Machine Learning in Finance: Forecasting purchase behavior for banking products

Masterarbeit

zur Erlangung des akademischen Grades „Master of Science“ (M.Sc.)

eingereicht

beim Prüfungsausschuss für den Masterstudiengang

Economics

der

Fakultät für Wirtschafts- und Sozialwissenschaften der

Ruprecht-Karls-Universität Heidelberg

2023

Viktor Reif

Geboren in Buchloe am 07.09.1994

Erklärung der Urheberschaft

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig und ohne unerlaubte fremde Hilfe verfasst habe und dass alle wörtlich oder sinngemäß aus Veröffentlichungen entnommenen Stellen dieser Arbeit unter Quellenangabe einzeln kenntlich gemacht sind.

_____ _____

(Ort, Datum) (Unterschrift)

Sperrvermerk

Die vorgelegte Masterarbeit basiert auf internen, vertraulichen Daten und Informationen des Unternehmens Commerzbank AG. In diese Arbeit dürfen Dritte, mit Ausnahme der Gutachter und befugten Mitgliedern des Prüfungsausschusses, ohne ausdrückliche Zustimmung des Unternehmens und des Verfassers keine Einsicht nehmen. Eine Vervielfältigung und Veröffentlichung der Bachelorarbeit ohne ausdrückliche Genehmigung — auch auszugsweise ist nicht erlaubt.

(Ort, Datum)	(Unterschrift)

Abstract

In an effort to reduce costs and yield ever better performances, machine learning demonstrates remarkable progress in the field of finance. There already exist algorithms that successfully identify credit card fraud, predict customer retention or recommend products for private banking. We consider the task of purchase behavior prediction for a complex product space, which is needed for instance to recommend products for corporate banking clients. We argue that this task is harder to handle than most other use-cases in finance, and we expect machine learning algorithms currently common in these use-cases to struggle. We propose a panel data model estimated by the "Long Short-Term Memory" (LSTM) variant of a recurrent neural network as an alternative. In an empiric case study we compare the panel approach to a collaborative filter and a cross-sectional decision tree classifier. The panel model performs well. The other two approaches show poor to mediocre performance.
Keywords: Purchase Behavior Prediction; LSTM; Multi Label Classification

1 Introduction

Our research question is how to model complex purchase behavior in the domain of finance. Knowing what your clients want to purchase next is an immensely valuable information. Private banks invest a lot of resources into their sales departments in an effort to be in touch with their clients and thus recommend them the right products. This is cost intensive. Especially for the diverse needs of corporate clients, predicting and recommending financial products is an intricate challenge. Banks' solution to the problem has always been a bulk of sheer manpower. For instance, German private banks such as Deutsche Bank AG or Commerzbank AG employ hundreds of corporate client consultants. Each consultant is supposed to predict their clients' purchase behaviors guided by their expertise and experience. With the rise of machine learning methods in all conceivable tasks, also banks start to automate processes that used to be ruled by human guesswork. Successfully implementing a machine learning algorithm, that is capable of predicting complex purchase behavior offers two benefits. Firstly, the bank can increase the degree of automation and quality in its sales processes, which yields great cost benefits and higher sales revenues. Secondly, the results contribute to pushing the development of applied machine learning both in finance and in general.

We analyze the research question by implementing and evaluating three approaches for a product recommender system for corporate clients in banking. The modelling problem behind predicting purchase behaviors in corporate banking is difficult. The product choices of corporate clients are more complex than for instance for Netflix customers look-

ing for a new tv-show to watch or for clients in private banking. The complexity arises by the sheer number of products. For instance, on the most granular level Commerzbank AG distinguishes between more than 800 corporate banking products. Also, the product choices are complex because the products are highly diverse. Corporate clients act in different industries and countries, have different business models and are subject to different regulatory requirements. Therefore, corporate clients are diverse and so are the products they demand. Due to this complexity, approaches that work for easier use-cases are expected to perform poorly in our setting.

We propose a panel data model estimated by a sophisticated variant of a recurrent neural network to predict purchase behavior. Also, we propose two baseline approaches. One baseline model is inspired by prior models to predict churn and credit card fraud in banking and is estimated by a decision tree. The second approach is a basic version of a user-based collaborative filtering algorithm. We propose a comprehensive analysis framework encompassing globally defined input features, target variables, and evaluation metrics. The framework compares the three approaches to identify the most adequate for complex purchase behavior forecasts. Within the framework, the panel approach performs well and clearly outperforms the two baseline models.

2 Literature

In banking, machine learning algorithms have been applied with auspicious results in several use-cases. Ghosh and Reilly (1994) and Patidar et al. (2011) are two of many papers showing successful use of Artificial Neural Networks (ANNs) to detect credit card fraud in banking. Ghosh and Reilly (1994) found that an ANN outperforms classic, rule based fraud detection procedures. Patidar et al. (2011) showed that their neural network tuned with a genetic algorithm reaches performances close to human intelligence levels. In the area of customer churn prediction, Xie et al. (2009) identified random forests as the most adequate method relative to other machine learning techniques. Rahman and Khan (2018) concluded in their analysis that the ANN is the best performing estimation method. In the same paper Rahman and Khan (2018) used the concept of recency (time between transactions), frequency (number of transactions in time period) and monetary value (volume of transactions in time period) to consolidate the input transaction data before predicting. In short, recency, frequency and monetary value are referred to as RFM. They found that representing customer behavior in this consolidated form improves predictive performance. Bahnsen et al. (2016), who analyzed which is the best feature engineering for credit card fraud detection, also utilized a transaction data aggregation strategy related to the RFM concept. They split the transaction time series into periods and use the von

Mises Distribution, a special continuous probability distribution, to consolidate the periods respectively. They showed that the von Mises Distribution is especially adequate for dimensional reduction with low information loss. Thanks to that feature engineering they maximized modelling performance which yielded additional cost reductions of more than 50% within their fraud detection system. Barreau (2020) predicted bank clients' purchase behavior of financial instruments like bonds, options and futures, with several machine learning algorithms for generating product recommendations. He identified collaborative filtering together with gradient boosted decision trees and deep learning as the most adequate methods.

There already exist well-established recommender systems. All firms, and especially those relying heavily on e-commerce, like Amazon, Netflix, or Booking.com have a high demand for adequate recommendation algorithms. The kaggle competition "The Netflix Prize" funded by Netflix in 2006 with a grand prize of $1,000,000 is a fitting example for this demand. Its objective was to predict how users would rate movies they had not yet watched at least 10% better than Netlix's own recommender "Cinematch". The Netflix Prize shook the domain of recommendation algorithms. Bennett et al. (2007) explained that the competition strongly pushed the development of new recommender systems. They noted, that most competitors' approaches are related to the collaborative filtering concept. As explained by Schafer et al. (2007), the collaborative filter algorithm clusters agents by similarities in their characteristics or their purchase behavior and then recommends products given the resulting groups. There are no predefined groups for clustering, which renders the task an unsupervised machine learning problem. The cluster problem within the algorithm possesses a lot of depth since there are countless options to solve it. Moreover, researchers developed effective feature engineering measures before clustering to enhance their recommender systems. Also, each approach can define and compute similarity on its own accord. As this algorithm is highly customizable and the core of numerous recommendation sytems, it is no surprise that a collaborative filtering approach won the competition. Koren (2009) presented his team's winning solution. They proposed a collaborative filtering architecture, in which nearly every step is computed by one or a chain of machine learning algorithms, like gradient boosted decision trees and generative stochastic artificial neural networks.

Apart from the Netflix Prize, Bernardi et al. (2015) implemented a collaborative filter approach for the e-commerce of booking.com. Sorokina and Cantu-Paz (2016) used natural language processing methods for feature engineering to enhance the collaborative filtering algorithm used by Amazon. Covington et al. (2016) proposed a recommender algorithm for Youtube videos that solves collaborative filtering steps via deep learning. In their approach user similarity is computed by a highly customized neural network that can handle the complex feature structure that represents a Youtube user's behavior and

characteristics. Recommendable products are identified via a multi-class classification model, which is estimated by another neural network. Their deep learning variant outperforms classic collaborative filtering models.
Most papers concerning machine learning work empirically. It is very common to propose a new model or a new variant and compare it to one or more established models. For example Patidar et al. (2011), Xie et al. (2009) and Rahman and Khan (2018) did this for churn and credit card fraud predictions. These papers use well-established statistical metrics, such as precision, F1-Score or area under curve (AUC) to evaluate and compare the proposed methods. The metric choice is highly reliant on which aspect of the recommendation is most relevant in the respective use-case. For instance, a wrong recommendation might be devastating in a certain use-case, which could motivate a researcher to use a metric that especially accounts for this context. Those contexts can be rather complex. Therefore, for instance Barreau (2020) defined a custom cost function, which quantifies the economic impact of each correct or false prediction, to compare his models. Also Bahnsen et al. (2016) compared different feature engineering and estimation method combinations by a custom cost function to account for the monetary context of the modelling outcomes. Outside of banking, it is also common to set up a comparative framework while proposing a new method or model. For instance, Gu et al. (2020) set up such a framework to analyse which machine learning algorithm is the most adequate for asset pricing.

3 Data

All primary data sources are copies or subsamples of copies from the tables in the productive database of Big Data & Advanced Analytics - Sales Analytics department of Commerzbank AG. As these tables are large, we use the big data framework Pyspark to load, format and join them to build our dataset. The resulting dataset contains information about 41,559 corporate clients within the time frame of 01/01/2019 until 31/06/2022 on a monthly level. In total, the productive database contains data about more than 100,000 clients. We consider a subset of all clients in an effort to create a dataset that is adequate for our analysis. Chapter 3.3 explains the details of this. The two keys of this dataset are the bank internal ID of the client and the monthly date. All 1,728,678 rows (41,559 clients times 42 months) are uniquely identifiable by this key combination. The dataset contains 237 columns, 235 of which contain feature information and two are keys.

Table 1: Dataset General Structure

	1	2	3	4	5	...	237
	Client ID	Date	...	...	...	...	...
1	000001	201901	Value	Value	Value	...	Value
2	000001	201902	Value	Value	Value	...	Value
3	000001	201903	Value	Value	Value	...	Value
...	...	...	...	...	...	...	...
41	000001	202205	Value	Value	Value	...	Value
42	000001	202206	Value	Value	Value	...	Value
43	000002	201901	Value	Value	Value	...	Value
...	...	...	...	...	...	...	...
1728676	041559	202204	Value	Value	Value	...	Value
1728677	041559	202205	Value	Value	Value	...	Value
1728678	041559	202206	Value	Value	Value	...	Value

Table 1 visualizes that in the dataset each row can be identified by its two keys. The row and column numbers on the borders of table 1 are for visualization only. The real dataset consists only the column names (”Client ID”, ”Date”, ...) and the values within the columns. This is indicated by the inner part of the table. For each client the set of features is a panel spanning over 42 months and 235 columns. For instance, rows 1 to 42 cover the panel for the client with the ID 000001 from 01/01/2019 to 30/06/2022.

3.1 Feature Variables

Given the key structure within the dataset, we can join any set of features, which are needed for modelling. The features we join into the dataset are client revenues per product group (as binaries, i.e. created any revenue: Yes/No), client industry, credit rating, internal client group (grouped by revenue and domestic/foreign), transaction details, liquidity and client independent market characteristics. Each of these features is extracted from a different table in the same productive database.

Table 2: All Feature Groups

	Data Type	Occupied Columns in Dataset
Revenues per Product Group (Yes/No)	Binary	21
Client Group	Categorical	43
Rating	Categorical	31
Industry	Categorical	91
Liquidity	Numeric	1
Transactions per Currency	Numeric	4
Transactions per Industry	Numeric	35
Market Indicators	Numeric	9

Table 2 lists all feature groups in the dataset. Depending on the preprocessing and the number of features within the group, the number of columns occupied within the dataset differs between groups. While joining the feature groups into the dataset, values might be missing. This can occur for example due to a feature not being available for a certain point in time or for a client, or due to a key mismatch. We fill the missing values depending on the data type of the feature thus that the dataset contains valid values for all its elements. We cover the handling of missing values and the different number of occupied columns when explaining each feature group in detail in the following paragraphs.

We extract the data for the feature group "Revenues per Product Group" from the respective table, which contains all revenues from the bank's entire corporate client sector. Each row in this table gives the date and the amount of a revenue that the bank generated off a client given a product. In the Commerzbank product universe there exists a hierarchical category mapping that groups all products into categories on four levels of granularity. The revenues table registers products on the lowest level of this product mapping. On this level there exist approximately 800 different products. Using this hierarchical mapping we combine these products into 21 categories. The reason for doing so are twofold: Firstly, for the bank's recommendation context more abstract categories are preferred as lower granularity levels are too detailed for a recommendation. Secondly, from a machine learning perspective fewer categories render the predictive problem easier to handle. We aggregate the revenue information per client per month into the binary variable "created revenue in product group": Yes/No (1/0) for each of the 21 product groups individually. This means that if a client created a revenue with any product on at least one day within the respective month and within a certain product group, we flag the field with 1. Whenever a client did not create any revenue for a given month, we fill the respective field with 0. The amount of each revenue is irrelevant. Since the revenue features consist of 21 bi-

nary variables, they span over 21 columns in the dataset. The motivation for the binary representation of the revenue information is that thus different clients or product groups cannot distort each other. If revenues differ greatly along one of these two dimensions and we feed the information in numeric format to a model, it may fail to notice smaller revenues. This would distort the analysis. Binaries prevent this problem.

Table 3: Summary Statistics Revenues per Product Group

	Mean	Empty Clients
Asset Management	0.02	39,369
Aval	0.09	35,439
Betriebliche Altersvorsorge	0.00	40,292
Bond-Emissionen	0.00	40,890
Bürgschaften und Garantien	0.14	33,569
Cash Pooling	0.00	40,238
Mobilienleasing	0.02	38,413
Export Dokumentengeschäft	0.04	36,278
Exportfinanzierung	0.00	40,754
Forderungsmanagement	0.01	40,202
Geldmarktkredit	0.06	36,977
Global Payment Plus	0.09	32,265
Import Dokumentengeschäft	0.01	39,677
KK-Kredit	0.02	38,990
Kapitalanlagen	0.01	38,780
Rohstoffmanagement	0.00	40,517
Sichteinlagen	0.72	3,872
Termin-Einlage	0.05	36,609
Unternehmensfinanzierung	0.15	32,457
Währungsmanagement	0.08	34,346
Zinsmanagement	0.00	40,354

Table 3 shows the summary statistics for these 21 binary variables. For instance, the product group "Asset Management" has a mean of 0.02 and is empty for 39,379 clients. As the revenue variables contain only zeros and ones, the mean quantifies the proportion between occurences of purchases and no purchases. Therefore, the mean indicates that 2% of all 1,728,678 rows in the dataset contain the information that the respective client created at least one revenue in the respective month in the product group "Asset Management". The empty client count implies that approximately 5% of all clients created at least

one revenue with this product group. Both the mean and the empty clients count clearly indicate that the revenue features show a high level of sparsity. Also, the correlation between mean and empty clients is imperfect in table 3. This means that product groups are unevenly distributed across clients. Per client, the revenue features can be represented as a matrix covering 21 product groups over 42 months.

Figure 1: Revenue Matrix of one Client

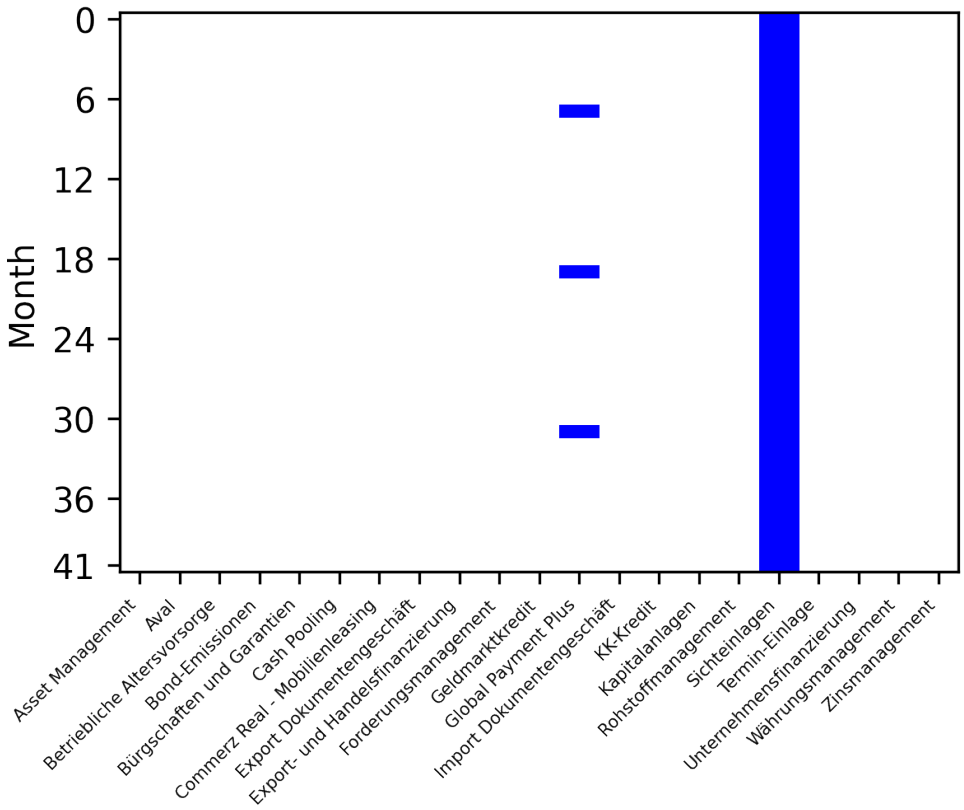


Figure 1 shows the revenue features as a matrix for an example client. The plot contains the time in months on the y-axis and the product groups (Asset Management, Aval, ..., Zinsmanagement) on the x-axis. The example customer created revenues with the product group ”Sichteinlagen” for all 42 months and with the product group ”Global Payment Plus” in four months. The client has not utilized any of the other product groups. Figure D.1 in the appendix shows the revenue matrix for ten more random clients. Considering these matrices for a few example clients yields two insights: It visually confirms our finding given the summary statistics that the revenue feature data is sparse. Moreover, after the first revenue, product groups may differ greatly in how long they keep generating revenues.

Table 4: Dataset General Structure including Revenues

	1	2	3	4	...	24	...
	Client ID	Date	Asset Management	Aval	...	Zinsmanagement	...
1	000001	201901	0	0	...	1	...
2	000001	201902	1	0	...	0	...
3	000001	201903	0	0	...	0	...
...	...	...	...	...	...	...	...
41	000001	202205	0	0	...	0	...
42	000001	202206	0	0	...	0	...
43	000002	201901	0	0	...	0	...
...	...	...	...	...	...	...	...
1728676	041559	202204	0	1	...	0	...
1728677	041559	202205	0	1	...	0	...
1728678	041559	202206	0	0	...	0	...

Table 4 shows how this feature group connects to the general structure of the dataset after joining via the keys ID and monthly date. For instance, the revenue matrix for the client with the ID 00001 covers the first 42 rows and columns 003 until 024 of the dataset. All remaining feature groups line up consecutively on the right hand side of these columns. Within the revenue features missing values are filled with zero. The assumption behind this is that when there is no information available, the client did not create a revenue.
The categorical features industry, credit rating and client group are static features different for each client. Static means that these features are fixed over time. For each client, they appear 42 times (once per month) as the same values in the dataset.

Table 5: Categorical Feature Client Group Summary

	Count	Percent
Corporates Inland (ohne Umsatzangabe)	9,492	23.06%
Corporates Inland (Umsatz EUR 500 Mio)	3,762	9.14%
Corporates Inland (Umsatz EUR 25 Mio)	2,968	7.21%
Corporates Inland (Umsatz EUR 50 Mio)	2,852	6.93%
Finanzierungsgesellschaften: Leasing...	2,714	6.59%
Other (37)	19,371	47.06%

Table 5 shows the summary statistics for client group, one of the three static features. See the statistics of the other two in the appendix (tables D.1 and D.2). Each client belongs to

one of 43 different groups. As an example for all three static characteristics, a client might be part of the client group ”Corporates Inland (Umsatz EUR 2,5 Mio)”, have no boni rating given by the bank (”None”) and operate in the energy sector (”Energieversorgung”). The three features cover 43, 31 and 91 columns respectively in the dateset, as we use one-hot encoding to store them. For instance, for the one-hot encoded feature client group, the first column contains the binary whether the client belongs to the category ”Corporates Inland (ohne Umsatzangabe)” or not. The second contains the same for ”Corporates Inland (Umsatz EUR 500 Mio)” and so forth until the 43th column for ”Corporates Inland (Umsatz kleiner EUR 1 Mio)”. Unlike in string format, one-hot encoded the information is directly processable by any kind of machine learning algorithm.

Table 6: Dataset General Structure including Client Group

	1	2	...	25	26	27	...
	Client ID	Date	...	C.I.(o. Umsatz)[1]	C.I.(EUR 500 Mio)	Leasing[2]	...
1	000001	201901	...	1	0	0	...
2	000001	201902	...	1	0	0	...
3	000001	201903	...	1	0	0	...
...	...	...	...	...	...	...	...
1728676	041559	202204	...	0	1	0	...
1728677	041559	202205	...	0	1	0	...
1728678	041559	202206	...	0	1	0	...

[1] Corporates Inland (ohne Umsatzangabe), [2] Leasinggesellschaften

Table 6 shows the static feature client group within the structure of the dataset. For instance, The client with the ID 00001 belongs to the group ”Corporates Inland (ohne Umsatzangabe)” and 041559 to ”Corporates Inland (Umsatz EUR 500 Mio)”. This feature covers the columns 25, 26, ...68. The same logic applies to the other two features boni rating and industry, which analogously occupy the following columns in the dataset. All three static features have no missing data, because they are available for all clients. Moreover, the features are contained in the same table as the client IDs. Hence, key mismatches are impossible.

The features liquidity, transactions per currency/ industry and market indicators are all numeric features. All data that is not on a monthly level is aggregated to fit the dataset’s time format. Consequently, a client’s liquidity is quantified by their monthly averaged account balance. Transaction details contain the sum measured in EUR each client transferred in each currency into each industry in each month. Negative transactions indicate that a client receives money. One example transaction could be that a client transferred an

amount of USD worth 500,000 EUR to an account tagged as a ”consulting firm”. The total number of 80 used currencies in the client transaction source table are consolidated into EUR, USD, GBP and ”other currencies”, as the former three currencies make more than 95% of the entire transaction volume. There are 35 industries including the placeholder ”Untagged Industry” from which or to which a transaction can take place. Market indicators are monthly averaged time series of the German stock market index ”DAX”, the exchange rate EUR/USD, interest rates on the European and US-American capital market for three different maturities (Euribor, Libor) and the international crude oil price. As these market indicator time series are the same for each client, they appear equally 41,559 times in the dataset.

Table 7: Numeric Features Summary

	Mean	Std Deviation	Empty Clients
DAX Index	13,109.17	1,557.65	0
EUR USD Exchange Rate	1.13	0.05	0
EURIBOR ON	-0.47	0.06	0
EURIBOR 3M	-0.45	0.10	0
EURIBOR 12M	-0.32	0.25	0
LIBOR USD ON	0.32	0.87	0
LIBOR USD 3M	0.87	0.91	0
LIBOR USD 12M	0.89	0.99	0
Oil Price	57.84	21.40	0
Liquidity	49,551.21	5,542,097.20	5,266
Transactions per Currency (4)	-432.81	9,004,309.43	53,931
Transactions per Industry (35)	-1,946.31	2,938,638.22	58,578

Table 7 shows the summary statistics of the numeric features within the dataset. All columns of numeric features are normalized on client level. When normalized, each variable's values range between $[0, 1]$ as their minimum and maximum are rescaled to zero and one. We normalize on client level, so that different clients cannot distort each other. The displayed mean and standard deviation are computed before normalization. Considering standard deviation, it is evident that transaction features and liquidity have a larger spread than the market features. Corporate clients can differ greatly from one another. For instance the mean liquidity across clients and over time is approximately $50,000$ EUR. The values also differ greatly for this variable as clients operate on different revenue levels. Considering again table 2, for instance ”Corporates Inland (Umsatz EUR 500 Mio)” contains clients with revenues 200 times larger than clients from ”Corporates Inland (Um-

satz EUR 2,5 Mio). The same intuition applies to the transaction variables. The market features are available for all clients. For the transaction features, however, the feature matrix lacks the data of many clients and thus becomes sparse.

Table 8: Dataset General Structure including all Features

	1	2	3	...	235	236	237
	Client ID	Date	Asset M.	...	T. Wasserv.[1]	T. Werbung[2]	Liquidity
1	000001	201901	0	...	0	0.23	0.78
2	000001	201902	1	...	0	0.31	0.81
3	000001	201903	0	...	0	0.33	0.83
...	...	...	...	... ...	...	...	...
1728676	041559	202204	0	...	0.50	0	0.23
1728677	041559	202205	0	...	0.51	0	0.14
1728678	041559	202206	0	...	0.56	0	0.15

[1] Transaction "Wasserversorgung und Abwasser", [2] Transaction "Werbung"

Table 8 shows the general structure of the dataset filled with all features. During the first three months that are visible in the example, the client 00001 transfers no money from or to the industry "Wasserversorgung und Abwasser", has some transaction volumes from or to "Werbung" and a rising liquidity. As the numeric features are normalized, they have no direct interpretation and it is latent whether their original value is larger or smaller than zero. All missing values in numerical features are filled with zero. We assume that when there is no information available, the client had zero liquidity and did not perform any transfers.

3.2 Target Variables and Sample Creation

The dataset contains the feature variables, but not the targets. The target variables are generated dynamically off the dataset every time a model is trained, but are always the same. We use the revenue features of the dataset to create the target variables in two steps. The first step is to convert the revenue information into purchase information, which we are solely interested in for the targets. A core assumption in this conversion process is that when a client is creating revenues with a product, they have purchased the product at the moment of the first revenue. This means a product group is only considered purchased if in the respective revenue matrix the values of a column switch from zero (no revenue created) to 1 (some revenue created). For instance, the example client in figure 1 purchases a product from the group "Global Payment Plus" three times during the time frame of the

dataset. We register no purchase in the column "Sichteinlagen", because in no field the revenue jumps from 0 to 1. Given the assumption that the moment of first revenue is the purchase moment, we convert the revenues into purchases. Per client, the result is a purchase matrix with the same shape as the revenue matrix. Figure D.2 shows these purchase matrices for the same 10 clients as in figure D.1 (revenue matrices). The second step is to aggregate several months (rows) of the purchase matrix into one target vector. The revenue feature data is sparse. Converting revenues into purchases will generate data that is even more sparse. In an effort to mitigate this sparsity, we aggregate several rows of each purchase matrix together. We take a slice of 6 months of the matrix and check for each of the 21 product groups if within that slice at least one purchase occurs. For one sample, the result of this operation is a vector of 21 binaries. Each of these binaries contains the information whether in this sample the client has purchased at least one product from the respective product group or not. These vectors are the target variables.
This second step, i.e. using a time window of 6 months to create the targets, goes hand in hand with the intuition of our sample creation. In an effort to obtain as many valid samples as possible for later feeding the predictive models, we cut three samples out of each client's panel. The result of this are 124,677 samples. The features of each client have a total range of 3.5 years (42 months). Each sample contains 24 months of feature data and a 6-months-long time window, which is converted into the target vectors.

Figure 2: Sample Creation Logic (Blue = Feature, Red = Target)

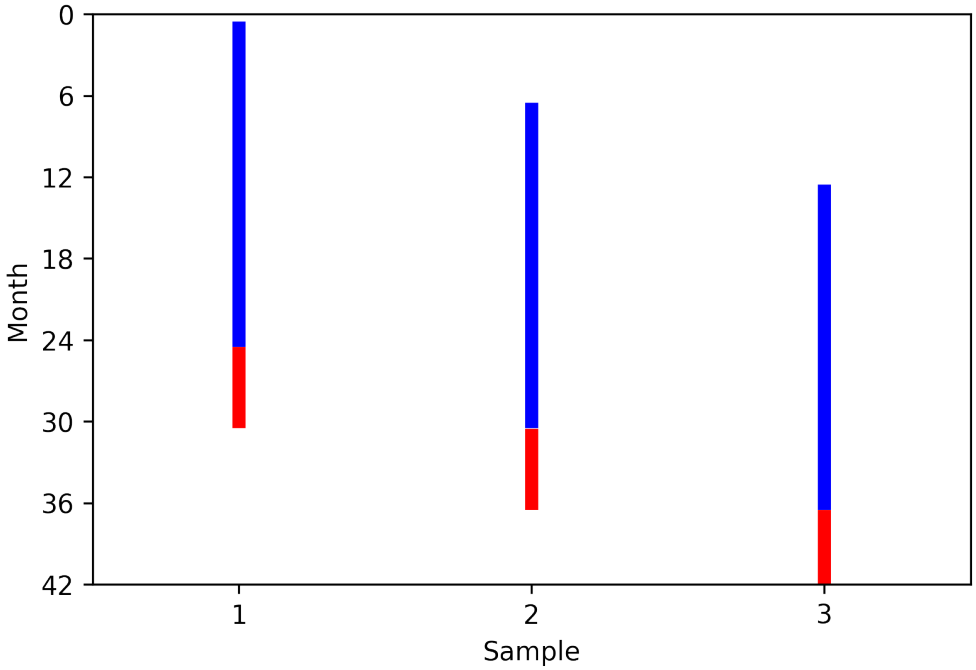


Figure 2 visualizes the logic of creating the feature and training parts per sample from the

panel of one client. For one sample, the feature part is a time-slice of the entire feature matrix. This is indicated by the blue line. The target part is a vector based on the consecutive time-slice of the revenue feature. This target time window is indicated in red. For instance, the first sample's feature time frame ranges from the first month (01/01/2019) to the 24th (31/12/2021). The feature data of this sample is the entire feature matrix of this client for that time window. The same window logic applies to the target part of this sample. The target time window ranges from the 25th month (01/01/2022) until the 31st (01/06/2022). The target vector is computed given the revenue matrix within that time window. To cut the other two samples out of this client's feature matrix, we push the feature and target time windows two times 6 months into the future. This is visualized by sample 2 and 3.

Figure 3: Target Products Count per Category

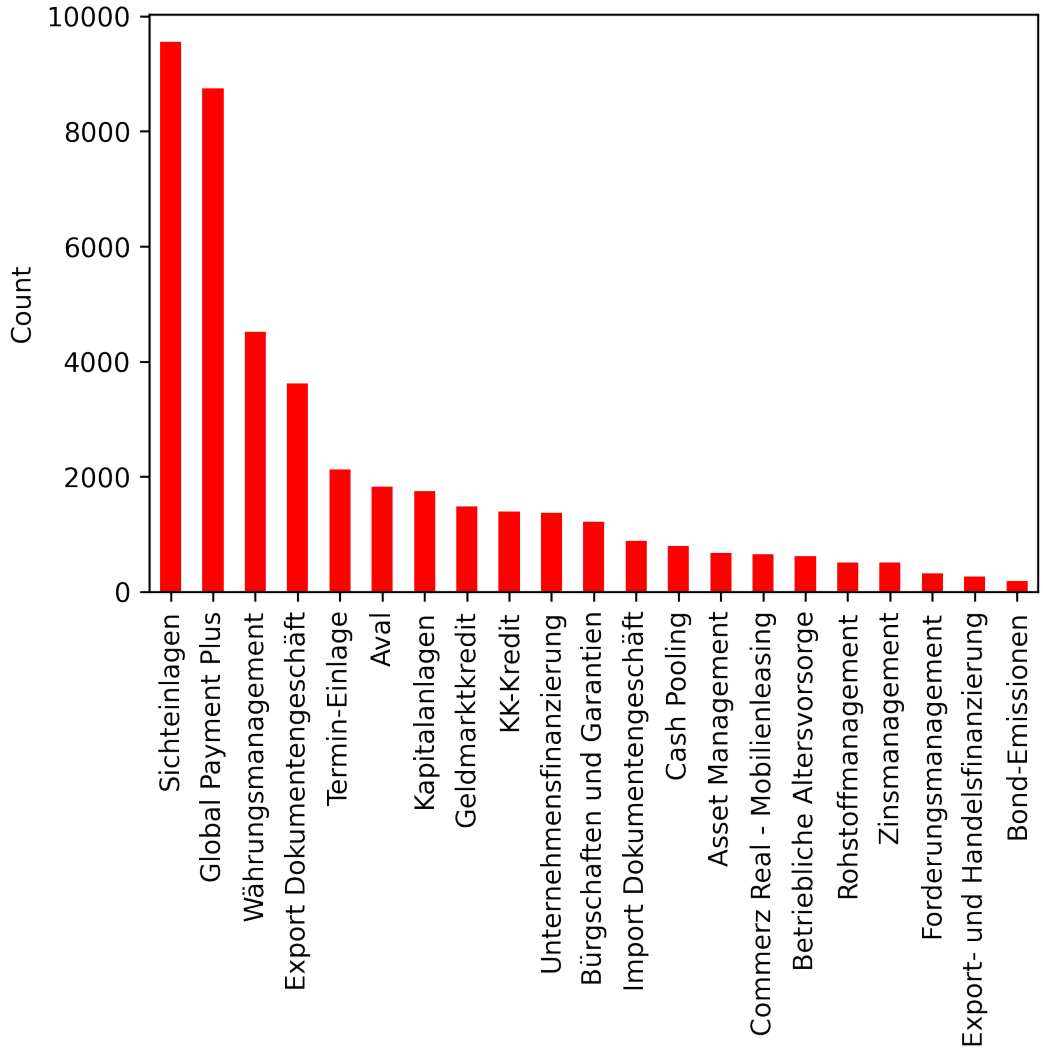


Figure 3 visualizes the distribution of the target variables, i.e. the purchases per product

group aggregated in 6-month windows as seen in figure 2, across clients. As each client's panel yields three samples for modelling, there are 124,677 target vectors. For instance, the product group "Sichteinlagen" has a value of approximately 95,000. This indicates that the respective client has newly purchased a product from this product group in 95,000 of all 124,677 samples. For most other categories new purchases occur a few thousand or even less than a thousand times, which causes the target vectors to be sparse.

3.3 Selecting Representative Clients

In total, the database contains data for more than 100,000 clients. Within our defined time frame there are 69,340 clients for which the data for the revenue features is available. We exclude all other clients from the dataset. Within this subset, there is the issue of idle clients. We consider clients idle, if they do not purchase any products in any of the three target time windows. Together, the three target time windows cover the last 18 of all 42 months in each client's feature panel. Factually, each of the 69,340 clients has created at least one revenue with at least one product during the total 42 months of the dataset. However, many of those clients happen not to create any new revenues during the last 18 months. If this occurs, our target data creation process does not identify any purchases by those clients. Given our time windows, 38,181 of all 69,340 clients are idle. We suspect that this number would strongly bias any kind of predictive model. Within those 69,340 clients there is a latent number of clients who stopped doing business with Commerzbank AG before or during our analysis time frame. Those clients' behavior is not what we are mainly interested in. Yet, there might exist clients that are still genuine clients of the bank, but just happen to remain idle during our analysis time frame. The more idle clients we include, the more challenging it will be to train a model that can identify purchases. The cause for this is that there are relatively fewer positive labels in the dataset if we include idle clients. This biases any given model. Yet, the fewer idle clients we include, the less our dataset represents the true behavior of all clients. Our dataset should have a good trade-off between these two considerations. Therefore, we tolerate a random subsample of 10,000 idle clients in the dataset. Thus, we drop 28,181 idle clients. This reduces the total size of the dataset to the final 41,559 clients, from which we generate 124,677 samples for modelling.

4 Approach

We work on the research question through an empiric case study. We propose three approaches to handle the problem of predicting purchase behavior of corporate banking clients. As all the approaches predict given the same data, we assume that they are comparable by our proposed metrics. We define a framework in which the predictive problem is approximated by a classification problem. Formally, this is a multi-label classification problem. Multi-label implies that one target variable contains non-exclusively several binary classes. In our context, non-exclusive means a client can purchase products from 1, 2... or N different product groups simultaneously. Binary indicates the target variable only considers whether a product group was purchased or not without considering the quantity. Thus, we only differentiate between whether the client purchases a product from a product group at least once or not at all.

4.1 Feature and Target Variables

All approaches have the same set of features available for predicting. The features are those described in the data chapter. For selecting those features there are two driving factors. Firstly, we select features that are easy to access and/or create. Secondly, we select features that characterize a customer and their financial activity in a meaningful way. We rely on the expertise within Commerzbank's sales analytics department to assess which features are adequate for that. For each feature's meaningfulness, the intuition is to find plausible scenarios, in which the client's feature has some connection to the client's demand for products. For instance, we consider it plausible that a client that starts lacking liquidity might demand a credit-like product. Another scenario would be that all clients operating in the same industry demand similar product groups at similar points in time. We assume that proposed features are plausible and thus meaningful. The features vary over time and for each sample. This can be represented as the feature matrix:

$$\mathbf{F} = \Big[feature\Big]_{s,t,k} \tag{1}$$

For each sample s, each feature k may change at every point in time t, where $s \in S$, $k \in K$ and $t \in T$. As seen in the data chapter, the sizes for these dimensions are $S = 124,677$, $K = 236$ and $T = 24$.

As target variables, the modelling problem contains a set of aggregated purchase behaviors. As described in the data chapter, there are two steps to create the targets. The first step is to turn the revenues into purchases. The resulting vector can be presented as inte-

ger booleans:

$$\overrightarrow{purchase_{s,t}} = [productgroup_1 : 1, productgroup_2 : 0, ...productgroup_N : 1]_{s,t} \tag{2}$$

The value 1 represents that in sample s the client purchased the respective product group at t. 0 indicates no purchase. Given (2), the second step to create the target variables is the aggregation over time of several purchase behaviors into one. Starting at a certain point in time t, we combine several future time steps into one vector of aggregated purchase behaviors. Thus, the vector

$$\overrightarrow{aggPurchase_{s,t}} = \max_{q=1}^{Q} \overrightarrow{purchase_{s,t+q}} \tag{3}$$

contains the value of 1 as an element if a product group was bought at least once in the future timeframe of size Q. Elsewise the element's value is 0. As seen in the data chapter, the target time window has a length of six months, thus $Q = 6$. The motivation for this second step goes beyond the effort to reduce sparsity in the target vector to facilitate modelling. The aggregation along the time axis renders the target vector more meaningful from a product recommendation perspective. The reason for this is the assumption that in our use-case, a client's desire to purchase a product group may last longer than one month. In addition, the time span between recommendation and purchase might be several months long. Financial products need time to be tailored exactly to a client's needs and due to high stakes clients might take their time to ponder a pending purchase. Thus, it is worthwile to recommend a product a few time steps, i.e. months, in advance.

4.2 Generalized Model

Given target and feature, the modelling problem for all proposed approaches can be formulated as:

$$\overrightarrow{aggBehaviors_{s,t}} = \alpha(\mathbf{C} \circ \mathbf{F}) \tag{4}$$

$\overrightarrow{aggBehaviors_{s,t}}$ are the targets to be estimated, $\alpha()$ is the feature engineering function, $\mathbf{C}$ is the feature selection matrix and $\mathbf{F}$ the feature matrix. The intuition of (4) is to estimate the targets $\overrightarrow{aggBehaviors_{s,t}}$ given a selection of transformed input variables based on the set of available features. The model is generalized, meaning each approach may have a different $\alpha()$, $\mathbf{C}$ and estimation method. $\alpha()$ can be any kind of feature engineering function for transforming the input data. An example for the feature engineering function would be to take the average of a feature over time or to sum several features together. The feature choice $\mathbf{C}$ is a matrix of binary weights with the same shape of $(S * T * K)$ as $\mathbf{F}$. "$\circ$"

implies that we take the Hadamard product of **C** and **F**, which is the result of an entry-wise multiplication. If **C** has the value 0 for a certain element, the positionally equal element in **F** will also be 0. Hence, $\mathbf{C} \circ \mathbf{F}$ describes how each approach is able to exclude certain features of all K features and omit certain time points of all T time points. For one sample $\mathbf{C} \circ \mathbf{F}$ could be:

$$\begin{bmatrix} 1,1,1,...,1,1,0 \\ 0,0,0,...,0,0,0 \\ 0,0,0,...,0,0,0 \\ ... \\ 0,0,0,...,0,0,0 \\ 0,0,0,...,0,0,0 \\ 0,0,0,...,0,0,0 \end{bmatrix}_{(T*K)} \circ \begin{bmatrix} 1,0,0,...,0,0.23,0.78 \\ 1,0,0,...,0,0.31,0.81 \\ 0,0,0,...,0,0.31,0.83 \\ ... \\ 0,0,1,...,0,0.76,0.05 \\ 0,0,0,...,0,0.80,0.02 \\ 0,0,0,...,0,0.80,0.10 \end{bmatrix}_{(T*K)} = \begin{bmatrix} 1,0,0,...,0,0.23,0 \\ 0,0,0,...,0,0,0 \\ 0,0,0,...,0,0,0 \\ ... \\ 0,0,0,...,0,0,0 \\ 0,0,0,...,0,0,0 \\ 0,0,0,...,0,0,0 \end{bmatrix}_{(T*K)} \tag{5}$$

In the example, **C** selects only the first time point $t = 1$. Also the last feature is excluded. $(T * K)$ indicates that these matrices have $T = 24$ rows and $K = 236$ columns. The rows represent the time dimension t and the columns the feature dimension k. For the sake of comparability approaches may not exclude specific samples.
Given (3) and (4), our generalized model, which all approaches adapt, predicts several time steps into to the future. We already argued that in our use-case a prediction a few time steps into the future is worth recommending in the present. Thus, we assume:

$$\overrightarrow{\hat{aggBehavior}}_{s,t} \approx recommendation_{s,t} \tag{6}$$

One of the three proposed approaches is a collaborative filtering algorithm. The time orientation of a recommender algorithm like that is contextual. For instance, Netflix uses a collaborative filter and gives customers the option to add a recommended movie to the list "watch later". Only if such a context is given, it is plausible to treat the recommendation algorithm like a predictor. The reason for this is that the clients behave according to the context and the recommender learns the clients' behavior. We state (6), as it not only implies that a prediction approximates a recommendation, but also vice versa a recommendation a prediction. Thus, the assumption allows to handle the collaborative filtering output like a prediction, since it is contextually plausible in our use-case. Therefore, we argue that the outputs of all proposed approaches are valid predictions and comparable to one another.

4.3 Evaluation Metrics

Our framework evaluates all approaches' adequateness by the same metrics. Generally, a model is adequate, if it correctly predicts many purchase behaviors. Correct implies that the predicted class within a label matches the true class.

Figure 4: A Confusion Matrix. (https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826 accessed 02/02/2023.)

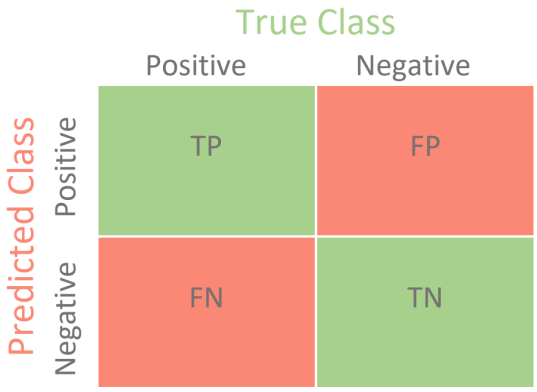


Figure 4 shows the confusion matrix, which contains all possible outcomes a single prediction of any proposed model can have. "Positive" (P) means in our setting that a client purchases a product than belongs to a certain product group. "Negative" (F) is the opposite, implying that a client does not purchase any product from a certain group. "True" (T) indicates that the predicted label equals the real label. "False" (F) works vice versa. For example, a client purchases a product from the product group "Cash Pooling". A model predicts that the client will not purchase that product. The actual value of the label is negative, but the predicted outcome is positive. Hence, the given result is a "False Positive" (FP). The other possible results TP, FN and TN work analogously. For instance,

$$\overrightarrow{aggBehaviors_{s,t}} = [1, 0, 1, ...0, 0, 1], \tag{7}$$

is the target variable of one sample and

$$\overrightarrow{\hat{aggBehaviors}_{s,t}} = [0, 1, 1, ...0, 0, 1] \tag{8}$$

is its prediction made by a given model. Checking all 21 labels results in:

$$[FN, FP, TP, ...TN, TN, TP] \tag{9}$$

For instance, the first element of (9) is a false negative, because the model wrongly pre-

dicted that the client in this sample would purchase the first product group.
The most basic evaluation metric is accuracy. This metric is computed as $(TP+TN)/(P+N)$. It quantifies how often a model predicts correctly relative to the sample size. Is accuracy for instance at 50%, then the model predicts correctly every second time. In the context of a multi-label problem, there are two ways to compute accuracy. The prediction for each sample can be taken as whole and counted as one single prediction. This means to take the entire vector of (9) as one. Only if the full estimated vector matches the true vector perfectly, the prediction is considered correct. In every other case the prediction is incorrect. In the multilabel context, this metric is referred to as (normal) accuracy. The other variant of this metric is called binary accuracy in a multi-label problem. Binary accuracy computes the accuracy for each element of the target vector separately and then averages element-wise within the target vector and across all target vectors simultaneously. In our use-case, the normal accuracy variant is expected to yield very low scores because for each target vector only one of 2^{21} different behavior combinations is entirely correct. The binary variant should give very high scores, as the target vectors contain mostly zeros, which is easy to learn for a given algorithm. Also, the binary variant tolerates some mistakes within a single target vector. For example, a predicted vector with 20 correctly classified labels and one wrong label gives a normal accuracy of 0% and a binary accuracy of 95.2%. Thanks to its simplicity and straightforward interpretation, accuracy is a popular choice for evaluation and is often used as a default metric.
However, the advantageous simplicity renders the metric blind to certain issues. One of these issues is the problem of highly sparse target data, as it occurs in our use-case. For the normal accuracy variant, the main issue are the idle clients. As discussed in the data chapter, there are 10,000 idle clients in the dataset, and they make up about one quarter of all clients. An example model that always predicts a client will not purchase any products, which would mean only zeros in (9), would have an accuracy of 25%. We know this value of accuracy without computation because 75% of clients purchased at least one product and the example model will always predict at least one element in (9) that is incorrect. For the other quarter, the model's prediction of only zeros matches perfectly the true labels of the idle clients. The problem is that the normal accuracy variant might evaluate the performance of a non-sense model, like the example model, higher than of a real model. The binary variant can suffer from the same problem, only that it does not punish deviation from predicting always zero as strongly as the other variant. Nonetheless, also in the binary variant the accuracy score for the example model might be considerably higher than for a real model due to sparse targets. Moreover, apart from statistical dangers, the metric is economically lacking. Within the context of a product recommender purchases are clearly more relevant than occurrences of clients not buying a product. Accuracy cannot differentiate between these cases.

Therefore, other, more specific metrics are preferable. Two relevant metrics are precision and recall. Precision describes how often a model correctly classifies a true label as such relative to all true labels. As a formula this translates to TP/(TP+FP). In our use-case that means how often is the model correct when it predicts a product category that the respective client buys in the future. Recall is interpreted as the percentage of all true labels that the model manages to identify as such. Hence, it is calculated as TP/P. Here, recall represents the ratio of all in the future purchased product groups that the model successfully identifies. The two metrics are well suited for our use-case, as they ignore true negatives. Thus, they have an emphasis on product purchases, which is what we are especially interested in. Also, the metrics do not suffer from the issues discussed before with accuracy. The mentioned example model that always predicts zeros would have a value of zero for recall and undefined for precision. The two metrics are computed for each label (product group) individually. From these individual scores one can derive averages. There are two options. The macro average calculates precision and recall for each label individually and takes the mean of these scores per label. The micro average also calculates per label scores individually and then takes the mean weighed by how many positive samples each label contains. Since this use-case contains highly unbalanced labels, the micro average give the less distorted information about the models' performance. Clearly, both higher precision and recall are indicator for a superior modelling performance. A mix of both metrics is the F1-score which is calculated as:

$$F1score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{10}$$

(10) is a harmonic mean. This implies that the average penalizes distorted combinations. For instance values of 0.1 and 0.9 for precision and recall would give a mean average of 0.5. The harmonic mean is 0.18. That value is considerably lower, because the precision is poor. That way the F1-score is robust to highly uneven combinations of precision and recall. In our framework the micro averaged F1-score is the main metric for model evaluation. The higher a model's out of sample F1-score, the higher its adequacy. As argued by Chicco and Jurman (2020), the F1-score as a standalone metric might lead to misleading conclusions. One issue is that a slightly distorted precision-recall combination might be adequate for our use-case, but we reject it due to the penalty in the harmonic mean. To potentially detect the issue, we also consider the micro averages of precision and recall. Another potential problem is that the micro averaged F1-score is blind to poor performances in underrepresented labels. To verify that, we plot the out of sample F1-score per label. If both the plot and precision/recall show that the two discussed problems are not prevalent, or at least not differently prevalent between approaches, we assume it is justified to interpret the micro averaged F1-score.

For further evaluation, we use Precision-Recall curves. For all classifiers, each output is a value within $[0, 1]$. The distribution of these outputs depends on the computational characteristics of the respective classifier. Proxy interpretations of this distribution are how likely according to the model this unknown label is true, or how "sure" the model is that the label is true. A threshold value between zero and one is used to turn each value of the distribution into firm predictions of positives and negatives. The most common value for the threshold is 0.5, which we also use. Increasing the value turns more of a model's output into negatives and vice versa. E.g. for a threshold of 0.8 the model has to be at least "80 % confident", i.e. very sure, that the given label is positive, to be accounted as such. The PR-curve shows the respective precision and recall trade-off for each possible threshold value. Considering the PR curves gives us several insights. Firstly, we can compare the average model performances for all their thresholds. Clearly, the closer each point of a curve to top right corner of the plot, i.e .the higher precision and recall for each trade-off, the better the model. Secondly, the curves show how versatile each model is in covering different precision-recall trade-offs. Independently of the threshold a model may not reach high values for one of the trade-off metrics. This would render the model less versatile. We prefer more versatile models. In our context, the recommender system might be tweaked later on to recommend more riskily or conservatively. For this, the recommender must be based on a versatile model. Lastly, the PR curves serve to justify our use of the default threshold of 0.5 to produce the other metrics. If there are no intersections between the curves, we know that the performance hierarchy in the main results is true independently of the threshold. In that case the threshold choice is justified. Furthermore, if the curves have no intersections and are smooth, we know that the performance differences within the hierarchy of the main results are approximately independent of the threshold. Due to the multi-label context, we consider the micro averaged PR-curves of each approach.

We use the framework to evaluate three different approaches. Firstly, we propose a basic, well-established collaborative filtering algorithm. The concept of this approach is to calculate client similarity by correlation of purchase behavior and then recommend/predict product groups for each client given what similar clients have purchased beforehand. Secondly, we propose cross-sectional data model estimated by a decision tree classifier. This model comes from the family of churn/credit fraud detection models. It uses the RFM concept in its feature engineering to turn the given panel data into cross-sectional input features. Thirdly, we propose a panel data model estimated by a recurrent neural network with "Long Short Term Memory" (LSTM) layers. We did not find this LSTM approach used in any research concerning machine learning in finance. The approach is expected to take advantage of the complex structures within the feature space to increase performance.

4.4 Collaborative Filter Approach

The concept of collaborative filtering is to find structural similarities in the context of product recommendation or purchase behavior analysis. We utilize user based collaborative filtering for this approach. The user based variant finds similar individuals by product purchases. It then recommends each individual products by checking what similar individuals bought beforehand.
We use collaborative filtering in its simplest form, in which the algorithm only takes purchases as a feature. Hence, the approach models future purchases by past purchases. This can be formulated as:

$$\overrightarrow{\hat{aggBehavior}_{s,t}} = \alpha(\mathbf{C} \circ \mathbf{F}) \tag{11}$$

Here, the feature selection matrix $\mathbf{C}$ contains weights which select the last 6 months until t and excludes all features except revenues. The feature engineering function $\alpha(...)$ transforms the revenues into aggregated purchases. Analogously to the target vectors creation, the intuition is to to turn the revenues into purchases and then aggregate six time steps into one vector. $\alpha(...)$ is defined as $\max_{q=1}^{6}(...)_{s,t-q}$. The feature engineering transforms ($\mathbf{C} \circ \mathbf{F}$) into the purchase matrix $\mathbf{P}$. This two-dimensional matrix contains all N available product groups along one axis and all S given samples on the other. Every element is 1 if the client of the sample of this row newly purchased the product group of this column, and 0 elsewise. Aside from the shared the creation logic, also the shape of $\mathbf{P}$ is the same as for the target vectors, if they were concatenated into a matrix. Or vice versa each row of $\mathbf{P}$ has the same shape as a target vector.

$$\begin{bmatrix} 0,0,1,...,0,1,0 \\ 1,0,0,...,0,0,0 \\ 0,0,1,...,1,1,0 \\ ... \\ 0,0,1,...,0,0,1 \\ 0,0,0,...,1,0,0 \\ 0,1,0,...,0,1,0 \end{bmatrix}_{(S*N)} \tag{12}$$

(12) shows example values for $\mathbf{P}$. The first row contains the aggregated purchases of the client during the feature time window for the sample $s = 1$. For instance, the client in sample $s = 1$ buys the product group $n = 3$ and $n = 20$. Those two are "Betrieblich Altersvorsorge" and "Währungsmanagement". The second row contains the same for the sample $s = 2$, the third for $s = 3$, and so on until the last row $s = S$.
The approach's estimation method uses a similarity function $s()$ to compute similarity scores. A similarity score quantifies how similar one sample's row in the purchases matrix

is relative to another samples's row. We formulate this as:

$$\overrightarrow{similarity_s} = s(\mathbf{P}) \tag{13}$$

$\overrightarrow{similarity_s}$ contains S vectors of length S. Each element is a vector of one sample s containing the similarity score between sample s and another sample j, where $j = 1, ...S$. Hence, the shape of the set of similarity score vectors is $(S * S)$. We use the pearson correlation coefficient to measure similarity between $\overrightarrow{similarity_s}$ and $y = \overrightarrow{similarity_j}$, where $s, j \in S$. As most other similarity measures, correlation ranges between $[-1, 1]$. 1 represents perfect correlation between two vectors, which translates to exact similarity between two clients' aggregated purchase behavior in the given time frame. 0 indicates that clients' purchases are independent of each other. The higher the value between 0 and 1, the higher the correlation and consequently the similarity. For values below 0, a lower value indicates that two clients are more systematically different in behavior than only being independent. That means in the sample s the client purchased product groups that the client of sample j did not and vice versa. At -1 clients are perfectly negatively correlated, indicating they are as different as possible in their product purchases. We choose correlation over other similarity measures. Correlation is simple and parsimonious. Furthermore, most researchers are familiar with it. Given the similarity vectors, the next step is to connect the similarity scores with the purchases of the feature time window. This step is applied to each sample s individually. For instance, sample $s = 0$'s similarity vector is

$$[1, -0.2, 0.8, ..., 0.6, -0.2, 0.5] \tag{14}$$

where the first element is the similarity score between sample $s = 0$ and sample $j = 0$, the second element between $s = 0$ and $j = 1$, and so on. The last element contains the similarity score betwen $s = 0$ and $j = S$. The first element of the vector is 1 because it measures $s = 0$'s similarity with itself. Sorting (14) results in

$$[1, 0.9, 0.88, ..., -0.3, -0.45, -0.5] \tag{15}$$

From (15) we take the L first samples. This subset contains the L samples with the most similar feature purchase behavior relative to the sample $s = 0$. Then, we multiply the similarity score with the product binary for each similar client and product group respectively. This means for each of the first L entries in (15), we find the row of the corresponding client in $\mathbf{P}$ and multiply each binary of that row with the entry in (15). The result is an

intermediary matrix, like for instance:

$$\begin{bmatrix} 0,0,1,...,0,1,0 \\ 0,0,0.9,...,0,0,0 \\ 0.88,0.88,0,...,0,0,0 \\ ... \\ 0.75,0,0,...,0,0,0 \\ 0,0.74,0,...,0.7.4,0.74,0 \\ 0,0,0,...,0,0.72,0 \end{bmatrix}_{(L*N)} \tag{16}$$

N is the total number of product groups. This intermediary matrix is then averaged per product group to get a recommendation vector, i.e. we take the mean of each column. The result is a vector of length N containing the algorithm's certainty that the example client in sample $s = 0$ could demand a product for each product group. This result vector might look as follows:

$$[0.35, 0.1, 0.75, ..., 0.05, 0.45, 0.15] \tag{17}$$

As purchases per product group is a binary variable and the similarity score is defined to return values in range of $[-1, 1]$, also all elements in the result vector will range from -1 to 1. To generate final recommendations/predictions, we use an threshold between $[-1, 1]$. Given an example threshold value of 0.5, (17) turns into:

$$[0, 0, 1, ..., 0, 0, 0] \tag{18}$$

(18) contains the N final product recommendations for the example client from the sample $s = 0$. Visible in this example, the algorithm recommends the client the product group "Betriebliche Altersvorsorge" ($n = 3$). Analogously, this process is repeated for each sample. This means we compute the purchase matrix $\mathbf{P}$ only once, and then recommend per sample via averaging the aggregated purchase behavior vectors of the L most similar samples, where each element of the vector is weighted by the similarity score.
As already mentioned, there are 10,000 clients (30,000 samples) in the dataset that do not purchase products during the target time windows of the last 18 months. The quantity of clients that are idle during the feature time windows is similar in magnitude. Given values for L that are lower than the number of idle feature samples, the algorithm will always predict that an idle client will remain idle. The reason for this is that all L similarity scores of one are multiplied by purchase behaviors of zero resulting in vectors of only zeros. Independently of the threshold, the final prediction will be also a vector of zeros. In our context , this means that for each client that purchases nothing during the feature time window, the most similar clients are those that also purchase nothing. Since the collab-

orative filter recommends what on average the most similar clients have purchased, the final recommendation will be nothing, too. How this issue impacts the forecast quality depends on how those idle clients behave in target time windows.
We argued that in this use-case a recommendation output is a valid forecast. We therefore consider most of the discussed recommendation settings, e.g. Netflix, similar to ours. Since in those settings the collaborative filter approach worked well, it should also be adequate for our proposed problem. Given the simplicity of the approach's estimation method, we expect the approach to yield reasonable results, even though it might fail to handle certain edge cases, like the idle clients issue. Furthermore, the approach is parsimonious both in concept and computationally, which is generally advantageous.

4.5 Cross Sectional Approach

The general idea of this approach is to reduce the dimensionality of the input data before estimation, so that the data is easier to handle by the estimation method and remains rich in feature information. For each sample, the feature data, which is originally a panel, is transformed into cross-sectional data by the approach's feature engineering function. We call the approach cross sectional, because its main characteristic is that it removes the time dimension from the feature data thus that only the cross sectional dimension remains. Given the transformed cross-sectional input data, the approach use a decision tree classifier to predict purchase behavior. The theoretical model can be formulated as:

$$\overrightarrow{aggBehavior_{i,t}} = \alpha(\mathbf{C} \circ \mathbf{F}) \tag{19}$$

where

$$\alpha(...) = RFM(...) \tag{20}$$

The feature selection matrix $\mathbf{C}$ contains binary weights which select the last 12 months until t and includes all features except market characteristics. The cross sectional approach omits the client-independent market indicators, as they only contain a time dimension, which after being consolidated would render these features empty in information. (20) represents that the approach leverages the concept of RFM to consolidate the feature matrix. We use RFM in a similar fashion as for instance Rahman and Khan (2018) or Bahnsen et al. (2016). In the concept, recency accounts for how many time steps ago the last change in a feature variable has occurred. For the revenue features that would be the time difference in months between the end of the feature time window and the last time the client create a revenue with a product group. A low value in recency indicates that a product was bought a short time ago. For a transaction feature recency quantifies how many

months ago the last transaction occurred. If a product group was not bought at all or a certain transaction type never occurred, the value of recency is missing. Frequency counts how many times a single feature changed. For revenues this means the number of times a certain product was newly purchased within the feature time frame. Monetary value represents the mean volume of a variable. E.g. the average liquidity in EUR over the feature time period. The recency and frequency dimensionality reduction are applied to the revenues feature, client liquidity and all transaction features, those being currency and target industry of the transaction. Monetary value consolidation is used for the transaction features and liquidity. Only RFM variables enter the model. This means all features in their raw form are omitted.
The approach uses a decision tree classifier as estimation method, as it is both parsimonious and intelligable. As described for instance by Loh (2011), its intuition is to recursively split the training data by one respective feature in an effort to create a set of rules that together can categorize all samples.

Figure 5: Decision Tree Classification Algorithm (https://www.geo.fu-berlin.de/en/v/geo-it/gee/3-classification/3-1-methodical-background/3-1-1-cart/index.html accessed 02/02/2023.)

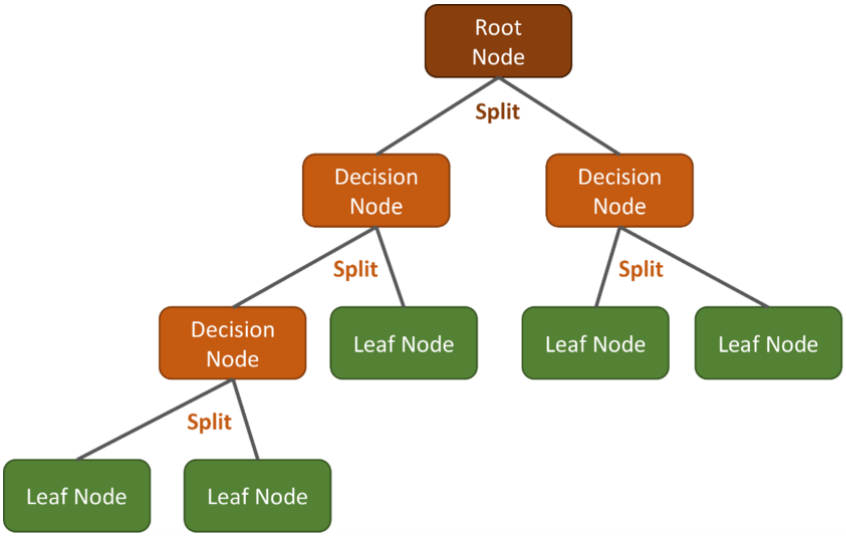


Figure 5 visualizes the tree's algorithm. At each of the tree's decision nodes, i.e. branch forks, the algorithm decides how to split the samples given one feature according to an impurity criterion. This criterion is a mathematical metric, which quantifies how well the node splits the data for successful classification. The split in each decision node has a binary conditional logic:

$$\begin{cases} 1, & \text{if condition is true} \\ 0, & \text{otherwise} \end{cases} \tag{21}$$

This could be for example

$$\begin{cases} 1, & \text{if } RFM(\mathbf{C} \circ \mathbf{F})_{k=x} > 20 \\ 0, & \text{otherwise} \end{cases} \tag{22}$$

In (22) the criterion is based on the feature x. An example criterion could be that at the node samples are split into two groups depending on whether their average liquidity, so $MonetaryValue(liquidity)$, is larger or smaller than 100,000 EUR. In the classifier tree, each lowest node is called leaf node and returns a final classification output. During training, the impurity criterion is maximized for each feature by choosing the best value for the split and then the features are sorted by the maximized criterion. Then, the feature with the best value becomes the root node, the second best becomes the following nodes, and so on. The result is a tree structure, which can classify new samples by passing them from root to leaf. A decision tree in its standard implementation handles single- or multi-class problems. To handle the given multi-label classification problem, we use, as proposed by Vens et al. (2008), the single-label classification concept. This means each binary class label of the problem is treated like an independent single-label classification task with a separate decision tree. The algorithm's result vector containing the multiple binary classes consists of the independently classified single labels.
This approach is adequate as it utilizes an established concept of feature engineering. For instance Rahman and Khan (2018) and Fu et al. (2016) successfully use the concept of RFM in less complex banking related classification problems. The decision tree estimation method is adequate because it is robust and parsimonious. Also, the method has a piecewise linear nature. Each node contains a linear logic, as it simply splits the data given one feature. Yet, a chain of nodes within a tree is non-linear, as there is no linear relationship between features and targets. Thus, the decision tree can control for nonlinear and interactive structures in data. This means the method can detect cross dependencies between features and use the information when predicting the labels. Assuming the existence of relevant interactions in the given input data, this methodological characteristic of the decision tree renders the cross sectional approach especially adequate for our modelling problem.

4.6 Panel Approach

This approach uses all available features along all available dimensions to find dependencies within and between those dimensions and thus enhance the forecast quality. Especially identifying time dependent patterns might strongly enhance this approach's predictive performance. Methodologically, the idea is to leverage a complex estimation method which is especially designed to handle high dimensionality sequential data and thus find

trends, patterns and dependencies. The theoretical model of this approach can be formulated as follows:

$$\overrightarrow{aggBehavior_{s,t}} = \alpha(\mathbf{C} \circ \mathbf{F}) \tag{23}$$

where

$$\alpha(\mathbf{C} \circ \mathbf{F}) = Panel(\mathbf{C} \circ \mathbf{F}) = 1 * (\mathbf{C} \circ \mathbf{F}) \tag{24}$$

and $\mathbf{C} = 1$. The feature choice matrix $\mathbf{C}$ selects all available features and the feature engineering function $\alpha()$ is an identity function, meaning no feature transformations occur. Consequently, we can reformulate (23) as

$$\overrightarrow{aggBehavior_{i,t}} = \mathbf{F} \tag{25}$$

(25) states that the approach predicts future purchases by the full set of features in their full set of dimensions.

We use the Long Short Term Memory method (LSTM) proposed by Hochreiter and Schmidhuber (1997) to estimate (25). The LSTM is a variant of a recurrent neural network (RNN), which in turn is a variant of the ANN. The ANN is the standard neural network. The RNN is an ANN, that is especially designed to handle sequential data by being able to store information in itself during several training steps. The LSTM is similar to the RNN, but its information storage system is more thorough. Thus, the LSTM can handle data with particularly long sequential patterns. The usual symptom of using a network variant that is not adequate for the input data is poor performance. We suspect that there are numerous long sequential patterns in the input data of this approach. Therefore, the LSTM is the most adequate method for this approach, as it should show the best performance. Before explaining the LSTM variant, we showcase both the ANN and RNN, since the LSTM shares several characteristics with the two.

Figure 6: ANN Architecture (https://www.researchgate.net/figure/Typical-examples-of-ANN-RNN-and-LSTM_fig4_334854519 accessed 02/02/2023.)

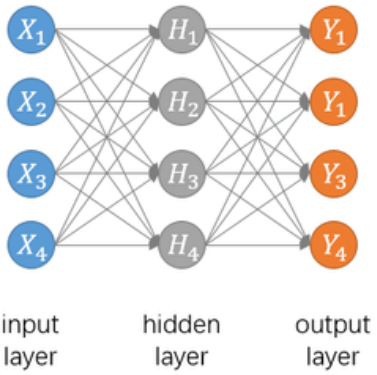


Figure 6 visualizes the general design of an ANN. As explained by Kubat (2017), the ANN mimics the neuron/synapses structure of a biological brain. In the input layer the features enter the network as signals. This layer accepts the features into the network and pass them on to first hidden layer. The input layer and the input features share the same shape. For instance, in our use-case the panel data model has per sample a feature shape of (237, 24), as there are 237 different characteristics that change monthly for 24 months. The input layer must have the same shape of (237, 24). As seen in figure 6, each signal sent from the input layer reaches each neuron in the first hidden layer. When a signal reaches a neuron, it is transformed by an activation function.

Figure 7: Tanh function (https://paperswithcode.com/method/tanh-activation accessed 02/03/2023.)

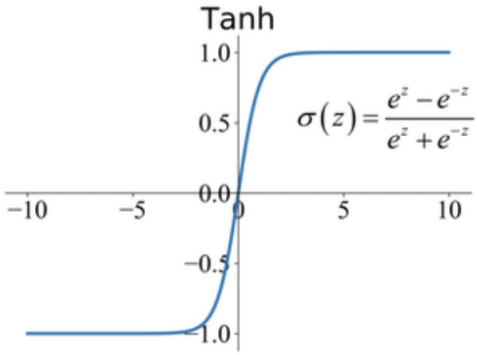


Figure 7 shows the Tanh function, where $\sigma(z) \rightarrow y, z \in \mathbb{R}, y \in [-1, 1]$, as an example activation function. Activation functions filter out irrelevant signals. If a signal is relevant, i.e. has a large enough value, the function returns a value larger zero (a number not close to zero). This is called activating the node. When the signals reach a neuron, each signal is first amplified or mitigated via a respective weighting factor θ, then they are summed together and then the sum passes the activation function. The result of these three operations is one value, which is then passed on as new signals to the neurons of the next layer. If the activation function of that node does not activate the summed signals, the node sends signals of zeros (or very close to zero), i.e. nothing, to the next nodes. All other hidden layers line up successively behind the first and their neurons receive weighted inputs via synapses from all neurons in the layer before. The last layer is the output layer, which merges all inputs from the last hidden layer into output signals. For a single label binary classifier, the output would be a single signal $\in [0, 1]$. To turn this output into a prediction a cutoff threshold within the possible range determines if the output will count as zero or as one. In the context of mulitilabel classification, the neural network would have a vector of these output signals as total output. Analogously to the input layer, the output layer shape must fit the shape of the target vectors. Therefore, in this approach the output

layer has a shape of (1, 21), as there are 21 different product groups that a client purchases or not in a given sample. This output layer contains 21 neurons. The ANN is estimated by backpropagation. The intuition of backpropagation is to repeatedly let the ANN try to classify the training data and then "teach" the ANN to classify better given the mistakes that occurred while trying. Appendix A shows the details of this.
As noted by Medsker and Jain (2001), the ANN is unable to handle sequential patterns within the input data, as it lacks a way to remember patterns. As an alternative, they propose the RNN. Recurrent means that each node in the hidden layers has not only synapse connections to nodes of the next (hidden) layer, but also to itself. These recurrent connections allow the RNN to remember sequential patterns. Apart from these self-connections, the RNN is equal to the ANN.

Figure 8: RNN-rolled (https://colah.github.io/posts/2015-08-Understanding-LSTMs accessed 02/02/2023.)

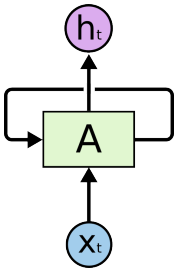


Figure 8 visualizes this self loop for a given node A, where x_t is the node's input and h_t its output for any given training step t. Each node within a recurrent hidden layer possesses this layout. At the step t a node receives the input from the self loop and adds it to the weighted signals received from the previous layer. To the result of that the node applies the Tanh activation function, to yield the output and self-loop information for next given step. The output is sent as a new signal from the node to next layer. The self-loop information stays within the node to be accessed by the same node at next step $t + 1$. That way the node can store information in one training step and access it in the next. Information may last for many steps within the RNN as each recurrent node accesses the information from its predecessor node and updates it for the next step.
However, RNN architectures often fail to handle patterns that are very long term. There exists a positive correlation between pattern length and likelihood that the RNN "forgets" the pattern. When this occurs, the network often stops improving (almost) entirely and performance is low. This phenomenon is called the vanishing gradient problem. Appendix B explains the details of this. In an effort to mitigate the gradient issue, Hochreiter and Schmidhuber (1997) proposed the LSTM architecture, which acts like an extension to the RNN.

We already discussed how in the RNN each node of the hidden layers receives new information from the input signals and its self loop. The LSTM variant adds a third information source whilst keeping the other two. This third source is the permanent self loop, which each node receives from itself from the previous step without having to pass an activation function, which could wipe out the information. Thanks to that third inlet of information without activation, remembering sequences is the LSTM's default behavior. To accommodate this permanent self loop feature, the LSTM node consists of three different sections. These sections are referred to as gates, and are in general a way to optionally send information from one place to another within the node. Each gate uses either Tanh or sigmoid, where $\theta(x) = \frac{1}{1+e^{-}x}$, or a combination of both for activation. The activation of each gate represents its decision to do or not do its purpose given the respective inputs. One of the three gates in the LSTM is the "forget gate". Given the information from the input signal and the normal self loop, it decides how much information will remain in the permanent self loop of the current step. The "input gate" takes the same input as the forget gate and decides if and what information to add to the permanent self loop information. The "output gate" takes all three information sources as input. Thus, it receives the input signal from the other nodes, its own normal self information loop input and its own permanent self loop information. The gate decides how to combine all the information and sends it as the output to next nodes as well as to itself in the next step as the normal self loop information. The workflow is as follows and takes place in every recurrent node of the network:

1. Forget Gate decides what remains in permanent self loop
2. Input Gate decides what is added to permanent self loop
3. Output Gate decides how to use permanent self loop information to create the output

Thanks to this more versatile structure within each node, the LSTM can remember longer sequential patterns while avoiding the gradient problem. In particular each node's option to discard redundant or confusing information via the forget gate renders the LSTM more reliable at remembering long term patterns.
We expect the panel approach to perform well because it uses the theoretical model with the most features and dimensions. Especially the inclusion of the time dimension enables the estimation method to find sequential behavioral patterns and dependencies and thus achieve better predictive results. Moreover, the approach works without the use of a feature engineering function to transform the selected features. We expect this characteristic to also enhance the predictive quality of the approach, as most transformations imply a

certain level of information loss. Also, the LSTM estimation method is especially adequate for the theoretical model, which should enhance performance. At the same, we deliberately omit even more complex estimation methods, as the input data contains only 24 months along the time dimension. For instance, one applicable model could be Bidirectional Encoder Representations from Transformers (BERT) as used by Yuan and Lin (2020). We assume that the input data is too short for more complex methods and that LSTM would outperform those more advanced methods.

5 Empiric Results

We estimate the different approaches presented in Section 4. The results are based on each approach's tuned estimation method. See appendix C for details on tuning.

Table 9: Main Results

	Precision	Recall	F1-Score
Collaborative Filter Approach	0.38	0.28	0.32
Cross Section Approach	0.66	0.39	0.49
Panel Approach	0.73	0.59	0.65

Table 9 contains the key results of the approaches. The main metric is the out-of-sample micro averaged overall F1-score. A clear hierarchy in performance is evident. The cross section approach outperforms the collaborative filter approach and the panel approach outperforms both. All three approaches converge to classification trade-offs, in which precision is higher than recall. This means all approaches classify conservatively, as they rather sparsely identify positive labels, but when they do, they do that with a relatively high degree of correctness. For the product recommender, this translates into an algorithm that only recommends a product group if it is very confident that the client will demand the product. The panel approach, for instance, has an out-of-sample performance of 73% precision and 59% recall. This means almost three quarters of the guesses, that a customer will buy a certain product group, were correct and the algorithm identified more than half of all purchases that occurred in the test samples.

As discussed in chapter 4, the F1-score averages precision and recall in a way that penalizes strongly uneven combinations of the two. All approaches show relatively similar combinations on different overall performance levels. We argue that in our results the F1-score metric does not discriminate any of the approaches' performance due to an uneven precision-recall combination. Therefore, the precision and recall scores confirm the

F1-Scores result that the panel approach performs best.

Figure 9: F-1 Score per Product Group

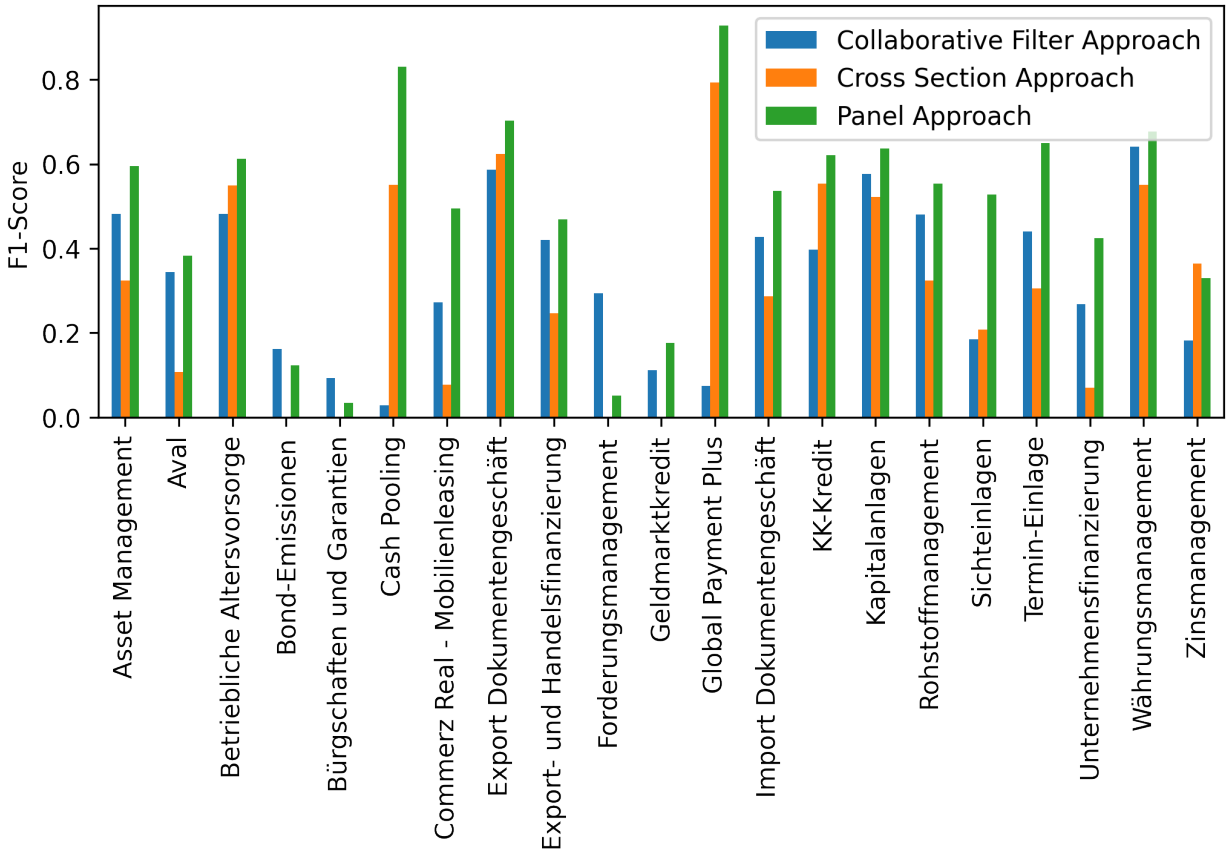


Figure 9 visualizes the F1-score per product group. All models show an uneven distribution of F1-scores across product groups. The collaborative filter approach and panel approach have F1-scores larger than zero for all product groups. This means the two models successfully predict at least some purchases in all product groups. The cross-sectional model fails to identify four categories entirely. All three approaches show very low F1-scores ($< 10\%$) in some product groups, implying poor performance. The collaborative filter approach shows low scores in four categories, cross sectional approach in six and the panel approach in three. In approximately 80% of all product groups the panel approach performs best. The collaborative filter approach cross-sectional share the remaining 20%. Considering F1-scores per product group, the panel model performs best. All approaches show rather similar distributions across product groups. This confirms the key results and its interpretation that the panel model shows the best performance.

Figure 10: Averaged Precision-Recall Trade-offs

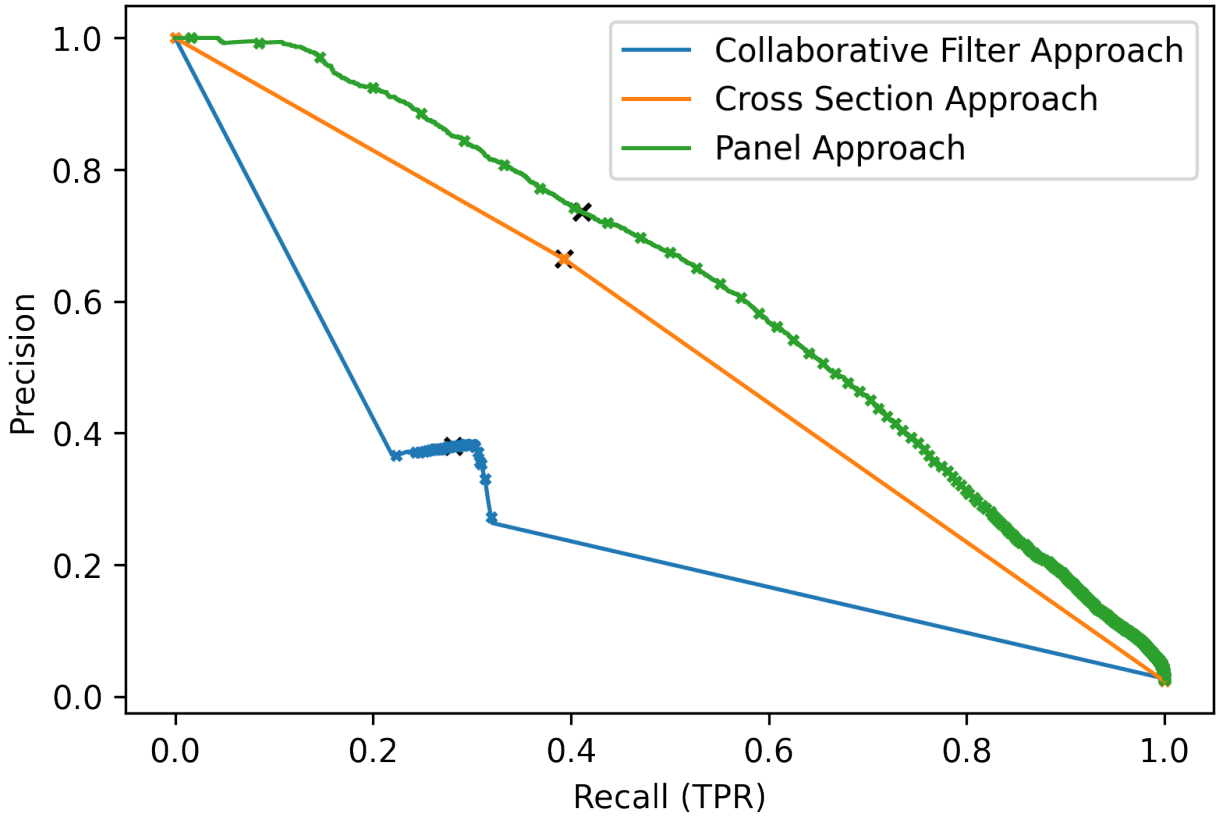


Figure 10 visualizes the distribution of possible micro averaged precision-recall trade-offs for each approach. Each cross on a curve in the plot represents one possible precision and recall trade-off for one approach given a threshold. The black cross on each line indicates the trade-off for the default threshold of 0.5. All trade-offs of the panel approach are above the cross sectional approach's curve. The lowest curve for all thresholds is the collaborative filter approach curve. Hence, for all possible thresholds the panel model outperforms the other two. Moreover, figure 10 shows that the panel model is the most versatile. The cross sectional model only has one valid trade-off at 66%/39% (precision/recall), independently of the threshold. The collaborative filter approach's trade-offs are all limited to the narrow space between 30%-40%/25%-30%. This means that no matter how we change the threshold, the performance stays within that range. Only the panel model is fully versatile, as its trade-offs range to performances of either very high precision or recall. For instance, two possible trade-off are approximately 90%/20% and 20%/85%. The three curves do no intersect. This means the key results hold for all thresholds. This finding justifies our use of the threshold 0.5 for the analysis. Yet, the curves are not equally smooth. The collaborative filter approach curve shows an outward spike on the right of

the default trade-off. At the default threshold the approach performs at 38%/ 28%. Setting the threshold to the peak of that spike would yield a performance of 39%/33%. This implies that the performance differences between approaches in the key results depend on the threshold. As the changes in differences are small, we neglect this.
It is worth noting, that the cross-sectional model is estimated by a decision tree, whose outputs are always concrete predictions of zeros and ones. So the outputs are $\in \{0, 1\}$ and not $\in [0, 1]$ like for the other two approaches. Therefore, the cross-sectional model only has three possible trade-offs on its curve. Two are the theoretical edge cases of setting the threshold either to zero or one. At the threshold of zero, the model predicts that every client always purchases every product group. In that case, recall is one and precision is very close to zero. At a threshold of one, the model predicts that every client is idle. In that case recall is zero and precision is undefined, i.e. NA%/0%. For every other threshold, the trade-off is the same, as the outputs are $\in \{0, 1\}$.

6 Discussion

The main implication of our results is that we can confirm that machine learning is useful in the domain of finance. However, our results indicate that rather complex modelling problems, like predicting the purchases for corporate banking products, can bring established machine learning approaches for that domain to their limits. Our results imply that more advanced methods, which require far more resources in implementation and computation, are better at handling those modelling problems.
We identify several limitations in our analysis. One is the choice of estimation methods in our proposed approaches. For each underlying model the options of estimation methods are endless. We choose correlation to estimate similarity between clients in the collaborative filer algorithm, and we estimate the cross section model with a decision tree classifier. Both of these methods are simplistic and well-established for similar problems. In the panel approach, the ANN LSTM estimation method variant is rather new for machine learning in finance, but known to work for tasks similar to ours in other fields. Choosing other methods for our proposed models might change the results drastically. As most authors working on similar problems, like Patidar et al. (2011) or Rahman and Khan (2018), we implicitly assume that this is not an issue. We only argue why our method choices are adequate and not why they are more adequate than others. This is a limitation for the analysis. A viable alternative would be to prove the assumption empirically. This can be achieved by proposing several estimation methods for each model and compare their performances, similarly to how to tune the hyperparameters of a single estimation method. We omit this alternative. Also, the analysis is constrained by the hyperparameter tuning.

Except for the collaborative filter algorithm, which is not a pure machine learning method and has a small hyperparameter space, the used methods' maximal margin of tuning is latent. That means we cannot guarantee that our tuning setup is able to find the globally best hyperparameters. The reason for this is that for any given method the total hyperparameter space is large, especially for the LSTM. As a consequence, further tweaking the approaches' estimation methods might change their performance significantly. We assume that defining our hyperparameter according to established rules of thumb renders the risk of suboptimal tuning unlikely.
Another limitation is threshold optimization. The output threshold impacts the quality of the trade-off between precision and recall. In chapter 5 we consider the average trade-off curve to evaluate the models and justify the default threshold value of 0.5. In a multilabel problem it is possible to define different thresholds for each category, which we omit.

Figure 11: Panel Approach: Precision-Recall Curve per Product Group

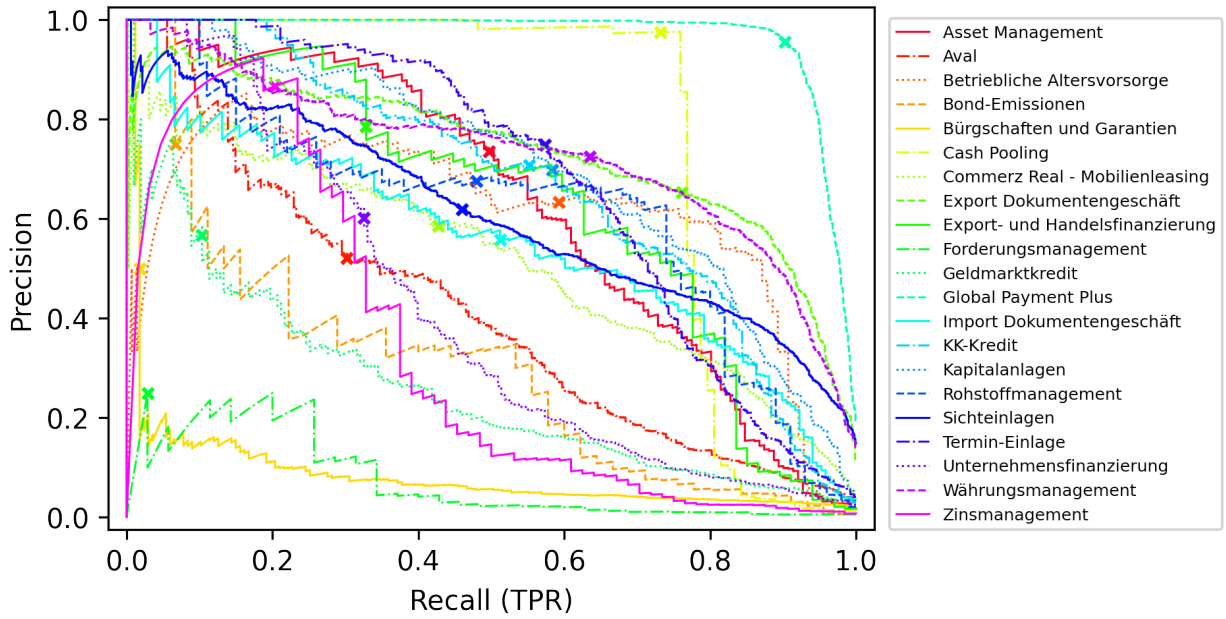


Figure 11 shows the precision-recall trade-offs per product group for the panel approach. Unlike its average trade-off curve, some curves for single categories are not smooth and show spikes. Inward, i.e. the bottom left corner, facing spikes imply a local minimum in trade-off quality, outward spikes vice versa. For instance the product group "Cash Pooling" shows an outward spike at approximately 95%/75% and "Forderungsmanagement" at 25%/25%. For the panel approach the individual trade-off curves show a high level of heterogeneity.

Figure 12: Collaborative Filter Approach: Precision-Recall Curve per Product Group

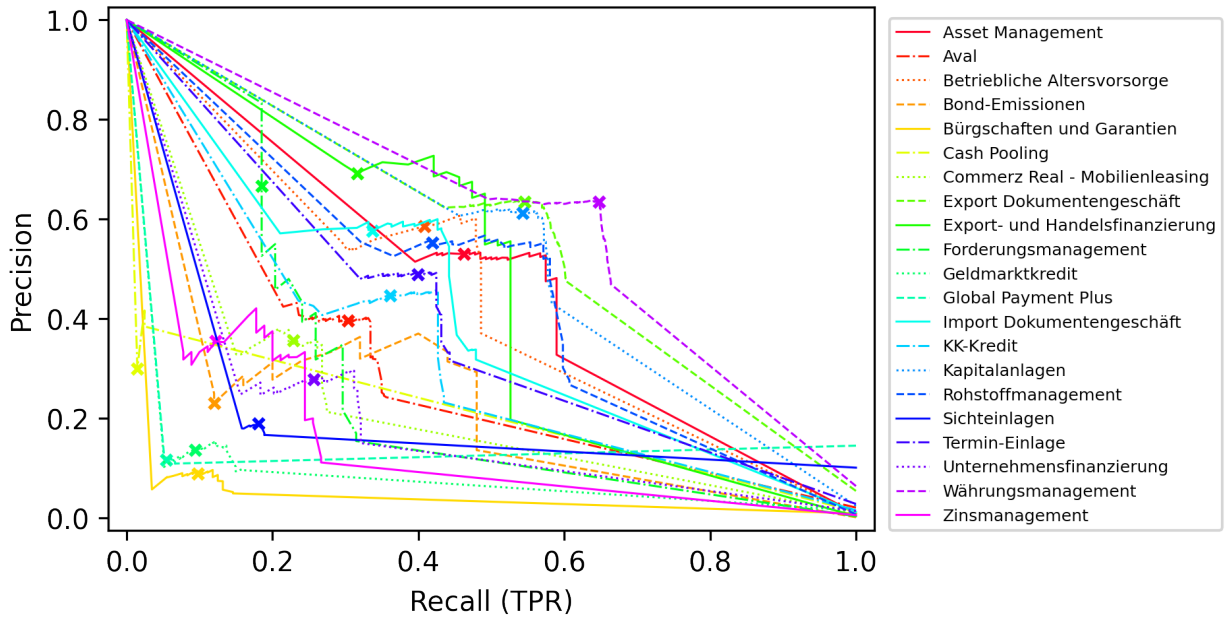


Figure 12 shows the per category trade-offs for the collaborative filter approach. The majority of trade-offs are similar to the micro averaged curve and thus have a high level of homogeneity. Again, there are no trade-offs for the cross section approach due to its estimation method. Clearly, setting a specific threshold for each product group would increase the overall performance of the collaborative filter and panel approach. Due to the higher heterogeneity in the individual trade-off curves in the panel approach, we suspect that it would receive the largest boost in performance from per category threshold optimization. How much this changes the overall results, remains latent. In our analysis we only consider the averaged trade-off curve and assume that category specific thresholds would not falsify the general results. Not proving this assumption empirically limits our analysis. Yet another impediment to our analysis is the unused performance potential within the estimation methods of the baseline approaches. For both approaches there exist more sophisticated variants, which might perform better in our use-case. For the collaborative filter approach, one example is the prize-winning solution based on a collaborative filter developed by Koren (2009) and his team. It is possible that their variant of the approach, or some other more complex variant outperforms the collaborative filter we proposed. For the decision tree method in the cross-sectional approach, we use the single-label classification concept to handle the multi-label context of the problem. There exists the alternative concept called Hierarchical Multi-label Classification (HMC). Vens et al. (2008) find that this alternative concept outperforms separate single label decision trees, because in the HMC variant the decision tree method can learn intra-label dependencies.

This means whereas the single-label classification concept accounts for cross dependencies in features but not in labels, the HMC variant accounts for both. Thus, for both estimation methods there might still be performance gains to be made. We omit that and thus limit the analysis. Moreover, the binary presentation of the revenue features limits the analysis. The binary format ensures that the revenue feature cannot be distorted. However, it also implies a large loss of information, as possibly relevant differences in the amounts of revenues are not available for the model. A viable alternative might be to keep the revenues as numeric features and normalize per product group and per client. This would lead to less information loss and therefore potentially better model performance.

The clearest avenue for further research is the transition from a behavioral classification model to a product recommendation algorithm. The motivation for the transition is that a recommendation system is practically applicable and the classification model is not. Considering again figure 9, a fundamental challenge is to define a kind of trade-off that turns the predictive model into an adequate product recommender. The equally fundamental problem in this is to find a definition for "adequate". On average and given the default threshold, our approaches predict with higher precision than recall. We argue that for our recommendation setting the added value of a successful recommendation outweighs the damage of an incorrect recommendation. Subsequently, the underlying algorithm should be tweaked preferring recall over precision. For other use-cases this might differ substantially. Another nuance of this challenge is that the theoretically most adequate precision-recall trade-off could be less performant. This aspect goes hand in hand with the question whether the recommendation problem might need different ways of evaluation altogether.

Another avenue is to strive for even better performances in the proposed classification problem. As discussed, we argue that this task differs in complexity compared to other use-cases of machine learning in finance, like for example customer churn prediction. In an effort to maximize performance, we solve the problem by modelling with a rich set of features together with a heavy estimation method to handle those features. We see potential for improvement in using a more holistic tuning approach, a larger feature space along all dimensions and more advanced estimation methods. Tuning is limited by its resources. The more computational power and the more time, the closer an estimation method can be nudged towards its maximum performance. This applies to our use-case, too, and tuning the LSTM method further could increase performance. There are potentially more client characteristics that are connected to purchase behavior. Those could be for instance geospatial data, i.e. where is the client based relative to the others, text based features, e.g. meeting notes of Commerzbank salespersons interacting with the client, or more general client information, e.g. foundation year, number of employees or legal status. Adding more meaningful features to the dataset could positively impact performance. Our setup

features 24 months for training, which is still relatively short. We expect that including longer time series will lead to better results. The longer the time dimension, the more information is available for the model to detect patterns that are connected to purchase behavior. Obviously, these patterns must be interactively connected to the revenue feature, as the revenues hold the purchase behavior information. Elsewise, the model cannot account for changes in purchase behavior itself along the time dimension.

Figure 13: F1 Score LSTM given different Features Lenths (Months)

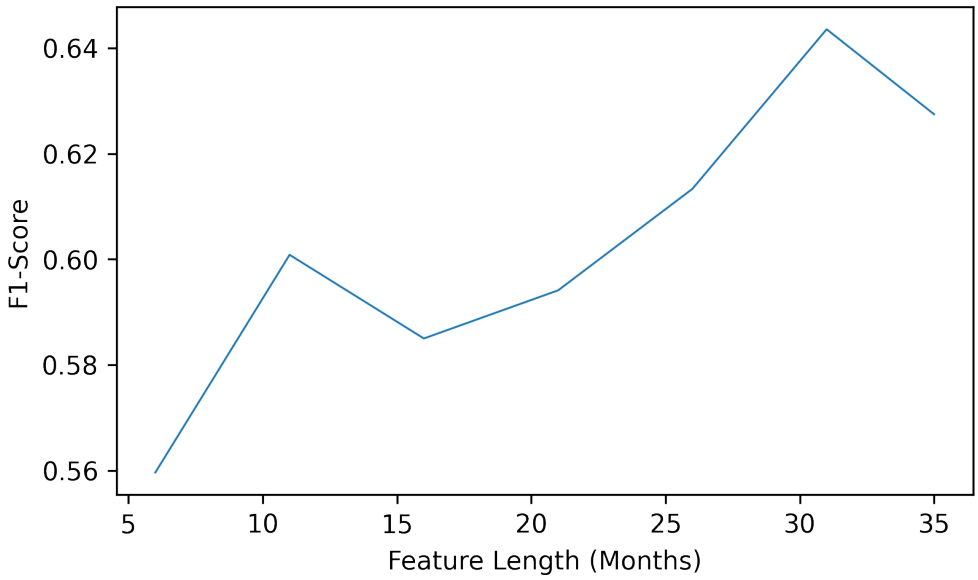


Figure 13 shows an estimate for the relationship between feature length along the time dimension and out of sample performance. The setup for this estimation is to newly train a LSTM network given the hyperparameters from tuning in the panel approach. We newly train the LSTM 20 times for different feature lengths ranging from 11 to 36 months. For each feature length, we save the largest F1-score of all runs. Figure 13 plots these maximum scores per feature length. The plot shows a positive correlation between feature length and performance. Due to the highly simplified nature of this estimation, we interpret it as an indication, that future research could find even better LSTM performances given longer feature time series. Furthermore, increasing the time dimension might also yield even more complex estimation methods, like for example BERT, adequate for the problem. More complex might mean better performance.

Another area for further research is to apply our proposed panel approach to established problems for machine learning in finance. Both credit fraud and churn identification are commonly solved by approaches similar to our cross sectional approach. As in this analysis the panel approach outperforms the cross section approach, it is likely that our results are reproducible in those tasks, too. The main characteristic of the cross-sectional

approach is its feature engineering to consolidate the time dimension of the input data. This implies that the underlying features are time series data, which the LSTM generally thrives on. The added value of that would be the prospect of better modelling performances, which ultimately saves (monetary) resources.

7 Conclusion

We formulate the task of modelling the purchase behavior within a complex product space as a multilabel classification problem. Our main finding is that the proposed panel approach estimated by an LSTM clearly outperforms the two baseline approaches. One of the two baseline approaches is a simple collaborative filter algorithm. The other is a model that consolidates its features with the concept of recency, frequency and monetary value and is estimated by a decision tree classifer.
There are two key implications of our results. Firstly, the proposed modelling problem is more complex than other problems common for machine learning in finance, such as fraud or churn identification. We argue that the product space for corporate clients is inherently more diverse and larger than for private clients, which makes their purchase behavior more difficult to predict. Secondly, our results imply that this more complex problem needs a more complex approach to be handled successfully.
With those main findings this paper gravitates towards the stream of publications that advocate the use of heavy machine learning methods in a conjunction with massive datasets. Yet, we also find that even though performing worse, simplistic algorithms can also handle complex modelling tasks with mediocre success.

References

Bahnsen, A. C., Aouada, D., Stojanovic, A., and Ottersten, B. (2016). Feature engineering strategies for credit card fraud detection. *Expert Systems with Applications*, 51:134–142.

Barreau, B. (2020). *Machine Learning for Financial Products Recommendation*. PhD thesis, Université Paris-Saclay.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Bennett, J., Lanning, S., et al. (2007). The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. Citeseer.

Bernardi, L., Kamps, J., Kiseleva, J., and Müller, M. J. (2015). The continuous cold start problem in e-commerce recommender systems. *arXiv preprint arXiv:1508.01177*.

Chicco, D. and Jurman, G. (2020). The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, 21:1–13.

Chollet, F. et al. (2015). Keras. `https://keras.io`.

Covington, P., Adams, J., and Sargin, E. (2016). Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198.

Fu, K., Cheng, D., Tu, Y., and Zhang, L. (2016). Credit card fraud detection using convolutional neural networks. In *International conference on neural information processing*, pages 483–490. Springer.

Ghosh, S. and Reilly, D. L. (1994). Credit card fraud detection with a neural-network. In *System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on*, volume 3, pages 621–630. IEEE.

Gu, S., Kelly, B., and Xiu, D. (2020). Empirical asset pricing via machine learning. *The Review of Financial Studies*, 33(5):2223–2273.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Koren, Y. (2009). The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 81(2009):1–10.

Kubat, M. (2017). *An introduction to machine learning*, volume 2. Springer.

Loh, W.-Y. (2011). Classification and regression trees. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 1(1):14–23.

Medsker, L. R. and Jain, L. (2001). Recurrent neural networks. *Design and Applications*, 5:64–67.

Patidar, R., Sharma, L., et al. (2011). Credit card fraud detection using neural network. *International Journal of Soft Computing and Engineering (IJSCE)*, 1(32-38).

Rahman, A. and Khan, M. N. A. (2018). A classification based model to assess customer behavior in banking sector. *Engineering, Technology & Applied Science Research*, 8(3):2949–2953.

Schafer, J. B., Frankowski, D., Herlocker, J., and Sen, S. (2007). Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer.

Sorokina, D. and Cantu-Paz, E. (2016). Amazon search: The joy of ranking products. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 459–460.

Sun, S., Cao, Z., Zhu, H., and Zhao, J. (2019). A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics*, 50(8):3668–3681.

Vens, C., Struyf, J., Schietgat, L., Džeroski, S., and Blockeel, H. (2008). Decision trees for hierarchical multi-label classification. *Machine learning*, 73(2):185–214.

Xie, Y., Li, X., Ngai, E., and Ying, W. (2009). Customer churn prediction using improved balanced random forests. *Expert Systems with Applications*, 36(3):5445–5449.

Yuan, Y. and Lin, L. (2020). Self-supervised pretraining of transformers for satellite image time series classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14:474–487.

Appendix

The purpose of this appendix is to further explain the algorithmic intuition of some concepts stated in the approach chapter. Moreover, it gives a short overview of how all proposed approaches are tuned. Lastly, this appendix contains all additional tables with a short description.

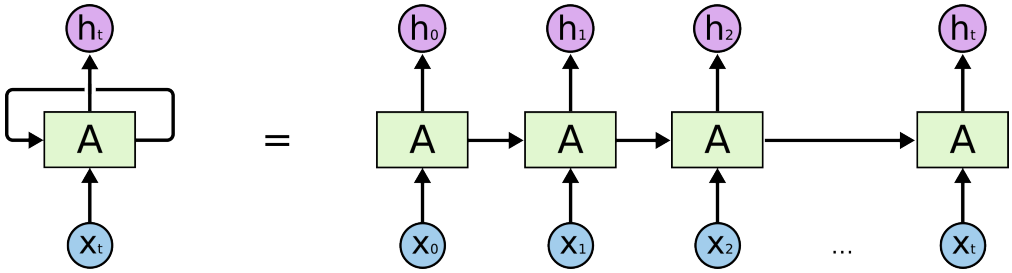
A Backpropagation

Backpropagation is the training process of most deep learning architectures, including ANNs, RNNs and LSTMs. The concept of this training algorithm is to move from back to front within the neural network for optimizing the weights. The first step of backpropagation is to calculate the error, i.e. the return value of the loss function, by comparing the estimated outputs given the current values of the network's weights and the real targets. Given the errors, the second step is to compute update values for the weights from the last hidden layer, which would minimize those errors. Those update values are calculated given the gradient, which is the derivative of the loss function given the current weights of the network. The gradient indicates how changing each weight impacts the error. This second step of calculating weight updates is then repeated for layers previous to the last hidden layer moving backwards through the network. The result of this are update values for all weights of the network given one training sample or one batch of several samples. Repeating that process for all training samples or batches and averaging the resulting weights changes, creates the final updates for all weights. Those averaged weight changes can be interpreted as the multidimensional direction towards which the neural net nudges its weights in an effort to find the solution that minimizes the error. Once the weights are updated, backpropagation can be applied again to update the weights. Each of these runs is called an epoch. Backpropagation is usually repeated as many epochs as necessary for the network to converge. How exactly in each layer for each sample the update values for the weights are calculated, depends on the optimization function. This function tries to achieve a good trade-off between being computationally efficient, converging in few training steps and getting as close to as possible to the best weight values, i.e. the weights that cause the lowest errors. Updating the weights is a minimization problem. Most optimization functions use derivatives to solve the problem. Sun et al. (2019) show that given how exactly each optimizer approaches the minimization, it has a different trade-off of the three mentioned convergence characteristics.

B The Vanshing Gradient Problem of the RNN

The vanshing gradient problem can occur in any neural network that is trained by backpropagation. It is a prominent cause for networks to fail to learn. A common symptom of this is that during trainig the network barely improves or stops improving at all. Depending on its computational details, the issue is called either the vanishing or the exploding gradient problem. Often both are referred to as just the vanishing gradient problem. Bengio et al. (1994) explain the mathematical process behind the problem and how it is especially common among RNNs due to their recurrent nodes. Like the ANN, the RNN is estimated by backpropagation. To account for the auto-connections in the recurrent nodes during estimation, the RNN can be understood as being unrolled into several ANNs. Each of them is like a duplicate of the original network at a certain previous time step.

Figure B.1: RNN Unrolled (https://colah.github.io/posts/2015-08-Understanding-LSTMs accessed 02/02/2023.)



Those ANNs are the units marked by $0, 1, 2, ...t$ in figure B.1. The network's error depends on all these units, as their stored information impacts the output of the last ANN. Computing the gradient of the unrolled RNN involves taking the derivative of a large term containing many duplicate elements. Due to the size of that term and the multiplicative connection of its elements, the risk of the gradient converging either to zero, i.e. to vanish, or converging to ∞, i.e. to explode, is high. In both cases the gradient fails to indicate the relation between each weight change and the error. This impedes the learning process of the network, which ultimately drowns the overall performance.

C Tuning Details

We optimize the hyperparameters of the estimation methods used in our proposed approaches. The exact hyperparameter spaces we use can be found in the respective jupyter notebook of the "codes" folder. For the experimental setup of the LSTM there is moreover a raw text document containing the relevant results for the random search runs.

For the collaborative filter and the decision tree classifier the hyperparameter space is manageably small. We define a subset of the total available space by general rules of thumb for these methods. We use a grid search algorithm to test every possible combination of hyperparameters. For both approaches we implement the tuning manually using common Python for-loops.

The LSTM method of the panel approach contains a large hyperparameter space. We select three optimizers from the space according to how adequate they appear for our problem given the theoretical properties as described by Sun et al. (2019). We further narrow down the hyperparameter space by using established rules of thumbs and accounting for computational limitations. For instance, the GPU on our virtual machine cannot handle hidden layers containing more than 2,000 units. The three optimizer in our hyperparameter subset are Stochastic Gradient Descent (SGD), Adaptive Moment Estimation (ADAM) and Root Mean Squared Propagation (RMSProp). We exhaustively use random search to identify RMSprop as the most adequate optimizer. Then we use RMSProp and its smaller hyperparameter space with a grid search to find the best combination for the LSTM. We utilize the "KerasTuner" framework developed by Chollet et al. (2015) for tuning the LSTM.

D Additional Figures and Tables

Figure D.1: Revenue Matrices for 10 Random Clients

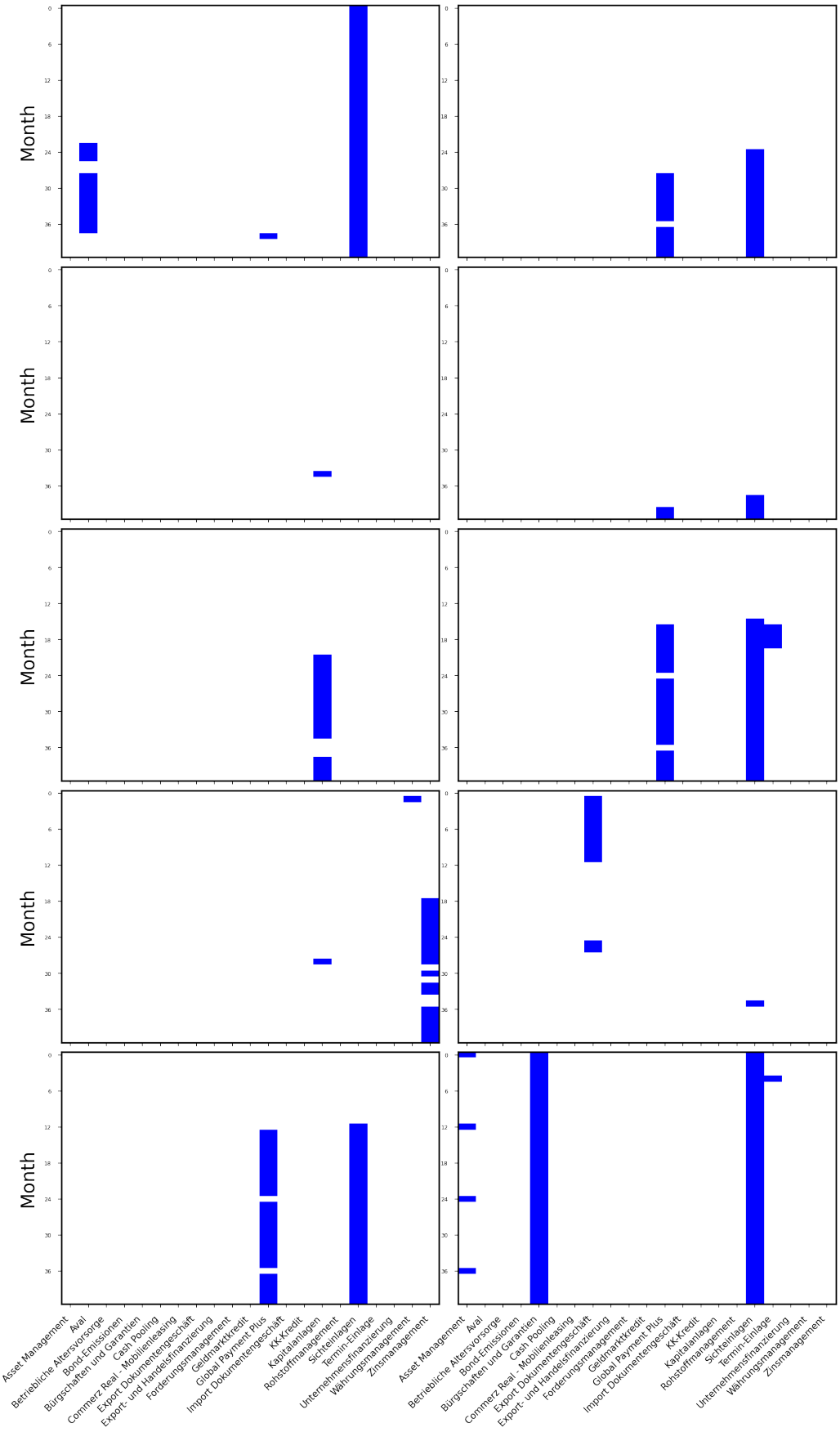


Figure D.2: Purchase Matrices for the same 10 Random Clients

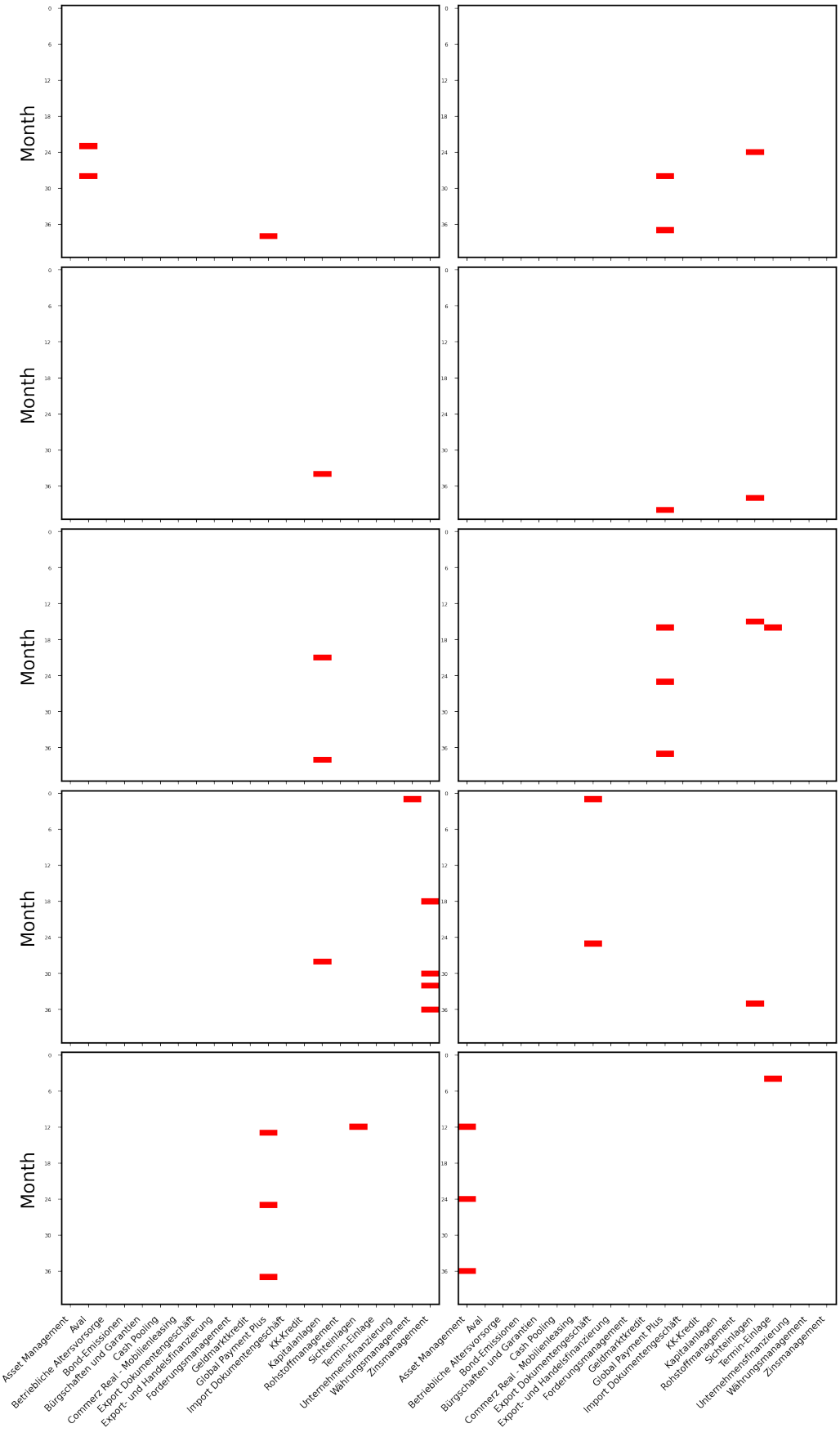


Figure D.1 and D.2 show the revenue and purchase matrices for the 10 randomly chosen clients. For instance, the left revenue matrix in the top row of D.1 shows that the client created revenues with products from the product group "Aval", "Global Payment Plus" and "Sichteilagen". The respective purchase matrix in D.2 shows that this client purchased the product groups "Aval", "Global Payment Plus". The product group "Sichteilagen" is not considered a purchase as at no point in the matrix the revenue feature changes from 1 to 0.

Table D.1: Categorical Features Summary: Boni Rating (2/3)

	Count	Percent
None	14,861	36.11%
24	4,370	10.62%
26	3,380	8.21%
22	3,145	7.64%
28	2,585	6.28%
Other (26)	12,818	31.14%

Table D.1 shows the summary statistics for Boni Rating, which is the second of three categorical features contained within the dataset. It shows "None" values for more than one third of all samples. "None" means that the bank did not evaluate the client's credit risk.

Table D.2: Categorical Features Summary: Client Industry (3/3)

	Count	Percent
Großhandel (ohne Handel mit Kraftfahrzeug	5,190	12.61%
Mit Finanz- und Versicherungsdienstleistu	2,808	6.82%
Maschinenbau	2,732	6.64%
Grundstücks- und Wohnungswesen	2,627	6.38%
Herst. von Metallerzeugnissen	1,465	3.56%
Other (86)	26,337	63.99%

Table D.2 shows the summary statistics for Client Industry, the third of three categorical features contained within the dataset. It gives the information in which industry the client operates.