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

Fady Morris Milad[†]

February 21, 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. It's solutions focus on automation and data analytics.[KGaa]

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

Arvato analytics is helping businesses to take important decisions ang 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 campain mailout data.

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

1.2 Problem Statement

The problem can be stated as: "Given the existing marketing campain 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 or 1, and the dataset is imbalanced as shown in subsection 2.1 - Data Exploration, We will use the following classification metrics:

1.3.1 ROC-AUC Score

Area Under the Receiver Operating Characteristic Curve. This is the metric used by Kaggle Competition to evaluate the test data.

The Receiver Operator Characteristic (ROC) curve is a plot of *true positive rate*(TPR) against *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 classes.

ROC-AUC Metric interpretation:

- **AUC = 1**: The classifier perfectly separates the positive and the negative classes.
- **AUC = 0**: The classifier predicts all negatives as positives and vice versa.
- **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 totally unable to distinguish between classes.

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:

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

Where:

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

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

In our project, log-loss objective metric is used by Amazon Sagemaker hyperparameter tuining 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 campain. It contians data for 42,982 individuals.

  This training daataset has 367 columns, 366 of them are demographic features and 1 label column `'RESPONSE'`

- `Udacity_MAILOUT_052018_TEST.csv`: The testing dataset, It contains data for 42,833 individuals and it has 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.

There are also 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 meaning of each value.

2.1.2 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.
  value: This column contains a valid range of values for each attribute.
  meaning: Meaning of each value of the values.

- The metadata file has 314 feature entries, while the training dataset contains 365 features, so there are 51 features that don't have metadata and we have to examine and validate manually.
- For value column, some entries are invalid or contain multiple values. Example: `"-1, 9"`, `"-1, 0"`, `"..."`. (See Figure 2)

Table 1: Frequency counts of unknown value numeric codes

code	frequency
-1	230
9	75
0	66
10	12

2.1.3 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 the following table:

Table 2: Label Counts and Frequencies

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

The training dataset features are numeric, nominal, and ordinal. The features are 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 examination, 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 `['CAMEO_DEUG_2015', 'CAMEO_INTL_2015']` have the invalid values `['X', 'XX']`
- Some columns have different name than those in the metadata (For example `'D19_.*'` columns), they should be renamed to match the metadata. The metadata will be used to check for valid range of values in the preprocessing step.
- There are 67 features that 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 > 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 are shown in the following table. Outliers are shown in 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

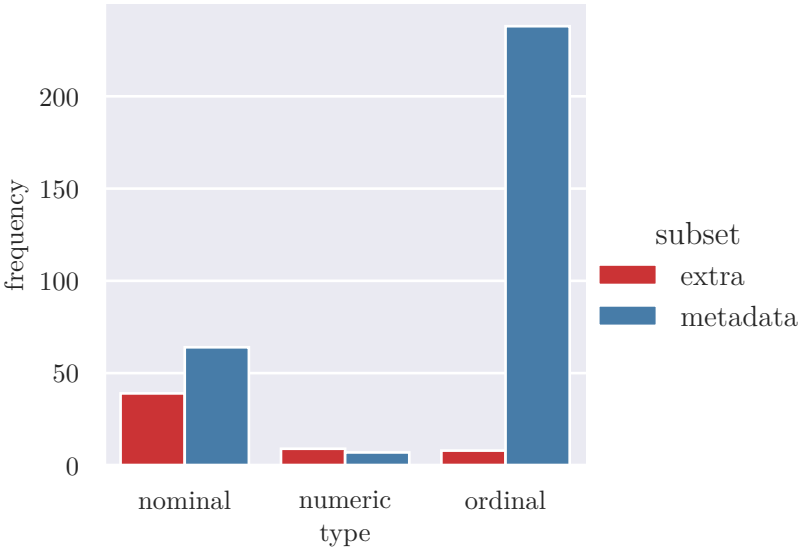


Figure 2: Frequency of unknown values representation in metadata

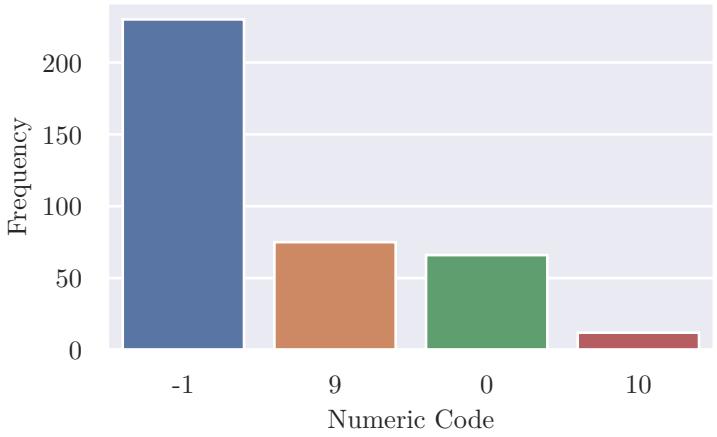


Figure 3: Percentage of duplicate samples.

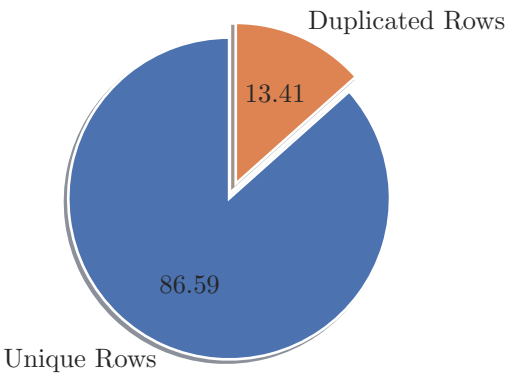


Figure 4: Features that have more than 30% of missing values

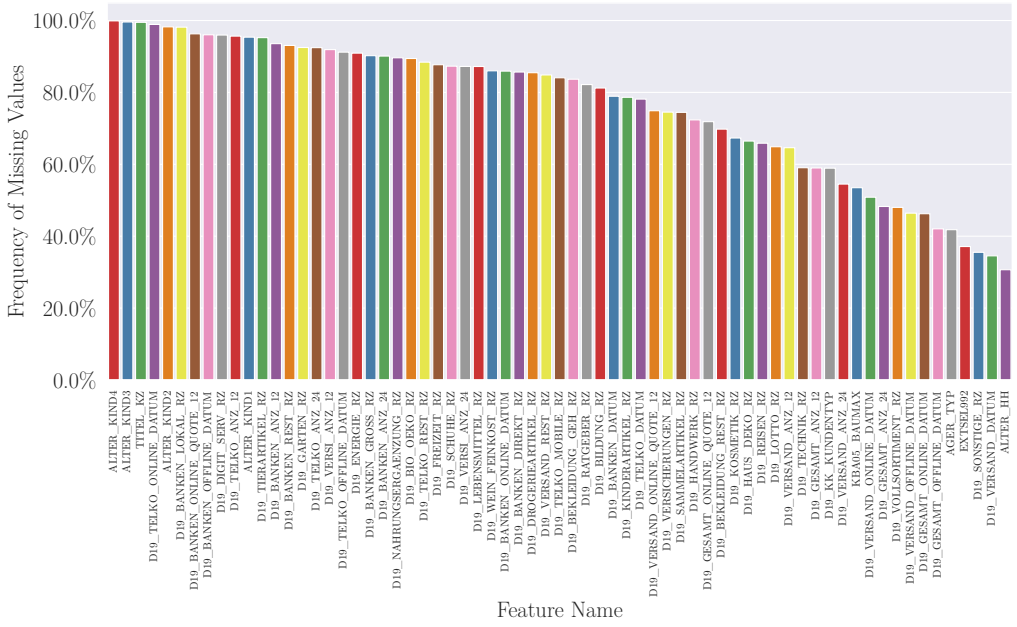


Figure 5: A heat map of highly correlated feature (> 0.75)

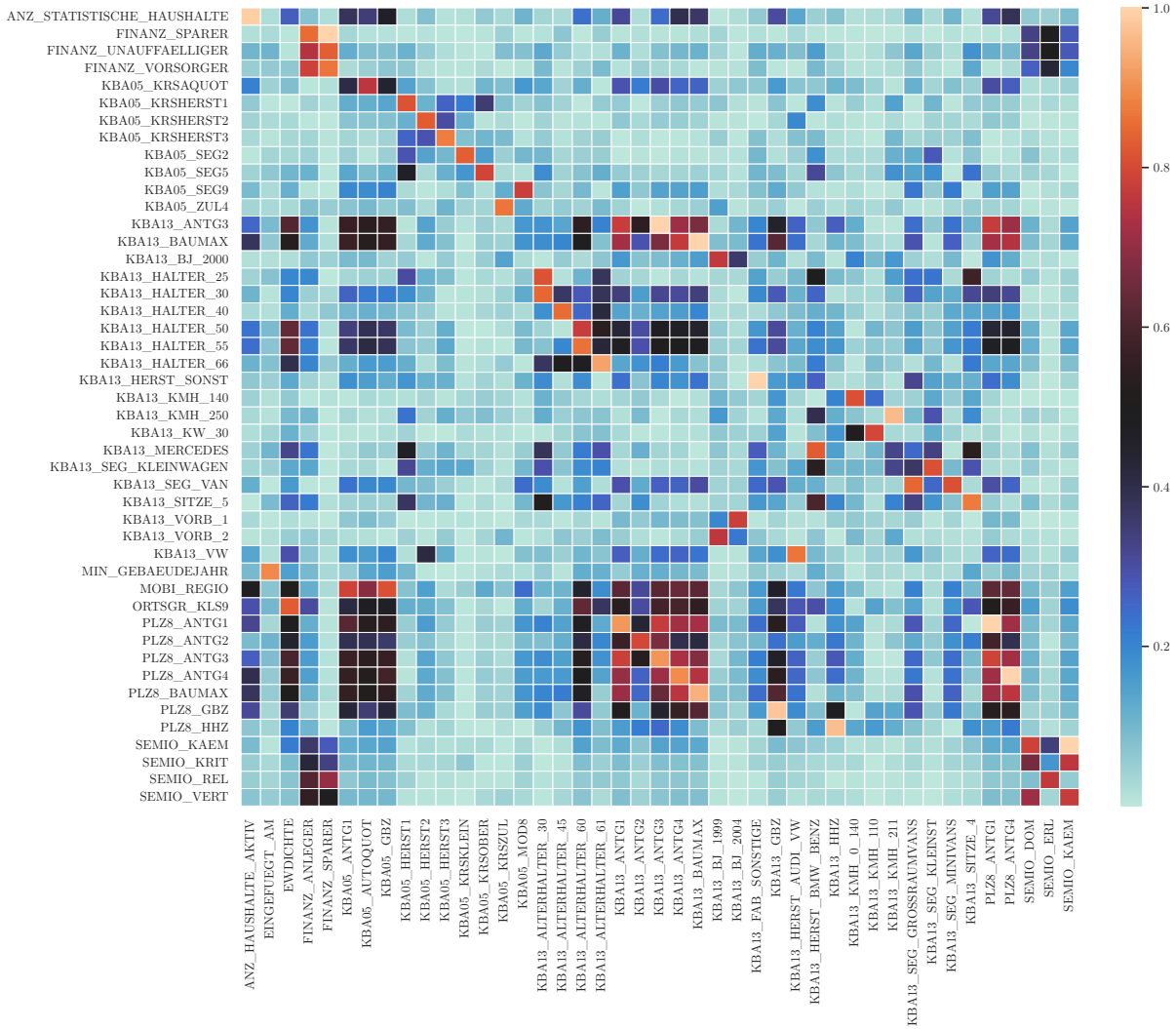
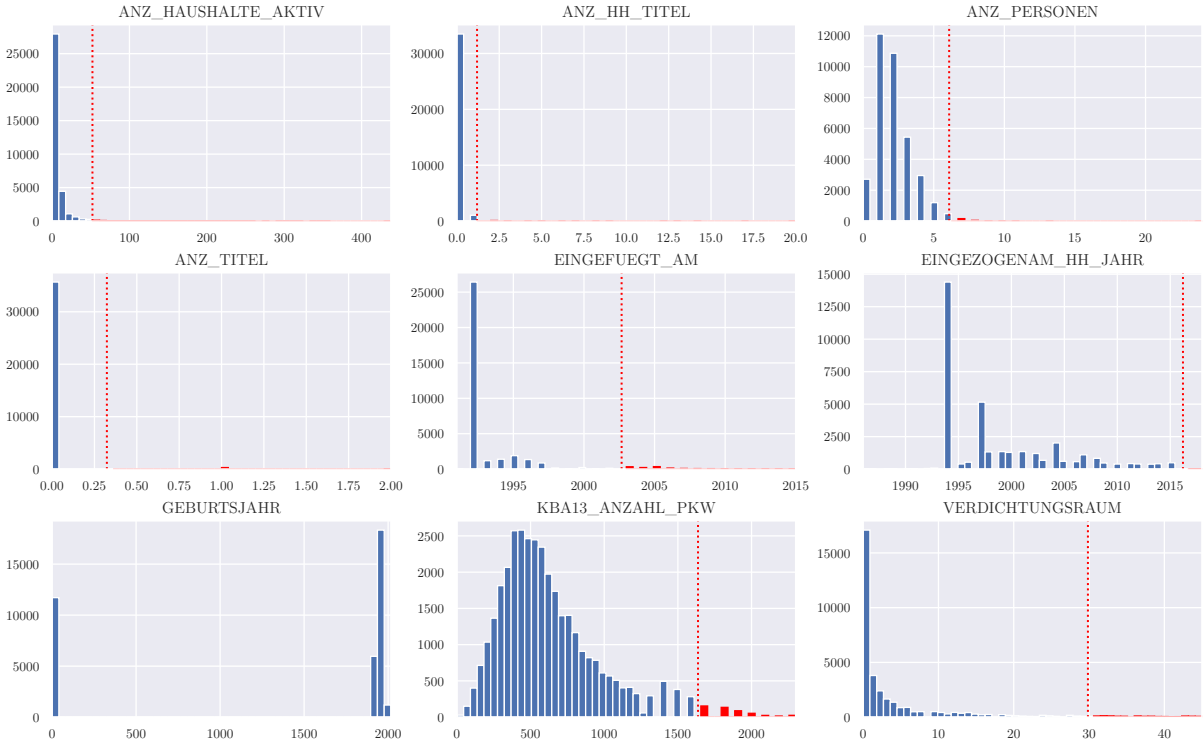


Figure 6: Outliers in numeric values



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 customer will be a customer (class 1) or not (class 0).

XGBoost stands for e**X**treme **G**radient **Boost**ing. is used in supervised learning problem like ours. 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 together their predictive power is enhanced and we obtain a strong learner.

The algorithm has many hyperparameters, but we will only tune the following hyperparameters using Amazon Sagemaker hyperparameter tuning jobs.

num_round	he 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`

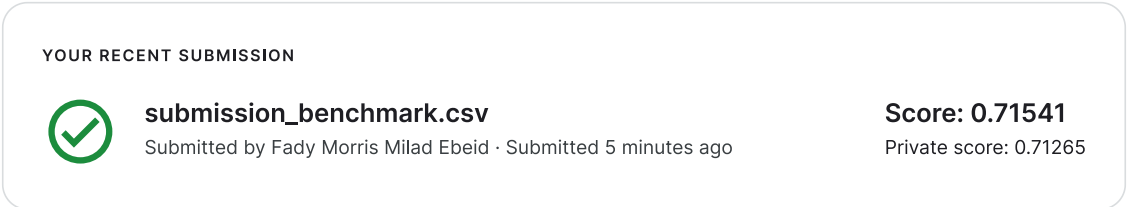
2.4 Benchmark

Our benchmark model is a simple Logistic Regression model from Scikit-Learn trained using the default hyperparameters.

The model was trained and then used for inference on the Arvato test dataset supplied, then the resulting probabilities were submitted to Kaggle Udacity+Arvato: Identify Customer Segments competition for evaluation.

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

Figure 7: Benchmark Model Score

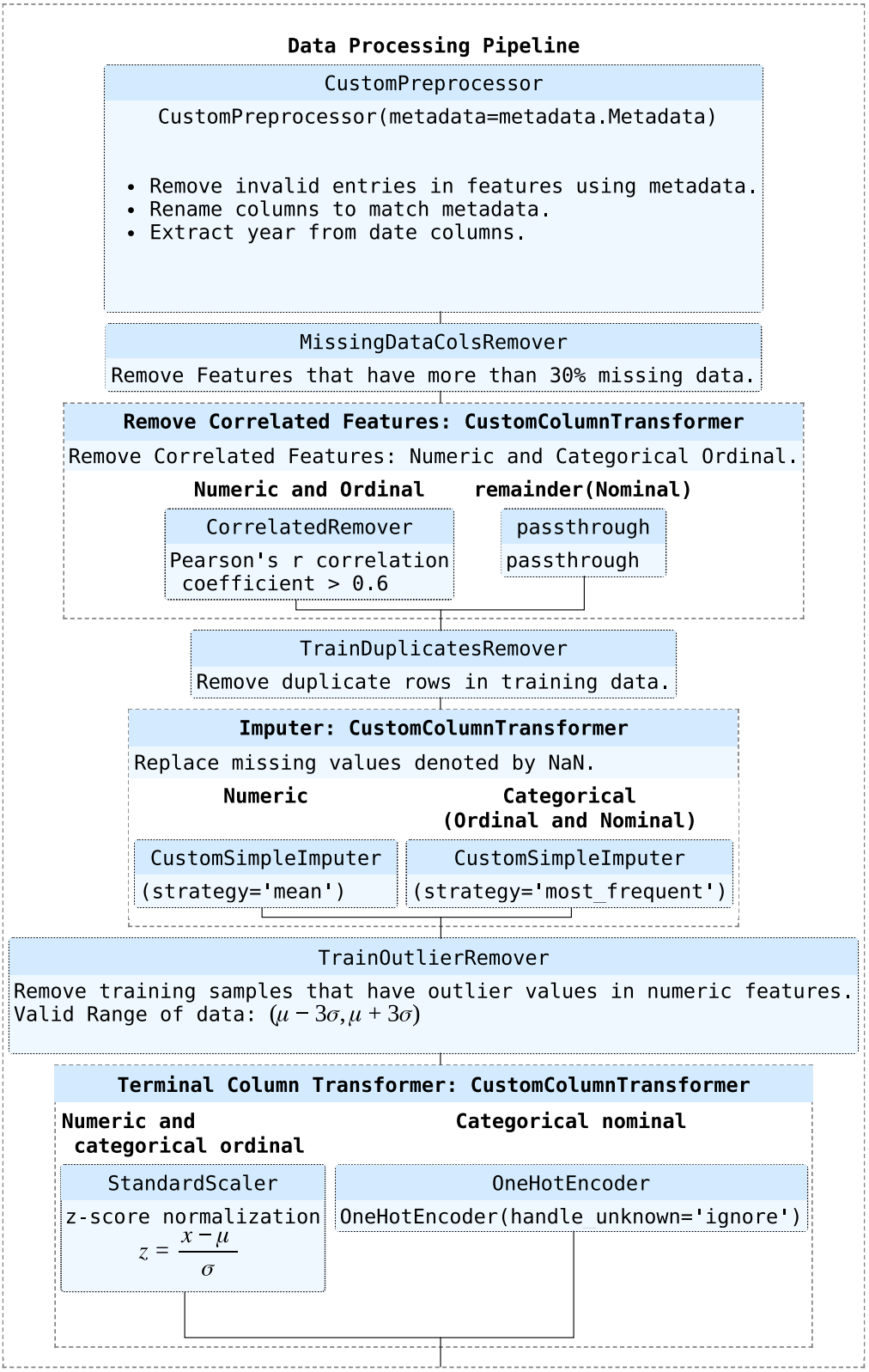


In subsection 4.1, our `XGBoost` model will be compared to this baseline model in terms of performance.

3 Methodology

3.1 Data Preprocessing

Figure 8: Data Processing Pipeline



The data processing pipeline is divided into the following steps:

1. `CustomPreprocessor`: a custom preprocessing class that performs the following:
   - Fix the columns that have mixed data types `['CAMEO_DEUG_2015', 'CAMEO_INTL_2015']`. Replace the invalid values `['X', 'XX']` by `NaN`
   - Drop the `['LNR']` column as it is an index column.
   - Rename all columns that differ in name between the dataset and the metadata either by regular expressions (For example `'D19_.*'` columns) or manually using a dictionary.
   - For categorical (ordinal and nominal) features that have metadata, check their values against their valid ranges from the metadata.
2. `MissingDataColsRemover`: Removes features that have more than 30% missing values.
3. `CorrelatedRemover`. Removes a correlated features that have a Pearson's r correlation coefficient greater than 0.6. It is wrapped in a `CustomColumnTransformer` that transforms *numeric* and *ordinal* features and pass *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: replace missing values with the mean.
   - Categorical (ordinal and nominal) features: replace missing values with most frequent (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 *numeric* and *ordinal* features.It performs z-score normalization according to the following equation:
   $$\mathbf{z} = \frac{\mathbf{x} - \mu}{\sigma}$$
   - `OneHotEncoder`: Transforms *categorical nominal* features by creating dummy variable of each value of the categorical feature.

3.2 Implementation

The project was implemented using Python, Pandas, Scikit-Learn and AWS Sagemakr.

Following are the general steps for project development:

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

   After that I created a `Metadata` wrapper class that is used to lookup metadata for invalid range of values, query for metadata 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__(self, file_path):
   def drop_unknown_values(self):
   def lookup_features(self,
       input_df, method='intersect',
       subsets=['metadata', 'extra'],
       types=['numeric', 'ordinal','nominal']
    ):
   def get_features(self, subsets=['metadata', 'extra'],
                    types=['numeric', 'ordinal','nominal']):
   def get_valid_ranges(self, col_name, col_dtype):
```

2. **Data Processing:** Data processing is 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 custom classes (`CustomPreprocssor()`, `CustomColumnTransformer()`, `MissingDataColsRemover()`, `CorrelatedRemover()`, `TrainDuplicatesRemover()`, `CustomSimpleImputer()`, and `TrainOutlierRemover()`) to model the pipeline steps. They provide `fit()` and `transform()` methods.

   (a) The *training* dataset is loaded from `Udacity_MAILOUT_052018_TRAIN.csv` in a pandas dataframe, then it is passed through the pipleline'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 splitted into *train* and *validation* dataset with a 80% : 20% ratio.

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

3. Upload the data to AWS S3 bucket using AWS cli. `aws s3 cp` command, then define a `sagemaker.inputs.TrainingInput()` data channel input for each csv file.

4. **Model training job:**

   We will use *XGBoost* as a built-in algorithm on Amazon Sagemaker. We will retrieve an *Amazon Machine Image (AMI)* of the algorithm, and run a training job to train our model. Then the trained model artifacts that contain model weights are saved as `model.tar.gz` in *Amazon S3 bucket* will be used by *Sagemaker Batch Transform* job to make inferences.

   (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 AWS Sagemeker `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 use AWS Sagemaker Batch Transform to obtain predictions from the *test* set.

   The transformer is created using the trained 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 AWS Sagemaker Hyperparameter Tuning job

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',
}
```

It runs 60 training jobs sequentially with different hyperparameters chosen from the hyperparamaeter ranges search space. The goal is find a set of hyperparameters that minimize the `'validation:logloss'` objective metric.

The top 5 hyperparameter tuner job results are displaying in Table 5.

Table 5: 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

Then an XGBoost estimator is trained with the best hyperparameters found from the hyperparameter tuning jobs, and inferences were obtained for test data using AWS Sagemaker batch transform.

4 Results

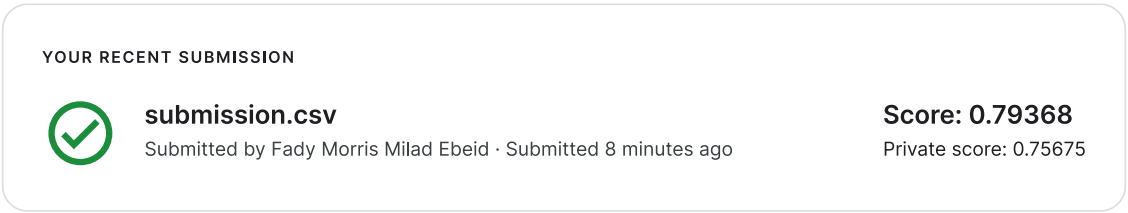
4.1 Model Evaluation and Validation

The final model trained with the best hyperparameters obtained in subsection 3.3 was used to make inferences by an AWS Sagemaker batch transform job, then the results were submitted to kaggle competition for validation.

For model testing we use 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 resulting final private AUC-ROC score is 0.75675 ans shown in Figure 9. Which shows that the model is able to distinguish between positive and negative classes.

Figure 9: Final Model Score



4.2 Justification

The final results are summarized in Table 6. It shows that for XGBoost model we obtained 0.79368 AUC-ROC score on Kaggle competition for test data, Improving the original benchmark score of 0.71541 by $\approx 11\%$.

Table 6: 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

5 Conclusion

5.1 Reflection

In this project we used the Arvato dataset to predict whether a person addressed by mailout campain 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, Scikit-learn pipeline and transformers came 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 explored for problems.
3. The metadata was cleaned and saved for use throughout the project. Different types of features were categorized into numerical, ordinal and nominal.
4. The training dataset was examined for mismatch with the metadata, missing data, correlated features, duplicate training samples, outliers.
5. A preprocessing pipeline was developed using best software engineering practices of code reuse. I extended Scikit-learn transformers and adapted them to the current problem at hand.
6. The training and testing dataset were processed using the developed pipeline.
7. The training dataset was split further into train and validation sets.
8. A benchmark was developed, tested on test data and score was recorded.
9. Data were uploaded to AWS S3 Bucket to be used by Amazon Sagemaker.
10. A preliminary Sagemaker estimator was created to train an XGBoost model using Amazon Sagemaker built-in algorithm image.
11. The model was further refined using Sagemaker hyperparameter tuning job to select the best hyperparameters according to an objective metric.

12. A final XGBoost model was trained using the best hyperparameters discovered by the tuning job.
13. An inference using the test data was made. The test data were transformed using the same processing pipeline fitted to the training data. Then the results were submitted to kaggle to get a score.
14. The final XGBoost model was evaluated and compared to the logistic regression 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 improved by the possible following suggestions:

1. Increasing the hyperparameter tuning job `'max_jobs'` parameter to allow the tuner to run for longer period and search for better hyperparameters in the search space.
2. Investigating oversampling techniques (such as SMOTE) and under sampling techniques to address the problem of class imbalance.
3. More thorough understanding of the dataset features.
4. Investigating feature importance and retraining the algorithm using k-most important features.
5. Examining correlation between categorical nominal variables using 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).