Machine learning approach for a self-scanning divergence detection

Joao Guilherme Routar de Sousa
Faculdade de Engenharia de Universidade do Porto

October 17, 2018

Abstract

Automation plays an important role in almost any industry. Automating tasks generally saves time and reduces or even eliminates the need for specialized staff, thus reducing overall costs. Self-scanning, a technology recently adopted by SONAE (big retail Portuguese company) is an example of task automation in retail, more specifically supermarkets. Although such technology enables a faster and more efficient shopping, it is susceptible to faulty transactions, known as divergences. There might be a damaged bar code or the client may scan extra products or miss another. In this paper, an approach is proposed to detect whether a transaction may be divergent. A set of time and product-based features were extracted and selected from the original pre processed data. Using Gradient Boosted Trees, the baseline F1-score (5.14%) obtained by Sonae quintuples (25.61%) using the extracted features.

1 Introduction

Process automation is a core asset for any industry or company that aims to grow. Physically demanding and repetitive tasks are often time consuming, consequently increasing the costs associated with such services. Automating processes tends to increase service quality and delivery time and reduce costs, hence adopting a technology-enabled automation of complex processes usually provides with a competitive advantage over the competition.

Retail is an example of an industry where process automation may be a reliable option. McDonald's has introduced touch screen ordering and payment systems, reducing the need for as many cashier employees. Many supermarkets have also introduced a self checkout mechanism, where customers themselves can scan the products they intend to buy.

SONAE, arguably the biggest Portuguese retail company, has introduced a self-scanning mechanism in 2014. Although self checkout machines offer reduced labour costs, the duration of the products processing tends to increase (since it depends on clients), propagating such delay to the following transactions. Self-scanning consists of a small portable machine able to scan the products as they are collected. The processing still depends on each client, but the time they take to scan the products doesn't propagate to other clients', avoiding extra waiting time.

2 Problem Statement

Even though self-scanning efficiently automates the products processing, it still poses a few risks, especially to the providing retail company. The following list enumerates three different deficient occurrences, named divergences, that may occur using the self-scanning mechanism.

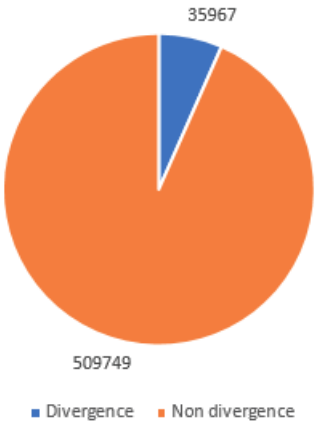

Figure 1: Distribution of divergence class

- Machine fails to scan a certain product (e.g. damaged bar code).
- Customer misses a product (e.g. scans a single bottle of milk instead of the whole pack).
- Customer adds extra products (e.g. scans same product multiple times).

In order to avoid divergences, a percentage of the clients who use the self-scanning system are audited before they checkout. Auditing transactions means the whole process of scanning the products is repeated by a cashier employee. This defeats the purpose of implementing such a system, since customers will still have to wait and there is still the need for employees, reducing neither time consumed or costs involved per transaction.

Being the ultimate goal of self-scanning fully automating the shopping process, it is crucial that auditions are reduced to the lowest possible amount. On this document I propose a machine learning approach to foresee whether a certain transaction is prone to be divergent and, consequently, audited.

3 Data

The original data provided consisted of a record of approximately forty million transacted products with forty-three attributes (40,000,000 x 43) related to the whole transaction the product was involved in, including the classification label, *total_divergence_fl*.

3.1 Data Preprocessing

With a supervised classification approach in mind, the first step was to filter out the instances that were not classified (or audited). The instances where the attribute *rescan_fl* wasn't set to *true* were removed. Such filtering reduced significantly (by 70%) the amount of useful data, going from forty (40) million to twelve (12) million of labeled instances.

A subset of features was also dropped because they weren't relevant to the predictive process (e.g. *currency_key*, *system_key*) and would eventually result in overfitted data. The final data set featured twelve million instances and twenty seven attributes (12,000,000 x 27). Table 1 shows some of the attributes used for prediction and their description.

Lastly, and before performing any feature extraction, the granularity of the data was changed. Each transacted product was aggregated into a single transaction so we could extract transactional patterns. The classes distribution is shown in figure 1.

Attribute	Description
time_key	Date of transaction
transaction_id	Transaction ID
card_key	Client ID
ss_register_time_key	Date of client first register on self-scan
shop_trip_start_hour_key	Start time of shopping trip
shop_trip_end_hour_key	End time of shopping trip
transactional_total_amt_eur	Total amount of the transaction in euros
transactional_total_qt	Total quantity of products in the transaction
total_divergence_fl	Divergence flag
ean_key	Product ID
product_added_times_qt	Number of times product was scanned
product_removed_qt	Number of times product was unscanned
unknown_product_fl	Unidentified product flag
product_total_amt_eur	Total product(s) cost
product_total_qt	Product total quantity
total_divergence_sign	Type of divergence ('=', '-', '+')

Table 1: Main data set original attributes

Yearly	Monthly	Daily
year	**month**	**day**
year day	week of month	**week day**
week of year	first month half	is weekend
is quarter start/end	second month half	shopping morning
is year start/end	is month start	shopping evening
is leap year	is month end	shopping night
	month days	

Table 2: It_1 time attributes. Attributes in bold were computed using *fast.ai* Python library.

4 Feature Extraction

The process of extracting features went through 4 different iterations. On iteration 0, It_0, a small set of original attributes was extracted from the data in order to define a baseline methodology and compare results within the same approach. The attributes used were *transactional_total_amt_eur*, *transactional_total_qt* and *unknown_product_fl*, plus the label *total_divergence_fl*.

On iteration 1, It_1, a bigger vector of features was added to the data. Logically, time plays an important role in shopping and retail in general. As an example, sales generally go up during holidays (e.g. Christmas) and people are more prone to shop in the beginning of the month, after receiving their paycheck, rather than in the end or middle. Thus, It_1 focus on time-based features. The *total_shopping_time*, *elapsed_register_time* and *mean_time_per_product* were computed among other metrics. The library *fastai* was used to split dates into relevant metrics. Table 2 summarizes time metrics extracted from the original data.

On Iteration 2, It_2, the focus was the products in each transaction. Regular features such as *average_price_per_product* and *number_of_different_products* were extracted. Table 3 summarizes the product metrics extracted.

On Iteration 3, It_3, a single attribute, also product-based, based on each product's divergence, was computed. The product divergence score, *prod_div_score*, shows how *divergent* a product can be, e.g. there should be more divergences when buying yogurts, since they come

Feature	Description
avg_price_product	Average price per product
product_added_times_qt	Number of times a product was scanned
product_removed_qt	Number of times a product was unscanned
number_different_products	Number of different products
diff_total_added_products	Diff. between total scanned products and total products
diff_total_removed_products	Diff. between total unscanned products and total products
ratio_total_added_products	Ratio between total scanned products and total products
ratio_total_removed_products	Ratio between total unscanned products and total products

Table 3: It_2 product attributes.

Feature
religious_holiday
optional_holiday
national_holiday
is_spring
is_summer
is_fall
is_winter

Table 4: It_4 attributes.

in packs, rather than shampoos, which are usually bought as a single unit. The score could be computed as the ratio between a product's total divergences and it's total frequency if all the products had the same frequency. Since that's not the case, each ratio must be multiplied by a predefined weight as shown in equation 1.

$$prod\_div\_score = weight * \frac{product\_total\_divergences}{product\_total\_frequency} \tag{1}$$

The frequency for each product is contained in the $[0, max\_product\_frequency]$ interval. The higher the frequency of a product, the higher the relevance of the ratio, which translates to a consequent bigger weight. The weight is computed as the *base weight* (predefined variable) added to the *step* (incremental predefined variable) multiplied by the index of the sub interval the product's frequency belongs to (e.g. $[0, 500], [501, 1000], ..., [109501, 110000]$). Equation 2 shows how the weight can be computed for each product and the default values considered.

$$weight = base\_weight + step * interval\_index, \tag{2}$$

$$\begin{gathered}\text{base\_weight} = 2,\\ \text{step} = 0.015,\\ \text{interval\_index} = 500\end{gathered}$$

Finally, Iteration 4, It_4, features 2 new attributes: holidays and yearly season. Using *holidays*, a Python real time holidays package, we considered whether the shopping was done in a holiday (e.g. Christmas, 25th April). We also considered the season (e.g. Winter). Table 4 summarizes the extra features added in the last feature extraction iteration.

By the end of each iteration, values were transformed using *Python sklearn*'s preprocessing technique, *MinMaxScaler*. Each feature was scaled to a $[0, 1]$ range, using equation 3.

$$scaled\_value = std * (max - min) + min \tag{3}$$

	True 0	True 1	Precision
Pred. 0	297292	683	**99.77%**
Pred. 1	29054	805	**2.7%**
Recall	**92.51%**	**54.1%**	

AUC : 0.51
Accuracy : 90.8%

Table 5: SONAE baseline results for **Logistic Regression** (0: non divergence, 1: divergence).

	True 0	True 1	Class precision
Pred. 0	84197	5266	**94.12%**
Pred. 1	17794	1896	**9.63%**
Class recall	**82.55%**	**26.51%**	

AUC : 0.590
Accuracy : 78.88%
Error : 21.12%

Table 6: Baseline (It_0) results for **Gradient Boosted Tree** (0: non divergence, 1: divergence).

5 Results

The training of the models was made using state of the art algorithms such as *Gradient Boosted Tree* and *Deep Learning*. The latter is based on a multi-layer feed-forward artificial neural network that is trained with stochastic gradient descent using back-propagation. The activation function used was *Rectifier* (*ReLu*).

For a better understanding of the results the main evaluation technique used was the *confusion matrix* itself rather than *accuracy*, since the main goal of the work was to detect possibly divergent transactions. The confusion matrix gives a deeper look at what the algorithms output, by providing the actual values and ratios of the actual values versus the predicted ones. The *divergence class recall* displays the percentage of divergences detected out of the total divergences, while the *divergence class precision* displays the percentage of divergences correctly predicted out of the total of divergences predicted.

Since the approach was a binary classification, other evaluation metrics were also considered such as *AUC* and *ROC*, for a more general visualization of the results produced.

5.1 SONAE (baseline) results

Before designing an approach, a baseline had already been established by SONAE using a different methodology that's out of scope for this document. The algorithms used were *Random Forest*, *k nearest neighbors* and *Logistic Regression*, being the latter the one that produced the best results, displayed in table 5.

5.2 It_0 (baseline) results

To establish a baseline we applied a set of different algorithms to better understand which would produce the best results. The best were chosen and applied on later iterations. Table 6 displays the results for *Gradient Boosted Tree*, table 7 for *Naive Bayes* and table 8 for *Deep Learning*.

Even though *Naive Bayes* produced the highest accuracy out of the three algorithms, by looking at it's confusion matrix (table 7), it is clear that it failed miserably to identify real divergent transactions. For this reason, we discarded this approach for the following iterations and used only *Gradient Boosted Tree* and *Deep Learning*. Only the results for the best algorithm are displayed.

	True 0	True 1	Class precision
Pred. 0	100101	6888	**93.56%**
Pred. 1	1890	264	**12.26%**
Class recall	**98.15%**	**3.69%**	

AUC : 0.581
Accuracy : 91.96%
Error : 8.04%

Table 7: Baseline (It_0) results for **Naive Bayes** (0: non divergence, 1: divergence).

	True 0	True 1	Class precision
Pred. 0	43210	2253	**95.04%**
Pred. 1	58781	4899	**7.69%**
Class recall	**42.37%**	**68.50%**	

AUC : 0.574
Accuracy : 44.08%
Error : 55.92%

Table 8: Baseline (It_0) results for **Deep Learning** (0: non divergence, 1: divergence).

5.3 It_1 results

The addition of time-based features produced the results seen in table 9.

5.4 $It_2 + It_3$ results

Since It_3 only added 1 new feature and both It_2 and It_3 are product-based iterations, the results for both were merged. They are displayed in table 10.

5.5 It_4 results

The results for the last and final iteration are displayed in table 11.

5.6 Feature importance

In the end of the feature extraction process, the weight of each predictive attribute on the final model was measured. Using only the most important attributes, i.e the ones with $weight > 5$ as seen in figure 2, a new model was generated. It's results are displayed in table 12.

6 Conclusion

The summary of the results are displayed in table 13. The best results for each evaluation method are highlighted in bold for the different iterations and baselines.

Although SONAE's model achieved a good overall accuracy and the best *divergence* class recall out of all the models, i.e. correctly identifies more than half of divergent transactions (54%) out of the total of divergences, the same class precision is really low, sitting at 2.7%. This means that a big number of non divergent transactions are identified as potentially divergent, forcing the need for auditions on several non divergent transactions. Figure 3 shows the ROC curve for SONAE's model.

	True 0	True 1	Class precision
Pred. 0	78899	4568	**9**4.53%
Pred. 1	23092	2584	**10.06%**
Class recall	**77.36%**	**36.13%**	

AUC : 0.618
Accuracy : 74.66%
Error : 25.34%

Table 9: It_1 results for **Gradient Boosted Tree** (0: non divergence, 1: divergence).

	True 0	True 1	Class precision
Pred. 0	97383	5425	**94.53%**
Pred. 1	4608	1727	**27.26%**
Class recall	**95.48%**	**24.15%**	

AUC : 0.700
Accuracy : 90.81%
Error : 9.19%

Table 10: $It_2 + It_3$ results for **Gradient Boosted Tree** (0: non divergence, 1: divergence).

	True 0	True 1	Class precision
Pred. 0	97351	5430	**94.72%**
Pred. 1	4640	1722	**27.07%**
Class recall	**95.45%**	**24.08%**	

AUC : 0.701
Accuracy : 90.77%
Error : 9.23%

Table 11: It_4 results for **Gradient Boosted Tree** (0: non divergence, 1: divergence).

	True 0	True 1	Class precision
Pred. 0	96973	5407	**94.72%**
Pred. 1	5018	1745	**25.80%**
Class recall	**95.08%**	**24.40%**	

AUC : 0.700
Accuracy : 90.45%
Error : 9.55%

Table 12: Results after feature selection for **Gradient Boosted Tree** (0: non divergence, 1: divergence).

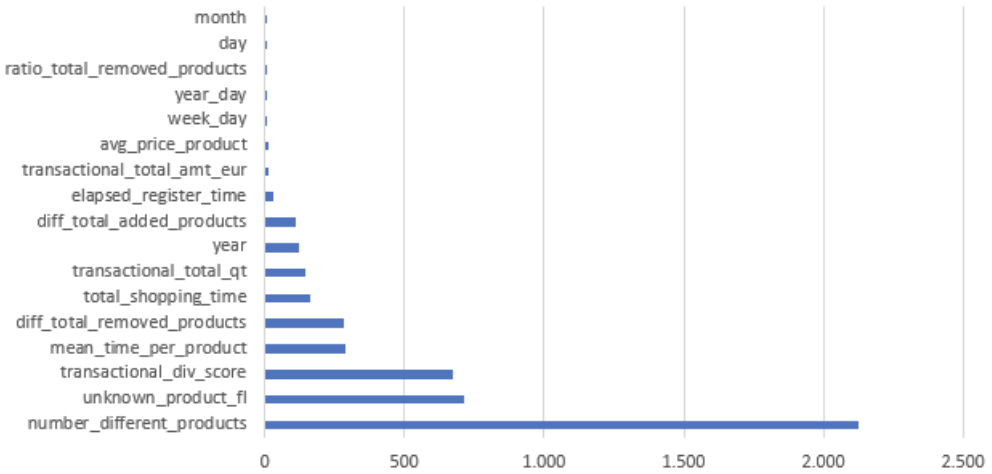

Figure 2: Attributes importance generated bar graph.

	AUC	Div. Precision	Div. Recall	Auditions
Sonae	0.51	2.7%	**54.1%**	29859
It0 (Baseline)	0.590	9.63%	26.51%	19690
It1	0.618	10.06%	36.13%	25676
It2 + It3	0.7	**27.26%**	24.15%	**6335**
It4	**0.701**	27.07%	24.08%	6362
Feat. Select.	0.7	25.80%	24.40%	6763

Table 13: It_4 results for **Gradient Boosted Tree** (0: non divergence, 1: divergence).

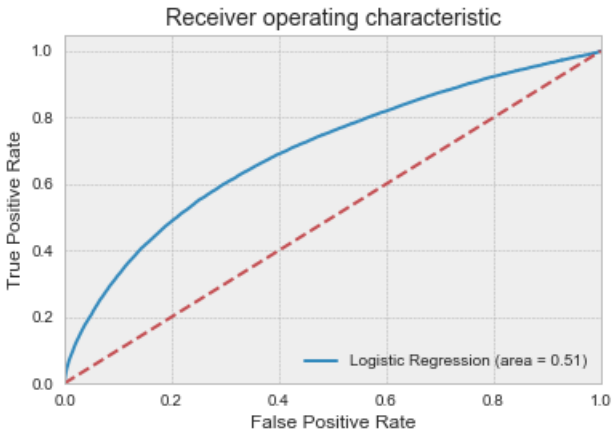

Figure 3: SONAE's ROC curve

Introducing time-based features (it_1) increased 10% the *divergence* class recall (number of identified divergent transactions), while the number of required auditions also slightly decreased. The addition of product-based features reduced the number of divergent transactions identified (10% less), but the amount of required auditions greatly decreased.

Since the goal was to balance both the *divergence* class recall and precision (detected divergences and number of auditions), maintaining a high precision and recall for the *non divergence* class, It_{2+3} was considered to be the best result according to it's F1 score (table 14) and ROC curve (figure 4).

	F1 score
Sonae	5.14%
Baseline	14.13%
It_{2+3}	25.61%

Table 14: F1-scores

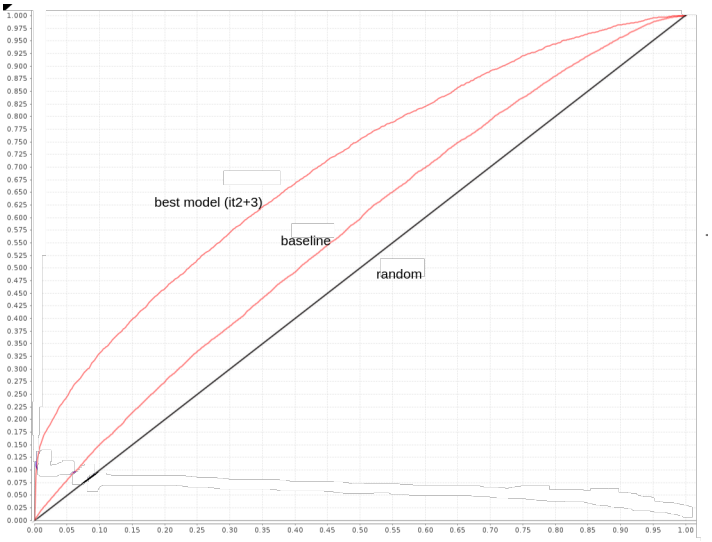

Figure 4: Comparison of ROC curves for the baseline and final model

7 Improvements

Due to the short development period, it was not possible to deepen the methodologies used and extract more and better features. The following list summarizes possible improvements that could be done in order to increase the performance of the models:

Focus on more product-based features: Even though recall generally lowered with the addition of these type of features, precision increased significantly. As mentioned earlier, a transaction involving a lot of packed products (e.g. milk) is more prone to be divergent than

one that doesn't. For this reason, identifying product's categories and subcategories would probably improve the results.

Add client-based features: A transaction is always performed by a client. Having information regarding the person that's shopping (e.g. age, social status, average income) might help filtering divergent transactions.

***prod_div_score* parameter tuning**: The product divergence score improved the detection of divergent transactions by 1%, which is quite significant for a single feature. The score was computed only once using the default parameters mentioned above. Tuning these parameters could possibly improve the results.

Try different deep learning architectures: Even though *Gradient Boosted Trees* outperformed every other algorithm used, deep learning generated quite similar results. Considering only a single neural network architecture was used, tuning hyper parameters and trying different architectures could possibly improve the performance of deep learning.