Ensemble-Based Prediction of Business Processes Bottlenecks With Recurrent Concept Drifts

Yorick Spenrath
Department of Mathematics and Computer Science
Eindhoven University of Technology
y.spenrath@student.tue.nl

Marwan Hassani
Department of Mathematics and Computer Science
Eindhoven University of Technology
m.hasssani@tue.nl

ABSTRACT

Bottleneck prediction is an important sub-task of process mining that aims at optimizing the discovered process models by avoiding such congestions. This paper discusses an ongoing work on incorporating recurrent concept drift in bottleneck prediction when applied to a real-world scenario. In the field of process mining, we develop a method of predicting whether and which bottlenecks will likely appear based on data known before a case starts. We next introduce GRAEC, a carefully-designed weighting mechanism to deal with concept drifts. The weighting decays over time and is extendable to adapt to seasonality in data. The methods are then applied to a simulation, and an invoicing process in the field of installation services in real-world settings. The results show an improvement to prediction accuracy compared to retraining a model on the most recent data.

KEYWORDS

Complex event processing, data streams, recurrent concepts, data mining, knowledge discovery, adaptation methods, process mining

1 INTRODUCTION

Concept drift expresses an occurrence of a shift in relations between input and output data over time. The challenge with concept drift is that it is difficult to asses, detect, and correct such evolutions. One method of solving concept drift is to retrain models periodically. This method poses two major disadvantages. First, this reduces the quantity of available data, which is a big issue when the number of data points per time unit is low, or when dealing with unbalanced datasets. The combination of these two causes over-fitting of the model on larger classes. Second, retraining a model with new data inherently discards former data, which could remove valuable information on seasonality.

This paper discusses a technique to cover both issues and applies it to reduce the duration of a process, by predicting if, and where, bottlenecks take place in the process. A simple example for a paper submission process is shown in Figure 1, where the bottlenecks are indicated for process instances at different points in time. Note the bottle activities for papers on Process Mining. The proposed solution uses multiple machine learning models trained over different intervals of previous data, and assigns weights to the predictions of each model based on the difference in time between the model and the test data, factoring in an exponential decay (giving priority to recent models) and a periodic function (giving priority to seasonally similar points in time). As such we develop a Gradual and Recurrent concept drift Adapting Ensemble Classifer, or GRAEC. The contributions of this paper are:

- A mathematical way for assessing bottlenecks in business processes
- A novel concept drift adaptation method, GRAEC
- An extensive experimental evaluation using a simulation study and a real-world dataset, which proves the superiority of our method compared to existing solutions which also proves the correctness of our concept

The rest of the paper is organised as follows. Section 2 discusses some related literature. Section 3 defines the problem that this paper aims to solve. Section 4 proposes a solution to adapting for concept drift with seasonality. Section 5 describes the experimental set-up for the simulation and the real-world dataset. Section 6 presents and discusses the results of the experiment. Section 7 concludes the paper and provides suggestions for future research.

2 RELATED WORK

Concept drift has been a topic in online data streams in recent years. Concept drift is the phenomenon where relations between input (features) and output (labels in the case of the experiments in this paper) change over time, because the underlying models and/or distributions change [5]. Much of existing literature is on detecting concept drift. Techniques for detection include sudden change in classification accuracy (such as in [1]), or changes in feature importance (such as in [17] and [3]). Detecting concept drift is a first step, adapting to it is the next. One way to do so is by simply retraining the model whenever a drift in concept is detected. A disadvantage of this technique occurs if the drift is periodic, i.e. a drift of concept is detected, but the previous concept may still be of use at a later point in time, just not right after the detected drift. [20] trains a model for every given number of data points, and then creates an ensemble classifier, weighting each model according to the accuracy assessed on the most recent data points.

The challenges of seasonal concepts, or what is more known in the literature as *recurrent concepts*, have been addressed recently in [4, 9, 13, 15, 16]. The general idea of approaches handling recurring concepts is not to lose knowledge gathered over time. Therefore, they maintain a pool of classifiers and use one or an ensemble of them each time. Our approach is bringing the benefits of ensemble-based recurrent concept drifts to the domain of business process intelligence for predicting bottlenecks.

First International Workshop on Data Science for Industry 4.0.

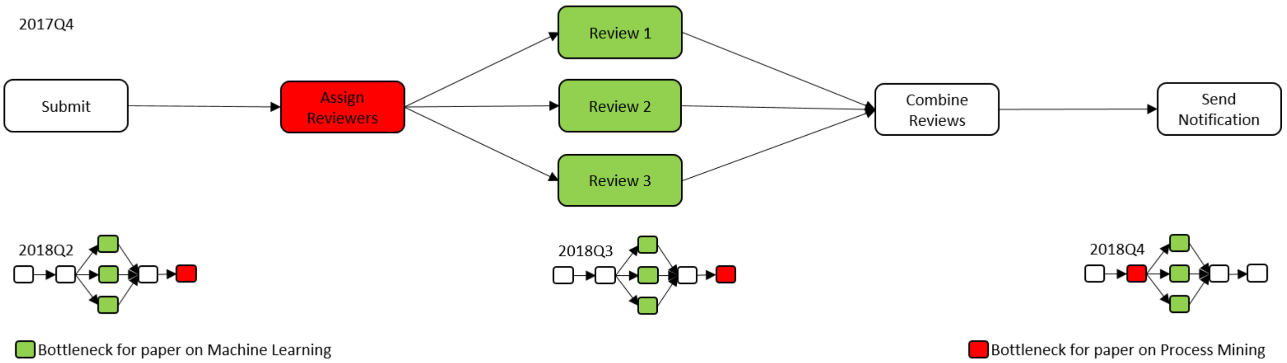

Figure 1: Example process with indicated bottlenecks at different quarters. The process is a simplified version of assigning reviewers and sending notifications of papers in a quarterly journal. We differentiated between two types of bottlenecks according to the paper topic. The bottlenecks are evolving over quarters although the process does not. Note that in the two quartile leading up to 2018Q4 the bottleneck for Process Mining papers is different from the bottleneck in 2018Q4 and 2017Q4.

In the field of prediction process bottlenecks, [18] predicts the duration of queue time, which can be regarded as one activity. In a study on handling concept drift in business processes, [12] investigates different solutions to adapt to concept drift when predicting process outcomes. The authors use information about the process execution (Control-flow perspective) as well as involved data (Data-perspective). They show the effectiveness of incremental machine learning algorithms such as Adaptive Hoeffding Option Trees [2] in handling different types of concept drift, amongst which recurrent concept drift. Different to this approach, our novel method introduced in this paper implements an ensemble classifier to adapt to recurrent concepts.

3 PROBLEM DESCRIPTION

The problem discussed in this paper is to adapt to concept drift when applied in the domain of business processes analysis. The focus of the paper is on the effectiveness of the proposed concept drift adaption method, finding the bottlenecks is the use case.

We aim to improve the throughput of the process by indicating bottleneck activities based on information we have before the case starts. In our toy example of paper submissions, such information could be the author(s), the research area, the number of publications each author has, and the number of pages in the paper, all of which are known once a submission is made. Using this information we then predict which of the activity will be the bottleneck activity. This bottleneck is then considered as a label for classifiers, which are trained using the information of the case as input.

We assume that the process is stable, i.e. our research is limited to processes whose structure does not change over time. The time evolution we are looking at is that of the relations between case information and bottleneck labels, i.e. the drift in the underlying statistical models.

4 PROPOSED SOLUTION

The problem this paper addresses is predicting bottleneck activities in business processes, and improve these predictions by accounting for concept drifts. The goal is not to indicate or explicitly detect concept drift, but to optimise the use of selected previously trained models to improve prediction quality. This section describes the proposed solution, first introducing the context of business processes (Subsection 4.1), next assessing what activity is a bottleneck activity (Subsection 4.2), and finally improving predictions using `GRAEC` (Subsection 4.3). We discuss them in general and also in applying them to a simulation and a case study in Section 5.

4.1 Business Processes

Process mining is field of research that bridges the gap between the computational possibilities of data mining with the industrial application of business processes, by learning process behaviour from event logs. Event logs consist of events, each describing what happened (*Activity*), at what time (*Time*), by whom (*Resource*), for which process instance (*Case ID*). Time can alternatively be represented by considering start and end times of events. The analysis in this paper only considers end times (the time when the activity is logged) but can easily be extended to include both. Resources are not considered in the process analysis part of the solution. To explain the algorithm used to define the bottlenecks, some notation is required.

An event log L is a set of *events* e, a subset of all possible events E. An event e is a combination of the Case ID, Activity, and Time, represented as the tuple $(e.cid, e.act, e.time)$. A *case* c is a sequence of events, $c \in E^*$. The first element of a sequence c is represented as $c(0)$. We further define the function $act(c) : E^* \longrightarrow A^*$, mapping each element of a case to the activity of the event, A being the set of all activities. $act(c)$ is said to be the *trace* of c, and two cases $c1$ and $c2$ are said to be of the same variant if $act(c1) = act(c2)$. The function is extended to a set C of cases; if all $c \in C$ have the same variant, $act(C)$ is the sequence of activities for which $act(c) = act(C)$ for all $c \in C$. The duration of a case c is defined as $duration(c) = c(|c| - 1).time - c(0).time$, i.e. the time difference between the first and the last event. This implies that the duration of the first event is not considered in the duration. If only the end times of events are logged, there is no way to asses the duration of the first event. In the example sketched above this is no issue: the submission merely marks the start of the case, no work is required by the committee. The same applies to the real-world scenario presented in Section 5. In other processes some notion of the start of the process (like an automated trigger caused by a client) needs to be identified to analyse the throughput of the first activity.

In the example of Figure 1, each case is one submission. The Case ID could be a DOI, the activity set A would be $\{$*Submit*, *Assign reviewers*, *Review1*, *Review2*, *Review3*, *Combine reviews*, *Send notification*$\}$. Events would relate these two together, providing information on when an activity for the given paper was completed. The process consists of only one variant, i.e. there is only one way to follow the process.

4.2 Bottleneck Assessment

Let C be a set of cases. We want to find bottlenecks in cases that have a long duration. We partition the set of cases into H_i and L_i, each representing cases that take up to a given time d (High-throughput) and cases that take more than d (Low-throughput) respectively, for variant i for $i \in 1 \ldots N$, N being the total number of variants in C. For activities to be considered bottlenecks, we have two requirements: Relevance and Resolvability.

If an activity has a long duration compared to other cases, but a short duration compared to other activities in the same case, it will be less effective to resolve a long duration of that activity than activities that take up a larger part of the case duration. We formalise this Relevance requirement by requiring that the fraction of the total case duration of the activity is at least α, where $\alpha \in [0, 1]$ is a parameter chosen on domain expert's recommendation and process characteristics (like the number of activities). Note that setting α to 0 drops the relevance requirement entirely. Further note that if α is set too high, no activities will meet this requirement. It will hence be important to find an appropriate balance or to have a fall-through mechanism to label cases.

An activity is resolvable if it has a longer duration than a benchmark average for that activity. We only address bottlenecks in L_i, as such we use the average duration of the activity in H_i as benchmark. This is the Resolvability requirement. We know that there must be at least one activity that satisfies this requirement, since the sum of all durations (i.e. the duration of the case) of any case in L_i is larger than the duration of any case in H_i, so at least one activity must take longer than the average of that activity in H_i.

Finally, of all activities that pass both requirements, we choose the activity that deviates the most (in terms of the standard deviation in H_i from the benchmark average). This way ensures that the chosen bottleneck is least explained by the intrinsic variability of real-world processes. This procedure is described in Algorithm 1.

Line 1 loops over each Variant of C. First we derive the Trace of the Variant in Line 2. Note that this is the same as $act(L_i)$. In Lines 3 and 4 we label each case in H_i as short. Lines 7 and 8 calculate the sample mean and standard deviation[1]. We next consider each case with a long duration in Lines 10 - 19. First we take all indices of the Trace in Line 11. Next we apply the Resolvability requirement in Line 12, and the Relevance requirement in Line 13. We then check if there is any activity that meets both requirements in Line 14. If there is, we apply as the label the activity whose duration deviates the most standard deviations from the mean throughput (assessed in Lines 8 and 7 respectively) in Line 15. If there are no activities that satisfies both requirements, we take the activity with the longest duration that satisfies the Resolvability requirement, in Line 17.

Algorithm 1: Selection of the bottleneck

```
   Result: Each case $c$ in $C$ gets a label $label(c)$
1  for $i = 1 \ldots N$ do
2      $\sigma \leftarrow act(H_i)$
3      for $c \in H_i$ do
4          $label(c) \leftarrow$ short
5      end
6      for $k \in 1 \ldots |\sigma| - 1$ do
7          $\bar{x_k} \leftarrow \frac{1}{|H_i|} \sum_{c \in H_i} c(k).time - c(k-1).time$
8          $s_k \leftarrow \sqrt{\frac{\sum_{c \in H_i} (c(k).time - c(k-1).time - \bar{x}_k)^2}{|H_i|}}$
9      end
10     for $C \in L_i$ do
11         $K_0 \leftarrow 1 \ldots |\sigma| - 1$
12         $K_1 \leftarrow \{k | k \in K_0 : c(k).time - c(k-1).time > \bar{x}_k\}$
13         $K_2 \leftarrow \left\{k | k \in K_1 : \frac{c(k).time - c(k-1).time}{duration(c)} \geq \alpha\right\}$
14         if $|K_2| \neq 0$ then
15             $label(c) \leftarrow$
               $c\left(\arg\max_{k \in K_2} \left[\frac{c(k).time - c(k-1).time - \bar{x}_k}{s_k}\right]\right).act$
16         else
17             $label(c) \leftarrow$
               $c(\arg\max_{k \in K_1} c(k).time - c(k-1).time).act$
18         end
19     end
20 end
```

4.3 Incorporating the Effect of Concept Drift

This subsection proposes a method of dealing with concept drift. The general idea is to split available training data into smaller sets, and give weights to the models created by each set. The method is general, and we will demonstrate its effectiveness in our use case of predicting bottlenecks.

Let D be our dataset, and the points in D be d_i. Each datapoint has input features $d_i.x$, a timestamp $d_i.time$, and a label $d_i.y$. In our use case of business process bottleneck prediction, d_i is a case, $d_i.x$ is information about a case, $d_i.time$ is the timestamp of the first event of a case, and $d_i.y$ is the bottleneck as labelled by Algorithm 1. In the light of Figure 1, $d_i.x$ would be information on the author(s), the research area, the number of publications each author has, and the number of pages in the paper. $d_i.time$ would be the time of the submission, and $d_i.y$ would be the bottleneck, as indicated in Figure 1.

D contains data over a time period of length $T = \max_{d_i \in D} d_i.time - \min_{d_i \in D} d_i.time$. This length is divided into smaller data sets D_j, each with length S, such that $d_i \in D_j \Leftrightarrow \lfloor \frac{d_i.time}{S} \rfloor = j$. In this paper S is regarded as a constant in time, and different values of

[1] We could also have taken the population standard deviation, it does not make a difference since the sample size is the same for each activity.

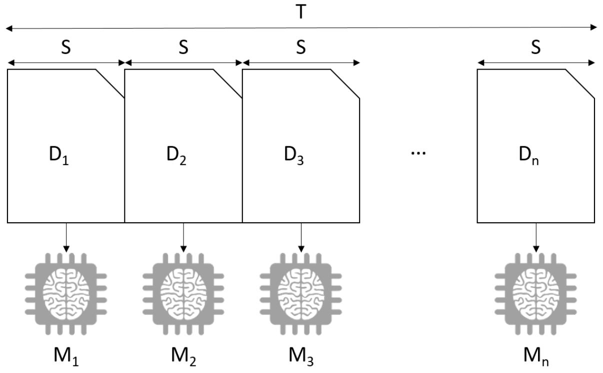

Figure 2: Schematic presentation of how the dataset is split into smaller subsets.

Algorithm 2: Determination of weights

```
Result: For the test prediction $d$, each model $M_k$ gets a weight $w_k$
$j \leftarrow \lfloor \frac{d.time}{S} \rfloor$
for $k = 0 \dots j-1$ do
    $w_k \leftarrow 10^{-(j-k+1)\beta}$
end
$t \leftarrow d.time - p$
while $t > 0$ do
    $k \leftarrow \lfloor \frac{t}{S} \rfloor$
    $w_k \leftarrow w_k + \tau$
    $t \leftarrow t - p$
end
```

S are explored. Picking the right value of S is important, having too short time-frames will mean too few datapoints, having too long time-frames might fail to capture short-term concept drift. The splitting of data is visualised in Figure 2.

We then train a machine learning model on each D_j, resulting in multiple classifiers M_j. This creates an ensemble classifier, and we use a weighting function explained below to decide the importance of each classifier in the ensemble. In the running example, one could decide to train a classifier monthly ($S = 1$ month), creating twelve models per year, each consisting of all papers that were submitted in a specific month.

When predicting a test data point, all models M_j receive a weight based on how old the model is. We propose a combination of two weights. The first weight is related to exponential decay, and is inspired by for example [5]. Let d be a test data point, which has a timestamp $d.time$. As such, d belongs to period $j = \lfloor \frac{d.time}{S} \rfloor$. We assign weight 1 to M_{j-1}, $10^{-\beta}$ to M_{j-2}, $10^{-2\beta}$ to M_{j-3} and so on; with $\beta \in \mathbb{R}_+$. In general we assign $10^{-(j-k+1)\beta}$ to model M_k, for $k < j$ (models M_k, $k \geq j$ are not available for testing d, since they are learned from data that is not available at $d.time$).

The second weight relates to giving extra importance to models that are seasonally similar. Say we want to reduce the effects of recurrent concept drift that occurs every $p \in \mathbb{R}_+$ time. We hence add weight $\tau \in \mathbb{R}_+$ to models that belong to $d.time - p$, $d.time - 2p \dots$. These additions are given on top of the exponential decay weight; though in most situations we will have that $p >> S$ (we train models far more often than the recurrent concept drift occurs), and hence the exponential weight will be insignificant for models around time $d.time - p$.[2] The determination of the weights is also presented in Algorithm 2.

We combine the models as follows: for datapoints in D_j we let each of the j existing trained prediction models ($M_0, M_1, \dots, M_{j-1}$) make a prediction, resulting in j vectors ($p_{1,k}, p_{2,k}, \dots, p_{n,k}$), where $p_{l,k}$ is the probability of label l as predicted by model k (where we have a total of n labels in all of D). We determine the combined prediction for the test datapoint d by taking the weighted sum:

$$p_l = \sum_{0 \leq k \leq j-1} w_k \cdot p_{l,k}$$

where w_k is the weight of the described above. The predicted label is then $\arg\max_l p_l$. The combination of β, τ and p uniquely identifies a weighting assignment, together with S we form a complete concept drift adaption technique. Since this technique adapts to gradual and recurrent drift, we will refer to it as Gradual and Recurrent concept drift Adapting Ensemble Classifier, or GRAEC. Finding the best combination of β, τ, p and S is part of the training process. In this paper we will restrict to a single value for p.

5 SIMULATION AND CASE STUDY SETUP

This section describes the simulated (Subsection 5.1) and real-world data set used (Subsection 5.2), the experimental setup (Subsection 5.3), and additional information about how the models are trained (Subsection 5.4). The goal of the case study is to test the effectiveness of our proposed method. The actual predicted bottlenecks is of interest to the company, but falls outside the scope of this paper.

The code to run the full Simulation experiment including the implementations of Algorithms 1 and 2 is available at https://github.com/yorick-spenrath/GRAEC.

5.1 Simulation Setup

We simulate an event log based on the running example. We use three features: the topic, the number of pages used in the paper, and the number of publications the author has. The topic is used to determine where a bottleneck will be, and the number of pages is used to determine if there will be a bottleneck. As such, the number of publications will not influence the simulation. The topic is one of the following: 'Machine Learning' (ML), 'Data Mining' (DM), 'Process Mining' (PM), and 'Operations Management' (OM). The number of pages and the number of publications are both integers in the interval [1,100], drawn at random. We simulate for 5 years of data, where we receive 200 papers of each topic each month (for a total of 800 cases per month, or 48000 over the simulation). Time is measured in months, and months consist of 30 days each, i.e. the start of all cases in the first month have a time between 0 and 1, and a day takes $\frac{1}{30}$ units of simulation time. The start of the case coincides with the timestamp of the *submit*

[2] In particular, the implementation only assigns exponential weights of at least 10^{-6}.

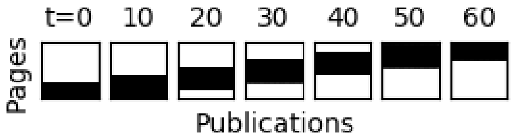

Figure 3: Gradual Concept Drift. The black area indicates for which values there is no bottleneck in the process.

Qt.	ML	DM	PM	OM
1	A	R	C	S
2	S	A	R	C
3	C	S	A	R
4	R	C	S	A

Table 1: Bottlenecks as determined by topic and quartile of the case. A = 'Assign reviewers', R = 'Review', C = 'Combine reviews', S = 'Send notification'

activity, the timestamp the other events is the timestamp of the preceeding event, increased by a random variable X. X is in days, with $X \sim \ Gamma(k = 5, \theta = 0.7)$ for a non-bottleneck activity (see below), and $X \sim \ Gamma(k = 25, \theta = 0.7)$ for a bottleneck activity. This way we will have cases without a bottleneck to take 14 days on average, and over 97% of the cases without bottlenecks will take less than 21 days. Cases with bottlenecks will on average take 28 days, and at least 97% of them take more than 21 days.

We add gradual concept drift by selecting an interval of size 20 in $[1, 100]$; if the number of pages is in this interval, we will not add a bottleneck to the case, if the number of pages is outside the interval we will. The interval changes over time; at time t, the interval starts at $80 \cdot \frac{t}{60}$ and ends at $80 \cdot \frac{t}{60} + 20$. This is visualised in Figure 3. The activity which will be the bottleneck is based on the topic and quartile as indicated in Table 1, resulting in recurrent concept drift. We test our approach for values of $S \in \{1, 2, 3, 4\}$.

5.2 Real-World Dataset

We analyze part of the repair process at a Dutch installation services company. Their repair process is roughly divided into three steps: preparation, repair, and invoicing. We have an event log of about two and a half years, containing about 200 000 cases and about 2 000 000 events, with information on the resource and the time stamp of when an activity ended. We are interested in the invoicing part, which, in the light of Subsection 4.1, is considered as the full process starting at the last repair activity[3]. We define the throughput of the invoice process as the time difference between the last repair activity occurrence and the last invoice activity occurrence. The invoice consists of three activities and one optional activity, but the event log contains cases that still have repair activities after the first invoice activity, and duplicate or missing invoice activities. In the case of repair activities after an invoice activity, the throughput is measured from the last non-invoice activity before the first invoice activity. We further require that the three main invoice activities (repair assessment, data check, and invoice creation) occur in the case, and that the case is completed (there is a final batching activity in the process that happens once a month). Again, each case from the dataset results in one datapoint. Cases have feature values, such as client, responsible resource, type of defect, client location, all of which are categorical.

The time in the dataset is measured in days, we use $S \in \{7, 14, 21, 28\}$. This is because the dataset is affected by working days, so taking any other value than a multiple of 7 will result with biased subsets.

5.3 Experimental Setup

This section describes how we use events logs, to which we refer as L, to assess different methods of predicting bottlenecks. Unless otherwise stated, the explanation holds for the simulation study as well as the real-world dataset.

From L we create four different datasets D^S, one for each value of S. For each S, L is split into smaller subsets L^S_j, each of duration S, as discussed in Section 4.3. This creates subsets D^S_j, where information about cases (topic, pages, and publications for the simulation, client information for the Real-world dataset) are related to labels, for period j. D^S_j contains data about cases that started in $[S \cdot j, S \cdot (j+1))$. Note that cases could have different bottleneck labels for different values of S, as a result of being compared within a different set in Algorithm 1. d and α are set to 21 days and 0.2 respectively. For the simulation dataset, this is set by the simulation design, for the real-world dataset this is per recommendation of the company's domain expert. After labelling each set, the number of datapoints in each subset is reduced using `k-medoids` with `Jaccard` distance, to ensure that the largest class in a set is at most twice as big as the smallest class, to reduce overfitting. This is not required for the simulation dataset, as it is already balanced by design. For each subset D^S_j a model M^S_j is trained using the method described in the next subsection.

We then compare three different approaches. The first approach uses only the first year of data to create a single model, which is then used for all testing. This will be referred to as the Naive method. The second approach uses the model created from the dataset that was most recently completed; if a datapoint of D^S_j is tested, model M^S_{j-1} is used. This approach will be referred to as the Recent method. Finally, we use our approach, which will be referred to as `GRAEC`. Each method is tested on each value of S. Note that this also creates four configurations for the Naive method, since a different S results may result in different labelling.

We next use the full last year of data for evaluation. This part is split in a stratified 80% - 20% train-test split. The train split is used to optimise the parameters of `GRAEC`, using a complete grid search with $\beta \in \{0, 1, 2\}$, and $\tau \in \{0, 0.01, 0.1, 1, 10, 100\}$ for each value of S. We also set p on one year (12 in the simulation, 365.25 in the real-world dataset), as per design and domain expert recommendation. The test split is used to assess all approaches.

5.4 Model Training

The model M^S_j is trained by optimising the hyperparameters of `Logistic Regression`, `kNN`, `Decision Tree`, and `Random Forest` classifiers using a stratified 5-fold cross validation on stratified 80%

[3] This also justifies not taking the first event of the case into account when determining the bottleneck

of a dataset, all using the implementation of the scikit learn library in Python [14]. The remaining 20% is then used to pick the best model out of the four optimised algorithms. Because of this, we require that D_j^S has at least 10 entries in each of its labels and that it has at least 2 different classes. For the real-world dataset, some D_j^S's might fail to meet one or both of these requirements. This means that those D_j^S's will not be used for training, and as such the respective model M_j^S may not exist. GRAEC is not limited by this, but having too few M_j^S's will have a negative impact on the effectiveness of GRAEC. If we cannot make a prediction for a datapoint in D_j^S because of a missing model M_{j-1}^S, we will use the most recent model instead (M_{j-2}^S, or even earlier models).

6 RESULTS AND DISCUSSION

In this section we show the results of applying our approach to the simulation dataset and the real-world dataset described in the previous section. Since the proposed solution aims to adapt to overcome the effects of concept drift and not to detect it, we are only interested in the score, but not in the drift in the process or derived models themselves. In Subsection 6.1 we present and discuss the results of the simulation dataset, in Subsection 6.2 we do the same for the real-world dataset.

6.1 Simulation Study

The results of the simulation study are presented in Figure 4 and Table 2. Figure 4 shows the scores of each of the twelve methods applied. The best scores for GRAEC are underlined in Table 2 to indicate the optimal values of β and τ. Table 2 further shows the F_1 and accuracy scores for all configurations of β, τ and S.

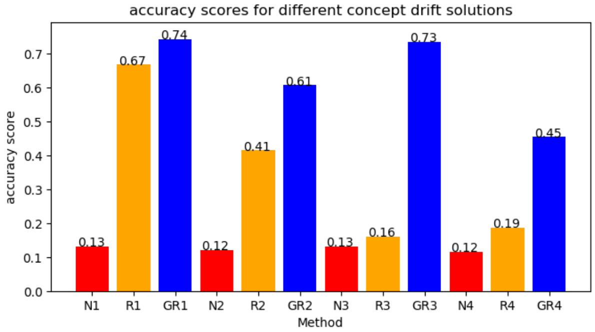

Figure 4: The results of the Simulation in terms of F_1 scores. *GR* indicates the GRAEC method, *R* uses the most Recent model, and *N* creates a model once on the first year of the dataset. Each method is tested for different values of S, indicated by the numbers. The values of β and τ for GRAEC are underlined in Table 2

S	τ	$\beta = 0$		$\beta = 1$		$\beta = 2$	
		ACC	F_1	ACC	F_1	ACC	F_1
1	0	0.140	0.109	0.669	0.672	0.667	0.671
	0.01	0.139	0.108	0.670	0.673	0.669	0.673
	0.1	0.154	0.130	0.671	0.674	0.670	0.674
	1	0.263	0.295	0.742	0.653	$\underline{0.742}$	$\underline{0.652}$
	10	0.649	0.605	0.723	0.642	0.722	0.642
	100	0.709	0.635	0.716	0.639	0.716	0.639
2	0	0.113	0.090	0.344	0.348	0.395	0.396
	0.01	0.114	0.092	0.352	0.355	0.442	0.444
	0.1	0.134	0.118	0.513	0.516	0.528	0.532
	1	0.245	0.279	$\underline{0.607}$	$\underline{0.549}$	0.605	0.545
	10	0.547	0.515	0.582	0.529	0.582	0.530
	100	0.575	0.526	0.577	0.527	0.577	0.527
3	0	0.123	0.090	0.161	0.171	0.161	0.171
	0.01	0.127	0.096	0.161	0.171	0.161	0.171
	0.1	0.142	0.119	0.162	0.172	0.162	0.172
	1	0.381	0.426	0.731	0.649	0.731	0.648
	10	0.724	0.641	0.734	0.646	0.734	0.646
	100	0.731	0.645	$\underline{0.734}$	$\underline{0.646}$	0.734	0.646
4	0	0.096	0.075	0.180	0.185	0.186	0.192
	0.01	0.097	0.077	0.185	0.190	0.188	0.194
	0.1	0.112	0.099	0.203	0.207	0.204	0.209
	1	0.215	0.241	$\underline{0.454}$	$\underline{0.430}$	0.454	0.428
	10	0.450	0.415	0.460	0.420	0.460	0.420
	100	0.461	0.422	0.461	0.422	0.461	0.422

Table 2: Scores for Simulation study. The underlined scores belong to the best accuracy scores for each S, as presented in Figure 4

6.1.1 *The effectiveness of* GRAEC.

The foremost result is the effectiveness of GRAEC. When comparing the scores for each method for a given value of S, GRAEC outperforms the other two methods. The best GRAEC score, for $S = 1$ outperforms every other option. This is expected, the design of the simulation was to make optimal use of the adaptions GRAEC could give for $S = 1$. The optimal parameters for GRAEC indicate the adaption for both gradual and recurrent concept drift.

6.1.2 *The ineffectiveness of Naive.*

The Naive method is completely ineffective for any value of S. This makes sense, since one year of data contains four different recurrent concepts. As a result the Naive model will not even be able to properly explain the dataset it was trained on, let alone datasets that only contain a single recurrent concept, and datasets on which the gradual drift has had more effect.

6.1.3 *The effect of S.*

We shortly discuss the effect of each value of S below. We will refer to the way topic the related to the bottleneck activity as a concept, and refer to this concept in quartile 1 as 'A', in quartile 2 as 'B', 3 as 'C', and 4 as 'D'. A D_j^S can contain one or more concepts. For example, D_0^1 contains only concept 'A', as do D_1^1 and D_2^1, where as D_3^1 has concept 'B'. On the other hand D_1^2 has concepts 'A' and 'B' in an equal number of cases. We hence refer to the concept of D_1^2 as 'AB', and in line, the concept of D_0^2 is 'AA', the concept of D_2^3 is 'CCC', and D_1^4 has concept 'BBCC'.

S = 1 For the Recent method, we have that in two out of the three periods, the previous period had the same concept. As a result, about one third of the predictions are made based on data from a different recurrent concept, decreasing the overall score of the Recent method.

S = 2 and **S = 4** For $S = 2$ we will have 6 D_j^2's per year, with concepts 'AA', 'AB', 'BB', 'CC', 'CD', 'DD'. As a result it will be difficult to train models on the second and fifth D_j^2 each year (with concepts 'AB' and 'CD' respectively). For the Recent method, predictions are suboptimal because:

- in the third and sixth period each year predictions will be made with a bad model
- in the second and fifth period each year, only half of the cases are the same as the previous concept
- in the first and fourth period each year, none of the cases are the same as the previous concept

As a result, the Recent method has a lower score than for $S = 1$. GRAEC only has the disadvantage of the poor models during predictions in the third and sixth periods, because of the optimal value for τ. The first, second, fourth, and fifth period belong to a single concept ('A' through 'D' respectively), and models of previous years are re-used. As a result, the negative effect on the score for GRAEC is weaker. The score is still weaker than for $S = 1$, since there is less adaption to gradual drift, due to the optimal value of τ. For S=4, the arguments follow the same reasoning, though their effect is more present (resulting in a lower score). This is because *all* consecutive periods have different recurrent concepts.

S = 3 The effect of the recurrent concept drift, and how Recent fails to adapt to it while GRAEC does, is most evident for $S = 3$. Each D_j^3 will be based off a completely different concept compared to D_{j-1}^3. This has a devastating effect on the Recent method, which can at best tell something about short cases. The GRAEC on the other hand scores almost as well as for $S = 1$. The slightly decreased score comes from the inability to capture the gradual concept drift. The optimal $\beta = 1$, combined with $\tau = 100$ causes the influence of models trained in different periods to be insignificant, and GRAEC only adapting to the recurrent concept drift.

6.2 Real-World Dataset

The results of the real-world study are presented in Figure 5 and Table 3. Figure 5 shows the overall scores of each of the twelve methods applied, and Table 3 presents the values of β and τ that belong to each value of S for GRAEC.

GRAEC					Recent		Naive	
S	β	τ	ACC	F_1	ACC	F_1	ACC	F_1
7	1	0	0.417	0.418	0.405	0.405	0.251	0.245
14	1	0.1	0.418	0.417	0.412	0.413	0.253	0.245
21	1	0	0.412	0.410	0.405	0.406	0.264	0.258
28	1	1	0.419	0.416	0.415	0.418	0.267	0.259

Table 3: Optimal values for S, β, and τ, and the Accuracy and F_1 scores for the real-world dataset.

The key result to take away from Figure 5 is that the improvement of GRAEC over Recent is present, albeit small. The most likely explanation is that there is no strong effect of recurrent concept drift present in the real-world dataset. This also matches the results for τ; only $S = 28$ has a value for τ that puts significant weight to periodically similar models, though for $S = 28$, there will be two such models at best. Please note that the method: Recent does not exist, and that we used it as an intuitive method for comparison. The results do not disprove the effectiveness of GRAEC. The goal of GRAEC was to allow adaption to both gradual and recurrent concept drift. As is evident from the results for τ, GRAEC is in essence self-correcting the lack of recurrent concept drift.

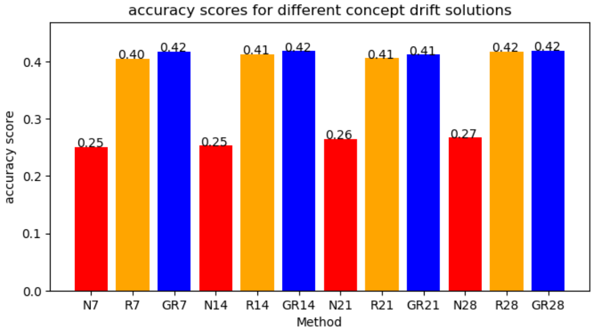

Figure 5: The results of the Real-world dataset in terms of accuracy scores. *GR* indicates the GRAEC method, *R* uses the most Recent model, and *N* creates a model once on the first year of the dataset. Each method is tested for different values of *S*, indicated by the numbers.

7 CONCLUSION AND FUTURE WORK

The results from the Simulation study show the potential of GRAEC. We have developed a method that can adapt to both gradual as well as recurrent concept drift. The case study on the real-world dataset shows improvement, albeit it small, over retraining the data periodically. The Simulation study stresses the importance of a correctly chosen S.

Like the simulation study, the real-world scenario was analysed in a post-mortem setting; there was data on completed cases available during analysis. A future research direction is to apply the simulation study as an online setting and include online optimisation of β, τ and possibly S, to see if the adaptions to gradual and recurrent concept drift are still optimal. In such an online setting, care should be taken that supervised information is available with a delay; cases that start in a certain period might complete (long) after the period ends. In such a scenario, the most recent model available may not be from the preceding period. This is because all cases that started in the preceding period might not have ended once the next period starts. As a result both the Recent and GRAEC method will deteriorate, though we expect the Recent method to be more affected by this.

The advantage of the use of a weighted ensemble, as GRAEC, is the availability of different machine learning models. Since we build one model on each period, we can freely choose the type of model each time. A different method, such as described in [12] is to use incremental learners, i.e. we do not create new models from scratch, but keep updating an existing model. As they show, especially the Adaptive Hoeffding Option Tree from [2] adapts well to recurring concept drift. The next step in this study is to compare the effectiveness of our proposed concept drift adaption to theirs, on their artificial data, our simulation study and our real-world data.

The weighting functions are based on related work and expert knowledge. In this work we have limited ourselves to one value for p. In future work, different values or a superposition of values could be created. Another update to GRAEC would be not to add weight to the single model from the period at $d.time - p$, but also (some) additional weight to the period before and after $d.time - p$.

One interesting further future direction is to borrow concepts from the online process discovery of process models [8] in order to extract process models on the fly and apply advanced stream mining techniques over those [7], [6] to extract the best classifiers for each period of time in an efficient manner.

Additionally, we would like to address the relatively different problem of drift detection in the more complicated and realistic scenario where interval-based events are considered [11]. In the case of overlapping of such events, more relations are considered between these temporal events appear. Also, we would like to investigate the implementation of available online drift detection techniques for observing deviations in streaming conformance checking applications [19] and anytime stream classification [10].

REFERENCES

[1] Manuel Baena-García, José Campo-Ávila, Raúl Fidalgo-Merino, Albert Bifet, Ricard Gavald, and Rafael Morales-Bueno. 2006. Early Drift Detection Method. *Fourth intl. Workshop on Knowledge Discovery from Data Streams* (01 2006), 77–86.

[2] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. 2009. New Ensemble Methods for Evolving Data Streams. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '09)*. ACM, New York, NY, USA, 139–148. https://doi.org/10.1145/1557019.1557041

[3] Avrim Blum. 1997. Empirical Support for Winnow and Weighted-Majority Algorithms: Results on a Calendar Scheduling Domain. *Machine Learning* 26, 1 (01 01 1997), 5–23. https://doi.org/10.1023/A:1007335615132

[4] João Gama and Petr Kosina. 2014. Recurrent concepts in data streams classification. *Knowledge and Information Systems* 40, 3 (01 Sep 2014), 489–507. https://doi.org/10.1007/s10115-013-0654-6

[5] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A Survey on Concept Drift Adaptation. *ACM Comput. Surv.* 46, 4, Article 44 (March 2014), 44:1–44:37 pages.

[6] Marwan Hassani. 2015. *Efficient Clustering of Big Data Streams*. Ph.D. Dissertation. RWTH Aachen University.

[7] Marwan Hassani. 2019. *Overview of Efficient Clustering Methods for High-Dimensional Big Data Streams*. Springer International Publishing, Cham, 25–42. https://doi.org/10.1007/978-3-319-97864-2_2

[8] M. Hassani, S. Siccha, F. Richter, and T. Seidl. 2015. Efficient Process Discovery From Event Streams Using Sequential Pattern Mining. In *2015 IEEE Symposium Series on Computational Intelligence*. 1366–1373.

[9] Paulo Mauricio Gonçalves Jr and Roberto Souto Maior de Barros. 2013. RCD: A recurring concept drift framework. *Pattern Recognition Letters* 34, 9 (2013), 1018 – 1025. https://doi.org/10.1016/j.patrec.2013.02.005

[10] Philipp Kranen, Marwan Hassani, and Thomas Seidl. 2012. BT* - An Advanced Algorithm for Anytime Classification. In *Scientific and Statistical Database Management - 24th International Conference, SSDBM 2012, Chania, Crete, Greece, June 25-27, 2012. Proceedings.* 298–315. https://doi.org/10.1007/978-3-642-31235-9_20

[11] Yifeng Lu, Marwan Hassani, and Thomas Seidl. 2017. Incremental Temporal Pattern Mining Using Efficient Batch-Free Stream Clustering. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management, Chicago, IL, USA, June 27-29, 2017.* 7:1–7:12. https://doi.org/10.1145/3085504.3085511

[12] Marco Maisenbacher and Matthias Weidlich. 2017. Handling Concept Drift in Predictive Process Monitoring. In *2017 IEEE International Conference on Services Computing (SCC)*, Vol. 00. 1–8. https://doi.org/10.1109/SCC.2017.10

[13] João Bártolo Gomes ; Mohamed Medhat Gaber ; Pedro A. C. Sousa ; Ernestina Menasalvas. 2014. Mining Recurring Concepts in a Dynamic Feature Space. *IEEE Transactions on Neural Networks and Learning Systems* 25, 1 (Jan 2014), 95–110. https://doi.org/10.1109/TNNLS.2013.2271915

[14] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* 12 (Nov. 2011), 2825–2830. http://dl.acm.org/citation.cfm?id=1953048.2078195

[15] Sasthakumar Ramamurthy and Raj Bhatnagar. 2007. Tracking recurrent concept drift in streaming data using ensemble classifiers. In *Sixth International Conference on Machine Learning and Applications (ICMLA 2007)*. 404–409. https://doi.org/10.1109/ICMLA.2007.80

[16] Sripirakas Sakthithasan, Russel Pears, Albert Bifet, and Bernhard Pfahringer. 2015. Use of ensembles of Fourier spectra in capturing recurrent concepts in data streams. In *2015 International Joint Conference on Neural Networks, IJCNN 2015, Killarney, Ireland, July 12-17, 2015.* 1–8. https://doi.org/10.1109/IJCNN.2015.7280583

[17] Erik Scharwächter, Emmanuel Müller, Jonathan F. Donges, Marwan Hassani, and Thomas Seidl. 2016. Detecting Change Processes in Dynamic Networks by Frequent Graph Evolution Rule Mining. In *ICDM'16*. 1191–1196.

[18] Arik Senderovich, Matthias Weidlich, Avigdor Gal, and Avishai Mandelbaum. 2014. Queue Mining – Predicting Delays in Service Processes. In *Advanced Information Systems Engineering*, Matthias Jarke, John Mylopoulos, Christoph Quix, Colette Rolland, Yannis Manolopoulos, Haralambos Mouratidis, and Jennifer Horkoff (Eds.). Springer International Publishing, Cham, 42–57.

[19] Sebastiaan J. van Zelst, Alfredo Bolt, Marwan Hassani, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. 2017. Online conformance checking: relating event streams to process models using prefix-alignments. *International Journal of Data Science and Analytics* (27 Oct 2017). https://doi.org/10.1007/s41060-017-0078-6

[20] Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. 2003. Mining Concept-drifting Data Streams Using Ensemble Classifiers. In *KDD '03*. ACM, 226–235.