AWS Machine Learning Engineer Nanodegree Capstone Project
Arvato Customer Acquisition Prediction Using Supervised Learning

Fady Morris Milad[†]

February 22, 2022

[†]Email: fadymorris86@gmail.com

Contents

1 Definition

1.1 Project Overview

ARVATO is a company that provides financial services, IT services, and supply chain management services solutions to other businesses. It has a large base of global customers. Its solutions focus on automation and data analytics.[KGaa]

Arvato's customer base comes from a wide variety of businesses, such as insurance companies, telecommunications, media education, and e-commerce.

Arvato analytics is helping businesses to take important decisions and gain insights from data. It uses data science and machine learning to meet business goals and gain customer satisfaction.

Arvato is owned by Bertelsmann [KGab], which is a media, services, and education company that operates in about 50 countries around the world.

In this project, Arvato is helping a mail-order company that sells organic products in Germany to build a model to predict which individuals are most likely to convert into becoming customers for the company by analyzing marketing campaign mailout data.

Customer retention and churn were addressed in the following academic research papers: [AF20] and [Zhu18]

We have a proprietary dataset that contains 42,982 training samples and 42,833 testing samples. Both the datasets have 366 features and the testing set doesn't have the label column.

We will use Amazon SageMaker to train a machine learning model using the training set, and then generate predictions using the testing set. We will validate the generated predictions using Kaggle API which has a dedicated competition for this problem.

1.2 Problem Statement

The problem can be stated as: "Given the existing marketing campaign demographic data of customers who responded to marketing mails, how can we predict whether a new person will be a potential customer for the mail-order company?"

Here we use *XGBoost*, which is a supervised learning algorithm, and train a model that will help the company make such predictions and decide whether is a person is a potential candidate to be a customer for the company or not.

1.3 Metrics

Our problem is a binary classification problem. Our classes are labeled either 0 (the individual is not a customer) or 1 (the individual became a customer), and the dataset is imbalanced as shown in Table 2 - Label Counts and Frequencies.

To measure how close our predictions to the ground truth labels, we will use two classification metrics: *ROC-AUC score* and *log-loss score*.

1.3.1 ROC-AUC Score

ROC-AUC stands for the **R**eceiver **O**perating **C**haracteristic **A**rea **U**nder **C**urve. This is the metric used by the Kaggle Competition to evaluate the test data.

The receiver operator characteristic (ROC) curve is a plot of the *true positive rate*(TPR) against the *false positive rate* at various threshold values.

The area under ROC curve is a metric that measures the ability of the classifier to distinguish between positive and negative classes.

ROC-AUC Metric interpretation:

- **AUC = 1**: The classifier perfectly separates the positive and the negative classes.
- **0.5 < AUC < 1**: The classifier can distinguish between negative and positive classes with a high probability. The higher is better.
- **AUC = 0.5**: is not different from a random guess. The classifier is unable to distinguish between classes.
- **AUC = 0**: The classifier predicts all negatives as positives and vice versa.

1.3.2 Log-Loss

Log-loss is a metric that indicates how close the prediction probability is to the corresponding actual value (0 or 1 in our case). The more the predicted probability diverges from the actual value, the higher is the log-loss value.

The log-loss function $\mathcal{L}$ can be defined by the following equation:

$$\mathcal{L}\left(p^{(i)}, y^{(i)}\right) = \frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(p^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-p^{(i)}\right)\right]$$

Where:

$y^{(i)}$: The actual (ground truth) label for the i^{th} training example.

$p^{(i)}$: The predicted probability of class 1 for the i^{th} training example.

In our project, the log-loss objective metric is used by Amazon SageMaker hyperparameter tuning to find the best combination of hyperparameters that minimize the loss function.

2 Analysis

2.1 Data Exploration

The dataset is a private dataset. It is used with permission from Arvato for use in the nanodegree project.

2.1.1 Raw Dataset Files

There are two files of the dataset that we are concerned with:

- `Udacity_MAILOUT_052018_TRAIN.csv`: Demographic data for people in Germany that were targeted by the marketing mailing campaign. It contains data for 42,982 individuals.

  This training dataset has 367 columns, 366 of them are demographic features, and one label column `['RESPONSE']`
- `Udacity_MAILOUT_052018_TEST.csv`: The testing dataset, contains data for 42,833 individuals and it has 366 columns of the demographic features, this dataset has no label columns and will be tested using Kaggle API for Udacity+Arvato: Identify Customer Segments competition.

Two metadata files that contain a data dictionary for the demographic features in the previous dataset files.

- `DIAS Information Levels - Attributes 2017.xlsx`: An excel sheet that contains a top-level organization of demographic features, their description, and some notes.
- `DIAS Attributes - Values 2017.xlsx`: An excel sheet that contains demographic features sorted alphabetically, their description, their values, and the meaning of each value.

2.1.2 The Dataset Metadata

The dataset metadata is described in `DIAS Attributes - Values 2017.xlsx`

- It has the following columns :

`'attribute'`	Name of the dataset feature.
`'description'`	Description of the dataset feature (corresponding to `'attribute'`.
`'value'`	Contains a valid range of values for each attribute.
`'meaning'`	Meaning of each value of the values (corresponding to `'value'`).

- The metadata file has 314 feature entries, while the training dataset contains 365 features, so, there is a mismatch between some of the dataset features and their metadata entries and we have to examine and correct them manually.
- For the `'value'` column, some entries are invalid or contain multiple entries. Example: `"-1, 9"`, `"-1, 0"`, and `"..."`.
- After metadata cleaning, it is noted that some codes represent *unknown values*. They are summarized in Table 1 and plotted in Figure 2

Table 1: Frequency counts of unknown value numeric codes

code	frequency
-1	230
9	75
0	66
10	12

2.1.3 The Training Dataset

The raw training dataset is in the file `Udacity_MAILOUT_052018_TRAIN.csv`.

A quick examination of the dataset showed that the dataset is highly skewed (imbalanced). only 1.24% of the individuals targeted by the marketing campaign would respond to it as shown in Table 2:

Table 2: Label Counts and Frequencies

response	count	percentage
0	42,430	98.76 %
1	532	1.24 %

After cleaning the dataset, the features were categorized into numeric, nominal, and ordinal. The features are also divided into two subsets: features that have metadata and extra features that don't have metadata.

Table 3 shows the distribution of feature types. And a plot is shown in Figure 1.

Table 3: Dataset feature types

type	subset	frequency
nominal	extra	39
nominal	metadata	64
numeric	extra	9
numeric	metadata	7
ordinal	extra	8
ordinal	metadata	238

There are many problems with this dataset:

- After exploratory data analysis, it was found that the training dataset has 56 columns that don't have metadata and should be categorized and examined manually.
- Some columns have mixed data types, for example the featrues `['CAMEO_DEUG_2015', 'CAMEO_INTL_2015']` have the invalid values `['X', 'XX']`.
- Some dataset columns have different names than their corresponding metadata entries (for example: `'D19_.*'` columns). They should be renamed to match the metadata. The metadata will be used later in the Data Preprocessing subsection to check for the valid range of values of every dataset feature.
- 67 features have more than 30% missing values. (See Figure 4)
- 98 features are highly correlated with other features in the dataset, with a Pearson's r correlation coefficient greater than 0.6. A heatmap of highly correlated features is shown in Figure 5.
- 13.41% of training samples are duplicates. (See Figure 3).
- Numerical features have outliers that are more than three standard deviations away from the mean. The percentages of outliers for every numeric feature are shown in Table 4 and Figure 6.

Table 4: Percentage of outliers for each numerical feature

feature	percentage of outliers
ANZ_HAUSHALTE_AKTIV	6.81%
ANZ_HH_TITEL	7.20%
ANZ_PERSONEN	3.88%
ANZ_TITEL	4.11%
EINGEFUEGT_AM	9.38%
EINGEZOGENAM_HH_JAHR	3.34%
GEBURTSJAHR	0.00%
KBA13_ANZAHL_PKW	7.25%
VERDICHTUNGSRAUM	9.27%

2.2 Exploratory Visualization

Figure 1: Dataset Feature Types

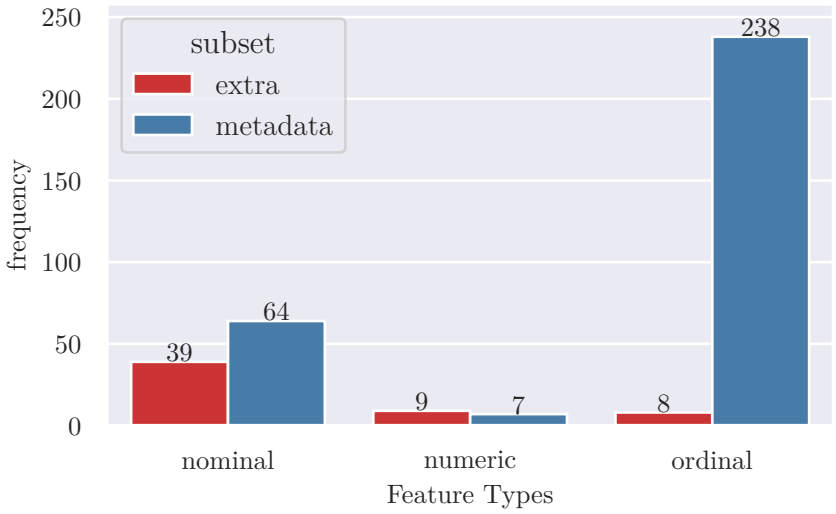


Figure 2: Frequency of unknown values representation in metadata

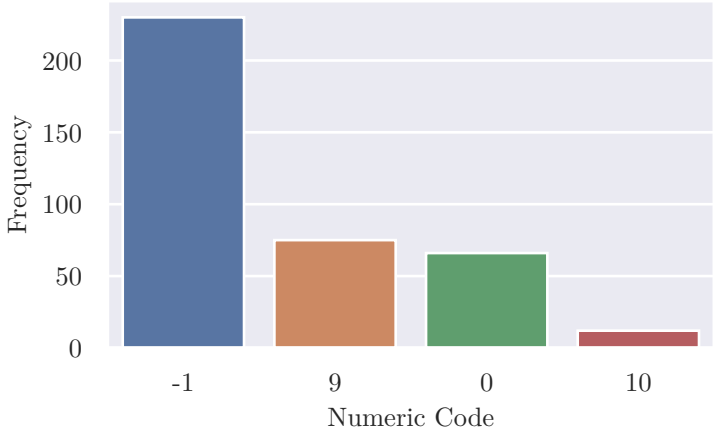


Figure 3: Percentage of duplicate samples.

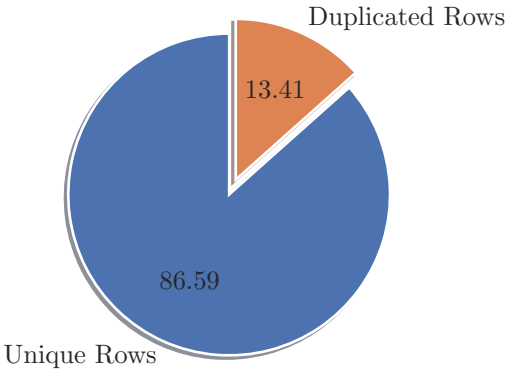


Figure 4: Features that have more than 30% of missing values

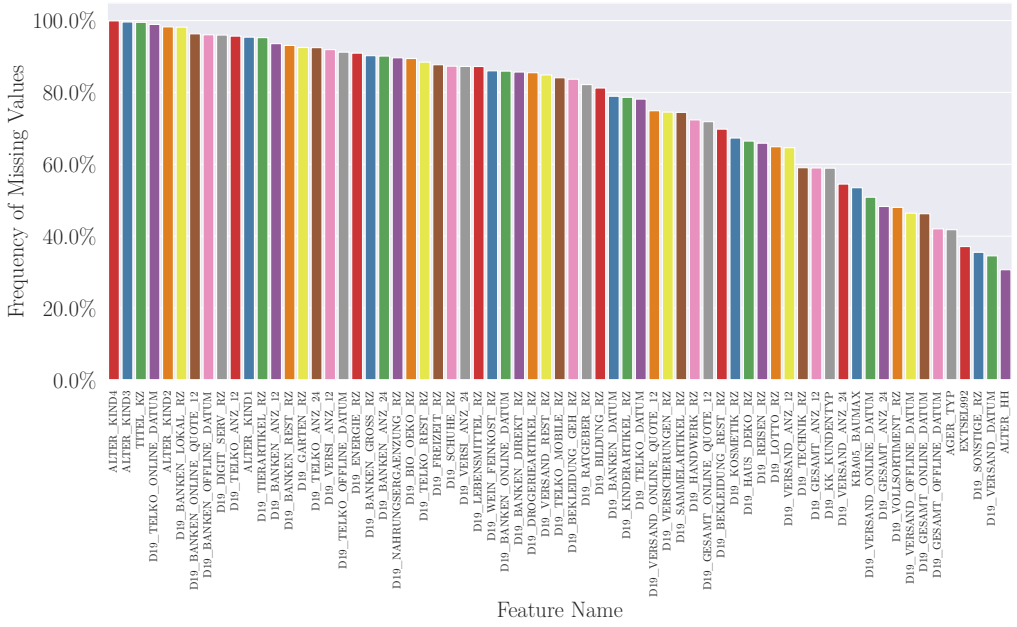


Figure 5: A heat map of highly correlated feature (> 0.75)

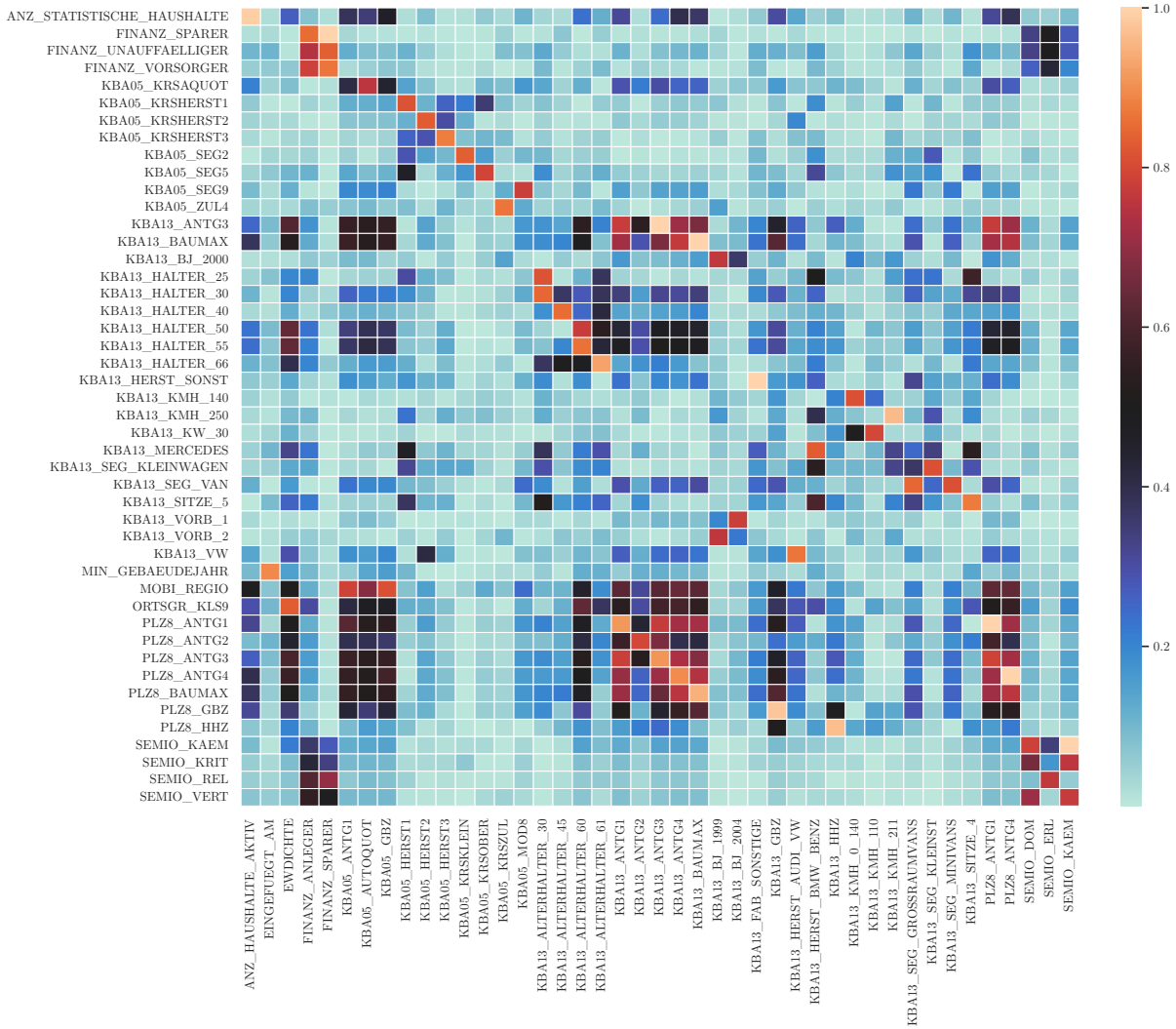
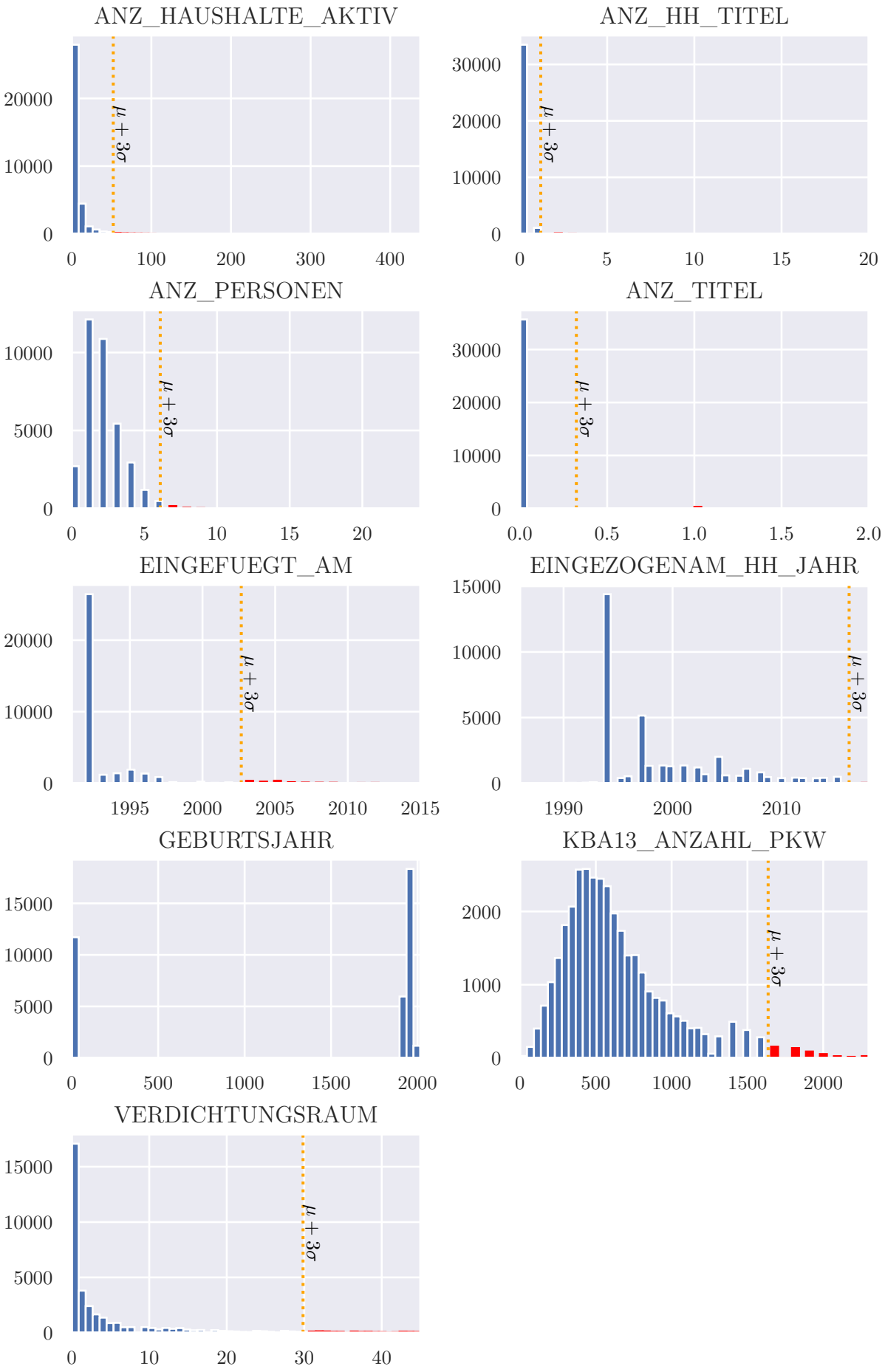


Figure 6: Outliers in numeric values



2.3 Algorithms and Techniques

Our problem is a binary classification problem. We will use the XGBoost supervised learning algorithm [Fri00] to predict whether a targeted individual will be a customer (class 1) or not (class 0).

XGBoost stands for e**X**treme **G**radient **Boost**ing. It can be used in supervised learning problems like ours and it is built on the concept of decision tree ensembles. A tree ensemble model is a collection of classification and regression trees (CART) that are called *weak learners*. When (CART) trees are combined their predictive power is enhanced and we obtain a *strong learner*.

The algorithm has many hyperparameters, but we will only tune the following hyperparameters in Table 5 using Amazon SageMaker hyperparameter tuning jobs:

Table 5: XGBoost Tunable Hyperparameters

Parameter Name	Description
`'num_round'`	The number of rounds to run the training.
`'alpha'`	L1 regularization term on weights. Increasing this value makes models more conservative.
`'colsample_bylevel'`	Subsample ratio of columns for each split, in each level.
`'colsample_bynode'`	Subsample ratio of columns from each node.
`'colsample_bytree'`	Subsample ratio of columns when constructing each tree.
`'eta'`	Step size shrinkage used in updates to prevent overfitting. After each boosting step, you can directly get the weights of new features. The eta parameter actually shrinks the feature weights to make the boosting process more conservative.
`'gamma'`	Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm is.
`'lambda'`	L2 regularization term on weights. Increasing this value makes models more conservative.
`'max_delta_step'`	Maximum delta step allowed for each tree's weight estimation. When a positive integer is used, it helps make the update more conservative. The preferred option is to use it in logistic regression. Set it to 1-10 to help control the update.
`'max_depth'`	Maximum depth of a tree. Increasing this value makes the model more complex and likely to be overfit. 0 indicates no limit. A limit is required when `'grow_policy'`=depth-wise.
`'min_child_weight'`	Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than `'min_child_weight'`, the building process gives up further partitioning. In linear regression models, this simply corresponds to a minimum number of instances needed in each node. The larger the algorithm, the more conservative it is.
`'subsample'`	Subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collects half of the data instances to grow trees. This prevents overfitting.

The hyperparameters that have the greatest effect on optimizing the XGBoost evaluation metrics are: `'alpha'`, `'min_child_weight'`, `'subsample'`, `'eta'`,and `'num_round'`.

The full list of hyperparameters can be found in Amazon SageMaker documentation.

We will use the `'validation:logloss'` evaluation metric described in subsection 1.3 as our objective function is `'binary:logistic'`

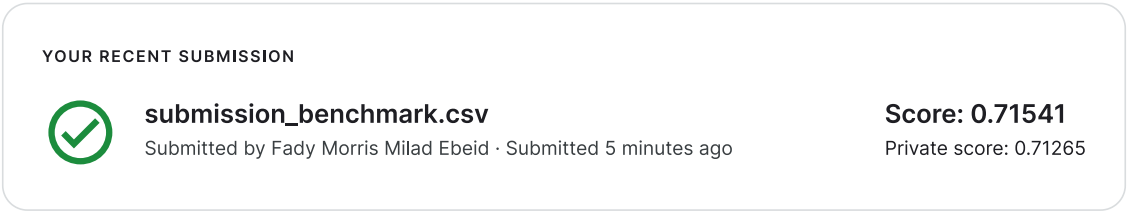
2.4 Benchmark

Our benchmark model is a simple scikit-learn logistic regression model trained using the default hyper-parameters.

The model was trained using the *train set*, used for inference on the *test set* to generate probabilities, and the resulting probabilities were submitted to Kaggle Udacity+Arvato: Identify Customer Segments competition for evaluation.

The resulting AUC-ROC private test score is **0.71265** as shown in Figure 7

Figure 7: Benchmark Model Score

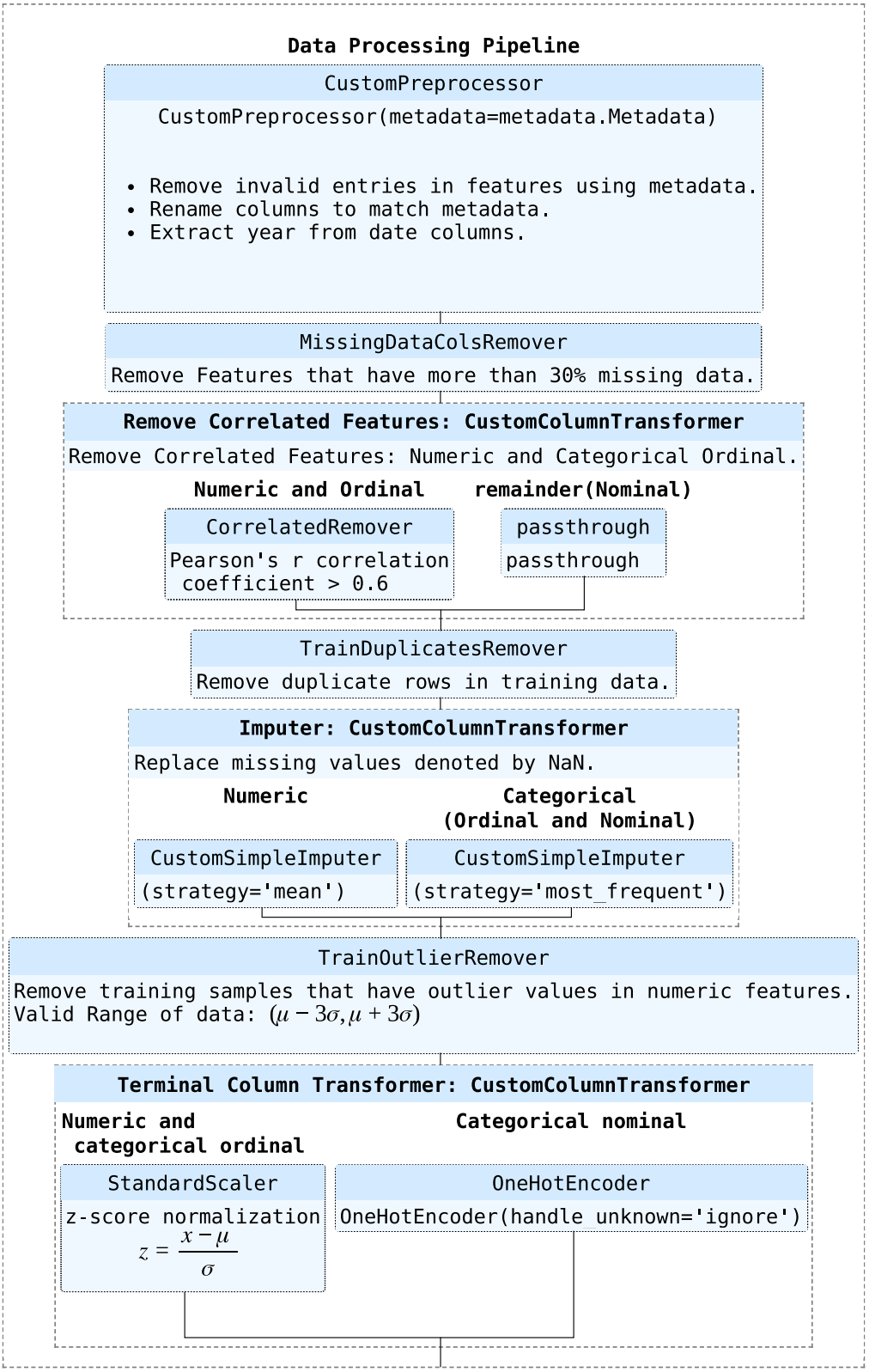


In subsection 4.2, the score of our *XGBoost* model will be compared to the score of this baseline model in terms of performance.

3 Methodology

3.1 Data Preprocessing

Figure 8: Data Processing Pipeline



The data processing pipeline sketched in Figure 8 is divided into the following steps:

1. `CustomPreprocessor`: a custom preprocessing class that performs the following:
   - Fixes the columns that have mixed data types `['CAMEO_DEUG_2015', 'CAMEO_INTL_2015']` by replacing the invalid values `['X', 'XX']` by `NaN`
   - Drops the `['LNR']` column as it is an index column.
   - Renames all the columns that differ in name between the dataset and the metadata either by regular expressions (For example `'D19_.*'` columns) or manually using a dictionary to match the metadata.
   - For categorical (ordinal and nominal) features that have metadata, it checks their values against their valid ranges from the metadata.
2. `MissingDataColsRemover`: Drops the features that have more than 30% missing values.
3. `CorrelatedRemover`. Removes correlated features that have a Pearson's r correlation coefficient greater than 0.6. It is wrapped in a `CustomColumnTransformer` that transforms the *numeric* and *ordinal* features and passes the *nominal* features unchanged.
4. `TrainDuplicatesRemover`: Removes duplicate training samples.
5. `Imputer`: A `ColumnTransformer` that contains two `CustomSimpleImputer` transformers that transform features according to different strategies:
   - Numeric features: replaces the missing values with the mean.
   - Categorical (ordinal and nominal) features: replaces the missing values with the most frequent value (the mode).
6. `TrainOutlierRemover`: Removes training samples that contain outliers. Outliers are values that fall out of the range of three standard deviations from the mean. Valid data range is $(\mu - 3\sigma, \mu + 3\sigma)$.
7. A terminal `ColumnTransformer` that wraps two transformers:
   - `StandardScaler`: is applied to the *numeric* and *ordinal* features. It performs z-score normalization according to the following equation:

     $$\mathbf{z} = \frac{\mathbf{x} - \mu}{\sigma}$$

   - `OneHotEncoder`: Transforms the *categorical nominal* features by creating a dummy variable of each value of every categorical feature.

3.2 Implementation

The project was implemented using Python, Pandas, Scikit-Learn, and Amazon SageMaker.

Following are the general steps of the project development:

1. **Metadata Extraction and Metadata Wrapper Class:** First I explored the metadata file `DIAS Attributes - Values 2017.xlsx`, extracted the relevant data from it in the form of [`'attribute'`, `'description'`, `'value'`,`'meaning'`, `'type'`], then saved the cleaned metadata to the file `input/data/processed/metadata.csv` to be used in the project.

   After that, we created a `Metadata` wrapper class that is used to look up the metadata dataframe for valid values and query it to get custom lists of features by type (nominal, ordinal, and categorical). The general outline of the class is in Listing 1.

Listing 1: Metadata

```
class Metadata():
   df_lookup = None
   df_metadata = None
   def __init__(file_path):
   def drop_unknown_values():
   def lookup_features(
       input_df,
       method='intersect',
       subsets=['metadata', 'extra'],
       types=['numeric', 'ordinal','nominal']
    ):
   def get_features(subsets=['metadata', 'extra'],
                    types=['numeric', 'ordinal','nominal']):
   def get_valid_ranges(col_name, col_dtype):
```

2. **Data Processing:** Data processing was described in subsection 3.1 - Data Preprocessing. The data processing pipeline (Illustrated in Figure 8 - Data Processing Pipeline) was developed using scikit-learn pipeline.

   The pipeline steps were modeled as subclasses of scikit-learn BaseEstimator and TransformerMixin classes.

   I created the custom classes (`CustomPreprocssor()`, `CustomColumnTransformer()`, `MissingDataColsRemover()`, `CorrelatedRemover()`, `TrainDuplicatesRemover()`, `CustomSimpleImputer()`, and `TrainOutlierRemover()`) to model the pipeline steps. They provide `fit()` and `transform()` methods that are called by `sklearn.Pipeline` during fitting and transformation of data. The main steps of preprocessing are:

   (a) The *training* dataset is loaded from `Udacity_MAILOUT_052018_TRAIN.csv` in a pandas dataframe, then it is passed through the pipeline's `fit_transform()` method. The *testing* dataset is loaded from `Udacity_MAILOUT_052018_TEST.csv` and passed through the fitted pipeline's `transform()` method.

   (b) The training dataset is split into 80% *train* subset and 20% *validation* subset.

   (c) The transformed *train*, *test*, and *validation* datasets are saved to `.csv` files that are compatible with Amazon SageMaker built-in algorithms. The first column is the label and the `.csv` file has no column headers.

3. The data were uploaded to an AWS S3 bucket using AWS CLI "`aws s3 cp`" command, then we defined a `sagemaker.inputs.TrainingInput()` data channel input for each subset to channel the data from the `.csv` files located in the S3 bucket to the EC2 instance that runs the training algorithm.

4. **Model training job:**

   We used *XGBoost* as a built-in algorithm on Amazon SageMaker. We retrieved an *Amazon Machine Image (AMI)* of the algorithm (`'683313688378.dkr.ecr.us-east-1.amazonaws.com/sagemaker-xgboost:1.3-1'`), then launched a training job that starts an `'ml.m5.2xlarge'` EC2 compute instance and runs the container AMI inside it to train our model.

   Finally, the trained model artifacts that contain the model weights are saved as `model.tar.gz` in the *AWS S3 bucket*. It was used later by the *SageMaker Batch Transform* job to make inferences on the test set.

The general steps of this process are:

(a) Get the training *XGBoost* image URI

```
xgboost_image = sagemaker.image_uris.retrieve("xgboost", region, version='1.3-1')
```

(b) Create a model training job using Amazon SageMaker `sagemaker.estimator.Estimator` class. The estimator is created with the following common keyword args and hyperparameters:

```
estimator_kwargs = {
    'image_uri':       xgboost_image,
    'role':            IAM_ROLE,
    'instance_count':  1,
    'instance_type':   'ml.m5.2xlarge',
    'volume_size':     5,
    'output_path':     f"s3://{BUCKET}/output/",
    'base_job_name':   "xgboost-training-arvato",
}
```

```
estimator_hyperparameters_common = {
   "objective": "binary:logistic",
   "num_round": 4000,
   "early_stopping_rounds": 400,
   "tree_method": 'exact',
}
```

```
xgb_estimator = sagemaker.estimator.Estimator(**estimator_kwargs,
                            hyperparameters=estimator_hyperparameters_common
                                           )
```

(c) Start a Model training job.

```
xgb_estimator.fit(data_channels, wait=True)
```

5. **Model inference job:** We used Amazon SageMaker batch transform to obtain the predictions from the *test* set as a vector of probabilities.

   The transformer is created from the fitted estimator as follows:

```
xgb_transformer = xgb_estimator.transformer(
    instance_count = 1,
    instance_type = 'ml.m5.2xlarge'
)

xgb_transformer.transform(
    f"s3://{BUCKET}/data/test.csv" ,
    content_type=content_type,
    split_type='Line',
    wait=True)
```

3.3 Refinement

Model refinement was done using Amazon SageMaker Hyperparameter Tuning

A hyperparameter tuner is created with the following hyperparameter ranges and keyword arguments:

```
tuner_hyperparameter_ranges = {
    'alpha':             ContinuousParameter(0, 1000),
    'colsample_bylevel': ContinuousParameter(0.1, 1),
    'colsample_bynode' : ContinuousParameter(0.1, 1),
    'colsample_bytree' : ContinuousParameter(0.5, 1),
    'eta':               ContinuousParameter(0.1, 0.5),
    'gamma':             ContinuousParameter(0, 5),
    'lambda':            ContinuousParameter(0, 1000),
    'max_delta_step':    IntegerParameter(0, 10),
    'max_depth':         IntegerParameter(0, 10),
    'min_child_weight':  ContinuousParameter(0,120),
    'num_round' :        IntegerParameter(1,4000),
    'subsample' :        ContinuousParameter(0.5, 1),
}
```

```
hyperparameter_tuner_kwargs = {
    'estimator':              sagemaker.estimator.Estimator(
        **estimator_kwargs,
        hyperparameters= estimator_hyperparameters_common
    ),
    'objective_metric_name': 'validation:logloss',
    'hyperparameter_ranges':  tuner_hyperparameter_ranges,
    'objective_type':        'Minimize',
    'max_jobs':               60,
    'max_parallel_jobs':      1,
    'base_tuning_job_name':  'arvato-hpo',
    'early_stopping_type':   'Auto',
}
```

Amazon SageMaker automatic model tuning was configured to run 60 sequential training jobs on the train set using the range of hyperparameters we specified above. It then chose the hyperparameter values that resulted in a model that performed the best, as measured by the chosen objective metric `'validation:logloss'`

We set `'max_parallel_jobs' = 1` as running one training job at a time achieves the best results with the least amount of compute time. (see SageMaker documentation on best practices for hyperparameter tuning).

The top 5 combinations of hyperparameters that were discovered by the hyperparameter tuner are displayed in Table 6.

Table 6: Hyperparameter Tuner Results

hyperparameter	1	2	3	4	5
'job_name'	bd859e2b	424417fa	a121354b	4e3dd959	4931eb1a
'obj_value'	0.058980	0.059020	0.059100	0.059200	0.059400
'time'	94	84	63	63	91
'num_round'	1,136	1,160	251	239	314
'alpha'	1.466071	0.000000	23.438728	30.701911	14.456794
'colsample_bylevel'	0.772286	0.266009	0.543753	0.564040	0.637091
'colsample_bynode'	0.669727	0.144735	0.485596	0.895001	0.714065
'colsample_bytree'	0.657077	0.696430	0.505232	0.744022	0.840035
'eta'	0.108203	0.265815	0.445710	0.138256	0.358915
'gamma'	0.485602	1.177283	0.380622	3.849457	0.577412
'lambda'	880.072219	914.404495	842.474563	114.649856	398.453567
'max_delta_step'	1	3	4	1	9
'max_depth'	8	8	9	2	10
'min_child_weight'	53.322057	8.474547	21.097838	77.382115	0.493639
'subsample'	0.952727	0.768599	0.944670	0.961180	0.789859

Next, an *XGBoost* estimator was trained using those best hyperparameters, and inferences were obtained for the test data using Amazon SageMaker batch transform.

4 Results

4.1 Model Evaluation and Validation

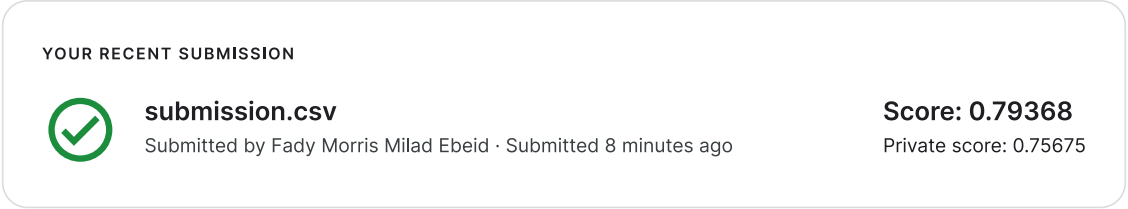
The final model was trained with the best hyperparameters obtained in subsection 3.3. Then, it was used by an Amazon SageMaker batch transform job to generate inferences for test data and get a vector of probabilities.

For model testing, we used the test dataset in `Udacity_MAILOUT_052018_TEST.csv` that the model hasn't seen before during training. The same preprocessing steps were applied to the data using the same pipeline that was used to transform the training data.

The results were submitted to the Kaggle competition for validation and the resulting final private AUC-ROC score is **0.75675** as shown in Figure 9.

The score is high, which shows that the model can distinguish between positive and negative classes.

Figure 9: Final Model Score



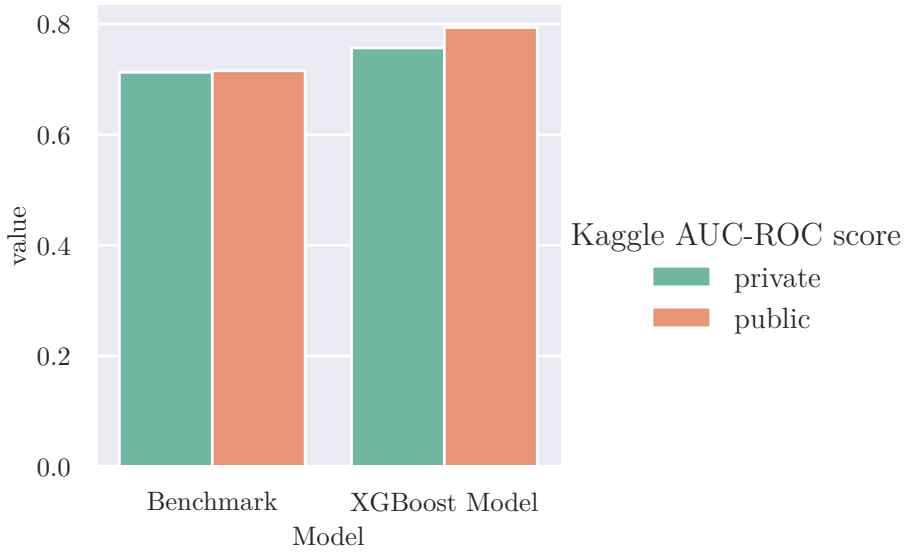
4.2 Justification

The final results are summarized in Table 7 and Figure 10. It shows that our final *XGBoost* model has achieved an AUC-ROC score of **0.79368** for the test dataset on the Kaggle competition, Improving on the benchmark score of **0.71541** by $\approx 11\%$.

Table 7: Comparison Between the Benchmark Model and the Final Model's AUC-ROC score

Model	private score	public score
Benchmark(Logistic Regression)	0.71265	0.71541
XGBoost Model	0.75675	0.79368

Figure 10: A Comparison Between Benchmark Model and Final Model Scores



5 Conclusion

5.1 Reflection

In this project, we used the Arvato dataset to predict whether an individual targeted by the mailout campaign will turn into a client or not.

The dataset was very challenging to work with, it was highly dimensional with lots of inconsistencies, missing values, and invalid values. For such a dataset, the scikit-learn pipeline and transformers came in handy to develop a robust preprocessing workflow.

From this project, it was clear that *XGBoost* is one of the top-performing machine learning algorithms. That's why it is used in Kaggle competitions.

The entire end-to-end solution to the problem can be summarized in the following steps:

1. The problem was clearly defined, then the candidate solution algorithms and benchmark metrics were selected according to the nature of the problem.
2. The training dataset and metadata were checked for problems and inconsistencies in the exploratory data analysis phase.
3. The metadata was cleaned and saved for use throughout the project. The dataset features were categorized into numerical, ordinal, and nominal.
4. The training dataset features were examined for mismatch with the metadata. Then we explored missing data, correlated features, duplicate training samples, and outliers.
5. A preprocessing pipeline was developed using the best software engineering practices of code reuse. We extended scikit-learn transformers and adapted them to the current problem at hand.
6. The training and the testing dataset were processed using the developed pipeline.
7. The training dataset was split further into train and validation sets.
8. A benchmark model was developed, tested on the test data and the metric score was recorded.
9. The datasets were uploaded to an AWS S3 Bucket to be used by Amazon SageMaker.
10. A preliminary SageMaker estimator was created to train an *XGBoost* model using an Amazon SageMaker built-in algorithm container image.
11. The model was further refined using an Amazon SageMaker hyperparameter tuning job to select the best hyperparameters according to the objective metric specified.
12. A final *XGBoost* model was trained using the best hyperparameters discovered by the tuning job.
13. The final model was used for inference to generate predictions from the test data again, obtaining a vector of probabilities. Then the results were submitted to Kaggle API to get another metric score.
14. The final *XGBoost* model was evaluated and compared to the benchmark model. It was clear that the final model outperformed the benchmark model and got higher AUC-ROC scores.

5.2 Improvement

The results achieved in this project could be possibly improved by the following suggestions:

1. Increasing the hyperparameter tuning job's `'max_jobs'` parameter to allow the tuner to run for a longer period and search for a better combination of hyperparameters in the search space.
2. Trying to use the logarithmic scales for some hyperparameters in hyperparameter tuning.
3. Investigating oversampling techniques (such as SMOTE) and undersampling techniques to address the problem of class imbalance.
4. A more thorough understanding of the dataset features.
5. Investigating feature importance and retraining the algorithm using k-most important features.
6. Examining the correlation between categorical nominal variables using the chi-square test.

References

[AF20] Atallah M. AL-Shatnwai and Mohammad Faris. "Predicting Customer Retention using XGBoost and Balancing Methods". en. In: *International Journal of Advanced Computer Science and Applications* 11.7 (2020). ISSN: 21565570, 2158107X. DOI: 10.14569/IJACSA.2020.0110785. URL: http://thesai.org/Publications/ViewPaper?Volume=11&Issue=7&Code=IJACSA&SerialNo=85 (visited on 01/23/2022).

[Fri00] Jerome Friedman. "Greedy Function Approximation: A Gradient Boosting Machine". In: *The Annals of Statistics* 29 (Nov. 2000). DOI: 10.1214/aos/1013203451.

[KGaa] Bertelsmann SE & Co KGaA (www.bertelsmann.com). *Arvato - Bertelsmann SE & Co. KGaA*. en. Publisher: Bertelsmann SE & Co. KGaA, Carl-Bertelsmann-Straße 270, D-33311 Gütersloh, www.bertelsmann.com. URL: https://www.bertelsmann.com/divisions/arvato/ (visited on 01/23/2022).

[KGab] Bertelsmann SE & Co KGaA (www.bertelsmann.com). *Company - Bertelsmann SE & Co. KGaA*. en. Publisher: Bertelsmann SE & Co. KGaA, Carl-Bertelsmann-Straße 270, D-33311 Gütersloh, www.bertelsmann.com. URL: https://www.bertelsmann.com/company/ (visited on 01/23/2022).

[Zhu18] Yayun Zhuang. "Research on E-commerce Customer Churn Prediction Based on Improved Value Model and XG-Boost Algorithm". en. In: *Management Science and Engineering* 12.3 (Sept. 2018). Number: 3, pp. 51–56. ISSN: 1913-035X. DOI: 10.3968/10816. URL: http://flrjournal.org/index.php/mse/article/view/10816 (visited on 01/23/2022).