Solving the false positives problem in fraud prediction using automated feature engineering

Roy Wedge[1], James Max Kanter[1], Kalyan Veeramachaneni[1], Santiago Moral Rubio[2], and Sergio Iglesias Perez[2]

[1] Data to AI Lab, LIDS, MIT, Cambridge, MA-02139
[2] Banco Bilbao Vizcaya Argentaria (BBVA), Madrid, Spain

Abstract

In this paper, we present an automated feature engineering based approach to dramatically reduce false positives in fraud prediction. False positives plague the fraud prediction industry. It is estimated that only 1 in 5 declared as fraud are actually fraud and roughly 1 in every 6 customers have had a valid transaction declined in the past year. To address this problem, we use the Deep Feature Synthesis algorithm to automatically derive behavioral features based on the historical data of the card associated with a transaction. We generate 237 features (>100 behavioral patterns) for each transaction, and use a random forest to learn a classifier. We tested our machine learning model on data from a large multinational bank and compared it to their existing solution. On an unseen data of 1.852 million transactions, we were able to reduce the false positives by 54% and provide a savings of 190K euros. We also assess how to deploy this solution, and whether it necessitates streaming computation for real time scoring. We found that our solution can maintain similar benefits even when historical features are computed once every 7 days.

1 Introduction

Fraud detection problems are well-defined supervised learning problems, and data scientists have long been applying machine learning to help solve them [2, 7]. However, false positives still plague the industry [10] with rates as high as 10-15%. Only 1 in 5 transactions declared as fraud be truly fraud [10]. Analysts have pointed out that these high false positives may be costing merchants more then fraud itself [3].

To mitigate this, most enterprises have adopted a multi-step process that combines work by human analysts and machine learning models. This process usually starts with a machine learning model generating a risk score and combining it with expert-driven rules to sift out potentially fraudulent transactions. The resulting alerts pop up in a 24/7 monitoring center, where they are examined and diagnosed by human analysts. This process can potentially reduce the false positive rate by 5% – but this improvement comes only with high (and very costly) levels of human involvement. Even with such systems in place, a large number of false positives remain.

[3] https://blog.riskified.com/true-cost-declined-orders/

In this paper, we present an improved machine learning solution to drastically reduce the "false positives" in the fraud prediction industry. Such a solution will not only have financial implications, but also reduce the alerts at the 24/7 control center, enabling security analysts to use their time more effectively, and thus delivering the true promise of machine learning/artificial intelligence technologies.

We use a large, multi-year dataset from BBVA, containing 900 million transactions. We were also given fraud reports that identified a very small subset of transactions as fraudulent. Our task was to develop a machine learning solution that: (a) uses this rich transactional data in a transparent manner (no black box approaches), (b) competes with the solution currently in use by BBVA, and (c) is deployable, keeping in mind the real-time requirements placed on the prediction system.

We would be remiss not to acknowledge the numerous machine learning solutions achieved by researchers and industry alike (more on this in Section 2). However, the value of extracting behavioral features from historical data has been only recently recognized as an important factor in developing these solutions – instead, the focus has generally been on finding the best possible model given a set of features, and even after this, few studies focused on extracting a handful of features. Recognizing the importance of feature engineering [5] – in this paper, we use an automated feature engineering approach to generate hundreds of features to dramatically reduce the false positives.

Key to success is automated feature engineering: Having access to rich information about cards and customers exponentially increases the number of possible features we can generate. However, coming up with ideas, manually writing software and extracting features can be time-consuming, and may require customization each time a new bank dataset is encountered. In this paper, we use an automated method called *deep feature synthesis*(DFS) to rapidly generate a rich set of features that represent the patterns of use for a particular account/card. Examples of features generated by this approach are presented in Table 4.

As per our assessment, because we were able to perform feature engineering automatically *via* Featuretools and machine learning tools, we were able to focus our efforts and time on understanding the domain, evaluating the machine learning solution for financial metrics (>60% of our time), and communicating our results. We imagine tools like these will also enable others to focus on the real problems at hand, rather than becoming caught up in the mechanics of generating a machine learning solution.

Deep feature synthesis obviates the need for streaming computing: While the deep feature synthesis algorithm can generate rich and complex features using historical information, and these features achieve superior accuracy when put through machine learning, it still needs to be able to do this in real time in order to feed them to the model. In the commercial space, this has prompted the development of streaming computing solutions.

But, what if we could compute these features only once every t days instead? During the training phase, the abstractions in deep feature synthesis allow features to be computed with such a "*delay*," and for their accuracy to be tested, all by setting a single parameter. For example, for a transaction that happened on August 24th, we could use features that had been generated on August 10th. If accuracy is maintained, the implication is that aggregate features need to be only computed once every few days, obviating the need for streaming computing.

What did we achieve?

DFS achieves a 91.4% increase in precision compared to BBVA’s current solution. This comes out to a reduction of 155,870 false positives in our dataset – a 54% reduction.

The DFS-based solution saves 190K euros over 1.852 million transactions - a tiny fraction of the total transactional volume. These savings are over 1.852 million transactions, only a tiny fraction of BBVA’s yearly total, meaning that the true annual savings will be much larger.

We can compute features, once every 35 days and still generate value Even when DFS features are only calculated once every 35 days, we are still able to achieve an improvement of 91.4% in precision. However, we do lose 67K euros due to approximation, thus only saving 123K total euros. This unique capability makes DFS is a practically viable solution.

	Information type	**Attribute recorded**
Verification results	Card	captures information about unique situations during card verification.
	Terminal	captures information about unique situations during verification at a terminal.
About the location	Terminal	can print/display messages
		can change data on the card
		maximum pin length it can accept
		serviced or not
		how data is input into the terminal
	Authentication mode	device type
About the merchant		unique id
		bank of the merchant
		type of merchant
		country
About the card		authorizer
About the transaction		amount
		timestamp
		currency
		presence of a customer

Table 1. A transaction, represented by a number of attributes that detail every aspect of it. In this table, we are showing *only* a fraction of what is being recorded in addition to the $amount$, $timestamp$ and $currency$ for a transaction. These range from whether the customer was present physically for the transaction to whether the terminal where the transaction happened was serviced recently or not. We categorize the available information into several categories.

2 Related work

Fraud detection systems have existed since the late 1990s. Initially, a limited ability to capture, store and process data meant that these systems almost always relied on expert-

Item	Number
Cards	7,114,018
Transaction log entries	903,696,131
Total fraud reports	172,410
Fraudulent use of card number reports	122,913
Fraudulent card reports matched to transaction	111,897

Table 2. Overview of the data we use in this paper

driven rules. These rules generally checked for some basic attributes pertaining to the transaction – for example, "Is the transaction amount greater then a threshold?" or "Is the transaction happening in a different country?" They were used to block transactions, and to seek confirmations from customers as to whether or not their accounts were being used correctly.

Next, machine learning systems were developed to enhance the accuracy of these systems [2, 7]. Most of the work done in this area emphasized the modeling aspect of the data science endeavor – that is, learning a *classifier*. For example, [4, 12] present multiple classifiers and their accuracy. Citing the non-disclosure agreement, they do not reveal the fields in the data or the features they created. Additionally, [4] present a solution using only transactional features, as information about their data is unavailable.

Starting with [11], researchers have started to create small sets of hand-crafted features, aggregating historical transactional information [1, 9]. [13] emphasize the importance of aggregate features in improving accuracy. In most of these studies, aggregate features are generated by aggregating transactional information from the immediate past of the transaction under consideration. These are features like "number of transactions that happened on the same day", or "amount of time elapsed since the last transaction".

Fraud detection systems require instantaneous responses in order to be effective. This places limits on real-time computation, as well as on the amount of data that can be processed. To enable predictions within these limitations, the aggregate features used in these systems necessitate a streaming computational paradigm in production [4], [5] [3]. As we will show in this paper, however, aggregate summaries of transactions that are as old as 35 days can provide similar precision to those generated from the most recent transactions, up to the night before. This poses an important question: When is streaming computing necessary for predictive systems? Could a comprehensive, automatic feature engineering method answer this question?

3 Dataset preparation

Looking at a set of multiyear transactional data provided to us – a snapshot of which is shown in Table 1 – a few characteristics stand out:

Rich, extremely granular information: Logs now contain not only information about a transaction's *amount*, *type*, *time stamp* and *location*, but also tangentially related mate-

[4] https://mapr.com/blog/real-time-credit-card-fraud-detection-apache-spark-and-event-streaming/

[5] https://www.research.ibm.com/foiling-financial-fraud.shtml

rial, such as the attributes of the terminal used to make the transaction. In addition, each of these attributes is divided into various subcategories that also give detailed information. Take, for example, the attribute that tells “*whether a terminal can print/display messages*”. Instead of a binary “*yes*” or “*no*,” this attribute is further divided into multiple subcategories: “*can print*”, “*can print and display*”, *“can display”*, “*cannot print or display*”, and *“unknown”*. It takes a 59-page dictionary to describe each transaction attribute and all of its possible values.

Historical information about card use: Detailed, transaction-level information for each card and/or account is captured and stored at a central location, starting the moment the account is activated and stopping only when it is closed. This adds up quickly: for example, the dataset we received, which spanned roughly three years, contained 900 million transactions. Transactions from multiple cards or accounts belonging to the same user are now linked, providing a full profile of each customer’s financial transactions.

Table 2 presents an overview of the data we used in this paper – a total of 900 million transactions that took place over a period of 3 years. A typical transactional dataset is organized into a three-level hierarchy: $Customers \leftarrow Cards \leftarrow Transactions$. That is, a transaction belongs to a card, which belongs to a customer. Conversely, a card may have several transactions, and a customer may have multiple cards. This relational structure plays an important role in identifying subsamples and developing features.

Before developing predictive models from the data, we took several preparative steps typical to any data-driven endeavor. Below, we present two data preparation challenges that we expect to be present across industry.

	Original Fraud	**Non-Fraud**
# of Cards	34378	36848
# of fraudulent transactions	111,897	0
# of non-fraudulent transactions	4,731,718	4,662,741
# of transactions	4,843,615	4,662,741

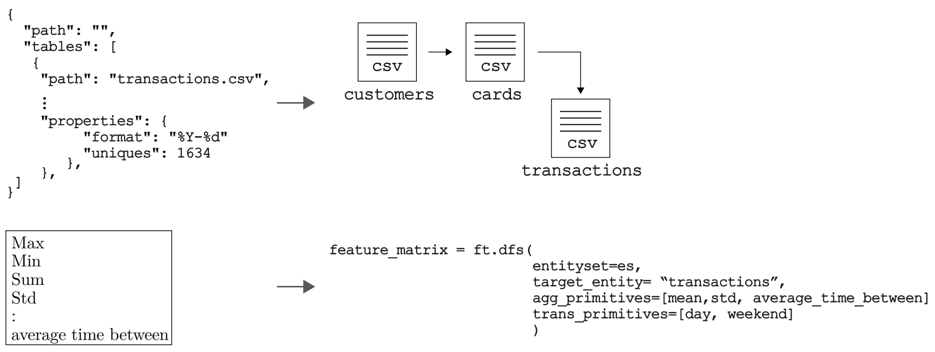
Table 3. The representative sample data set we extracted for training.

Identifying a data subsample: Out of the 900 million transactions in the dataset, only 122,000 were fraudulent. Thus, this data presents a challenge that is very common in fraud detection problems – less then 0.002% of the transactions are fraudulent. To identify patterns pertaining to fraudulent transactions, we have to identify a subsample. Since we have only few examples of fraud, each transaction is an important training example, and so we choose to keep every transaction that is annotated as fraud.

However, our training set must also include a reasonable representation of the non-fraudulent transactions. We could begin by sampling randomly – but the types of features we are attempting to extract also require historical information about the card and the customer to which a given transaction belongs. To enable the transfer of this information, we have to sample in the following manner:

1. Identify the cards associated with the fraudulent transactions,
   – Extract all transactions from these cards,
2. Randomly sample a set of cards that had no fraudulent transactions and,
   – Extract all transactions from these cards.

Table 3 presents the sampled subset. We formed a training subset that has roughly 9.5 million transactions, out of which only 111,897 are fraudulent. These transactions give a complete view of roughly 72K cards.



```
{
  "path": "",
  "tables": [
   {
    "path": "transactions.csv",
    ⋮
    "properties": {
        "format": "%Y-%d"
        "uniques": 1634
       },
    },
  ]
}
```

```
feature_matrix = ft.dfs(
                        entityset=es,
                        target_entity= "transactions",
                        agg_primitives=[mean,std, average_time_between]
                        trans_primitives=[day, weekend]
                        )
```

Fig. 1. The process of automatic feature generation. User specifies a `metadata` file that describes the relationships between multiple `csvs`, the path to the data files and several properties for each of the fields in each of the `csvs`. The three files for this problem are `customers` that has the list of all unique customers, `cards` that has the list of all unique cards, and `transactions`. The arrows represent one-to-many relationships. Given these two pieces of information, a user can select primitives in the featuretools library and compute the features. The library is available as open source at: https://github.com/featuretools/featuretools/

4 Automated feature generation

Given the numerous attributes collected during every transaction, we can generate hypotheses/features in two ways:

- **By using only transaction information**: Each recorded transaction has a number of attributes that describe it, and we can extract multiple features from this information alone. Most features are binary, and can be thought of as answers to yes-or-no questions, along the lines of *"Was the customer physically present at the time of transaction?"*. These features are generated by converting categorical variables using `one-hot-encoding`. Additionally, all the numeric attributes of the transaction are taken as-is.
- **By aggregating historical information**: Any given transaction is associated with a *card*, and we have access to all the historical transactions associated with that *card*. We can generate features by aggregating this information. These features are

mostly numeric – one example is, *“What is the average amount of transactions for this card?”*. Extracting these features is complicated by the fact that, when generating features that describe a transaction at time t, one can only use aggregates generated about the $card$ using the transactions that took place *before* t. This makes this process computationally expensive during the model training process, as well as when these features are put to use.

Broadly, this divides the features we can generate into two types: (a) so-called “transactional features,” which are generated from transactional attributes alone, and (b) features generated using historical data along with transactional features. Given the number of attributes and aggregation functions that could be applied, there are numerous potential options for both of these feature types.

Our goal is to automatically generate numerous features and test whether they can predict fraud. To do this, we use an automatic feature synthesis algorithm called Deep Feature Synthesis [8]. An implementation of the algorithm, along with numerous additional functionalities, is available as open source tool called `featuretools` [6]. We exploit many of the unique functionalities of this tool in order to to achieve three things: (a) a rich set of features, (b) a fraud model that achieves higher precision, and (c) approximate versions of the features that make it possible to deploy this solution, which we are able to create using a unique functionality provided by the library. In the next subsection, we describe the algorithm and its fundamental building blocks. We then present the types of features that it generated.

Deep Feature Synthesis The purpose of Deep Feature Synthesis (DFS) is to automatically create new features for machine learning using the relational structure of the dataset. The relational structure of the data is exposed to DFS as *entities* and *relationships*.

An entity is a list of instances, and a collection of attributes that describe each one – not unlike a table in a database. A transaction entity would consist of a set of transactions, along with the features that describe each transaction, such as the transaction amount, the time of transaction, etc.

A relationship describes how instances in two entities can be connected. For example, the point of sale (POS) data and the historical data can be thought of as a “Transactions” entity and a ”Cards” entity. Because each card can have many transactions, the relationship between Cards and Transactions can be described as a “parent and child” relationship, in which each parent (Card) has one or more children (Transactions).

Given the relational structure, DFS searches a built-in set of *primitive feature functions*, or simply called “primitives”, for the best ways to synthesize new features. Each primitive in the system is annotated with the data types it accepts as inputs and the data type it outputs. Using this information, DFS can stack multiple primitives to find *deep features* that have the best predictive accuracy for a given problems.

The primitive functions in DFS take two forms.

– Transform primitives: This type of primitive creates a new feature by applying a function to an existing column in a table. For example, the `Weekend` primitive could accept the transaction date column as input and output a columns indicating whether the transaction occurred on a weekend.

– Aggregation primitives: This type of primitive uses the relations between rows in a table. In this dataset, the transactions are related by the id of the card that made them. To use this relationship, we might apply the `Sum` primitive to calculate the total amount spent to date by the card involved in the transaction.

Synthesizing deep features: For high value prediction problems, it is crucial to explore a large space of potentially meaningful features. DFS accomplishes this by applying a second primitive to the output of the first. For example, we might first apply the `Hour` transform primitive to determine when during the day a transaction was placed. Then we can apply `Mean` aggregation primitive to determine average hour of the day the card placed transactions. This would then read like `cards.Mean(Hour(transactions.date))` when it is auto-generated. If the card used in the transaction is typically only used at one time of the day, but the transaction under consideration was at a very different time, that might be a signal of fraud.

Following this process of stacking primitives, DFS enumerates many potential features that can be used for solving the problem of predicting credit card fraud. In the next section, we describe the features that DFS discovered and their impact on predictive accuracy.

Features aggregating information from all the past transactions	
Expression	Description
cards.MEAN(transactions.amount)	Mean of transaction amount
cards.STD(transactions.amount)	Standard deviation of the transaction amount
cards.AVG_TIME_BETWEEN(transactions.date)	Average time between subsequent transactions
cards.NUM_UNIQUE(transactions.DAY(date))	Number of unique days
cards.NUM_UNIQUE(transactions.tradeid)	Number of unique merchants
cards.NUM_UNIQUE(transactions.mcc)	Number of unique merchant categories
cards.NUM_UNIQUE(transactions.acquirerid)	Number of unique acquirers
cards.NUM_UNIQUE(transactions.country)	Number of unique countries
cards.NUM_UNIQUE(transactions.currency)	Number of unique currencies

Table 4. Features generated using DFS primitives. Each feature aggregates data pertaining to past transactions from the card. The *left* column shows how the feature is computed via. an expression. The *right* column describes the feature in English. These features capture patterns in the transactions that belong to a particular card. For example, what was the mean value of the amount.

5 Modeling

After the feature engineering step, we have 236 features for 4,843,615 transactions. Out of these transactions, only 111,897 are labeled as fraudulent. With machine learning, our goal is to (a) learn a model that, given the features, can predict which transactions have this label, (b) evaluate the model and estimate its generalizable accuracy metric, and (c) identify the features most important for prediction. To achieve these three goals, we utilize a random forest classifier, which uses subsampling to learn multiple decision trees from the same data. We used `scikit-learn`'s classifier with 100 trees by setting `n_estimators=100`, and used `class_weight = 'balanced'`.

5.1 Evaluating the model

To enable comparison in terms of “false positives”, we assess the model comprehensively. Our framework involves (a) meticulously splitting the data into multiple exclusive subsets, (b) evaluating the model for machine learning metrics, and (c) comparing it to two different baselines. Later, we evaluate the model in terms of the financial gains it will achieve (in Section 6).

Machine learning metric To evaluate the model, we assessed several metrics, including the area under the receiver operating curve (AUC-ROC). Since non-fraudulent transactions outnumber fraudulent transactions 1000:1, we first pick the operating point on the ROC curve (and the corresponding threshold) such that the true positive rate for fraud detection is $> 89\%$, and then assess the model's `precision`, which measures how many of the blocked transactions were in fact fraudulent. For the given true positive rate, the precision reveals what losses we will incur due to false positives.

Data splits: We first experiment with all the cards that had one or more fraudulent transactions. To evaluate the model, we split it into mutually exclusive subsets, while making sure that fraudulent transactions are proportionally represented each time we split. We do the following:

- we first split the data into training and testing sets. We use 55% of the data for training the model, called D_{train}, which amounts to approximately 2.663 million transactions,
- we use an additional 326K, called D_{tune}, to identify the threshold - which is part of the training process,
- the remaining 1.852 million million transactions are used for testing, noted as D_{test}.

Baselines: We compare our model with two baselines.

- **Transactional features baseline**: In this baseline, we only use the fields that were available at the time of the transaction, and that are associated with it. We do not use any features that were generated using historical data via DFS. We use `one-hot-encoding` for categorical fields. A total of 93 features are generated in this way. We use a random forest classifier, with the same parameters as we laid out in the previous section.
- **Current machine learning system at BBVA**: For this baseline, we acquired risk scores that were generated by the existing system that BBVA is currently using for fraud detection. We do not know the exact composition of the features involved, or the machine learning model. However, we know that the method uses only transactional data, and probably uses neural networks for classification.

Evaluation process:

- Step 1: Train the model using the training data - D_{train}.
- Step 2: Use the trained model to generate prediction probabilities, P_{tu} for D_{tune}.
- Step 3: Use these prediction probabilities, and true labels L_{tu} for D_{tune} to identify the threshold. The threshold γ is given by:

$$\gamma = \underset{\gamma}{\operatorname{argmax}}\ \text{precision}_\gamma \times u\left(\text{tpr}_\gamma - 0.89\right) \tag{1}$$

where `tpr`$_\gamma$ is the true positive rate that can be achieved at threshold γ and u is a unit step function whose value is 1 when `tpr`$_\gamma \geq 0.89$. The true positive rate (tpr) when threshold γ is applied is given by:

$$\mathrm{tpr}_\gamma = \frac{\sum_i \delta(P^i_{tu} \geq \gamma)}{\sum_i L^i_{tu}}, \forall i, \quad where \quad L^i_{tu} = 1 \tag{2}$$

where $\delta(.) = 1$ when $P^i_{tu} \geq \gamma$ and 0 otherwise. Similarly, we can calculate fpr_γ (false positive rate) and $precision_\gamma$.

– Step 4: Use the trained model to generate predictions for D_{test}. Apply the threshold γ and generate predictions. Evaluate `precision`, `recall` and `f-score`. Report these metrics.

Metric	Transactional	Current system	DFS
Precision	0.187	0.1166	**0.41**
F-Score	0.30	0.20	**0.56**

Table 5. `Precision` and `f-score` achieved in detecting non-fraudulent transactions at the fixed `recall` (a.k.a true positive rate) of $>=$ 0.89. We compare the performance of features generated using the deep feature synthesis algorithm to those generated by "`one-hot-encoding''` of transactional attributes, and those generated by the baseline system currently being used. These baselines are described above. A

Results and discussion DFS solution: In this solution we use the features generated by the DFS algorithm as implemented in `featuretools`. A total of 236 features are generated, which include those generated from the fields associated with the transaction itself. We then used a random forest classifier with the hyperparameter set described in the previous section.

In our case study, the transactional features baseline system has a false positive rate of 8.9%, while the machine learning system with DFS features has a false positive rate of 2.96%, a reduction of 6%.

When we fixed the true positive rate at $> 89\%$, our precision for the transactional features baseline was 0.187. For the model that used DFS features, we got a precision of 0.41, a >2x increase over the baseline. When compared to the current system being used in practice, we got a >3x improvement in precision. The current system has a precision of only about 0.1166.

6 Financial evaluation of the model

To assess the financial benefit of reducing false positives, we first detail the impact of false positives, and then evaluate the three solutions. When a false positive occurs, there is the possibility of losing a sale, as the customer is likely to abandon the item s/he was trying to buy. A compelling report published by Javelin Strategy & Research reports that these blocked sales add up to $118 billion, while the cost of real card fraud only amounts to $9 billion [10]. Additionally, the same [10] study reports that 26% of shoppers whose cards were declined reduced their shopping at that merchant following the decline, and 32% stopped entirely. There are numerous other costs for the merchant when a customer is falsely declined [6].

From a card issuer perspective, when possibly authentic sales are blocked, two things can happen: the customer may try again, or may switch to a different issuer (different card). Thus issuers also lose out on millions in interchange fees, which are assessed at 1.75% of every transaction.[7] Additionally, it also may cause customer retention problems. Hence, banks actively try to reduce the number of cards affected by false positives.

Method	False postitives		False Negatives		Total Cost (€)
	Number	Cost (€)	Number	Cost (€)	
Current system	289,124	319,421.93	**4741**	**125,138.24**	**444,560**
Transactional features only	162,302	96,139.09	5061	818,989.95	915,129.05
DFS	**53,592**	**39,341.88**	5247	638,940.89	678,282.77

Table 6. Losses incurred due to false positives and false negatives. This table shows the results when `threshold` is tuned to achieve *tpr* $\geq$ 0.89. **Method**: We aggregate the `amount` for each false positive and false negative. False negatives are the frauds that are not detected by the system. We assume the issuer fully reimburses this to the client. For false positives, we assume that 50% of transactions will not happen using the card and apply a factor of 1.75% for interchange fee to calculate losses. These are estimates for the validation dataset which contained approximately 1.852 million transactions.

To evaluate the financial implications of increasing the precision of fraud detection from 0.1166 to 0.41, we do the following:

- We first predict the *label* for the 1.852 million transactions in our test dataset using the model and the threshold derived in Step 3 of the “evaluation process”. Given the true *label*, we then identify the transactions that are falsely labeled as frauds.
- We assess the financial value of the false positives by summing up the amount of each of the transactions (in Euros).
- Assuming that 50% of these sales may successfully go through after the second try, we estimate the loss in sales using the issuer’s card by multiplying the total sum by 0.5.

[6] https://blog.riskified.com/true-cost-declined-orders/

[7] ”Interchange fee” is a term used in the payment card industry to describe a fee paid between banks for the acceptance of card-based transactions. For sales/services transactions, the merchant’s bank (the ”acquiring bank”) pays the fee to a customer’s bank (the ”issuing bank”).

– Finally, we assess the loss in interchange fees for the issuer at 1.75% of the number in the previous step. This is the cost due to false positives - $cost_{fp}$
– Throughout our analysis, we fixed the true positive rate at 89%. To assess the losses incurred due to the remaining $\tilde{1}0\%$, we sum up the total amount across all transactions that our model failed to detect as fraud. This is the cost due to false negatives - $cost_{fn}$
– The total cost is given by

$$totalcost = cost_{fp} + cost_{fn}$$

By doing the simple analysis as above, we found that our model generated using the DFS features was able to reduce the false positives significantly and was able to reduce the $cost_{fp}$ when compared to BBVA's solution (€39,341.88 *vs.* €319,421.93). But it did not perform better then BBVA overall in terms of the total $cost$, even though there was *not* a significant difference in the number of false negatives between DFS based and BBVA's system. Table 6 presents the detailed results when we used our current model as if. This meant that BBVA's current system does really well in detecting high valued fraud. To achieve similar effect in detection, we decided to re-tune the threshold.
Retuning the threshold: To tune the threshold, we follow the similar procedure described in Section 5.1, under subsection titled "Evaluation process", except for one change. In Step 2 we weight the probabilities generated by the model for a transaction by multiplying the amount of the transaction to it. Thus,

$$P^i_{tu} \leftarrow P^i_{tu} \times amount^i \tag{3}$$

We then find the `threshold` in this new space. For test data, to make a decision we do a similar transformation of the probabilities predicted by a classifier for a transaction. We then apply the threshold to this new transformed values and make a decision. This weighting essentially reorders the the transactions. Two transactions both with the same prediction probability from the classifier, but vastly different amounts, can have different predictions.
Table 7 presents the results when this new `threshold` is used. A few points are noteworthy:

– **DFS model reduces the total cost BBVA would incur by atleast 190K euros**. It should be noted that these set of transactions, 1.852 million, only represent a tiny fraction of overall volume of transactions in a year. We further intend to apply this model to larger dataset to fully evaluate its efficacy.
– **When threshold is tuned considering financial implications, precision drops**. Compared to the precision we were able to achieve previously, when we did not tune it for high valued transactions, we get less precision (that is more false positives). In order to save from high value fraud, our threshold gave up some false positives.
– **"Transactional features only" solution has better precision than existing model, but smaller financial impact**: After tuning the threshold to weigh high valued transactions, the baseline that generates features only using attributes of the transaction (and no historical information) still has a higher precision than the existing

model. However, it performs worse on high value transaction so the overall financial impact is the worse than BBVA’s existing model.

– **54% reduction in the number of false positives**. Compared to the current BBVA solution, DFS based solution cuts the number of false positives by more than a half. Thus reduction in number of false positives reduces the number of cards that are false blocked - potentially improving customer satisfaction with BBVA cards.

Method	False postitives		False Negatives		Total Cost (€)
	Number	Cost (€)	Number	Cost (€)	
Current system	289,124	319,421.93	4741	125,138.24	444,560
Transactional features only	214,705	190,821.51	**4607**	686,626.40	877,447
DFS	**133,254**	**183,502.64**	4729	**71,563.75**	**255,066**

Table 7. Losses incurred due to false positives and false negatives. This table shows the results when the `threshold` is tuned to consider high valued transactions. **Method**:We aggregate the `amount` for each false positive and false negative. False negatives are the frauds that are not detected by the system. We assume the issuer fully reimburses this to the client. For false positives, we assume that 50% of transactions will not happen using the card and apply a factor of 1.75% for interchange fee to calculate losses. These are estimates for the validation dataset which contained approximately 1.852 million transactions.

7 Real-time deployment considerations

So far, we have shown how we can utilize complex features generated by DFS to improve predictive accuracy. Compared to the baseline and the current system, DFS-based features that utilize historical data improve the precision by 52% while maintaining the recall at $\tilde{9}0\%$.

However, if the predictive model is to be useful in real life, one important consideration is: how long does it take to compute these features in real time, so that they are calculated right when the transaction happens? This requires thinking about two important aspects:

– Throughput: This is the number of predictions sought per second, which varies according to the size of the client. It is not unusual for a large bank to request anywhere between 10-100 predictions per second from disparate locations.
– Latency: This is the time between when a prediction is requested and when it is provided. Latency must be low, on the order of milliseconds. Delays cause annoyance for both the merchant and the end customer.

While throughput is a function of how many requests can be executed in parallel as well as the time each request takes (the latency), latency is strictly a function of how much time it takes to do the necessary computation, make a prediction, and communicate the prediction to the end-point (either the terminal or an online or digital payment system). When compared to the previous system in practice, using the complex features computed with DFS adds the additional cost of computing features from historical data, on top of the existing costs of creating transactional features and executing the model. Features that capture the aggregate statistics can be computed in two different ways:

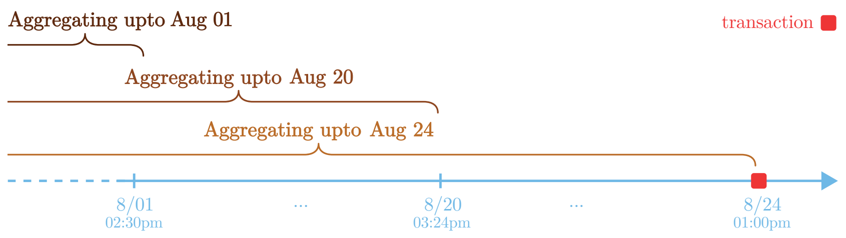


Fig. 2. The process of approximating of feature values. For a transaction that happens at 1 PM on August 24, we can extract features by aggregating from transactions up to that time point, or by aggregating up to midnight of August 20, or midnight of August 1, and so on. Not shown here, these approximations implicitly impact how frequently the features need to be computed. In the first case, one has to compute the features in real time, but as we move from left to right, we go from computing on daily basis to once a month.

- **Use aggregates up to the point of transaction**: This requires having infrastructure in place to query and compute the features in near-real time, and would necessitate streaming computation.
- **Use aggregates computed a few time steps earlier**: We call these approximate features – that is, they are the features that were current a few time steps t ago. Thus, for a transaction happening at 1PM, August 24, we could use features generated on August 1 (24 days old). This enables feature computation in a batch mode: we can compute features once every month, and store them in a database for every card. When making a prediction for a card, we query for the features for the corresponding card. Thus, the real-time latency is only affected by the query time.

It is possible that using old aggregates could lead to a loss of accuracy. To see whether this would affect the quality of the predictions, we can simulate this type of feature extraction during the training process. `Featuretools` includes an option called *approximate*, which allows us to specify the intervals at which features should be extracted before they are fed into the model. We can choose `approximate = "1 day"`, specifying that Featuretools should only aggregate features based on historical transactions on a daily basis, rather than all the way up to the time of transaction. We can change this to `approximate = "21 days"` or `approximate = "35 days"` Figure 2 illustrates the process of feature approximation. To test how different metrics of accuracy are effected – in this case, the precision and the f1-score – we tested for 4 different settings: $\{$1 day, 7 days, 21 days, and 35 days$\}$.

Using this functionality greatly affects feature computation time during the model training process. By specifying a higher number of days, we can dramatically reduce the computation time needed for feature extraction. This enables data scientists to test their features quickly, to see whether they are predictive of the outcome.

Table 8 presents the results of the approximation when `threshold` has been tuned to simply achieve > 0.89 `tpr`. In this case, there is a loss of 0.05 in precision when we calculate features every 35 days.

In Table 9 presents the `precision` and `f1-score` for different levels of approximation, when `threshold` is tuned taking the financial value of the transaction into account. Surprisingly, we note that even when we compute features once every 35 days, we do not loose any `precision`. However, we loose approximately $67K$ euros in money.

Implications: This result has powerful implications for our ability to deploy a highly precise predictive model generated using a rich set of features. It implies that the bank can compute the features for all cards once every 35 days, and still be able to achieve better accuracy then the baseline method that uses only transactional features. Arguably, a 0.05 increase in `precision` as per Table 8 and €67K benefit as per Table 9 is worthwhile in some cases, but this should be considered alongside the costs it would incur to extract features on a daily basis. (It is also important to note that this alternative still only requires feature extraction on a daily basis, which is much less costly than real time.)

	DFS with feature approximation			
Metric	1	7	21	35
Precision	0.41	0.374	0.359	0.36
F1-score	0.56	0.524	0.511	0.512
Total-cost	678,282.77	735,229.05	716,157.54	675,854.12

Table 8. `Precision` and `f-score` achieved in detecting non-fraudulent transactions at the fixed `recall` (a.k.a true positive rate) of $>= 0.89$, when feature approximation is applied and `threshold` is tuned only to achieve a $tpr >= 0.89$. A loss of 0.05 in precision is observed. No significant loss in financial value is noticed.

	DFS with feature approximation			
Metric	1	7	21	35
Precision	0.22	0.223	0.23	0.236
F1-score	0.35	0.356	0.366	0.373
Total-cost	255,066	305,282.26	314,590.34	322,250.67

Table 9. `Precision` and `f-score` achieved in detecting non-fraudulent transactions at the fixed `recall` (a.k.a true positive rate) of $>= 0.89$, when feature approximation is applied and `threshold` is tuned to weigh high valued transactions more. No significant loss in `precision` is found, but an additional cost of approximately 67K euros is incurred.

Bibliography

[1] Bhattacharyya, S., Jha, S., Tharakunnel, K., Westland, J.C.: Data mining for credit card fraud: A comparative study. Decision Support Systems **50**(3), 602–613 (2011)

[2] Brause, R., Langsdorf, T., Hepp, M.: Neural data mining for credit card fraud detection. In: Tools with Artificial Intelligence, 1999. Proceedings. 11th IEEE International Conference on. pp. 103–106. IEEE (1999)

[3] Carcillo, F., Dal Pozzolo, A., Le Borgne, Y.A., Caelen, O., Mazzer, Y., Bontempi, G.: Scarff: a scalable framework for streaming credit card fraud detection with spark. Information fusion (2017)

[4] Chan, P.K., Fan, W., Prodromidis, A.L., Stolfo, S.J.: Distributed data mining in credit card fraud detection. IEEE Intelligent Systems and Their Applications **14**(6), 67–74 (1999)

[5] Domingos, P.: A few useful things to know about machine learning. Communications of the ACM **55**(10), 78–87 (2012)

[6] Feature Labs, I.: Featuretools: automated feature engineering (2017)

[7] Ghosh, S., Reilly, D.L.: Credit card fraud detection with a neural-network. In: System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on. vol. 3, pp. 621–630. IEEE (1994)

[8] Kanter, J.M., Veeramachaneni, K.: Deep feature synthesis: Towards automating data science endeavors. In: Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on. pp. 1–10. IEEE (2015)

[9] Panigrahi, S., Kundu, A., Sural, S., Majumdar, A.K.: Credit card fraud detection: A fusion approach using dempster–shafer theory and bayesian learning. Information Fusion **10**(4), 354–363 (2009)

[10] Pascual, Al, M.K., Van Dyke, A.: Overcoming false positives: Saving the sale and the customer relationship. In: Javelin strategy and research reports. (2015)

[11] Shen, A., Tong, R., Deng, Y.: Application of classification models on credit card fraud detection. In: Service Systems and Service Management, 2007 International Conference on. pp. 1–4. IEEE (2007)

[12] Stolfo, S., Fan, D.W., Lee, W., Prodromidis, A., Chan, P.: Credit card fraud detection using meta-learning: Issues and initial results. In: AAAI-97 Workshop on Fraud Detection and Risk Management (1997)

[13] Whitrow, C., Hand, D.J., Juszczak, P., Weston, D., Adams, N.M.: Transaction aggregation as a strategy for credit card fraud detection. Data Mining and Knowledge Discovery **18**(1), 30–55 (2009)