Customer Segmentation Report for Arvato Financial Services

Pedro Szloma Herr Zaterka

Abstract—Customer behavior and marketing campaign efficiency are a common topic in business, and a prosperous field for machine learning practitioners. As an example, the problem tackled is both a customer segmentation and marketing response problem. This problem is presented in two steps: A clustering problem (Unsupervised Learning) and a classification problem (Supervised Learning). The problem was approached in two separed ways, with the unsupervised learning results being used in the supervised task and the results presented compare a general population with a customer database as well as predicting the customer response to a marketing campaign and submiting predictions into a Kaggle competition.

Index Terms—Machine Learning, Customer Segmentation, Marketing, Udacity, Arvato, Kaggle.

1 Definition

1.1 Project Overview

THE proposal presented here is one of the 3 suggested by the Machine Learning Engineer Nanodegree, the goal of it is to analyse demographics data for customers of a mail-order sales company form Germany, to perform customer segmentation using unsupervised learning to identify what customer profile in the general population would be more likely to become new customers.

After that, using a mail-out dataset, a new model will be created to predict whether or not a customer who receives the mail will convert or not. This task is also part of a Kaggle competition [1] and its results will be submitted to it as well.

1.2 Problem Statement

The problem consists of 3 parts:

- Perform a unsupervised learning task to identify patterns in customer data and compare it to Germany's population and create a customer segmentation report
- Perform a supervised learning task to predict likelihood of people to respond positively yo a campaign
- Use the second model in a Kaggle competition

1.3 Metrics

1.3.1 Unsupervised Task

The unsupervised task had two separate models, Principal Component Analysis and K-Means Clustering. The PCA metric was the explained variance provided by the principal components, with a determined threshold of 95% of explained variance regarding the source dataset.

For the K-Means Clustering Model, two metrics where initially chosen, Bayesian information criterion (BIC) [2] and the Elbow Method [3], the latter ultimately being the one used as it provided a clearer optimum value.

1.3.2 Supervised Task

The evaluation metric for the supervised model will AUC for the ROC [4] curve, as it is the metric defined by the Kaggle competition to which the model will be submitted to.

2 Analysis

The analysis was carried mainly on the Unsupervised datasets, as the databases were larger and findings about them were transferred to the supervised task.

2.1 Data Exploration

The Data Exploration used four datasets mainly:

- **"Udacity_AZDIAS_052018.csv"**: Demographics data for the general population of Germany with 891,211 persons (rows) x 366 features (columns);
- **"Udacity_CUSTOMERS_052018.csv"**: Demographics data for customers of a mail-order company with 191,652 persons (rows) x 369 features (columns);
- **"DIAS Information Levels - Attributes 2017.xlsx"**: top-level list of attributes and descriptions, organized by informational category;
- **"DIAS Attributes - Values 2017.xlsx"**: mapping of data values for each feature in alphabetical order.

The former two present the data to be used for the unsupervised task and clustering analysis. The latter are provided as a description of the attributes presented in the first datasets. The datasets provided for the supervised task were "MAILOUT TRAIN" (with 42,962 rows and 367 columns) and "MAILOUT TEST" (with 42,833 rows and 366 columns).

2.1.1 Missing Values

The first step taken to clean the datasets were to check if any attributes were labeled as "unknown" or other *NA* equivalent and were not present as such in the datasets. Checking for the word "known" inside the *Meaning* column of the

Attributes data the following terms (with their recurrence) are shown: unkown (232), no transaction known (34), no transactions known(22). Which means *NA* values are not labeled as such. for that the following function was applied to get the missing attributes:

```
def get_unknown(attritutes: pd.DataFrame)
-> pd.DataFrame:

    na = attributes.ffill()

    na = na.loc[(na['Meaning'].
    str.contains('unknown')) |
    (na['Meaning'] == 'no transaction known') |
    (na['Meaning'] == 'no transactions known')]

    na = pd.concat([pd.Series(row['Attribute'],
    str(row['Value']).split(','))
    for _, row in na.iterrows()]).reset_index()

    na.rename(columns={'index':'Value',
    0: 'Attribute'}, inplace = True)

    return na
```

The *NA* data is presented in the following tables for unsupervised and supervised tasks' datasets, divided by row and column:

Statistics of NaN values per column (in %)		
Metric	AZDIAS (Gen. pop.)	Customers
Mean	15.89	24.76
Std. Dev.	20.63	19.75
Min	0.00	0.00
25%	8.24	24.31
50%	11.87	26.76
75%	16.62	29.21
Max	99.86	99.88

Statistics of NaN values per row (in %)		
Metric	AZDIAS (Gen. pop.)	Customers
Mean	15.90	24.76
Std. Dev.	21.30	30.52
Min	0.82	0.54
25%	5.46	4.88
50%	7.65	6.50
75%	11.20	65.58
Max	78.14	77.51

2.1.2 Data type per column

The **AZDIAS** dataset has 360 columns with numerical values and 6 columns with non-numerical values. Those being: CAMEO_DEU_2015, CAMEO_DEUG_2015, CAMEO_INTL_2015, EINGEFUEGT_AM, OST_WEST_KZ.

Taking a closer look at number of unique values:

- Feature D19_LETZTER_KAUF_BRANCHE has 35 values
- Feature CAMEO_DEU_2015 has 45 values
- Feature CAMEO_DEUG_2015 has 19 values
- Feature CAMEO_INTL_2015 has 43 values
- Feature EINGEFUEGT_AM has 5162 values
- Feature OST_WEST_KZ has 2 values

OST_WEST_KZ is a binary class, *EINGEFUEGT_AM* is a date column, *CAMEO_INTL_2015* is a numeric columns, which has *NA* values filled as a string "XX" or "X". The rest are categorical classifications.

Regarding the **CUSTOMERS** dataset, it has 3 columns that are not present in the general population data: PRODUCT_GROUP, CUSTOMER_GROUP and ONLINE_PURCHASE which are post-buy information and depend on the person being a customer to make sense.

How the categorical and missing data are dealt with will be adressed in section **3.1**.

2.2 Exploratory Visualization

Some data presented here was discussed in the previous section and its presented in a more visual way.

2.2.1 AZDIAS Dataset - General Population

For the AZDIAS dataset, figure1 shows the distribution of the percentage of missing values in each column. It can be seen that most columns have less then 20% of missing values, which is a good sign, but some outliers have close to 100% of missing values.

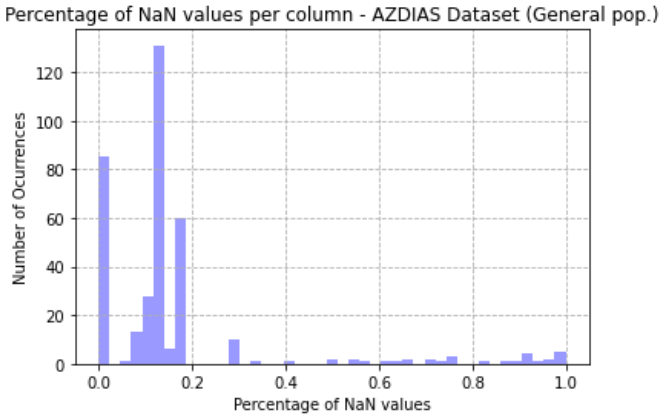

Fig. 1. Percentage of NaN values per column - AZDIAS Dataset

Analysing the rows, figure 2 shows that most rows also have less then 20% of missing values, but larger proportional values are more prominent.

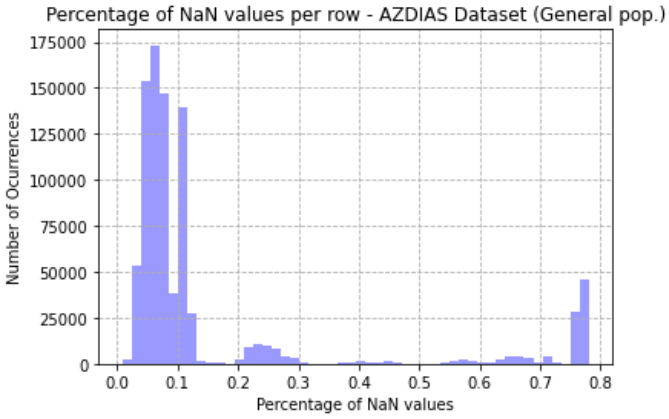

Fig. 2. Percentage of NaN values per row - AZDIAS Dataset

Figure 3 shows data types present in the dataset's features before preprocessing. Most columns are numerical, but

as discussed in the previous section, 6 of them have the ”object” data type.

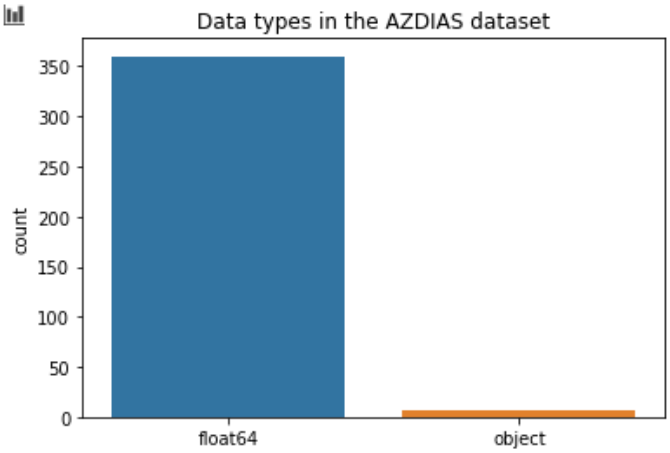

Fig. 3. Data types of columns - AZDIAS Dataset

One of the 6 categorical columns is a classification if it is from former FRG or GDR. The proportion between the classes in the general population is shown in figure 4

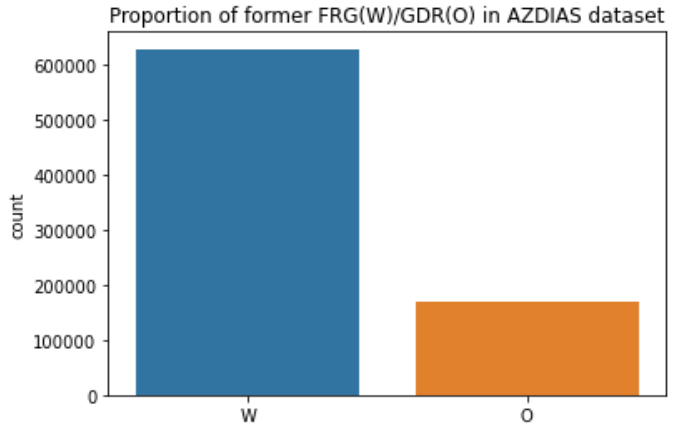

Fig. 4. Proportion of former FRG/GDR - AZDIAS Dataset

2.2.2 Customers Dataset

As seen with the AZDIAS dataset, the percentage of missing values per column for the customers dataset is shown in figure 5. For the customers dataset, the most common values lay between 20% and 40% of missing values, considerably higher than the dataset for the general population, and also contains some outlier values which near the 100% of missing values.

The number of missing values per row present a more similar behaviour when compared to the AZDIAS dataset. The distribution seen in figure 6 shows most values between 0% and 10%, which a considerable spike close to 80%.

For the data types, the customers dataset has more string columns, which indicates the type of customer and the product they bought. As seen in figure 7, most columns are still numeric, with data type float64.

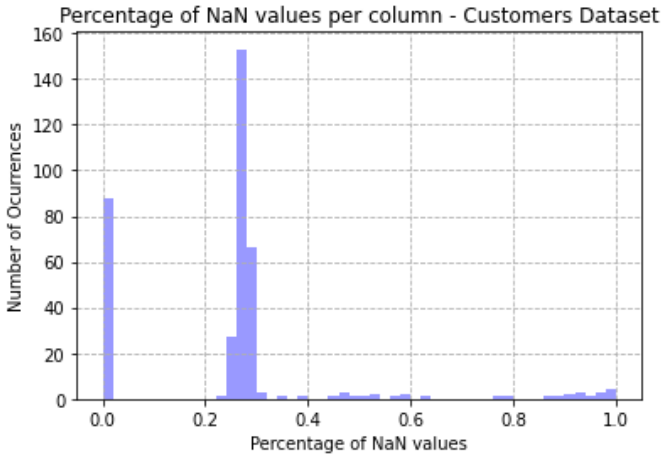

Fig. 5. Percentage of NaN values per column - Customers Dataset

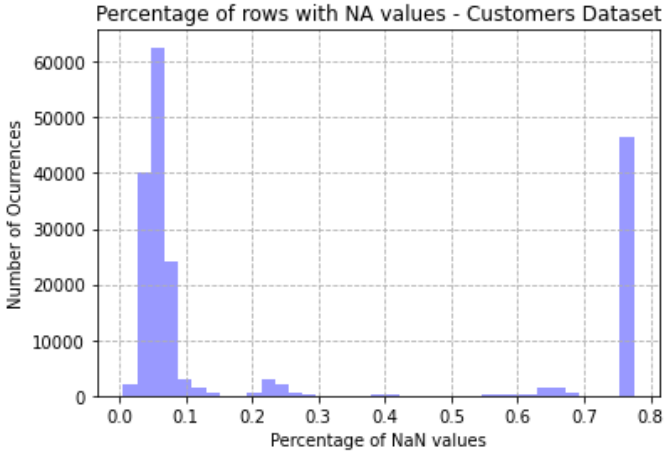

Fig. 6. Percentage of NaN values per row - Customers Dataset

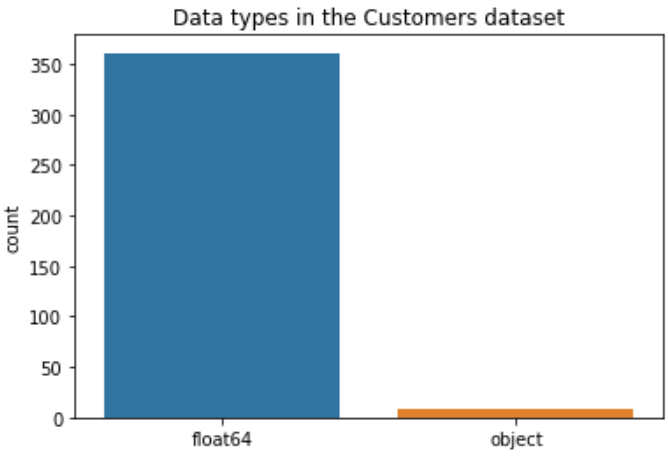

Fig. 7. Data types of columns - Customers Dataset

Figure 8 shows an interesting contrast with general population, as in proportion the customer database contains less former GDR entries, showing a potential concentration of a certain type of customer.

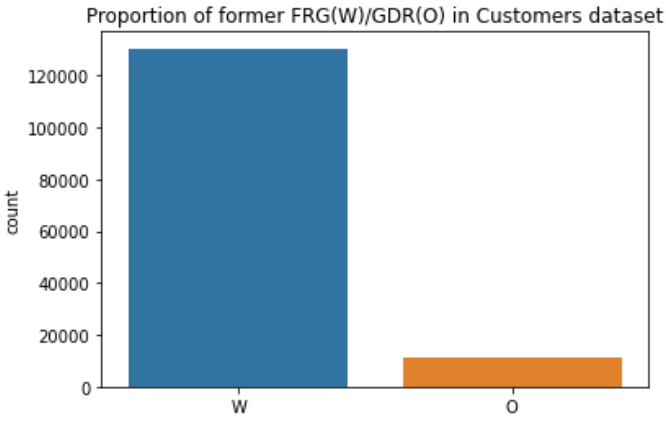

Fig. 8. Proportion of former FRG/GDR - Customers Dataset

2.2.3 Mailout Train Dataset

For the Mailout dataset, an important feature to analyse is the proportion between values in the RESPONSE variable. Figure 9 shows that the number of positive responses(1) is much lower than the number of negatives (0), representing 1.24% of total values.

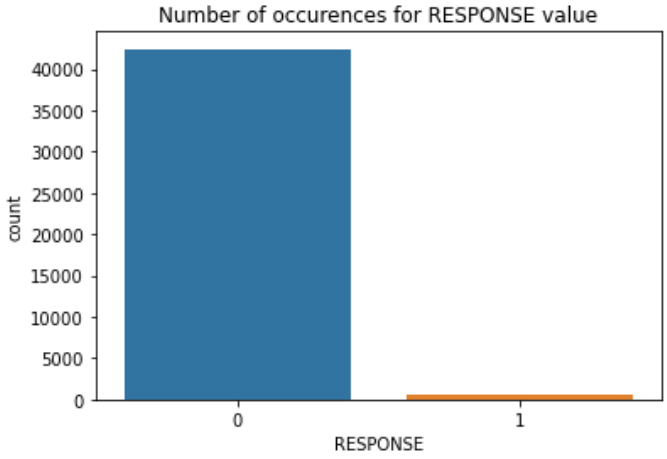

Fig. 9. Number of occurences for RESPONSE value

2.3 Algorithms and Techniques

The algorithms and techniques used differ for the Supervised and Unsupervised task, although the cluster classification was carried on to the supervised model.

2.3.1 Unsupervised Task

The main technique used in the unsupervised task, aside from the algorithms, is the filtering and scaling of the data.

The algorithms used where:

- **Principal Component Analysis** [5]:Reduce dimensionality and make it easier for the subsequent clustering method to work properly;
- **K-Means Clustering**: This unsupervised machine learning method helps finding patterns and classifying customers in the database. The number of clusters was defined by BIC and Elbow methods.

2.3.2 Supervised

For the supervised tasks, a plethora of models were tried, the results being firstly chosen using Repeated Stratified KFold cross-validation, with 5 splits and 3 repeats. The best perfoming models were then optimized using the Optuna [6] package.

As the classes to be predicted are very unbalanced (0.0125:1), a way to deal with it was also tested, using the synthetic minority over-sampling technique (SMOTE) [7].

- Regression Models:
  - **Logistic Regression** [8]: Simplest of the models, act both as a sanity check for the data and a baseline.
- Bagging Models:
  - **Random Forest Classifier** [9]: RF Classifier is the second most used algorithm in Kaggle competitions [10], thus it was deemed worthy to give it a shot.
- Boosting Models:
  Boosting models in general tend to perform well in Kaggle competitions [11], therefore some popular boosting models were tried, with the inclusion of CatBoost as suggested in the Capstone Proposal review. The included models are listed below:
  - **Gradient Boosting Classifier** [12]
  - **XGBoost Classifier** [13]
  - **LightGBM Classifier** [14]
  - **CatBoost Classifier** [15]
- Neural Networks
  - **TabNet** [16]: The authors state that this attention-based architecture can outperform boosting models in classification problems, so it was included as it had a very intuitive implementation on top of PyTorch called *pytorch-tabnet* (github.com/dreamquark-ai/tabnet)

2.4 Benchmark

The proposed benchmark for the supervised model was the top 10 score of the Kaggle Competition, a ROC AUC score of 0.808.

3 Methodology

3.1 Data Preprocessing

The data preprocessing had 3 main steps, defining criteria do drop columns and rows with too many *NA* values, filling those values, and defining the approach with the categorical columns.

The criteria was based on the AZDIAS (General population) dataset, as it is the most general dataset and will be the base for the clustering model.

3.1.1 Dropping NA columns and rows

From previous NA data, most columns and rows have little proportional number of NA values, with 75% having less then 17% of NA values for columns and 11.20% for rows. Although the max values are considerable (99.86% for columns and 78.14% for rows).

Dropping columns should have a more rigorous criteria, as unnecessarily removing one is removing more then 890,000 values, whilst dropping a row results in around 360 dropped values. For that reason, the criteria adopted for columns was to drop columns that surpass the 0.95 quantile of *NA* values (29%), and for rows, mean plus two standard deviations (51%) what in a normal distribution would correspond to dropping 2.5% of rows.

3.1.2 Filling NA values

For the imputing methods, two functions were used, one for boolean variables, which were filled with the mode, and numerical varibles, which were filled with the median, as it is more robust to outliers.

```
def fill_bool(df: pd.DataFrame, ref_df: pd.DataFrame
    ) -> pd.DataFrame:

    for column in df.columns:
        if df[column].nunique() == 2:
            try:
                df[column] = df[column].fillna(
                    ref_df[column].mode()[0])
            except:
                pass
    return df

def fill_num(df: pd.DataFrame, ref_df: pd.DataFrame)
     -> pd.DataFrame:

    for column in df.columns:
        if df[column].nunique() != 2:
            try:
                df[column] = df[column].fillna(
                    ref_df[column].dropna().median()
                    )
            except:
                pass
    return df
```

3.1.3 Assessing categorical variables

Categorical variables had a different approach for the supervised and unsupervised tasks.

As Principal Component Analysis does not work well with Label Encoder as well as One Hot Encoding, most categorical values were dropped for the unsupervised task. Based on the 6 columns identified as strings, the pproach by column was:

- D19_LETZTER_KAUF_BRANCHE: Columns will be dropped, as categories are many and last purchase sector may be more situational that may not reflect the potential customer's profile;
- CAMEO_DEU_2015: Has too many categories, and its nor necessarily hierarchical, so the column will be dropped;
- CAMEO_DEUG_2015: String rows seem like input error, will be treated as NAs;
- CAMEO_INTL_2015: String rows seem like input error, will be treated as NAs;
- EINGEFUEGT_AM: Input date will be dropped;
- OST_WEST_KZ: As there are only two categories, W will be replaced by 0 and O by 1.

As for the supervised task, the difference is that CAMEO_DEU_2015 and D19_LETZTER_KAUF_BRANCHE will be treated with one hot encoding, the rest will have the same approach as with the unsupervised task.

For the TabNet model, data will also be scaled, as most other models are based on decision trees, which generally are not affected by scaling the data.

3.2 Implementation

The data wrangling and preprocessing was done all with pandas with some custom preprocess functions, those different for unsupervised and supervised learning as the assessment of the categorical values as well as the presence of the "RESPONSE" column required it.

Also, the unsupervised task was incorporated in the supervised data pipeline to use cluster information in the models.

3.2.1 Unsupervised Task

For metrics, unsupervised learning required an implementation of Bayesian Information Criterion as well as an automated way to identify the most significant change in slope with the Elbow method. The *compute_bic* function was implemented as follows:

```
def compute_bic(kmeans,X):

    centers = [kmeans.cluster_centers_]
    labels  = kmeans.labels_

    m = kmeans.n_clusters

    n = np.bincount(labels)

    N, d = X.shape


    cl_var = (1.0 / (N - m) / d) * sum([sum(distance
        .cdist(X[np.where(labels == i)], [centers
        [0][i]],
             'euclidean')**2) for i in range(m)])

    const_term = 0.5 * m * np.log(N) * (d+1)

    BIC = np.sum([n[i] * np.log(n[i]) -
              n[i] * np.log(N) -
             ((n[i] * d) / 2) * np.log(2*np.pi*
                 cl_var) -
             ((n[i] - 1) * d/ 2) for i in range(m)])
                  - const_term

    return BIC
```

Which resulted in the figure 10:

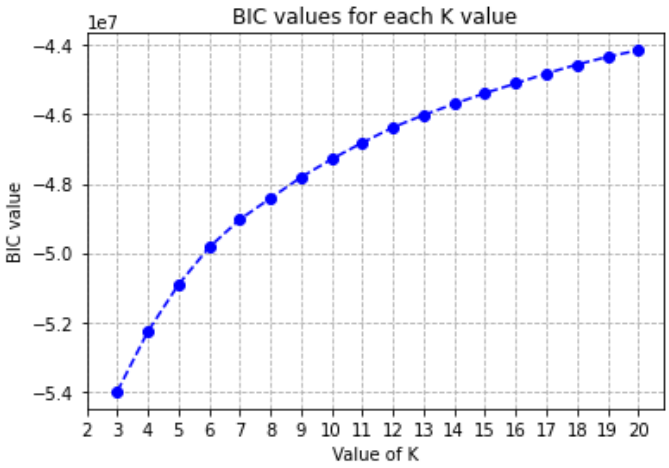

Fig. 10. BIC Values for K-Means Models

And for the elbow method, the average within-cluster sum of squares was calculated for each K ranging from 3 to 20, with the angle threshold of 0.99 radians. The results can be seen in figure 11:

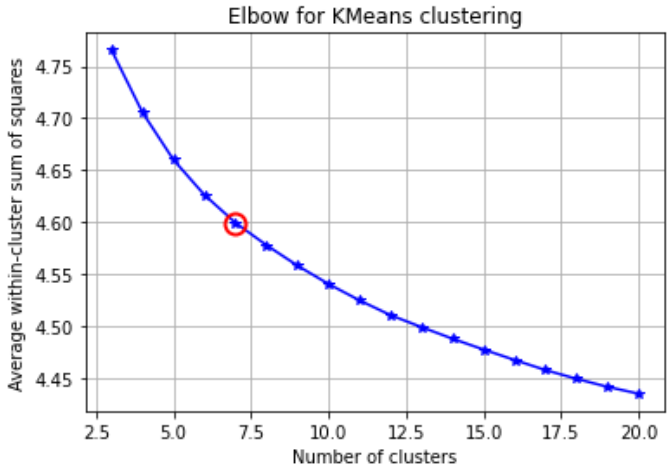

Fig. 11. Elbow Method for K-Means Models

All code implementation can be seen in the *Jupyter Notebook* and *aux_functions.py* files, present in the Github repository.

After clustering and comparing customer data with general population, clusters most prevalent in the customer dataset were analysed, having the main features present in them recovered.

3.2.2 Supervised Task

For the supervised task, most models were compatible with *ScikitLearn*, so the models were evaluated using *RepeatedStratifiedKFold* for cross validation and *roc_auc_score* for the evaluation metric.

Logistic Regression, Gradient Boosting Classifier and Random Forest Classifier were imported from *ScikitLearn*. The remaining models (XGBClassifier, LGBMClassifier and CatBoostClassifier) were imported from their respective's packages.

The exception to this pipeline was the TabNet model, using the implementation from *PyTorchTabNet*. As it was a more laborous model to train, it was trained in a separated script and data was scaled pre-training, as well as the *ROC AUC* metric was defined as follows:

```
from pytorch_tabnet.metrics import Metric
from sklearn.metrics import roc_auc_score

class ROCAUC(Metric):
    def __init__(self):
        self._name = "rocauc"
        self._maximize = True

    def __call__(self, y_true, y_score):
        return roc_auc_score(y_true, y_score[:, 1])
```

3.3 Refinement

First approach to refine and improve the models was the use of SMOTE to counterbalance the disproportion between classes to predict. This approach improved considerably the Random Forest performance, but added little to no improvement in the boosting models.

The opmization process was more challenging, as *RepeatedStratifiedKFold* was to computationally expensive to run with Optuna. For that, the optimization criterion was implemented with a split of the data into training e validation sets, the models were trained with the training set and evaluated regarding their ROC AUC score when predicting in the validation set.

To save time, an early stopping callback was defined for the optimization loop, it can be seen below, with OPTUNA_EARLY_STOPPING being defined as 150 steps without improvement:

```
class EarlyStoppingExceeded(optuna.exceptions.
    OptunaError):

    early_stop = OPTUNA_EARLY_STOPPING
    early_stop_count = 0
    best_score = None


def early_stopping_opt(study, trial):


    if EarlyStoppingExceeded.best_score == None:
        EarlyStoppingExceeded.best_score = study.
            best_value

    if study.best_value < EarlyStoppingExceeded.
        best_score:
        EarlyStoppingExceeded.best_score = study.
            best_value
        EarlyStoppingExceeded.early_stop_count = 0
    else:
        if EarlyStoppingExceeded.early_stop_count >
            EarlyStoppingExceeded.early_stop:
            EarlyStoppingExceeded.early_stop_count =
                0
            best_score = None
            raise EarlyStoppingExceeded()
        else:
            EarlyStoppingExceeded.early_stop_count=
                EarlyStoppingExceeded.
                early_stop_count+1

    return
```

Also, Optuna's optimization did not seem to work well with CatBoost, as it was very memory demanding when working in conjunction, what compromised the feasibility of achieving the best CatBoost model possible.

4 Results

4.1 Model Evaluation and Validation

4.1.1 Unsupervised Task

For the unsupervised task, two models where used to transform the data, first evaluation was the explained variance of the Principal Component Analysis. Figure 12 shows that the chosen number of components was sufficient to match the defined threshold of 95%.

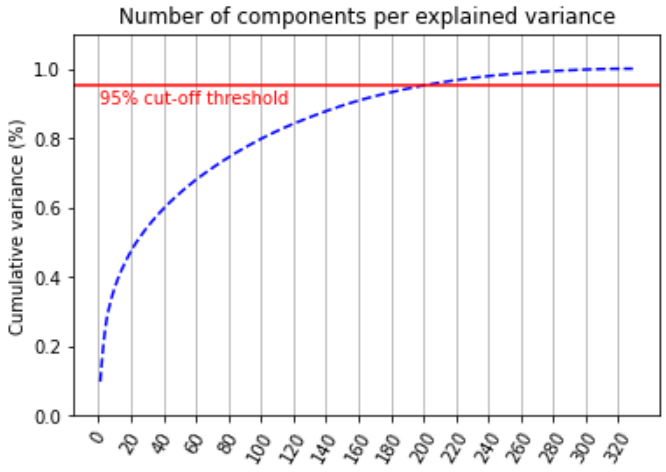

Fig. 12. Model performance for cross-validation

As for the K-Means Clustering model, metrics used to define number of clusters were presented in section **3.2.1**, but figure 13 shows no particular unbalance between clusters, showing that the method successfully classified the entries in an almost regular set of clusters.

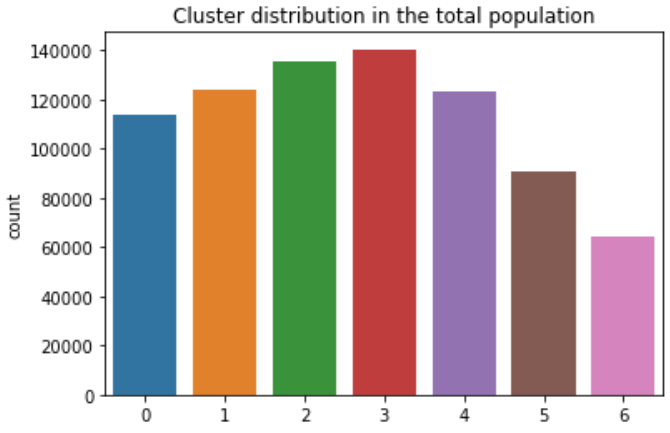

Fig. 13. Values by Cluster for the AZDIAS dataset

4.1.2 Supervised Task

The supervised task can be divided in the traditional Machine Learning models and the Deep Learning model.

For the Regression, Bagging and Boosting models, models were firstly evaluated with ScikitLearn's RepeatedStratifiedKFold cross validation. As the dataset is heavily unbalanced, SMOTE was used, both only oversampling the less frequent class and then both oversampling the less frequent class and undersampling the more frequent one. The table below summarises the results of the crossvalidation. For clarity in the table and images, the models names were abbreviated as follows:

- LR: Logistic Regression
- GBM: Gradient Boosting Classifier
- RGR: Random Forest Classifier
- XGB: XGBoost Classifier
- LGB: LightGBM Classifier
- CAT: CatBoost Classifier

ROC AUC Score (mean ± std)			
Model	No SMOTE	SMOTE(Over)	SMOTE(Over&Under)
LR	0.69 ± 0.02	0.68 ± 0.02	0.69 ± 0.02
GBM	0.77 ± 0.02	0.78 ± 0.02	0.77 ± 0.02
RFR	0.61 ± 0.02	0.60 ± 0.02	0.68 ± 0.02
XGB	0.74 ± 0.02	0.74 ± 0.02	0.74 ± 0.01
LGB	0.76 ± 0.02	0.77 ± 0.02	0.77 ± 0.02
CAT	0.77 ± 0.02	0.77 ± 0.02	0.77 ± 0.03

Figure 14 shows a boxplot for each model with the results with no SMOTE applied. Random Forest Classifier seemed to not deal well with the unbalanced data, being outperformed by the Logistic Regression Model. Boosting Models performed similarly, with a slight edge by CatBoost and Gradient Boosting, followed closely by LightGBM.

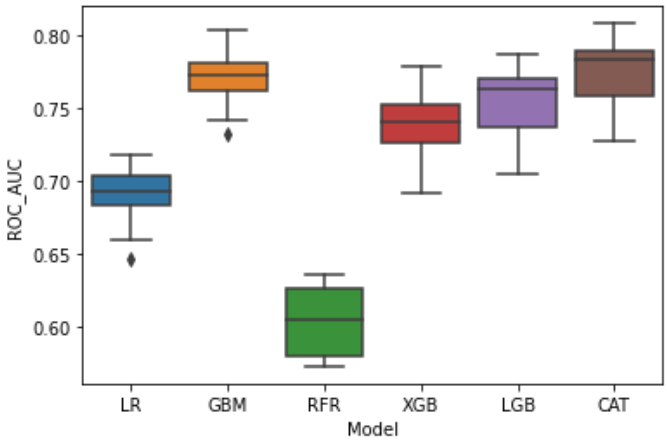

Fig. 14. Model performance for cross-validation

The results applying over and under sampling can be seen in figure 15, where performance for the Random Forest model was the most affected, having a considerable increase in performance, but still being outperformed by the other models.

From the results obtained, CatBoost, LightGBM and Gradient Boosting were selected for the optimization step. In this step a study was defined using the same cross

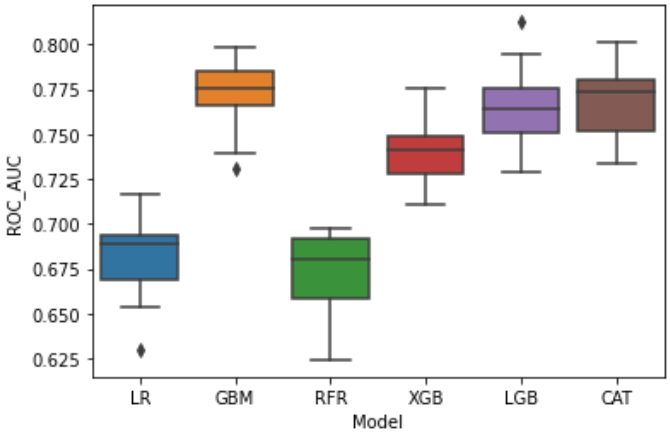

Fig. 15. Model performance for cross-validation using SMOTE

validation used for model selection, but the computational cost made it unfeasible, so the approach changed for a split in training e validation sets and optimization with the ROC AUC score of the predictions in the validation set.

Although CatBoost was the best model in the crossvalidation, results in the optimization were not as good, so the other two models were combined in the following optuna study:

```
def objective(trial):

    model_type = trial.suggest_categorical('
        model_type', ['lightgbm', 'gradientboost'])

    if model_type == 'lightgbm':
        param = {
            'objective': 'binary',
            'metric': 'binary_logloss',
            'verbosity': -1,
            'boosting_type': 'gbdt',
            'lambda_l1': trial.suggest_loguniform('
                lambda_l1', 1e-8, 10.0),
            'lambda_l2': trial.suggest_loguniform('
                lambda_l2', 1e-8, 10.0),
            'num_leaves': trial.suggest_int('
                num_leaves', 2, 256),
            'feature_fraction': trial.
                suggest_uniform('feature_fraction',
                0.4, 1.0),
            'bagging_fraction': trial.
                suggest_uniform('bagging_fraction',
                0.4, 1.0),
            'bagging_freq': trial.suggest_int('
                bagging_freq', 1, 7),
            'min_child_samples': trial.suggest_int('
                min_child_samples', 5, 100),
        }

        classifier_obj = LGBMClassifier(**param)

    else:
        param= {'n_estimators': trial.suggest_int('
            n_estimators', 50, 1500),
                'max_features': trial.
                    suggest_categorical('
                    max_features', \
                ['auto', 'sqrt', 'log2']),
                'max_depth': trial.suggest_int('
                    max_depth', 3, 20),
                'min_samples_split': trial.
                    suggest_int('min_samples_split',
                     2, 10),
                'min_samples_leaf': trial.
                    suggest_int('min_samples_leaf',
                    1, 5)
                }
        classifier_obj = GradientBoostingClassifier
            (**param)


    model_try = classifier_obj.fit(X_train, y_train)

    return roc_auc_score(y_valid, model_try.predict(
        X_valid))
```

With the best found model being a LightGBM model with the following parameters:

- lambda_l1: 0.023595797435052576
- lambda_l2: 0.10444065712521429
- num_leaves: 2
- feature_fraction: 0.9228058648353306
- bagging_fraction: 0.7627300352038621
- bagging_freq: 7
- min_child_samples: 13
- scale_pos_weight: 83.54634055886325

An intesting point to note is that the parameter responsible for dealing with unbalanced data, *scale_pos_weight* ended up in a very close value to the calculated value (79.16 after the preprocessing).

ROC curve and AUC values for the chosen model can be seen in figures 16 and 17, which very similar performances.

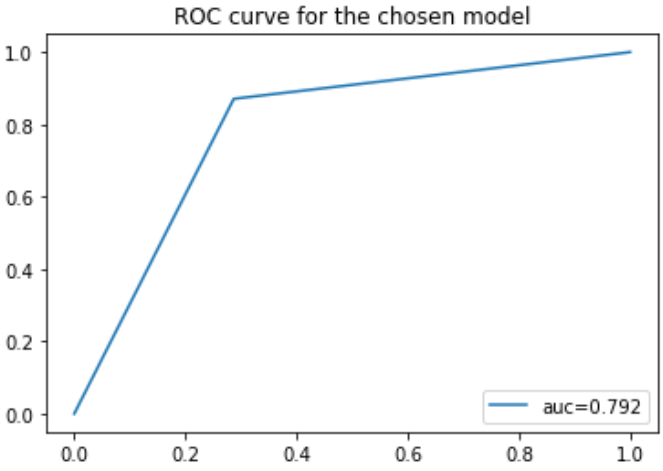

Fig. 16. ROC Curve for the final model

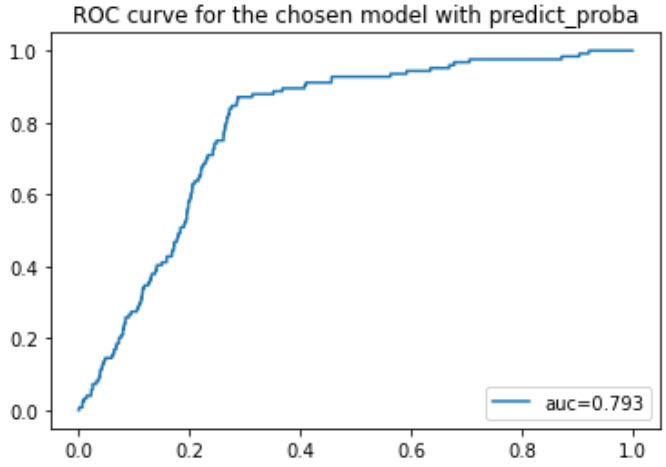

Fig. 17. ROC Curve for the final model predicting probability

In parallel, a TabNet model was developed, but due to time restrictions, a proper hyperparameter optimization could not be done. The model was defined as:

```
clf = TabNetClassifier(optimizer_fn=torch.optim.Adam
    ,
                       optimizer_params=dict(lr=5e
                           -3),
                       scheduler_params={"step_size"
                           :20,
                                         "gamma"
                                             :0.7},
                       scheduler_fn=torch.optim.
                           lr_scheduler.StepLR)
```

Using a small step size and PyTorch's implementation of the Adam optimizer [17]. The training loop was easily set with *pytorch-tabnet*:

```
clf.fit(
    X_train=X_train, y_train=y_train.values,
    eval_set=[(X_train, y_train.values), (X_valid,
        y_valid.values)],
    eval_name=['train', 'valid'],
    eval_metric=['rocauc'],
    max_epochs=1000 , patience=200
    )
```

One recurrent problem observed was the tendency to overfitting, which can be circumvented by regularization methods or dropouts, but due to time constraints they could not be tested. Alas, figure 18 shows the evolution of the ROC AUC metric along the training process, where a clear overfit is seen with training scores approaching one and validation scores decreasing.

Figure 19 shows the model loss by epoch, which steadly decreased during training.

The best score was achieved in epoch 33 with a ROC AUC score on the validation dataset of 0.77889. Even being worse then LightGBM, this was a very untuned and premature model, it was noticed that TabNet has the potential to surpass the presented boosting models performance given the proper tuning and optimization.

4.2 Justification

4.2.1 Unsupervised Task

The objective of the unsupervised task was to identify characteristics in the customers database using a clustering

Fig. 18. ROC AUC score by epoch for training and validation datasets

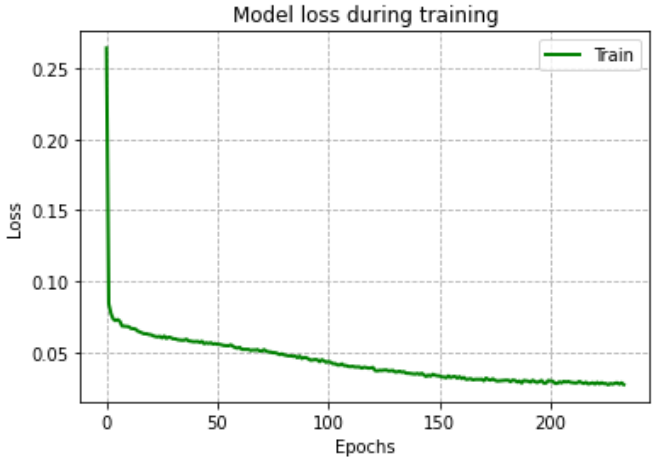

Fig. 19. Model loss by epoch

method comparing it to the general population. That being said, figure 20 shows a high concentration of custumers into the Cluster 0, with an absence of customers in clusters 3, 5 and 6.

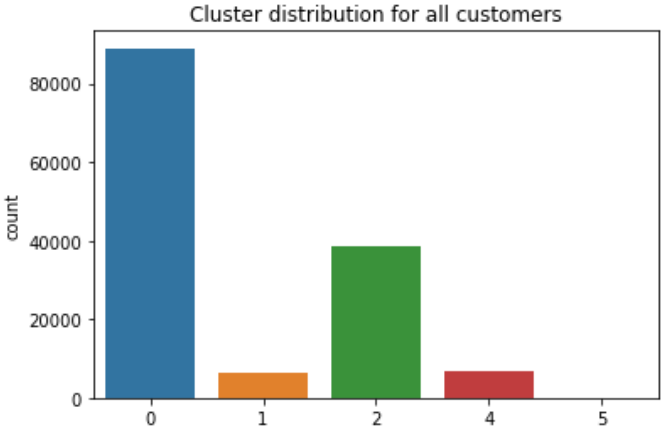

Fig. 20. Clusters for the Customer Database

When dividing the customer database by customer group, as seen in figure 21, clusters 0 and 2 show promi-

nence, but the difference is less significant for single buyers than for multi buyers. Cluster 4 also tend to be more present with single buyers.

Fig. 21. Cluster for the Customer database by Customer Group

When splitting by product group, proportions seem to be similar between the three groups, as seen in figure 22, with a notable lack of customers of "cosmetic_and_food" in cluster 4.

Fig. 22. Cluster for the Customer database by Product Group

At least, a proportional comparison between general population and customers can be seen in figure 23, with the already notable concentration of customers in the cluster 0, as well as considerable amount in cluster 2.

This suggests that those clusters can engulf a target demographic fitting for the products offered, and thus, people classified in those clusters have more potential to become customers than people classified as cluster 3, 5 or 6.

Getting the main original features that contribute for a person to be classified as a cluster 0 person, the top 5 features are:

- AKT_DAT_KL: Not present in the Attributes dataset
- ANZ_HAUSHALTE_AKTIV: number of households known in the building
- ANZ_HH_TITEL: number of academic title holders in building

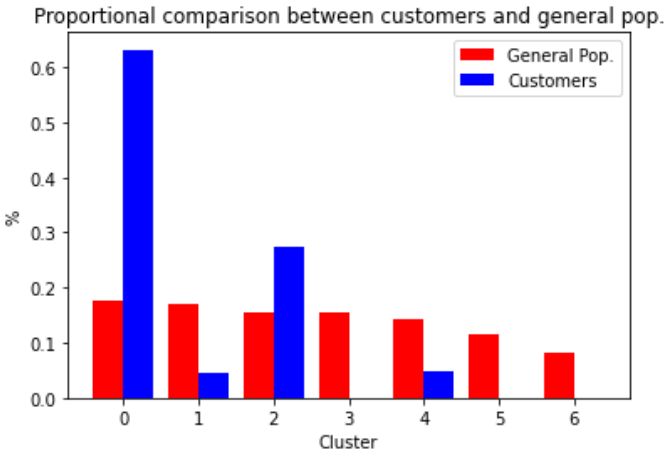

Fig. 23. Comparison between customers and general population clusters

- ANZ_KINDER: number of children in the holsehold
- ANZ_PERSONEN: number of adult persons in the household

4.2.2 *Supervised Task*

The supervised task was evaluated by the models performance in the **Udacity+Arvato: Identify Customer Segments**, the achieved results can be seen in the table below:

Performance by model	
Model	Score
LightGBM	0.80261
CatBoost	0.79376
TabNet	0.72468

The chosen submission was the LightGBM, which achieved a ROC AUC score of 0.80261, being 0.0054 short of the proposed benchmark of 0.808. While I believe that with more time the target was achievable, the score did not end up that far, but it only got to the 62^{nd} position, being in the top 19%.

5 Closing Remarks and Potential Development

The work can be further developed and improved with a little bit more tuning of the supervised methods as well as a deeper study into *pytorch-tabnet*'s implementation, to better tune the model. More preprocessing layers can be added to extract patterns and information as well as different thresholds for the unsupervised models.

CatBoost also showed as a great algorithm, which can also outperform LightGBM given proper time and tuning.

REFERENCES

[1] "Kaggle competition: Udacity+arvato: Identify customer segments." https://www.kaggle.com/c/udacity-arvato-identify-customers. Accessed: 2021-01-25.

[2] G. Schwarz *et al.*, "Estimating the dimension of a model," *Annals of statistics*, vol. 6, no. 2, pp. 461–464, 1978.

[3] R. L. Thorndike, "Who belongs in the family? psychometrika [internet]," 1953.

[4] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[5] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 61, no. 3, pp. 611–622, 1999.

[6] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[8] P.-F. Verhulst, "Mathematical researches into the law of population growth increase," *Nouveaux Mémoires de l'Académie Royale des Sciences et Belles-Lettres de Bruxelles*, vol. 18, pp. 1–42, 1845.

[9] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1, pp. 278–282, IEEE, 1995.

[10] Kaggle, "State of machine learning and data science 2020," 2020.

[11] L. Biewald, "Gradient descent podcast: How to win kaggle competitions with anthony goldbloom," Sept. 2020.

[12] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[13] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), pp. 785–794, ACM, 2016.

[14] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, pp. 3146–3154, Curran Associates, Inc., 2017.

[15] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," *arXiv preprint arXiv:1706.09516*, 2017.

[16] S. O. Arik and T. Pfister, "Tabnet: Attentive interpretable tabular learning," *arXiv preprint arXiv:1908.07442*, 2019.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.