model inference, or integrating models into web applications.

By encapsulating these tasks into reusable helper functions, machine learning practitioners can write cleaner, more modular code, reduce the likelihood of errors, and improve the efficiency of their workflow. Additionally, helper functions promote code maintainability and make it easier for team members to collaborate on machine learning projects.

B. Outliers in Machine Learning: In machine learning, outliers refer to data points that significantly differ from other observations in the dataset. These data points may be unusually high or low compared to the rest of the data and can have a substantial impact on the analysis or modeling process if not properly handled. Outliers can occur due to various reasons, including measurement errors, experimental errors, or natural variations in the data.

Outliers can affect machine learning models in several ways:

1. Impact on Statistical Analysis: Outliers can skew statistical measures such as the mean and standard deviation, leading to biased estimates of central tendency and variability.

2. Impact on Model Performance: Outliers can distort the relationship between variables and affect the performance of machine learning models. For example, outliers can influence the coefficients in linear regression models, leading to incorrect parameter estimates.

3. Robustness of Models: Some machine learning algorithms are sensitive to outliers and may produce suboptimal results if outliers are present in the dataset. For example, clustering algorithms like K-means can be heavily influenced by outliers, leading to the formation of clusters that do not accurately represent the underlying data distribution.

4. Generalization: Outliers can also affect the generalization ability of machine learning models by introducing noise into the training process. Models trained on datasets with outliers may not generalize well to unseen data, leading to poor performance on real-world applications.

Handling outliers in machine learning involves various techniques:

1. Detection: The first step in handling outliers is to identify them in the dataset. This can be done using statistical methods, such as z-score or IQR (Interquartile Range), or visualization techniques, such as box plots or scatter plots.

2. Removal: In some cases, outliers can be removed from the dataset if they are the result of measurement errors or data anomalies. However, it's essential to exercise caution when removing outliers, as doing so can lead to information loss and bias in the data.

3. Transformation: Another approach is to transform the data to reduce the impact of outliers. For example, logarithmic or square root transformations can help stabilize the variance in the data and make it more normally distributed.

4. Robust Algorithms: Using machine learning algorithms that are robust to outliers can also mitigate their impact on model performance. Algorithms such as decision trees, random forests, and support vector machines are less sensitive to outliers compared to linear models like linear regression or logistic regression.

5. Model Ensembles: Ensemble methods, such as bagging and boosting, can also help improve the robustness of models to outliers by combining multiple weak learners to make more accurate predictions.

Overall, handling outliers is an essential aspect of the data preprocessing pipeline in machine learning, and careful consideration should be given to their detection and treatment to ensure the robustness and generalization of the models.

C. MICE and SMOTE Function in Machine Learning: MICE (Multiple Imputation by Chained Equations) and SMOTE (Synthetic Minority Over-sampling Technique) are two different techniques used in machine learning for handling specific challenges in data preprocessing, specifically related to missing data and imbalanced datasets, respectively.

1. MICE (Multiple Imputation by Chained Equations): MICE is a method used to impute missing values in a dataset by iteratively imputing each variable's missing values based on the observed values of other variables in the dataset. It's a popular technique for handling missing data because it can handle different types of variables (categorical, numerical) and preserves the relationships between variables.

Here's how MICE works:

- In the first step, missing values in each variable are replaced with some initial guess (often the mean or median of the observed values for numerical variables, or the mode for categorical variables).

- Then, each variable with missing values is imputed one at a time, using regression models (e.g., linear regression for numerical variables, logistic regression for binary variables) that predict the missing values based on the observed values of other variables in the dataset.

- This process is repeated for multiple iterations until convergence, with each variable being imputed multiple times, and the imputed values are updated in each iteration based on the most recent imputations for other variables.

MICE is useful when dealing with datasets with missing values because it provides a flexible and principled approach for imputing missing data, allowing for uncertainty estimation and preserving the variability in the dataset.

2. SMOTE (Synthetic Minority Over-sampling Technique): SMOTE is a technique used to address class imbalance in a dataset, particularly in binary classification problems where one class (the minority class) is significantly underrepresented compared to the other class (the majority class). Imbalanced datasets can lead to biased models that perform poorly on the minority class.

Here's how SMOTE works:

- SMOTE generates synthetic examples for the minority class by interpolating between existing minority class instances in the feature space.

- For each minority class instance, SMOTE identifies its k nearest neighbors (often using Euclidean distance) in the feature space.

- Synthetic examples are then created by selecting random points along the line segments connecting each minority class instance to its k nearest neighbors and adding a random amount of the difference to the original instance's feature values.

- This process is repeated until the desired balance between the minority and majority classes is achieved.

SMOTE helps in balancing the class distribution, allowing the model to learn from sufficient examples of the minority class and make more accurate predictions. It's a popular technique for improving the performance of classifiers, especially in scenarios where class imbalance is prevalent.

D. Hyperparameter Tuning and Oversampling in Machine Learning: Hyperparameter tuning and oversampling are both techniques used in machine learning to improve the performance of models, especially in scenarios where imbalanced datasets or suboptimal model configurations can affect results. Here's an overview of each:

1. Hyperparameter Tuning: Hyperparameters are parameters that are set before the learning process begins. They control aspects of the learning process such as the complexity of the model, the learning rate, the number of trees in a random forest, etc. Unlike model parameters, which are learned during training, hyperparameters are typically set manually before training begins or are selected through optimization techniques.

Hyperparameter tuning, also known as hyperparameter optimization or search, is the process of finding the best hyperparameters for a machine learning model. This process involves searching through a defined hyperparameter space and selecting the combination of hyperparameters that results in the best performance on a validation dataset. Common techniques for hyperparameter tuning include grid search, random search, Bayesian optimization, and more recently, automated machine learning (AutoML) approaches.

The goal of hyperparameter tuning is to find the optimal configuration of hyperparameters that maximizes the performance of the model on unseen data, ultimately leading to better generalization and improved predictive accuracy.

2. Oversampling: Oversampling is a technique used to address class imbalance in a dataset. Class imbalance occurs when one class (the minority class) is significantly underrepresented compared to the

other classes (the majority class or classes). Imbalanced datasets can lead to biased models that perform poorly on the minority class or classes.

In oversampling, the instances of the minority class are artificially increased by either replicating existing instances or generating synthetic examples. The goal is to balance the class distribution, allowing the model to learn from sufficient examples of the minority class and make more accurate predictions.

Common oversampling techniques include:

- Random Oversampling: Randomly duplicating instances of the minority class until the class distribution is balanced.

- SMOTE (Synthetic Minority Over-sampling Technique): SMOTE generates synthetic examples for the minority class by interpolating between existing instances, effectively creating new samples along the lines connecting similar instances in the feature space.

- ADASYN (Adaptive Synthetic Sampling): ADASYN is an extension of SMOTE that focuses on generating synthetic examples in regions of the feature space where the density of minority class instances is low.

Oversampling helps improve model performance by ensuring that the model is exposed to a more balanced representation of the data, thereby reducing bias towards the majority class and improving the model's ability to generalize to unseen data. However, it's essential to be cautious with oversampling techniques to avoid overfitting or introducing bias into the model. Cross-validation and careful evaluation are crucial when using oversampling methods.

E. EDA in Machine Learning: Exploratory Data Analysis (EDA) is a critical phase in the process of building machine learning models. It involves examining and understanding the structure, patterns, and relationships in the data before applying any modeling techniques. EDA helps in uncovering insights, identifying anomalies, and making informed decisions about data preprocessing and feature engineering. Below is a detailed overview of EDA in machine learning:

1. Data Collection: The first step in EDA is gathering the data from various sources, which may include databases, files, APIs, or scraping from the web. It's essential to collect both the feature variables (inputs) and the target variable (output) if it's a supervised learning problem.

2. Data Cleaning: Raw data often contains errors, missing values, outliers, or inconsistencies, which can adversely affect the model's performance. Data cleaning involves tasks like handling missing values (imputation), removing duplicates, correcting errors, and dealing with outliers.

3. Statistical Summary: It's crucial to get an overview of the data through statistical summaries. Descriptive statistics such as mean, median, mode, standard deviation, minimum, maximum, and quartiles provide insights into the central tendency, dispersion, and shape of the data distribution.

4. Data Visualization: Visualization plays a significant role in EDA as it helps in intuitively understanding the data. Various plots like histograms, box plots, scatter plots, pair plots, and correlation matrices are used to visualize distributions, relationships between variables, and identify patterns.

5. Univariate Analysis: Univariate analysis focuses on exploring individual variables in the dataset. For categorical variables, frequency tables, bar plots, and pie charts are used to visualize the distribution of categories. For numerical variables, histograms, density plots, and box plots help understand the distribution and identify outliers.

6. Bivariate Analysis: Bivariate analysis examines the relationship between pairs of variables. Scatter plots are commonly used to visualize the relationship between two numerical variables. For categorical variables, stacked bar plots and contingency tables are used to explore dependencies.

7. Correlation Analysis: Correlation analysis measures the strength and direction of the linear relationship between two numerical variables. Correlation coefficients such as Pearson's correlation, Spearman's rank correlation, or Kendall's tau are computed, and correlation matrices are visualized to identify highly correlated variables.

8. Feature Engineering: EDA helps in generating new features or transforming existing ones based on insights gained from data exploration. Feature engineering aims to improve the model's performance by creating more informative representations of the data.

9. Outlier Detection: Outliers are data points that significantly deviate from the rest of the data. EDA helps in identifying outliers through visualization techniques or statistical methods such as z-score, IQR (Interquartile Range), or Tukey's method.

10. Dimensionality Reduction: High-dimensional data can be challenging to visualize and analyze. Dimensionality reduction techniques such as Principal Component Analysis (PCA) or t-distributed Stochastic Neighbor Embedding (t-SNE) can be applied to visualize high-dimensional data in lower dimensions while preserving its structure.

11. Conclusion and Insights: Finally, EDA concludes with summarizing insights gained from the exploration process. These insights guide further steps in the machine learning pipeline, such as selecting appropriate models, feature selection, and model evaluation.

Overall, EDA is a critical stage in the machine learning workflow, laying the foundation for building robust and effective predictive models. It enables data scientists to understand the data better, make informed decisions, and improve the quality of machine learning models.

F. Fancyimpute in Machine Learning: Fancyimpute is a Python library designed for imputing missing values in datasets, primarily in the context of machine learning and data analysis. Imputation refers to the process of replacing missing data with estimated values based on the available information.

Fancyimpute provides various sophisticated imputation techniques beyond simple methods like mean or median imputation. Some of the techniques implemented in Fancyimpute include:

1. Matrix Factorization: Methods like Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF) are used to decompose the data matrix and fill in missing values based on the decomposed matrices.

2. K-Nearest Neighbors (KNN) Imputation: This technique imputes missing values by finding the K nearest neighbors to each data point with missing values and then averaging or taking weighted averages of the neighbors' values.

3. Iterative Imputation: Fancyimpute implements iterative imputation techniques like Iterative Singular Value Thresholding (SoftImpute) and Matrix Completion by Alternating Least Squares (ALS) to estimate missing values iteratively based on the observed data.

4. Deep Learning Imputation: This involves using deep learning models, such as autoencoders, to learn patterns from the observed data and generate imputations for missing values.

These advanced techniques aim to provide more accurate imputations and preserve the underlying structure of the data while handling missing values. Fancyimpute is commonly used in data preprocessing pipelines before applying machine learning algorithms that cannot handle missing data.