Scuola universitaria professionale
della Svizzera italiana
SUPSI

University of Applied Sciences and Arts of Southern Switzerland
Department of Innovative Technologies

Data Challenge 3

ONLINE SALES DATASET CHURN ANALYSIS

Christian Berchtold
christian.berchtold@student.supsi.ch

Christian Pala
christian.pala@student.supsi.ch

Professor: Sandra Mitrovic
SUPSI, Lugano Switzerland

06/11/2022

Abstract

Project report for the first data challenge project on churn analysis and prediction, with a focus on feature engineering and selection. The topics covered in the report are:

1. *Problem definition*
2. *Dataset analysis*
3. *Preprocessing*
4. *Feature engineering*
5. *Feature selection*
6. *Dimensionality reduction.*
7. *Experimental setup and results*
8. *Hyper-parameter tuning*
9. *Critical reflections*
10. *Conclusions*

The entire project was developed with the python language, version 3.10, following the material provided by professor Mitrovic [1]

Table of Contents

List of Figures

1. Problem definition

We received, from professor Mitrovic, an online sales dataset, divided into two excel sheets, sorted by invoice numbers, containing:

- stock codes
- product descriptions
- quantities of the product bought
- the dates of the invoices.
- prices for the products
- the customer ID
- the country of purchase

from an unspecified company, for a period ranging from December 2009 to December 2011.

Our main objectives were to define a churn rate, extract useful features that may help in predicting it, do feature selection on said features, test the results of our work with a model of our choice and present our findings visually with dimensionality reduction.

1.1 Churn definition

Our initial idea, for the churn variable, defined by the Merrian-Webster dictionary as "a regular, quantifiable process or rate of change that occurs in a business over a period of time as existing customers are lost and new customers are added", was to use a moving window of thirty days. This follows the method used with a classic subscription model, where churn is defined as not renewing a periodic subscription. This proved problematic since many customers, which regularly bought products in the period we had data for, would be classified as churners in one of our windows and non-churners in another. We studied the distribution of churned customers based on different hard thresholds. We defined a hard threshold, as a calendar date after which, if the customer does not buy products, he is classified as a churner. After our initial data exploration, data preprocessing and discussion with professor Mitrovic, we settled on a reasonable division, assuming if the customer did not buy anything for more than half of the available data period, starting from the tenth of December 2010, we would consider them churners. This produced a 27% churn rate in the two years available in the dataset, with a binary classification task to distinguish whether a customer was a churner or not.

2. Dataset analysis

To better understand the data we were given, we started with some basic data exploration (EDA). The first task we tackled was loading the dataset.

2.1 Data loading

We wrote a small function to save the provided excel sheets in a pandas DataFrame, we included a check for the operating system, since we worked on two different environments. We also provided an option to load the two comma-separated values (csv) file sheets for the years 2009-2010 and 2010-2011 separately, to speed up the process, if those files were provided. Finally, we converted the date to a DateTime object. Having a central function to load and save the data helped a lot with consistency.

2.2 Initial dataset

Checking the initial dataset, after casting dates from strings to DateTime objects, we found some notable problems:

- 243007 missing values for customer IDs and 4382 missing descriptions.
- prices saved as strings, with commas instead of dots to mark fractional values
- both negative prices and quantities

2.2.1 Numerical variables

After correcting the problem with the `Price` column, we had two numerical variables to examine, we briefly plotted distributions and boxplots, but given the problem with negative numbers and the adjustments we knew we would need to make, it seemed premature to start making judgments on outliers and distribution shapes, we could, however, check the correlation between the two variables:

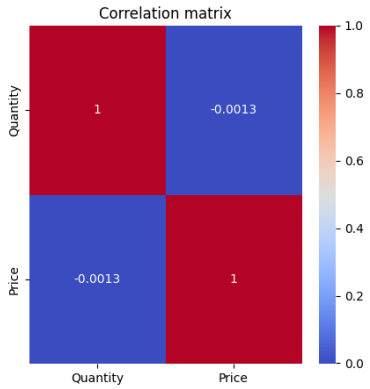


Figure 1: Correlation between quantity and price

We found no significant correlation between the two, which informed our later choices.

2.2.2 Categorical variables

We initially found 5305 products in the dataset, just based on visual inspection we noticed there was a lot to clean. Christian Berchtold noted an interesting point, stock codes which just differed by a single letter, were the same product with minor variations. Regarding the country of purchase, the vast majority of the orders were made in the United Kingdom (almost 92%). In general, examining the other countries, there is a lot of data about sales made in Europe, with some presence from English-speaking countries like Australia.

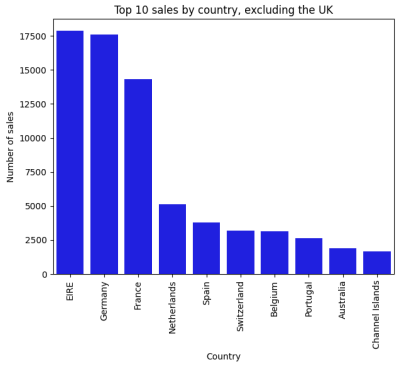
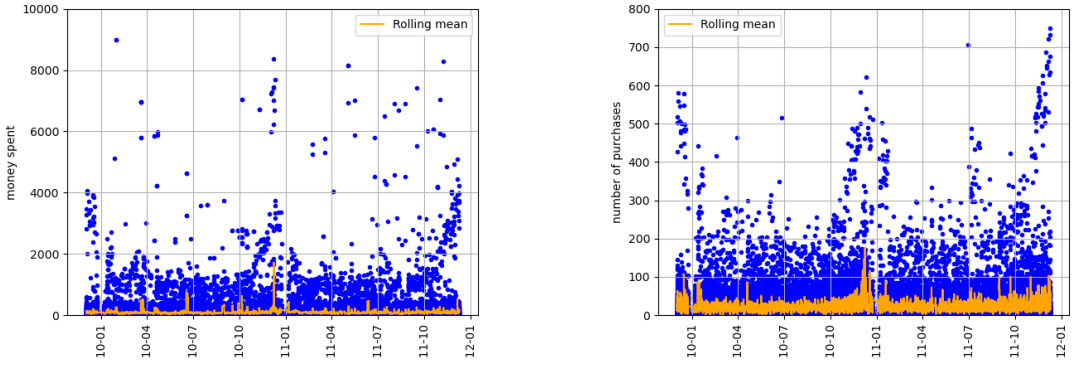


Figure 2: Number of sales by country, excluding the United Kingdom

2.2.3 Trends

Since this was a sales dataset we wanted to get a closer look at the trends over the 2-year period. Plotting the total spent and the number of orders, especially with the rolling mean, the peaks in the three Christmas periods present in the dataset, 2009, 2010, and 2011 respectively are highlighted.[1]



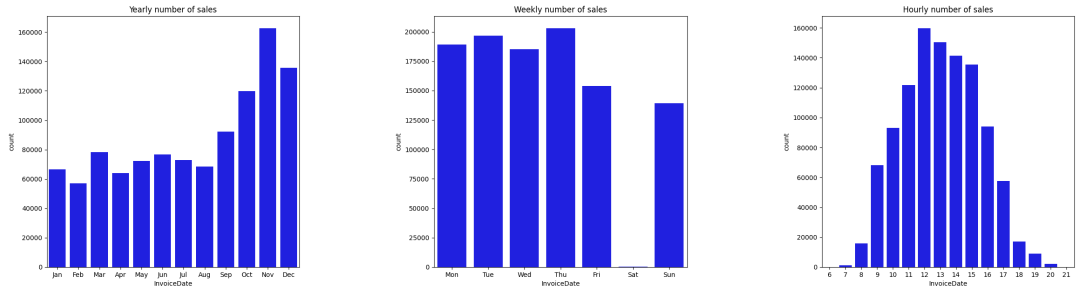
(a) Total money spent

(b) Total number of purchases

Figure 3: Two year trends

We also plotted a few histograms, to see the distributions of the monthly, daily, and hourly sales. The monthly sales confirm what we noticed in the first two plots. We briefly researched the topic and having strong November sales months seems to be the normal behavior for online stores, it also makes sense if we consider black Fridays and other November promotional events. We were particularly surprised by the results in 4b since it appears the store is closed or does not process orders on Saturdays, which seems quite unusual. Our running hypothesis was that Saturdays are the employee rest days.

[1]The plots have been limited on the Y axis, to have better readability.



(a) Monthly sales (b) Weekly sales (c) Hourly sales

Figure 4: Sales distribution

3. Data preprocessing

As with all real data and as mentioned in the previous chapter, there were a lot of problems with the initial data. The dataset was split into two sheets, each covering roughly a year. After a closer inspection, we noticed that the year span began from the $9^{\text{th}}/10^{\text{th}}$ of December and ended on the 31^{st} of December of the next year. This made us think a duplicate check was necessary. This proved true, the rows covering the period from 09/12/2010 to 31/12/2010 of the first sheet were repeated in the second file.

3.1 Missing customer identification numbers

The first big issue we encountered was customers with missing IDs. Given our interest in the dataset, it did not make sense to work with anonymous customers. During the first week of the project, Christian Berchtold tried using k-nearest-neighbor imputation to recover missing IDs, based on similarities with other rows, but the results we obtained did not make sense. We hypothesized that the missing IDs are mainly customers who decided to buy products without accounts, and thus unrecoverable. Luckily our dataset was fairly large to begin with, with 1067371 observations, hence we dropped the 243007 rows with missing customer IDs. Customer IDs were saved as strings in the dataset, we cast them to integers after cleaning, which made working with them as indexes easier.

3.2 Missing descriptions

We also found 4382 missing product descriptions. In this case, they could all be recovered by matching the stock code with instances where the description was provided.

3.3 Cancelling orders

Professor Mitrovic told us invoice numbers starting with C were canceling orders. They needed to be matched with regular invoices, which needed to be adjusted, with either total or partial cancellations. Since canceling invoice numbers and affected orders could not be matched using their invoice numbers, this proved to be the hardest part of our preprocessing pipeline. Given the task, our main interest was at the customer level, hence our initial approach was to correct the aggregated values for each customer, which could be easily implemented since both orders provided the customer ID. This later proved to be a problem when working on the disaggregated dataset, therefore Christian Pala implemented a function to recover the original order from the canceling one, with the assumption that both orders were executed in a 3-day time window. Following this procedure, we managed to find the likeliest original order for almost 19000 of the 21000 canceling orders, we had to delete the other 2000 unmatched ones.

3.4 Price and quantity imputations

We found some cases where the prices for products were set or rounded to 0, we imputed most of the values by matching the stock code where the price was positive, and deleted those that we could not recover. The remaining rows with negative quantities were also deleted.

3.5 Data cleaning

We found rows not containing products, including adjustments, postage charges, carriage charges, manual orders, sample orders, test orders, and a stock code used for testing, we removed all of them. Following and double-checking Christian Berchtold's observation about product variation stock codes differing only by an ending alphanumeric letter, we decided to consider all these stock codes as the same product.

3.6 Results

We cast the data in our preferred data types, in preparation for the feature engineering task, namely:

1. Invoice as an integer
2. StockCode as an integer
3. Description as a string
4. Quantity as an integer
5. InvoiceDate as a pandas DateTime object.
6. Price as 2 digits after the comma floating point number.
7. CustomerId as an integer
8. Country as an enumerator, matching each country with a value.

We obtained a data frame with seven features and 800901 rows, this is a snapshot of the dataset we produced:

Invoice	StockCode	Description	Quantity	InvoiceDate	Price	CustomerId	Country
499763	15056	EDWARDIAN PARASOL BLACK	1	2010-03-02 13:08:00	5.95	12346	38
499763	15056	EDWARDIAN PARASOL NATURAL	1	2010-03-02 13:08:00	5.95	12346	38
499763	15056	EDWARDIAN PARASOL PINK	1	2010-03-02 13:08:00	5.95	12346	38
499763	20679	EDWARDIAN PARASOL RED	1	2010-03-02 13:08:00	5.95	12346	38
499763	20682	RED SPOTTY CHILDS UMBRELLA	1	2010-03-02 13:08:00	3.25	12346	38
513774	20685	DOORMAT RED SPOT	1	2010-06-28 13:53:00	7.49	12346	38
513774	21523	DOORMAT FANCY FONT HOME SWEET HOME	1	2010-06-28 13:53:00	7.49	12346	38
513774	21524	DOORMAT SPOTTY HOME SWEET HOME	1	2010-06-28 13:53:00	7.49	12346	38
513774	21955	DOORMAT UNION JACK GUNS AND ROSES	1	2010-06-28 13:53:00	7.49	12346	38
513774	22365	DOORMAT RESPECTABLE HOUSE	1	2010-06-28 13:53:00	7.49	12346	38
513774	22366	DOORMAT AIRMAIL	1	2010-06-28 13:53:00	7.49	12346	38
513774	22660	DOORMAT I LOVE LONDON	1	2010-06-28 13:53:00	7.49	12346	38

Figure 5: Dataset for feature engineering

4. Feature engineering

Feature engineering and feature selection were our main interests for this project. In order to perform feature selection sensibly, a large number of features was required, to engineer them we followed the three main approaches requested by professor Mitrovic, namely:

- graph creation
- natural language processing
- time series

together with our initial approach, described below.

4.1 Recency, frequency, and monetary model

Recency, frequency, and monetary (RFM) analysis is a marketing tool that companies usually employ to monitor their customer habits, developed in the nineties [2], in order to perform targeted marketing campaigns and identify their best clients. The three quantitative factors are:

1. **Recency**: how recent the customer purchase is.
2. **Frequency**: how often the customer orders something.
3. **Monetary value**: how much the customer spends on orders.

From the preprocessed dataset we built an aggregated one, grouping by customers via their customer ID. For each customer, we then created features for the total quantity purchased, the total amount spent, the number of different products bought, the number of times they ordered something, a list of all descriptions of the items they bought, and a list of countries they bought from. To implement our churn status according to the definition we settled on, we also extracted the last purchase date for each customer, we could then easily create a Boolean variable, `CustomerChurned`, depending on whether the customer last purchased something before or after our reference date. Since the last purchase date was used to create our target label, we dropped that column from the aggregated data frame. The way we defined the train test split, combined with the way we generated the churn variable, made it impossible to define a variable for recency. It also limited us from implementing other features we thought about, like customer loyalty. We will explore this more in the critical reflection chapter. We obtained a dataset with information on 5849 customers, 1590 of which we labeled as churned, below we present a snapshot of the result, the feature names are shortened for viewing purposes:

Cus...	Recency	NumberOfPu...	TotalS...	TotalQu...	NumberOfPr...	Desc...	Country	Custome
12346	164	24	169.36	24	22	EDWARDIAN P…	38	True
12347	2	253	5633.32	3286	119	JUMBO BAG W…	16	False
12348	73	46	1658.4	2704	24	SET OF 72 S…	12	False
12349	42	172	3678.69	1621	136	WRAP PINK F…	18	False
12350	37346	16	294.4	196	16	BLUE POLKAD…	26	False
12351	10	21	300.93	261	21	BOX OF 9 PE…	39	True
12352	10	88	1609.21	651	68	SET/10 RED …	26	False
12353	43	24	406.76	212	23	FIRST CLASS…	2	False
12354	37346	58	1079.4	530	58	BLUE POLKAD…	32	False
12355	202	35	947.61	543	29	GOLD TEDDY …	2	False

Figure 6: Aggregated dataset

4.2 Graphs

Creating graphs and extracting features from the produced morphology is a novel approach for us. Initially, we were not sure if we needed to split the creation of the graph between the training and the testing sets to avoid information about the test set being used in training, after extracting the features from the graphs. Discussing with professor Mitrovic, we determined we could assume we were in a transductive case for this part of the project, meaning we knew about the characteristics of the training and testing sets a priori, so we could generate graphs on the whole dataset, which simplified the task.

4.2.1 Graph creation

Our initial approach was to create a graph with customers as nodes, using the total spent in common with other customers, as a criterion to create edges. Both customers would spend different amounts if they shared a basket of products, so we used the total amount spent as the weight of the edge and obtained a directed graph with 5849 nodes and 20367974 edges. Thinking about how to improve this part of the project and discussing with professor Mitrovic, we realized there was no reason to limit ourselves to just one graph, therefore we created two extra graphs. Trying to capture the relationships between products, we used their stock codes as nodes and whether they were present in the same invoice as the discriminant for the edges, we obtained an undirected graph

with 3857 nodes and 2946842 edges. Finally, we related customers to the countries they purchased from, by creating a third undirected graph with the 5849 customers as nodes, and 14225170 edges, representing they had a shared country of purchase.

4.2.2 Feature extraction

After graphically modeling the relationships we were interested in, the next task was to extract quantitative features. Taking inspiration from the approach we later describe for time series analysis, we briefly checked an automated feature extractor, giotto-tda [3], but after consulting with professor Mitrovic, we decided on a different solution. We divided the task into two main avenues, first we extracted centrality measures from each graph.

4.2.3 Centrality measures

As seen during the course, there are various ways of estimating the importance of a node in a graph, following what professor Mitrovic presented and the available metrics in NetworkX [4], the python library we used to build our graphs, we extracted the following centrality measures:

- degree centrality
- closeness centrality
- betweenness centrality
- eigenvector centrality
- PageRank

The assumption is, they should help our model in capturing the important relationships, in particular, we expected customers with high centrality measures to not be churners.

4.2.4 Deepwalk embeddings

The other method we explored was creating embeddings using the Deepwalk algorithm provided by the Karate Club library [5], which attempts to capture the relationships between nodes in a graph by

1. creating a set of random walks amongst nodes, the size of the walks can be tuned to the specific graph, each walk is then treated as a sentence
2. using SkipGram, a natural language processing algorithm, to update the routes and try to determine the maximum likelihood of a neighbor being part of a node's route, for more details we point you at the original paper explaining the procedure [6]

Tuning the parameters for our specific case we obtained 128 embeddings for each graph, we decided to discard the result for the graph of the products from the final dataset, due to a large number of missing values being generated.

4.2.5 Aggregation

To aggregate the information on products with the rest of the extracted features, we created a simple average measure on each of the features described in the centrality measures chapter. Explaining with an example, the feature ProductDegreeOfCentrality for each customer is the simple average of the degree of centrality of each product they bought. We obtained 271 features from the graph feature engineering side of the project.

4.3 Natural language processing

Natural language processing (NLP) is also a field in which we have not had, as of yet, a formal course. We learned during the project and professor Mitrovic's presentation [1] about text tokenizing, reducing sentences, in our case item descriptions, to a set of tokens, in a way that accounts for different spellings and other problems that may occur by just dealing directly with strings. Besides extracting the description length for each customer, our approach for feature extraction in this part of the project was based on creating clusters.

4.3.1 Clustering

To cluster customers who bought products with similar item descriptions, the first thing required was a distance metric. Discussing with professor Mitrovic, we decided to use the Jaccard similarity, which defines the similarity between two sets U and V as:

$$Jaccard(U, V) = \frac{|U \cap V|}{|U \cup V|}$$

outputting a score between 0 and 1. We then proceeded to tokenize the aggregated descriptions for each customer using the Natural Language Toolkit (nltk) library [7], removing punctuation, and other irrelevant tokens for our analysis. For this part of the project, it was important to define how we would split the aggregated dataset into training and testing. It was also paramount to keep the same split for all other parts of the project. We wrote our own function in an auxiliary library so we could reuse it during all phases, we chose a 0.8, 0.2 split for training and testing, with the possibility of holding 0.2 of the training set for validation, in retrospect a better choice could have been made. We then performed some analysis on the purity of the clusters we would generate on the training set with different thresholds, ranging from 0.5, the minimum we intuitively felt was required to describe two customers' descriptions as similar, to 0.8. We decided to fix the similarity threshold at 0.8 and extracted 23 clusters from the training set, we then proceeded to match the average description in each cluster with the description for each customer in the test set, to decide whether to assign them to the cluster or not. Anticipating the chapters on results and reflections, NLP is the weakest part of our project, nonetheless combining the RFM features with the features we extracted here did provide a small improvement to the base model.

4.4 Time series

Another requirement was to perform a time series analysis on the dataset. At first, since we did not have a background on how to perform it, we were a bit lost. After some research, we found a library called Time Series Feature Extraction Library (TSFEL) [8] and started working with it. There were a lot of instances where the package did not work as expected, so we had to modify the library source code to fit our needs. Once we solved the issues with the library, we had to remove customers who only made one purchase from the extraction, finally, we had to figure out which original features it made sense to use for the time series extraction. We decided to use `TotalSpent` since the evolution of the spending habits of the customers seemed like valuable information, and we later added `AvgDays`, representing the average time span between purchases for each customer. One of the features extracted from this variable proved to be very important for our final model. To speed up the process of extracting all the features, we used multiprocessing. The default extractor utilizes a JSON file containing all the features it has to calculate, some of which were specific to biosignals (for instance electrocardiograms), since they were not applicable to our project we excluded them.

The final result was a data frame with 5753 customers and 600 features. Many of the extracted features contained more than 70% missing values, so we dropped them, we decided to keep the rest and impute their missing values with the median of the feature.

4.5 Preparation for feature selection

Since we only had 96 customers missing from the time series dataset, compared to all the other data frames we generated during the feature engineering phase, we decided to merge all data on

the customers obtained from the time series. Christian Pala implemented an auxiliary script to merge the aggregated dataset from the RFM model with the graph, natural language processing, and times series features to obtain a dataset with 5753 customers and 413 features.

5. Feature selection

Professor Mitrovic's requirements for the feature selection part of the project were to implement at least one supervised and one unsupervised method. During the lecture [1], we learned we could place feature selection strategies into three broad categories:

- filtering methods
- wrapper methods
- embedded methods

To explore the topic and try to achieve the best results possible, we experimented and implemented methods from all three categories.

5.1 Filtering methods

"Filter feature selection methods use statistical techniques to evaluate the relationship between each input variable and the target variable, and these scores are used as the basis to choose (filter) those input variables that will be used in the model" [9]. For our project, we first implemented an unsupervised filter with variance thresholding, followed by a supervised mutual information filter.

5.1.1 Unsupervised

After some research and discussion, we decided to implement a variance threshold for the unsupervised part of feature selection, the assumption being variance in the variable is a proxy for the information it contains. Since, in this case, we do have the target variable, we tried to be conservative with the unsupervised filtering, to avoid inadvertently losing important features. The Scikit-learn library [10] provided us with a built-in function, VarianceThreshold; if the filter is used with the default value of 0 for the variance, there is no need for preprocessing, and only features with constant values are discarded. In our case however we wanted to use the variance of the feature as a measure of the information it contained, to be fair in the way we selected the features, it was, therefore, necessary to rescale all features to the same range, but **not** to also scale the variance as well, as would have been the case for instance by using the standard scaler provided by Scikit-learn [10]. We applied a MinMaxScaler [10], which scales all the features to the range [0, 1], to the dataset and, after some testing, used a threshold of 10^{-3} for the minimum variance required to pass the filter. From the 413 incoming features, we selected 308.

5.1.2 Supervised

As a safety measure and comparison, we decided to also implement a supervised learning filter, in order to both compare which features were discarded by each method and also to complement the selection with different filters. For supervised filtering, Christian Pala made use of mutual information, a probabilistic measure that quantifies the amount of information you may gain from one random variable, through another. In supervised learning, the variable of interest is the target and mutual information provides a score of how much each feature contributes to information on the target variable. We used the same threshold of 10^{-3} for easier comparison. In our project, the features selected by the filters were pretty similar, with mutual information discarding more. We decided to implement the filters sequentially, selecting 205 features at the end of the process.

5.2 Wrapper methods

Wrapper methods are based on a machine learning model, that is then fitted on a dataset. It is a greedy approach, evaluating all possible combinations of features, and evaluating them with a performance measure until the best one is found. [11] Both forward and backward selection methods were implemented, we later included recursive feature elimination and exhaustive search from the MLxtend library [12].

5.2.1 Forward selection

In forward feature selection, we start from a null model, we then select the feature with the best performance and add it to the model, continuing until a given performance threshold is not reached, regardless of which feature we may add. We chose to take advantage of the Sequential Feature Selector from the library Scikit-Learn [10], as it is straightforward to implement. Since we are dealing with an unbalanced dataset and to make the process more robust, we implemented the selection with repeated stratified k-fold cross-validation.

5.2.2 Backward selection

Scikit-Learn [10] offered both forward and backward methods in the same function, and we implemented them simultaneously. The backward version works exactly the opposite of the forward one. We start with the full model, and each time delete a feature until we don't decrease the performance by more than a given threshold. This method of selecting features proved to be too slow for us to test extensively.

5.2.3 Recursive feature elimination

Recursive Feature Elimination is similar to Backward feature selection. It works by searching for a subset of features, starting with all features in the training dataset and removing them until either the desired number remains, or a certain performance metric is not satisfied. This is achieved by fitting the given machine learning algorithm used in the core of the model, ranking features by importance, discarding the least important features, and re-fitting the model. This process is repeated until a specified number of features remains. This approach did not provide improvements compared to the previous two mentioned.

5.2.4 Exhaustive search

This method works by brute-forcing the evaluation of each feature subset. This means that it tries every possible combination of features and returns the best-performing subset. This method is very slow, but it helped us to find the most important feature for our project, `AvgOrderDaysAUC` automatically extracted by the TSFEL library [8] from the `AvgDays`, which measures the area under the curve of the average days between purchase for each customer.

5.3 Embedded methods

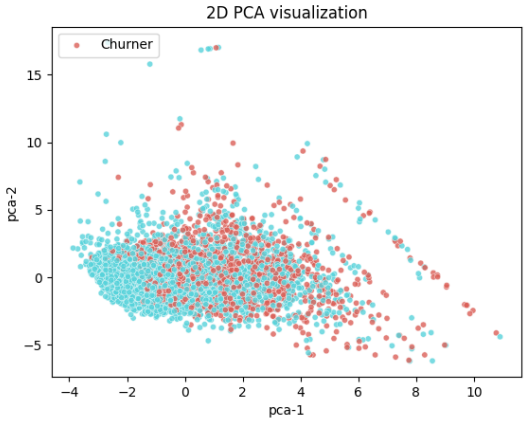
Embedded feature selection methods are integrated into the learning process of the algorithm. We elaborate more on our model decision in the next chapter, where we talk about model selection, but we made sure to select one which had this built-in mechanism. Feeding the full feature set to the classifier we chose provided the best results, compared to all other methods. Despite built-in regularization, the model still used most of the features provided, so after cross-referencing with the other methods and checking performance, we selected the features with feature importance values above $5 \cdot 10^{-3}$, a natural cutoff based on the model's importance scores, to have a reasonably sized dataset for dimensionality reduction.

6. Dimensionality Reduction

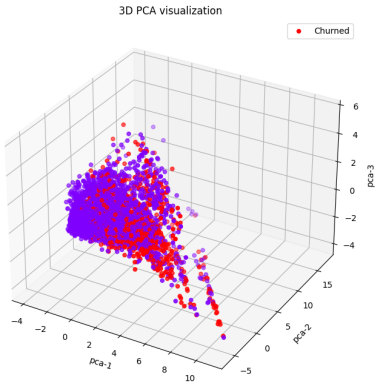
The last project requirement by professor Mitrovic was to transform the data to a lower dimension, using a dimensionality reduction technique. Based on the discussion we had in class on the topic, we selected two methods, principal component analysis (PCA), which we covered in previous courses, and t-distributed Stochastic Neighbor Embedding (t-SNE), which professor Mitrovic presented during her lecture [1]. We decided to use PCA compared to the alternatives provided since we had a very dense dataset, which should have enabled PCA to work well.

6.1 Principal Component Analysis

Principal Component Analysis (PCA) is a technique used to explain high-dimensional data, transforming it into lower dimensions while retaining as much information as possible. Taking advantage of the various available solvers, the implementation was straightforward, with the caveat that the features must be scaled to ensure the variance selection is fair. In our case, we selected a standard scaler and since our objective was data visualization, we used the whole dataset for both PCA and t-SNE. Just using PCA we were not able to separate the classes well, as highlighted by the two plots below.



(a) PCA 2D

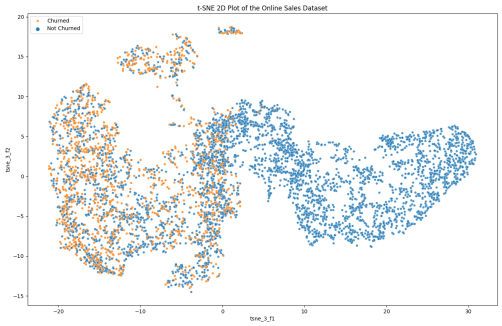


(b) PCA 3D

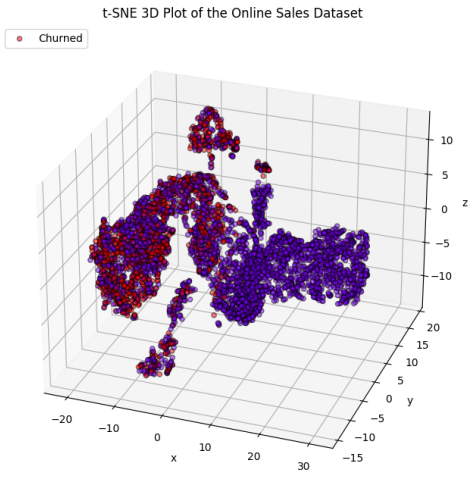
Figure 7: PCA dimensionality reduction

6.2 T-Distributed Stochastic Neighbor Embedding

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a different dimensionality reduction method for data visualization, explained in a class by professor Mitrovic. For visualization, it should work better than PCA. When implementing it, the interesting points were the tuning of the perplexity parameter and deciding whether to use PCA as an initializer. After reading t-SNE's library documentation [13] and discussing with both professor Mitrovic and professor Cannelli, we tested visualizations with perplexities ranging between 5 and 75, the square root of the number of customers, we obtained the best results in the 30-50 range and settled with a visualization obtained with 40 as the perplexity value. We also noticed a decent improvement using PCA as the initializer.



(a) t-SNE 2D



(b) t-SNE 3D

Figure 8: t-SNE visualizations

6.3 Dimensionality reduction on the full dataset

Finally, we tried applying dimensionality reduction with both PCA and t-SNE on the whole dataset, before implementing feature selection, but the results were quite a bit worse.

7. Experimental setup and results

During the project, we gradually evaluated the results of the datasets we produced, since we were often dealing with new topics it was important for us to verify that what we were doing made sense and was improving our churn modeling.

7.1 Model selection

During the feature engineering portion of the project, we debated which model to select. Given the task, we initially decided to implement a random forest, but since this is a binary and not a multi-class classification, together with the fact we have an unbalanced dataset, and also since feedback we have often received from teachers in previous projects where we used random forests, was that we could have improved our score using this alternative, we selected XGBClassifier from the XGBoost library [14]. For the purpose of this project, since modeling was not the focus, we can consider XGBoost as an optimized version of a random forest, for the task we needed to perform.

7.2 Evaluation metric selection

Since our main objective was to identify churning customers, the minority or positive class, evaluating the model using accuracy would have been a poor choice. For these kinds of tasks, the F_1 score, defined as:

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

is a much more robust metric, since it balances precision and recall on the positive class, the one we are mostly interested in. For each experiment, we also saved the confusion matrix, and the full classification report. We also plotted both the receiver operating characteristic (ROC) and the precision-recall curves. Since the cost of a false negative is much higher in this case, the client might opt to accept a larger recall in order to get better precision, making the precision-recall curve a valuable addition.

7.3 Hyperparameter tuning

When discussing how to perform the train test split globally, we decided to add an option to hold out a portion of the training set, as a validation dataset. Christian Pala noticed the library Hyperopt [15], he was not familiar with it and wanted to experiment with how it compared with the libraries we used in previous courses, for instance, Optuna, so he implemented a tuning function, which uses an advanced version of bayesian optimization, to look for the best hyper-parameters for the XGBClassifier. It was important to define the loss properly, in our case we selected the logloss of 1 - the F_1 score.

7.4 Results

As mentioned, we obtain the best results with embedded feature selection, providing the XGBoost Classifier with the complete dataset, it achieved an F_1 score of **0.650**. To obtain the best score for the full dataset, we also had to use a longer tuning process. The base RFM model we started with, had an F_1 score of **0.402**, we tested this result with 10 stratified folds repeated 10 times and the results were very stable, the standard deviation was **0.034**. After tuning the model the F_1 score improved to **0.431** in the run we reported. Below we provide a breakdown of the results of each phase of the project, the models are always tuned with a fast tuning, using two-fold cross-validation, so there is some variability in the results presented. The NLP score is combined with the RFM model since the score on its own is not meaningful.

7.4.1 Feature engineering performance

RFM model		
f-score: **0.431**	precision: **0.445**	recall: **0.418**

Graph model		
f-score: **0.461**	precision: **0.500**	recall: **0.428**

RFM + NLP model		
f-score: **0.433**	precision: **0.378**	recall: **0.505**

Time Series model		
f-score: **0.562**	precision: **0.577**	recall: **0.547**

7.4.2 Feature selection performance

Full dataset model		
f-score: **0.650**	precision: **0.638**	recall: **0.662**

Variance threshold model		
f-score: **0.631**	precision: **0.630**	recall: **0.632**

Mutual information model		
f-score: **0.614**	precision: **0.600**	recall: **0.628**

Forward selection model		
f-score: **0.608**	precision: **0.603**	recall: **0.630**

Backwards selection model		
f-score: **0.633**	precision: **0.612**	recall: **0.655**

Recursive feature elimination model		
f-score: **0.611**	precision: **0.611**	recall: **0.611**

Exhaustive 5 feature selection		
f-score: **0.607**	precision: **0.507**	recall: **0.757**

7.4.3 Feature importance

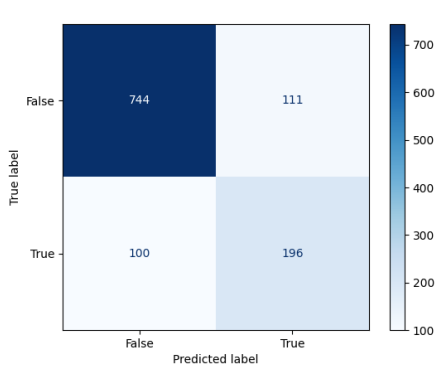
After testing extensively, cross-examining the results from the embedded model, the mutual information score, and the results from the various wrapper feature selection methods, we selected the features below as the most important for our task:

1. AverageDaysBetweenPurchaseAUC
2. AverageDaysSpectralSlope
3. TotalSpendECDF0
4. TotalSpendECDF1
5. ProductBetweennessCentrality
6. EigenvectorCentrality
7. ProductEigenvectorCentrality
8. TotalSpendECDF7
9. AverageDaysSpectralVariation
10. DegreeCentrality
11. ProductClosenessCentrality
12. CIDeepwalkEmb20-128

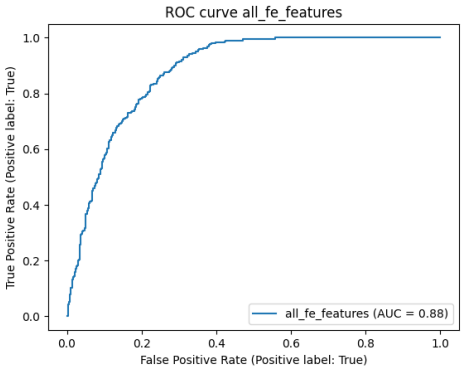
We noticed the RFM approach to the problem makes a lot of sense, many of the features selected capture aspects related to the frequency and monetary aspects. We are also pleased that both the Deepwalk embeddings, usually selected by the wrapper methods, and the centrality measures, are useful.

7.4.4 Plots

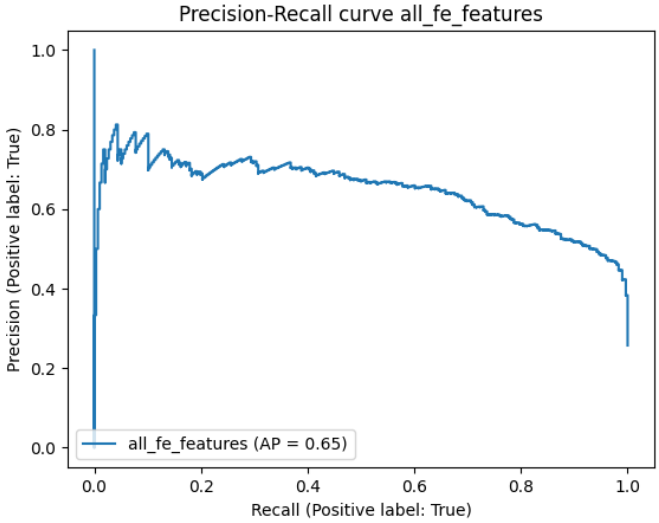
Below are the relevant plots for the best model, with embedded feature selection:



(a) Confusion matrix for the best model



(b) ROC curve for the best model



(c) Precision-recall curve for the best model

Figure 9: Best model performance plots

8. Error analysis

It was outside of the project's scope to explain the model's inner workings, but since we managed to have a reasonably small set of features, we decided to explore the model's decision-making using Shapley values and the methods provided by the SHAP library [16]. The SHAP library describes itself as follows: "SHAP (SHapley Additive exPlanations) is a game theoretic approach to explain the output of any machine learning model". Implementing it is fairly straightforward, an explainer object needs to be initialized on a trained model, the results can then be analyzed. Below we present two of the most common ways to check and explain model behavior.

8.0.1 Overall decision process explanation

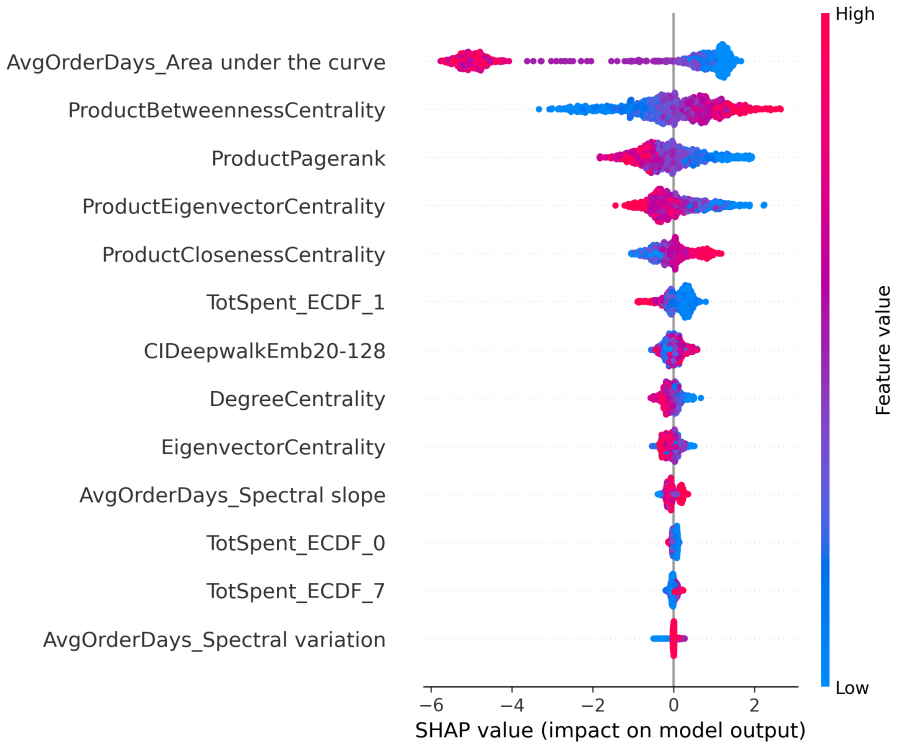


Figure 10: Model decision making

From the plot, you can see how, if the model finds high values for average days between purchases area under the curve, it will likely classify the sample as a non-churner. It also uses a different mix of information on the products bought and the amount spent to perform the classification.

8.0.2 Prediction examples

Below we showed two explained predictions, a false negative and a true positive, to showcase how the SHAP library can be used to explain the model's predictions. In this case, the ProductBetweennessCentrality misleads the model into classifying the sample as a non-churner.

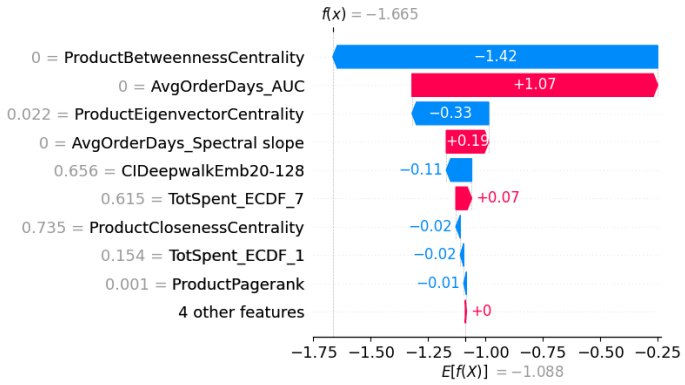


Figure 11: Incorrect sample prediction

For completeness, below we provide an example where ProductBetweennessCentrality is giving the model the right information.

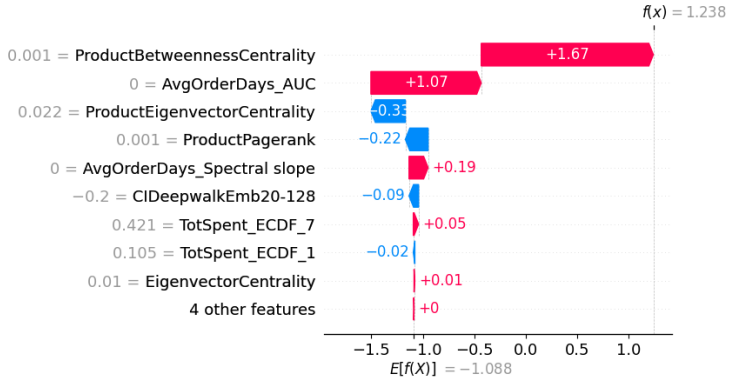


Figure 12: Correct sample prediction

9. Critical reflection

Below the main takeaways and critiques we have about how we executed the project:

- We have one big regret about the way we set up the churn label and how we split the data, which did not allow us to create features using the invoice date, like loyalty and recency, it's very likely the model could have been improved by using information on the invoice date better.
- One point which is becoming more and more apparent during our studies, and which was especially noticeable during this project, is how slow some parts of the code can be, this was really one of the main bottlenecks for us during the project.
- We were not expecting to have to clean the dataset so much, in particular not being able to easily figure out which orders the canceling orders referred to threw us for a loop. We spent a bit too long looking for quick solutions when the best approach would have been to do the hard work from the start.
- Regarding the natural language processing part, the path we selected via clustering did not lead to many concrete results, which we find a bit regrettable, we are curious to see if better solutions have been found by the other groups working on the project.

10. Conclusions

All things considered, we are reasonably satisfied with our work during the project, the models make sense and we learned how to extract and select features from new data types, like text and graph structures. We also managed time well, which allowed us to double-check our work and correct or mitigate the inevitable mistakes we made, working on new topics. We also gained some domain-specific knowledge in analyzing churn, if we were to repeat the project, we are confident we would be able to improve on what we reported here. Thank you for reading our report.

References

[1] S. Mitrovic, *Data Challenge 3 Slides*, (accessed: 05.11.2022).

[2] J. R. Bult and T. Wansbeek, "Optimal selection for direct mail," *Marketing Science*, vol. 14, no. 4, pp. 378–394, 1995.

[3] G. Tauzin, U. Lupo, L. Tunstall, *et al.*, *Giotto-tda: A topological data analysis toolkit for machine learning and data exploration*, 2020. arXiv: 2004.02551 [cs.LG].

[4] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11–15.

[5] B. Rozemberczki, O. Kiss, and R. Sarkar, "Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs," in *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*, ACM, 2020, pp. 3125–3132.

[6] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," *CoRR*, vol. abs/1403.6652, 2014. arXiv: 1403.6652. [Online]. Available: http://arxiv.org/abs/1403.6652.

[7] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[8] M. Barandas, D. Folgado, L. Fernandes, *et al.*, "Tsfel: Time series feature extraction library," *SoftwareX*, vol. 11, p. 100 456, 2020, ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2020.100456. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711020300017.

[9] M. Kuhn and K. Johnson, *Applied Predictive Modeling*. Springer, 2013.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[11] V. Verma, "Feature selection using wrapper method - python implementation," 2022. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/10/a-comprehensive-guide-to-feature-selection-using-wrapper-methods-in-python/.

[12] S. Raschka, "Mlxtend: Providing machine learning and data science utilities and extensions to python's scientific computing stack," *The Journal of Open Source Software*, vol. 3, no. 24, Apr. 2018. DOI: 10.21105/joss.00638. [Online]. Available: http://joss.theoj.org/papers/10.21105/joss.00638.

[13] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008. [Online]. Available: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[14] 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*, ser. KDD '16, San Francisco, California, USA: ACM, 2016, pp. 785–794, ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939785.

[15] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox, "Hyperopt: A python library for model selection and hyperparameter optimization," *Computational Science Discovery*, vol. 8, no. 1, p. 014 008, 2015. [Online]. Available: http://stacks.iop.org/1749-4699/8/i=1/a=014008.

[16] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., Curran Associates, Inc., 2017, pp. 4765–4774. [Online]. Available: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.