Expert Systems With Applications 104 (2018) 67–85

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Adapting dynamic classifier selection for concept drift

Paulo R.L. Almeida[a,*], Luiz S. Oliveira[a], Alceu S. Britto Jr.[b,c], Robert Sabourin[d]

[a] Federal University of Parana (UFPR), Rua Cel. Francisco H. dos Santos, Curitiba, PR 100-81531-990, Brazil
[b] Ponta Grossa State University, Av. General Carlos Cavalcanti, Ponta Grossa, PR, 4748-84030-900, Brazil
[c] Pontifical Catholic University of Parana (PUCPR), R. Imaculada Conceição, Curitiba, PR 1155-80215-901, Brazil
[d] École de Technologie Supérieure, 1100 Notre-Dame Street West, Montreal, Quebec, Canada

ARTICLE INFO

Article history:
Received 20 December 2017
Revised 12 March 2018
Accepted 13 March 2018
Available online 15 March 2018

Keywords:
Concept drift
Dynamic classifier selection
Dynamic ensemble selection
Concept diversity

ABSTRACT

One popular approach employed to tackle classification problems in a static environment consists in using a Dynamic Classifier Selection (DCS)-based method to select a custom classifier/ensemble for each test instance according to its neighborhood in a validation set, where the selection can be considered region-dependent. This idea can be extended to concept drift scenarios, where the distribution or the *a posteriori* probabilities may change over time. Nevertheless, in these scenarios, the classifier selection becomes not only region but also time-dependent. By adding a time dependency, in this work, we hypothesize that any DCS-based approach can be used to handle concept drift problems. Since some regions may not be affected by a concept drift, we introduce the idea of concept diversity, which shows that a pool containing classifiers trained under different concepts may be beneficial when dealing with concept drift problems through a DCS approach. The impacts of pruning mechanisms are discussed and seven well-known DCS methods are evaluated in the proposed framework, using a robust experimental protocol based on 12 common concept drift problems with different properties, and the PKLot dataset considering an experimental protocol specially designed in this work to test concept drift methods. The experimental results have shown that the DCS approach comes out ahead in terms of stability, i.e., it performs well in most cases requiring almost no parameter tuning.

1. Introduction

Classical classification methods often rely on a static environment assumption, where a classifier may be trained using historical labeled data, and then be used in the classification of unlabeled instances during an indefinite amount of time. Although this assumption may be valid in many applications, it can, however, be unrealistic in some scenarios, such as those in which the distributions or the *a posteriori* probabilities change over time in a phenomenon known as concept drift.

To illustrate the concept drift problem, we may consider social media networks. Since social media applications hold a vast collection of user images, they could employ these images to recognize user faces in future images. Under this scenario, some challenges may arise as a result of the fact that users' faces are constantly changing due to aging factors, use of makeup, beard growth/shaving, different haircuts, etc. This scenario may, therefore, suffer from a concept drift, since images collected from the user in the past may not be suitable for recognizing him/her in the present. As with many concept drift problems, two important questions arise, namely: Which learned information is useful in the current scenario (concept)? How do we keep track of changes?

Dynamic Classifier Selection (DCS) represents an alternative to concept drift, as we demonstrated in Almeida, Oliveira, Britto, and Sabourin (2016). A DCS method basically involves selecting the best classifier/ensemble based on a local region of the feature space, usually defined as the neighborhood of the test instance in a validation set (although in the literature DCS may refer to a single classifier selection, and Dynamic Ensemble Selection (DES) refers to an ensemble selection, we will refer to both scenarios as DCS for the sake of simplicity). The rationale behind this is the possibility that we may have a pool of classifiers specialist in different regions of the feature space, and the classifiers are chosen according to the test instance location. However, region dependency alone is not sufficient for scenarios containing concept drifts, since the problem evolves over time. With the framework introduced in Almeida et al. (2016), the Dynamic Selection Based Drift Handler (Dynse), we made DCS not only region-dependent, but also time-dependent.

* Corresponding author.
E-mail addresses: prlalmeida@inf.ufpr.br (P.R.L. Almeida), lesoliveira@inf.ufpr.br (L.S. Oliveira), alceu@ppgia.pucpr.br (A.S. Britto Jr.), robert.sabourin@etsmtl.ca (R. Sabourin).

https://doi.org/10.1016/j.eswa.2018.03.021
0957-4174/

However, in our preliminary study, several important aspects were left behind, such as the time dependence modeling for real and virtual concept drifts, the neighborhood size, the impact of the pruning, and the comparison of multiple DCS approaches under concept drift scenarios. To fill this gap, in this work we address all these issues and hypothesize that by associating the time dependency to the nature of the concept drift, any neighborhood-based DCS approach can represent a natural answer to the concept drift problem, regardless of its properties, such as speed, severity, and the possible presence of recurrences. To validate this hypothesis, we tested seven well-known DCS methods by adding the time dependency (through the Dynse framework) to deal with 12 popular concept drift problems, which include real world and artificial scenarios. We also include a novel experimental protocol to use the PKLot (Almeida, Oliveira, Britto, Silva, & Koerich, 2015) real world problem as a concept drift benchmark, which can be seen as a concept drift scenario where new labeled data may arrive in batches.

Our experimental results show that DCS is capable of adapting to different concept drift scenarios by selecting the most promising classifier/ensemble for each test instance, regardless of the concept used to train the selected classifiers. In addition, we show that the DCS performs well in most cases, and requires almost no parameter tuning. We also developed the idea of concept diversity by showing that the method can benefit from very large pools, i.e., pools containing classifiers trained under different concepts. Finally, our experiments on scenarios containing real and virtual concept drifts indicated that the nature of the concept drift must be considered when modeling the time dependency in the validation dataset. The Dynse framework was implemented using the Massive Online Analysis (MOA) framework (Bifet, Holmes, Kirkby, & Pfahringer, 2010), and it is publicly available for download in Almeida, Oliveira, Britto, and Sabourin (2018). To the best of our knowledge, few works address the concept drift problem by the use of the DCS idea (Pan, Wu, Zhang, & Li, 2010; Pan, Zhang, & Li, 2012; Tsymbal, Pechenizkiy, Cunningham, & Puuronen, 2008). This study may serve as a basis for other researchers to adapt their DCS methods for concept drift.

The remainder of this work is structured as follows: Section 2 gives a brief formalization of the concept drift phenomenon, while Section 3 presents some important contributions related to the concept drift problem. Section 4 discusses important issues tackled in this work, such as the time dependency of DCS-based methods under real and virtual concept drifts, the local region size, and concept diversity when using a DCS-based approach in dealing with a concept drift. Still in Section 4 we modify the Dynse framework in order to accommodate a pruning module. Section 5 presents our experiments, including the experimental protocol for the PKLot dataset. Finally, Section 6 presents some perspectives for future developments and concludes this work.

2. Concept drift background

A concept drift may be fit into one of two main categories: virtual or real concept drift. In a virtual concept drift, the *a priori* probabilities $P(y)$ or the unconditional distribution $P(\mathbf{x})$ may change over time, although the best boundary that separates the classes remain unaltered; that is, $P_t(y) \neq P_{t+1}(y)$ or $P_t(\mathbf{x}) \neq P_{t+1}(\mathbf{x})$, where the index t denotes a moment in time. A real concept drift, on the other hand, is caused by a change in the *a posteriori* probabilities, where $P_t(y|\mathbf{x}) \neq P_{t+1}(y|\mathbf{x})$, followed or not by a virtual concept drift (Gama, Žliobaitė, Bifet, Pechenizkiy, & Bouchachia, 2014; Hoens, Polikar, & Chawla, 2012; Kolter & Maloof, 2007; Krawczyk, Minku, Gama, Stefanowski, & Woźniak, 2017).

Note that when the concept drift is only virtual, the known labeled instances at time t do not change their target classes at $t+1$. Conversely, in a real concept drift scenario, the instances that were known at t may change their target classes at $t+1$, causing an *a posteriori* probability change, and as a result, even if it was possible to create the best boundary at t, this boundary would be incompatible with the concept present at $t+1$ (Gama et al., 2014; Hoens et al., 2012; Kolter & Maloof, 2007; Krawczyk et al., 2017). According to Minku, White, and Yao (2010), a real concept drift may be considered *Severe* if all instances change their classes in the next concept, otherwise it may be considered *Intersected*. A similar definition is used by Tsymbal et al. (2008), where a change in a subregion of the instance space is called a *local concept drift*.

In some scenarios the concept may change gradually, thus requiring several steps for the new concept take place completely; this generates a period of uncertainty between two stable states (concepts). In other cases, the concept may change abruptly, when the new concept completely replaces the old one in just one step. Formally, the concept drift speed can be defined as the inverse of the number of steps taken for a new concept to replace the old one, and if this number of steps is greater than one, the concept drift will be considered gradual, otherwise, it will be considered abrupt (Gonçalves, de Carvalho Santos, Barros, & Vieira, 2014; Hoens et al., 2012; Krawczyk et al., 2017; Minku et al., 2010).

Under some scenarios, a previously seen concept may reoccur in the future, characterizing a concept recurrence. Recurrent concepts are often related to seasonal changes. An example is an application that must detect people in an outdoor environment, and must, therefore, adapt to a new concept due to snow in the winter season, and return to the old concept in the spring. When dealing with recurrent concepts, keeping the information (i.e. trained models or training samples) acquired in older concepts may lead to better results when these concepts reappear (Gama et al., 2014; Hoens et al., 2012; Krawczyk et al., 2017; Minku et al., 2010).

To make the adaption to new concepts possible, methods created to handle real concept drifts must often rely not only on the feature distribution $P(\mathbf{x})$ of the unlabeled incoming data but also on some labeled data. This data must be fed to the system regularly, and it represents the current concept. In some publications, it is assumed that the labeled samples arrive in a stream fashion (e.g., test-then-train) (Bifet, Holmes, Pfahringer, Kirkby, & Gavaldà, 2009; Jian-guang, Xiao-feng, & Jie, 2010; Sun & Li, 2011), while in other publications, such as in this work, it is assumed that a few labeled instances will be given to the system from time to time, in batches containing N labeled samples (Elwell & Polikar, 2011; Kapp, Sabourin, & Maupin, 2011; Kolter & Maloof, 2007; Siahroudi, Moodi, & Beigy, 2018).

3. Related work

In this section, we review some important contributions that have been proposed to deal with the concept drift phenomenon, where the methods were organized in six different categories.

The methods in the **window-based** category constitute one of the simplest approaches to deal with the concept drift problem, where a window containing the M latest labeled samples is used to train/update the classifier. A window-based method will basically "forget" old training instances, which could represent a previous concept. Besides its simplicity, window-based methods raise the question of how large M should be. A small window can generate a system with a fast reaction to changes, but at a cost of a suboptimal accuracy under stable regions; a large window, on the other hand, could create a well-trained classifier that slowly adapts when the concept changes (Kuncheva, 2004; Widmer & Kubat, 1996). Examples of methods based on this classical approach can be found in Rakitianskaia and Engelbrecht (2012), Widmer and Kubat (1996). Some authors have extended the original window idea in order to maintain a variable window size, as in Jian-guang et al. (2010),

Kuncheva and Žliobaitė (2009), Sun and Li (2011), Widmer and Kubat (1996).

The **gradual forgetting-based** methods represent another category that follows basically the same approach as the Window-based one, in which old training data is "forgotten" over time, but instead of abruptly discarding training samples, these methods gradually decrease the importance of old instances by applying a certain aging factor (Gama et al., 2014; Hoens et al., 2012). When a new training sample arrives, the gradual forgetting methods update the weights of all samples according to their ages, and the model is then retrained. Some examples of gradual forgetting-based methods can be found in Elwell and Polikar (2011), Krawczyk and Woźniak (2014), Schlimmer and Granger (1986).

Unlike the window- or gradual forgetting-based methods, which are "always adapting" their models, **trigger-based** methods adapt to the new concept by taking some action, such as discarding the old data and updating the models, only when a change is detected (Gama et al., 2014; Gonçalves et al., 2014). These methods have the advantage that when the concept is stable, the classification system can remain unaltered, thus reducing the overhead; alternatively, every new information collected in this stable region can be aggregated in the training set to improve the model accuracy. However, these methods can suffer from some problems, such as false alarms and delayed or undetected drifts. A false alarm case can cause an unnecessary discard of still relevant knowledge. Delayed alarms or undetected drifts will cause a performance loss since the classification system will keep classifying incoming test instances based on a previous concept. A few examples of trigger-based approaches can be found in Alippi, Boracchi, and Roveri (2013); Baena-Garcıa et al. (2006); Bifet and Gavaldà (2007, 2009); Chen, Koh, and Riddle (2016); Gama, Medas, Castillo, and Rodrigues (2004); Minku and Yao (2012).

Many methods can be fitted in the **ensemble-based** category, where most of them basically maintain a pool of weighted classifiers, and the weights are calculated with respect to the current concept. Ensemble-based techniques can facilitate the use of non-incremental learners in the classification task since, for instance, every new labeled batch received can be used to train a new classifier that will be added to a pool. Another advantage is that recurring concepts can be handled by reactivating relevant previously trained classifiers (Hoens et al., 2012). A popular strategy when using an ensemble-based approach is to weight the available classifiers and then classify new instances by means of the Weighted Majority method, where the weight of each classifier may refer to its performance in the latest labeled instances received (Gama et al., 2014; Kuncheva, 2004). Some important contributions that can be fitted in the ensemble-based methods category can be found in Bifet, Holmes, and Pfahringer (2010); Bifet et al. (2009); Brzeziński and Stefanowski (2011); Gomes et al. (2017); Jaber, Cornuéjols, and Tarroux (2013); Siahroudi et al. (2018); Street and Kim (2001); Wang, Fan, Yu, and Han (2003).

The **local region-based** methods use an estimation of the local competence of the available classifiers to tackle the concept drift problem. When classifying a test instance $\mathbf{x}$, these methods basically find the local region of $\mathbf{x}$ and use the most promising classifier(s) with respect to this region. Some methods in this category try to define the region of competence of the classifiers in advance by, for instance, defining it as the feature region of their training datasets as in Chan, Zhang, Ng, and Yeung (2011); Polikar, Krause, and Burd (2003); Zhu, Wu, and Yang (2004), while other methods may use some DCS-based technique in order to define the competence of the classifiers in the pool with respect to $\mathbf{x}$ dynamically. In the former scenario, the local region of $\mathbf{x}$ may be defined by the neighborhood of $\mathbf{x}$ in a validation dataset Q. The neighborhood of $\mathbf{x}$ can then be employed to estimate the classifiers' competence before classifying $\mathbf{x}$ itself. Variations of this idea can be found in Almeida et al. (2016); Pan et al. (2010, 2012); Sethi, Kantardzic, and Hu (2016); Tsymbal, Pechenizkiy, Cunningham, and Puuronen (2006). One fundamental problem with this strategy is how to update the validation set Q in order to keep track of the current concept. A possible approach is to keep Q with the M latest labeled instances received; however, the strategy used to define the size of M should vary depending on the concept drift properties of the problem, as discussed in Section 4.

Finally, the **distribution analysis-based** methods rely solely on the $P(\mathbf{x})$ and/or $P(y)$ analysis in order to detect and adapt to possible concept drifts. Distribution analysis-based approaches often deal with virtual concept drifts, since changes in the *a posteriori* probabilities $P(y|\mathbf{x})$ are not necessarily reflected in a change in $P(\mathbf{x})$ or in $P(y)$, making these methods blind to some real concept drifts (Kuncheva, 2004; Markou & Singh, 2003). Some examples of distribution analysis-based category can be found in Cavalcante, Minku, and Oliveira (2016); González-Castro, Alaiz-Rodríguez, and Alegre (2013); Pérez-Gállego, Quevedo, and del Coz (2017); Radtke, Granger, Sabourin, and Gorodnichy (2014); Raza, Cecotti, and Prasad (2016), where the works in González-Castro et al. (2013); Pérez-Gállego et al. (2017); Radtke et al. (2014) were designed to deal specifically with concept drifts in $P(y)$.

4. DCS under concept drift scenarios

Under static environments, a DCS method works by selecting the best ensemble for the test instance $\mathbf{x}$ based on its local region in a validation dataset Q. Considering N_x being the neighborhood of $\mathbf{x}$ in the validation dataset Q ($N_x \subseteq Q$), the DCS can be seen as a function $E_x = DS(N_x, P)$, where P is a pool of classifiers, from which each classifier will be tested using N_x, and E_x is a custom selected ensemble for the test instance $\mathbf{x}$ with respect to its local region N_x (Britto, Sabourin, & Oliveira, 2014; Cruz, Sabourin, & Cavalcanti, 2018; Didaci, Giacinto, Roli, & Marcialis, 2005).

Hence, under a static environment, the ensemble E_x is region-dependent only. Nevertheless, under a concept drift scenario, a time dependency must be incorporated since the region dependency alone may not suffice. To ensure that this work is self-contained, Section 4.1 briefly describes the Dynse framework which has been updated to accommodate a pruning module. In Section 4.2, we discuss some issues that must be considered when adapting a DCS-based method for concept drift scenarios, including the time dependency of a DCS-based method under real and virtual concept drifts scenarios. Also in Section 4.2 we present the benefits derived from estimating the classifiers competence in a local region, as well as the impact of varying its size, and the pool diversity problem under a concept drift scenario.

4.1. The Dynse framework

In our previous work (Almeida et al., 2016) we proposed the Dynamic Selection Based Drift Handler (Dynse) framework as a modular tool to cope with concept drift scenarios using Dynamic Classifier Selection (DCS). The framework basically trains new classifiers over time with the available labeled data and estimates the classifiers' competence for a test instance $\mathbf{x}$ based on the local region of $\mathbf{x}$ in a validation dataset, which contains the latest labeled data. In Almeida et al. (2016), we assumed that the pool of classifiers could increase indefinitely, nevertheless, due to resources constraints (e.g., available memory), it can be impossible to maintain all trained classifiers in the pool. In the light of this, we introduce

a pruning module for the Dynse framework, and make the updated version of the framework fully available in Almeida et al. (2018).

The updated version of the Dynse framework is presented in Algorithm 1, where a stream of batches is required as its input.

Algorithm 1: The Dynse framework algorithm. Underlined items refers to the main interchangeable components of the framework.

```
   Input: Stream of batches {B_1, B_2, ..., B_t},
   Maximum pool Size (D),
   Accuracy Estimation Window Size (M),
   Neighborhood Size (K),
   Classification Engine (CE),
   Pruning Engine (PE),
   Base Classifier (BC)
1  W ← ∅
2  P ← ∅ foreach Batch B ∈ Stream do
3      if B is a Labeled Batch then
4          W ← W ∪ B
5          if |W| > M then
6              removeOldestBatch(W)
7          end
8          C ← trainNewClassifier(BC, B)
9          P ← PE(P, W, C, D)
10     end
11     else
12         foreach test instance x ∈ B do
13             N_x ← KNearestNeighbors(x, k, W)
14             E_x ← CE(N_x, P)
15             x_class ← classify(x, E_x)
16             makeAvailable(x_class)// The result of the
                   classification is available to the user
17         end
18     end
19 end
```

In the beginning, both the accuracy estimation window W and the pool P are set to empty (steps 1 and 2). For each batch available in the stream, if the next batch is labeled, the following steps are performed:

- In steps 5 to 8, the accuracy estimation window W is updated to accommodate only the M latest labeled batches received ($|W| = M$). This window can be seen as the validation dataset Q in a DCS method, and its size should be adjusted according to the type of the concept drift to correctly estimate the classifiers' competence, as discussed in Sections 4.2.1 and 4.2.2.
- In step 9, a new classifier C is trained using the labeled batch. In this step any Base Classifier (BC) may be used for building a new classifier (e.g. SVM, MLP, KNN, ...).
- Finally, in step 10, the current Pool P, the accuracy estimation window W, the newly created classifier C and the Maximum pool size D are handed to the pruning engine PE, which must make a decision to maintain or to prune classifiers in P (or add C in the P), keeping the pool from increasing in size beyond the threshold D. The Pruning Engine can be seen as a function $PE(P, W, C, D) = P_p$, where P_p is the pruned pool, and $|P_p| \leq D$. Since PE is a parameter in Algorithm 1, any pruning strategy can be implemented in the framework.

On the other hand, if the batch being processed is not labeled, the steps 13 to 18 are executed, where for each test instance $\mathbf{x}$ in the current batch, the following steps are performed:

- In step 14, the k nearest instances in the accuracy estimation window W are selected to represent the local region of $\mathbf{x}$, where the neighborhood is defined as $N_x = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$, $N_x \subseteq W$.
- In step 15, the Classification Engine CE uses the set of neighbors N_x to select a custom classifier/ensemble E_x to $\mathbf{x}$ using the classifiers in P. The classification engine CE is a parameter in the Dynse framework that can be seen as a function $E_x = CE(N_x, P)$, and thus any DCS method based on the neighborhood of the test instance can be used.
- In step 16, the custom classifier/ensemble E_x is used to classify $\mathbf{x}$ and, finally, in step 17, the result of the classification is made available to the user.

As discussed earlier, we consider that the stream is composed of batches, each batch containing a number of instances. This scenario is considered in many works (Brzeziński & Stefanowski, 2011; Elwell & Polikar, 2011; Kolter & Maloof, 2007; Pan et al., 2012) and can be illustrated by, for instance, an anti-malware system, which may receive new labeled batches regularly in order to adapt to possible changing threats (*viruses*) (Jordaney et al., 2017), or by the parking spaces recognition problem, further discussed in Section 5.6, where we may receive small labeled image batches from previous days to adapt the classification system. For test-then-train scenarios, where we may receive one instance at a time, and commonly the true label of the test instance is given right after its classification, the Dynse framework can be adapted by keeping the latest S instances received in W, and by training a new classifier for every L instances accumulated (however, the test-then-train scenario where we receive one instance at a time is not in the scope of this work).

We reinforce that the framework is meant to be general and our implementation of the framework is public available in Almeida et al. (2018). Nevertheless, in order to give a concrete implementation example that is simple to reproduce, consider the main interchangeable components of the Dynse framework, which are underlined in Algorithm 1. The Hoeffding Tree (Domingos & Hulten, 2000) can be implemented as the *Base Classifier*, which is used to train new classifiers as new labeled batches arrive in line 9. An age-based pruning approach can be implemented in the *Pruning Engine* module. Thus, in line 10 of Algorithm 1, we can simply remove the oldest classifier if the pool reaches its limit. The KNORA-Eliminate (Ko, Sabourin, & Britto, 2008) can be used as the *Classification Engine* in line 15, thus all classifiers that correctly classify the entire neighborhood (N_x) will be part of the final ensemble. This concrete implementation refers to the default Dynse configuration, further discussed in Section 5.5.

The Dynse framework obviously can be fit in the *local region-based* methods discussed in Section 3, although it could also be fit in the *ensemble* or in the *window-based* (due to the use of a window to keep W up to date) categories. The ability to adapt any neighborhood-based DCS approach for concept drift scenarios by the addition of a time dependency (i.e. the update of W and the train/pruning of classifiers over time), is the main difference when comparing the Dynse to the other local region-based methods presented in Section 3, since these methods often define a fixed DCS method to deal with concept drift scenarios, instead of a general framework.

4.2. Important issues to be considered

In this Section some issues that must be considered when adapting a DCS method to a concept drift scenario are presented. The expected behavior and necessary adaptations for DCS-based methods under real and virtual concept drift scenarios are presented in Sections 4.2.1 and 4.2.2, respectively. A discussion about the local region of competence is presented in Section 4.2.3 and

the possible benefits of maintaining a pool diverse in terms of the training region and time are given in Section 4.2.4.

4.2.1. Dynamic classifier selection under real concept drifts

Under a real concept drift, the *a posteriori* probabilities of the instances may change over time (i.e., $P_t(y|\mathbf{x}) \neq P_{t+1}(y|\mathbf{x})$). As discussed earlier, under a static environment a DCS-based method selects a classifier/ensemble based on the classifiers' competence with respect to the neighborhood N_x of the test instance $\mathbf{x}$ in a validation dataset Q. Since the classifiers' competence is estimated using a subset of Q ($N_x \subseteq Q$), it is necessary to keep Q up to date with the current *a posteriori* probabilities.

To illustrate this problem, we should consider the two discriminant features $f_1 \in [0, 10]$ and $f_2 \in [0, 10]$ of the SEA Concepts benchmark (Street & Kim, 2001). We should also consider that between the times t and $t+1$ there is a (real) concept drift from Concept 1, where $\theta = 8$, to Concept 2, where $\theta = 9$ (in this problem, if $f_1 + f_2 \leq \theta$ the instance belongs to the positive class, or to the negative class otherwise). Fig. 1a shows a validation dataset (i.e., one containing labeled samples) Q_t containing instances collected at time t, while in Fig. 1b, the validation dataset Q_{t+1} contains instances collected at $t+1$ only. Circles are used to denote the $k = 5$ nearest neighbors of a test instance $\mathbf{x}$ in Fig. 1a (time t) and squares are used to identify the neighbors of $\mathbf{x}$ in Fig. 1b (time $t+1$).

Considering $t+1$ as the current time, and a pool P containing classifiers trained under Concept 1 (t) and classifiers trained under Concept 2 ($t+1$), the neighbors N_x of $\mathbf{x}$ in Q_{t+1} may lead to a good competence estimation of the classifiers in P, since N_x represents the local region of $\mathbf{x}$, and N_x is a subset of Q_{t+1}, which represents the current *a posteriori* probabilities (see Fig. 1b). To better understand the time dependency of the validation dataset, consider Fig. 1c, which contains a merge of the validation samples collected at both t and $t+1$. Under a static environment, this bigger validation dataset could improve the DCS performance, since it would possibly have a better coverage of the feature space. Nevertheless, since the boundary changed between t and $t+1$, this dataset may have some conflicting information.

As can be seen in Fig. 1c, only two neighbors of the test instance $\mathbf{x}$ come from the current concept $t+1$, while 3 neighbors come from the old concept t. Thus, the neighbors depicted in Fig. 1c may lead to a poor estimation of the classifiers' competence. This time dependence of the validation dataset Q raises the question of how to keep Q always up to date with the current concept or, as formulated at the beginning of this work, we may ask "How do we keep track of changes?". A possible solution for this problem is to keep only the latest M labeled samples/batches received in Q, as implemented in the Dynse framework as the *accuracy estimation window W*, where $|W| = M$.

This approach can be seen as a windowing strategy in the validation dataset, where a bigger value for M could lead to a better competence estimation of the classifiers under regions where the concept is stable (better coverage of the feature space). Nevertheless, the instances belonging to an old concept would take more time to be pruned in the presence of a concept change, and thus, a bigger window could lead to a poor estimation of the classifiers' competence under concept changing regions (i.e., some neighbors may belong to the old concept).

4.2.2. Dynamic classifier selection under virtual concept drifts

Let us analyze how the DCS deals with changes in $P(\mathbf{x})$ only (changes in $P(y)$ are not within the scope of this work). Under a virtual concept drift scenario, the *a posteriori* probabilities do not change over time. Thus, if we accumulate data acquired under previous concepts in the validation dataset Q, the classifiers' competence estimation will not be negatively affected if we consider only the neighborhood of the test instance $\mathbf{x}$ in Q. Instead, keeping as much data as possible in the validation set Q may be beneficial due to the better coverage of the feature space this will provide. To better understand this, let us consider an artificial two-feature $d_1 \in [0, 1]$ and $d_2 \in [0, 1]$ binary problem, where the instances belong to the *positive* class if $\sin(\pi d_1) \times (\frac{7}{9}) > d_2$, or to the *negative* class otherwise.

Let us consider that at the beginning of the system runtime, at time t, only labeled samples located in a region B_1 are available, and so the pool P contains only classifiers trained using instances located in B_1. If we consider a validation dataset Q_t at time t, composed of some samples from B_1, and a test instance $\mathbf{x}_1$, the neighborhood of $\mathbf{x}_1$ in Q_t may be considered as the "closest known local region" of $\mathbf{x}_1$. Thus, these instances can be used to estimate the classifiers' competence. This scenario is depicted in Fig. 2a, where the $k = 5$ nearest neighbors of $\mathbf{x_1}$ in Q_t are considered.

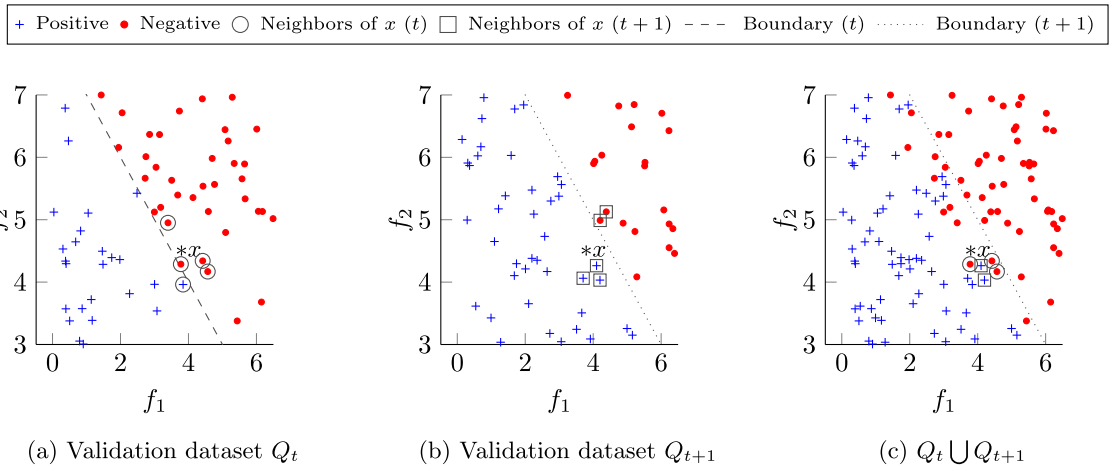
If new data belonging to a region B_2 become available at $t+1$, this new labeled data can be used to train new classifiers and to estimate the classifiers' competence. Since the concept drift here is only virtual, the labeled samples collected in t are still relevant for estimating the classifiers' competence in $t+1$ if we consider a DCS-based approach. This idea is presented in Fig. 2b, where instances collected at both t and $t+1$ are used in the validation dataset Q_{t+1}. As can be observed in Fig. 2b, the $k = 5$ validation samples collected at $t+1$ (i.e., in the B_2 region) are the closest to $\mathbf{x}_1$, and thus, these instances may be used to estimate the classifiers' competence with respect to $\mathbf{x}_1$. Still in Fig. 2b, a test instance $\mathbf{x}_2$ is introduced, and as we can see, the neighborhood of $\mathbf{x}_2$ is divided between the instances received at t (B_1) and $t+1$ (B_2), indicating that instances in both regions B_1 and B_2 may contribute to estimating the classifiers' competence with respect to $\mathbf{x}_2$. Note that under a scenario where the test instances have drifted to another region in the feature space before new training data arrived with respect to this feature region, a DCS-based approach is still able to estimate the classifiers competence based on the closest "known" regions of the feature space.

In summary, considering M as the number of the latest labeled batches/instances accumulated in the validation dataset Q (e.g., the *accuracy estimation window W* in the Dynse framework), we may say that a good policy would be to set M to be as big as possible in order to deal with a virtual concept drift, whilst a small value of M should be used in a real concept drift scenario in order to procure a faster adaptation to changes, as discussed in Section 4.2.1. These conclusions are empirically demonstrated in Section 5.2, where the tests showed that larger values for M generated a slower adaptation under a real concept drift scenario, while in a virtual concept drift scenario larger values for M leaded to better results.

4.2.3. The local region of competence

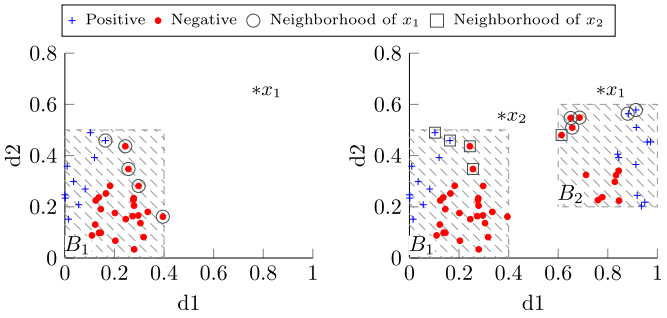
By taking into account the neighborhood of the test instance $\mathbf{x}$ under the current validation set Q by means of a DCS method, we are assuming that some information can be shared between concepts (i.e., "which information learned is still useful?"), and a classifier trained with an old concept may be still suitable under the current concept in some regions of the feature space (Tsymbal et al., 2006; 2008). To illustrate the rationale behind this thinking, see the example in Fig. 3a that shows the changed region between Concepts 1 and 2 in the SEA Concepts problem (Street & Kim, 2001). Consider the instances $\mathbf{x}_1$ and $\mathbf{x}_2$ in Fig. 3a, as well as Concept 2 ($\theta = 9$) as the current one, and that Q only contains instances labeled according to the current concept.

If we use the entire current validation dataset Q to estimate the classifiers' competence for both instances, as in Brzeziński and Stefanowski (2011); Karnick, Ahiskali, Muhlbaier, and Polikar (2008); Wang et al. (2003), then the very same set of classifiers will be selected to classify both $\mathbf{x}_1$ and $\mathbf{x}_2$. However, if we take into account



(a) Validation dataset Q_t (b) Validation dataset Q_{t+1} (c) $Q_t \bigcup Q_{t+1}$

Fig. 1. Neighborhood N_x of a test instance $\mathbf{x}$ in a validation dataset collected at t (1 a) and $t+1$ (1 b). In (1 c) N_x was computed using a merged validation dataset containing both t and $t+1$ instances.



(a) Validation dataset Q_t (b) Validation dataset Q_{t+1}

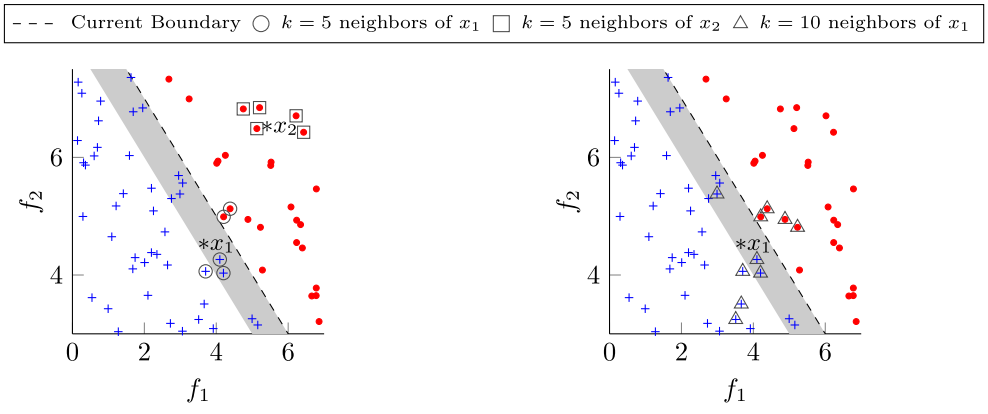
Fig. 2. Virtual concept drift caused by a change in $P(\mathbf{x})$. Neighbors of a test instance $\mathbf{x}_1$ in the validation dataset at t (1a), where only region B_1 is known. Neighbors of the instances $\mathbf{x}_1$ and $\mathbf{x}_2$ at $t+1$ (1 b), where both regions B_1 and B_2 are known.

only the local region of the test instances (see the neighbors of $\mathbf{x}_1$ and $\mathbf{x}_2$ in Fig. 3a), it becomes clear that classifiers trained under Concept 1 or 2 can classify $\mathbf{x}_2$, since the *a posteriori* probabilities did not change in the region where $\mathbf{x}_2$ was placed. On the other hand, it will be possible to verify that only the classifiers trained under the presence of Concept 2 will be able to classify $\mathbf{x}_1$, since it is expected that at least part of the neighborhood of $\mathbf{x}_1$ changed its *a posteriori* probabilities when the Concept changed (i.e., some instances are in the changed area).

As we estimate the classifiers' competence using the k nearest neighbors of the test instance, we may wonder how big k should be to define the local region. This is a fundamental problem with any DCS-based approach, regardless of whether or not the environment is static. Under a concept drift scenario, we may relate the neighborhood size to the DCS plasticity of the method, especially when the concept drift affects only part of the feature space (as in the SEA Concepts problem).

To demonstrate this, consider the neighborhood of $\mathbf{x}_1$ in Fig. 3a and b. As we increase the neighborhood size, as in Fig. 3b, there is a greater probability for some validation samples to be taken from regions that did not change; consequently, classifiers that are oblivious to this change may be wrongly selected to classify the test instance, depending on the DCS strategy implemented (e.g., a classifier trained in the old concept may be selected, since it would have a high accuracy as most neighbors are outside the changed region). On the other hand, a smaller neighborhood (Fig. 3a) could better represent the local region of the test instance in the current concept, thus giving a better competence estimation for the classifiers with respect to the test instance.

It is important to note, though, that even a smaller neighborhood may contain validation instances from areas where the con-



(a) $K = 5$ nearest neighbors (b) $K = 10$ nearest neighbors

Fig. 3. Neighbors of test instances in a validation dataset Q that contains only instances with respect to the concept 2 in the SEA Concepts problem. The gray area depicts the region that has a change between Concepts 1 ($\theta = 8$) and 2 ($\theta = 9$).

cept did not change, especially when the test instance is close to the boundary that changed between concepts. Estimating the optimal local region size may be a challenging task (Gomes, Bardal, Enembreck, & Bifet, 2017), but as discussed in this section, some DCS-based methods may decrease their performances when using larger local regions. Based on the fact that under static environments, DCS methods often provide good results with small neighborhoods (e.g., $k = 5$ for the DCS-LA methods (Woods, Bowyer, & Kegelmeyer, 1996)), we thought that using the same values that produced good results in the static environment would be a good start. This conclusion was taken into account in Section 5, where a neighborhood of size 5 was used in all tests.

4.2.4. Concept diversity

When dealing with classification problems in a static environment using a DCS strategy, we must rely on a diverse pool of classifiers P, where P is often considered constant (e.g., the classifiers in P are trained before the system is deployed, and then P is kept unchanged during the entire system runtime) (Britto et al., 2014; Didaci et al., 2005). However, under a concept drift where unconditional and/or the *a posteriori* distributions may change over time, it is necessary to train and add new classifiers in P as new labeled information arrives, in order to guarantee that the pool contains at least one classifier trained with the latest information received (as done in the Dynse framework).

Since new classifiers are added to P over time, a classifier pruning approach may be necessary in order to keep P from increasing in size indefinitely. This pruning mechanism must be implemented in the *pruning engine* module in the Dynse framework, where we should keep a diverse pool. The diversity is important for the Dynse due to the use of a DCS based approach to select the most promising ensemble for the test instances. Under a static environment, a DCS based method may benefit from a pool that contains classifiers specialized in different regions of the feature space, thus the diversity is region dependent. The region dependency alone may also suffice in scenarios that suffer from virtual concept drifts only. Nevertheless, the region dependency may be not sufficient in a real concept drift scenario, where it is also necessary to consider a time dependency in order to keep classifiers trained under different concepts (here, a different concept refers to a different *a posteriori* probability).

A DCS-based method may benefit from a pool containing classifiers trained under past concepts since some regions of the feature space may not be affected by the concept drift. Also, the presence of classifiers trained under previous concepts in the pool may lead to a seamless reaction under the presence of a recurrent concept (Tsymbal et al., 2006; 2008). In this work, a pool that contains classifiers specialized in different regions of the feature space (i.e. region diversity), and trained under different concepts (i.e. time diversity) is defined as a *Concept Diverse* pool. Thus, a pruning method may take into consideration the Concept Diversity when deciding to prune some classifier from P, or at least we should maintain as many past classifiers as possible (Almeida et al., 2016). As an example, consider Fig. 4, where the classifiers C_1 and C_2 are specialized in the region R with respect to Concept 1, while the classifier C_3 is specialized in the same region R, but with respect to Concept 2. In this example, a good candidate to be pruned from the pool should be the classifier C_1 or C_2, since both are specialists in the same region and in the same concept.

This idea is empirically demonstrated in Section 5.4, where it can be observed that a pool of "infinite size" generated the best results in the majority of the benchmarks. The infinite size pool can be considered Concept Diverse, since the classifiers trained at different times, and possibly covering different regions of the feature space, are kept in the pool.

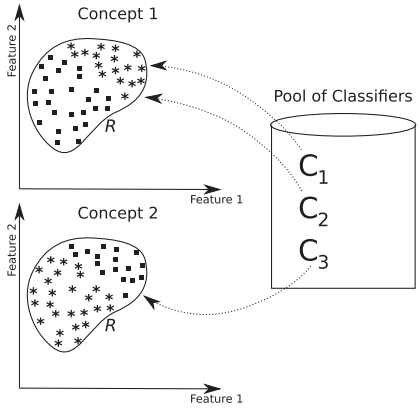


Fig. 4. Classifiers C_1 and C_2 are specialized in the region R with respect to Concept 1, while classifier C_3 is specialized in the same region under the presence of Concept 2.

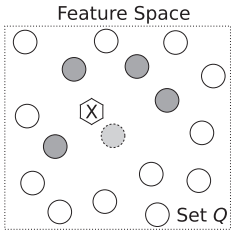


Fig. 5. The $k = 5$ nearest neighbors of a test instance x in a validation dataset Q (gray instances). The closest instance to x (dashed lighter gray) is noise.

5. Experiments

In this section, we present the experiments that were conducted to validate the discussions presented in the preceding sections of this work. We used the Dynse framework as a DCS-based approach to tackle concept drifts, and some classical DCS methods were used as the *Classification Engines* of the framework. Table 1 describes the tested *Classification Engines*, where the *Oracle* method refers to a hypothetical perfect *DSC* method, which always selects the first classifier able to correctly classify the test instance when the pool contains such a classifier. We will consider the *Oracle* as an upper bound, although it is important to note that the *Oracle* may be an overly optimistic accuracy limit (Didaci et al., 2005).

A modified version of the K-E (Ko et al., 2008) method is considered. In the modified version, given a set of neighbors N_x, where $|N_x| = k$, it will select all classifiers that correctly classify at least $k - l$ neighbors in N_x, where l is defined as the slack variable (Almeida et al., 2016). If no classifier is able to correctly classify at least $k - l$ neighbors, the value of l is increased by one. In the original version of the K-E, the selected classifiers must correctly classify the entire neighborhood and, when no classifier is able to classify all k neighbors, the value of k is reduced, which may lead to noise related problems. To illustrate this, consider Fig. 5, where a noisy instance is the closest one to the test instance x. In this scenario, the original version of the K-E would probably decrease the neighborhood size k until only the noise is present (presumably no classifier would be able to classify the entire neighborhood according to its labels due to the noisy instance), thus the classifiers would be tested using a single wrongly labeled sample.

To assess the pruning engine module in the Dynse framework we implemented two classical methods: *Age*—Remove the oldest classifier (Kapp et al., 2011; Pan et al., 2012) and *Accuracy*—Remove the worst performing classifier with respect to the current Accu-

Table 1
Relation between the acronyms and the DCS methods implemented as classification engines.

Acronym	Description
OLA	DCS-LA overall local accuracy (Woods et al., 1996)
LCA	DCS-LA local class accuracy (Woods et al., 1996)
Priori	A Priori (Giacinto & Roli, 1999)
Posteriori	A Posteriori (Giacinto & Roli, 1999)
K-U	KNORA-Union (Ko et al., 2008)
K-UW	KNORA-Union-W (Ko et al., 2008)
K-E	KNORA-Eliminate (Ko et al., 2008) modified as in Almeida et al. (2016)
Oracle	The Oracle theoretical perfect DCS method (Giacinto & Roli, 1999; Woods et al., 1996)

Table 2
State-of-the-art methods used in the experiments.

Acronym	Method	Category
AUE	The Accuracy Updated Ensemble (Brzeziński & Stefanowski, 2011)	Ensemble
AWE	The Accuracy-Weighted Ensembles method Wang et al. (2003).	Ensemble
ADACC	The Anticipative Dynamic Adaptation to Concept Change method (Jaber et al., 2013)	Ensemble
DDM	The Drift Detection Method (DDM) (Gama et al., 2004)	Trigger
EDDM	The Early Drift Detection method (Baena-Garcıa et al., 2006)	Trigger
HAT	The Hoeffding Adaptive Tree (Bifet & Gavaldà, 2007)	Trigger
LevBag	The Leveraging Bagging method (Bifet et al., 2010)	Trigger/ensemble
OzaAD	The method proposed in Bifet et al. (2009) using the ADWIN Bifet and Gavaldà (2007) trigger.	Trigger/ensemble
OzaAS	The method proposed in Bifet et al. (2009) using Adaptive-Size Hoeff. Trees.	Trigger/ensemble
ARF	The Adaptive Random Forests method (Gomes et al., 2017).	Trigger/ensemble
NaiveComb	A *naive combination* of all created classifiers	–

racy Estimation Window (Kuncheva, 2004; Tsymbal et al., 2006; Wang et al., 2003). To have a better insight into the pruning impacts, we also consider an "infinite pool size" (*Infinite*).

For comparison purposes, we also present the results obtained by some state-of-the-art concept drift methods. We used the default implementations and parameters of these methods available in the MOA framework (Bifet et al., 2010). The acronyms of the tested state-of-the-art methods are available in Table 2. In some tests, we also show the result achieved by a method called *Naive Combination*, which just trains and adds classifiers to its pool for every new labeled batch received, and then combine all classifiers by the majority voting rule to classify the test instances. It can be considered as a lower bound, since it is a method that takes no action to adapt to the drift, in spite of the arrival of new information.

All implemented approaches use Hoeffding Trees (Domingos & Hulten, 2000) as base learners (the HAT method uses a variation of the Hoeffding tree as defined in the original paper (Bifet & Gavaldà, 2007)), which is a common choice for concept drift scenarios (Gomes et al., 2017). For uniformity, the DDM and EDDM triggers use a pool of Hoeffding Trees as the learner, where every new labeled batch received is used to train a new classifier, and all classifiers are combined using the majority voting. Depending on the benchmark used in the tests, a holdout or a test-then-train approach is used. In the holdout strategy, for each time step, a train set is given, followed by an independent test set. In the test-then-train approach, at each time step the system must classify the received batch and, after the classification, the true labels of the batch are given and used for training (Ditzler, Roveri, Alippi, & Polikar, 2015; Krawczyk et al., 2017). The acronyms and a brief discussion about the datasets used in the tests are follow given.

STAGGER: The STAGGER Concepts problem was introduced in Schlimmer and Granger (1986). This synthetic dataset contains abrupt real concept drifts, and its instances are represented by three discrete features: $color \in \{r, g, b\}$, $shape \in \{c, t, r\}$ and $size \in \{s, m, l\}$. The problem is binary and a holdout evaluation is used. During the tests, there is an abrupt concept change for every 10 time steps. At each time step 20 labeled samples are given for training, and then 200 unlabeled samples are given for testing. The *positive* class is defined by $color = r \wedge size = s$ in the steps 0–9, $color = g \vee shape = c$ in the steps 10–19, and $size = m \vee size = l$ in the steps 20–29. In the steps 30–39 the first concept is repeated.

SEA: The SEA Concepts problem (Street & Kim, 2001) contains three randomly generated real features f_1 , f_2 and f_3 in the range [0, 10], and two possible classes $y \in \{pos, neg\}$, where the boundary that separates the classes is given by $f_1 + f_2 \leq \theta$. Noise is introduced by changing the class of 10% of the instances, and concept drifts are introduced by changing the θ value. During the test, at each time step, a labeled batch containing 250 samples is given for training, and another batch containing 250 samples from the same concept is generated for testing (holdout test). The concept is changed abruptly for each 50 steps, and the concepts are presented in the order $\theta = 8, \theta = 9, \theta = 7$ and $\theta = 9.5$ (Elwell & Polikar, 2011). Note that in this problem much of the information is shared between concepts (i.e. local concept drift).

SEARec: This benchmark follows a similar specification of the SEA benchmark. Nevertheless, in this configuration there is a concept change for every 25 steps. As in the SEA benchmark, there is a total of 200 steps in the problem, thus all concepts are repeated once in the order $\theta = 8, \theta = 9, \theta = 7, \theta = 9.5, \theta = 8, \theta = 9, \theta = 7, \theta = 9.5$.

CkrC, CkrP, CkrS and CkrE: These benchmarks refer to the Rotating Checkerboard problem considering *constant, Gaussian pulse, sinusoidal* and *exponential* changes, respectively (Elwell & Polikar, 2011). The datasets present real concept drift problems, where the boundaries are always changing gradually. Noise is added in 10% of the instances, and there is a total of 400-time steps in each benchmark. At each time step, 25 samples are given for training, and 1024 are given for testing (holdout test). This version of the benchmark and experimental protocol is proposed in Elwell and Polikar (2011), where the authors also provide a link to download the datasets.

Gauss: The Gauss dataset with class Addition/Removal used in Elwell and Polikar (2011). The concept drift is real, and the problem contains 300-time steps, where at each time step 20 samples are given for training, and 1024 samples are used for testing in a holdout fashion. All classes are constantly and gradually drifting. At the time step 120 a new class is introduced, and at time 240

Table 3
Main properties of each benchmark used. The train/test sizes refer to the batch size given for training/testing at each time step.

Benchmark	Drift type	Test type	Featur.	Classes	Num. steps	Train size	Test size
Real concept drift benchmarks							
STAGGER (Schlimmer & Granger, 1986)	Abrupt real	Holdout	3	2	40	20	200
SEA (Elwell & Polikar, 2011; Street & Kim, 2001)	Abrupt real	Holdout	3	2	200	250	250
SEARec[a] (Street & Kim, 2001)	Abrupt real	Holdout	3	2	200	250	250
Ckr[b] (Elwell & Polikar, 2011)	Gradual real	Holdout	2	2	400	25	1024
Gauss (Elwell & Polikar, 2011)	Gradual real	Holdout	4	2	300	20	1024
Real world benchmarks							
Nebr (Elwell & Polikar, 2011; Escovedo et al., 2013)	–	Test-train	8	2	604	30	30
For (Lichman, 2013)	–	Holdout	54	7	264	200	2000
Virtual concept drift benchmarks							
Dig (Lichman, 2013)	Virtual	Holdout	64	10	56	50	50
Let (Lichman, 2013)	Virtual	Holdout	16	26	200	50	50

[a] The SEARec is a variation of the original SEA benchmark containing recurrences.
[b] Configuration valid for all checkerboard benchmark variants (CkrC, CkrP, CkrS and CkrE).

one of the classes is removed. A more detailed description of the benchmark and download instructions are available in Elwell and Polikar (2011).

Nebr: The real world Nebraska Weather dataset using the same configuration employed in Elwell and Polikar (2011); Escovedo, Da Cruz, Vellasco, and Koshiyama (2013), where only the eight features with a missing feature rate less or equal than 15% were used, and the remaining missing values are replaced by the mean of the features in the preceding and following samples. The dataset is ordered in a chronological order and a test-then-train approach is used. At each time step, 30 samples are used for training, and the next 30 samples are given for testing. In the next time step, the testing samples are used as a training batch (the real labels are given), and the subsequent 30 samples are given for testing. This procedure is repeated until all samples are used, generating a test procedure containing 604-time steps.

For: The Forest Cover Type problem available at the University of California, Irvine (UCI) repository (Lichman, 2013). Besides its popularity as a test-then-train dataset (Bifet et al., 2010; Bifet et al., 2009; Chen et al., 2016; Krawczyk & Woźniak, 2014), in Bifet (2017) it is argued that this dataset may present a high correlation between its data, making it a trivial problem. Thus, in order to increase the problem difficulty, the instances are presented in the same order of the original dataset, and at each time step, a batch containing 200 samples is given for training, and a batch containing the next 2000 samples is given for testing. The dataset is used in a holdout form, thus, the testing samples are never used for training and, conversely, the training samples are never used for testing. This process is repeated until all samples are used, generating a test with 264-time steps.

Dig and Let: The Digit (Alpaydin & Kaynak, 1998) and Letters (Frey & Slate, 1991) problems available at the UCI (Lichman, 2013) repository with artificially introduced virtual concept drifts. To artificially introduce a virtual concept drift in the datasets, the following strategy is employed: at each time step, a random sample and its 49 nearest neighbors are taken from the dataset to be used in the training phase (i.e., 50 training samples). The training samples are removed from the dataset, and then 50 samples are randomly taken for the testing phase (differently from the training phase, all 50 samples are taken randomly during the testing phase). The testing samples are also removed from the dataset, and a new step begins (holdout test). This process continues until the dataset is empty. This procedure results in highly biased training sets with data concentrated in a single region of the feature space.

The main properties of each benchmark are shown in Table 3. The experiments are further organized as follows. In Section 5.1 we validate the DCS approach under local changes. Different accuracy estimation window sizes are put to test under real and virtual concept drift problems in Section 5.2. In Section 5.3 the results achieved using several DCS methods as classification engines are presented. In Section 5.4 some classical pruning strategies are tested and compared with a pool of "infinite size". Finally, in Section 5.5, the results achieved by a default configuration of the Dynse framework are compared with some state-of-the-art-methods. All results reported are the average of 10 replications.

5.1. The DCS approach under local changes

So far we have argued that a DCS-based method should be able to select a good set of classifiers for the test instance according to the current concept, regardless of the training concept of the selected classifiers. To validate this, first we use a configuration of the Dynse framework adopting the K-E as the classification engine without any pruning strategy and considering $k = 5; l = 0; M = 4$, to check its behavior in the SEA Concepts problem without noise[1] Table 4 presents the average accuracy of the Dynse framework in each concept of the SEA Concepts problem, as well as the average proportion of classifiers trained in each concept selected to classify the instances under the presence of the individual concepts of this dataset. For instance, in the presence of Concept 2, 46.5% of the classifiers selected to classify the test instances were trained under the presence of Concept 1. The numbers in parentheses show the perfect proportion (i.e., considering perfect classifiers and a perfect *Classification Engine*) that should be selected in each concept.

As can be observed, the proportions presented in Table 4 are close to the theoretical perfect values, indicating that with the Dynse framework, it is possible to create a DCS capable of correctly select classifiers trained under the current concept and also capable of reusing classifiers trained under old concepts in regions where they are still useful. Note that we repeated the first concept as the last one, where it can be observed an increase in the accuracy of the recurrent concept compared when it first appeared (97.9% versus 97.2%, respectively). This higher accuracy and the proportion of selected classifiers close to the theoretical ones in the recurrent concept show that a DCS-based method can handle a recurrent concept scenario without any further modification.

5.2. The impact of the accuracy estimation window size

In this Section, we evaluate the behavior of the Dynse framework for different accuracy estimation window sizes (M) under real and virtual concept drift scenarios. To verify the impact of M in a real concept drift scenario, the original configuration of the

[1] The noise was removed in order to better visualize the behavior of the method.

Table 4
Average accuracy and proportion of classifiers trained in each concept selected to classify the instances in the SEA Concepts problem without noise. The number in parenthesis indicate the perfect proportion that should be selected.

		Test concept				
		Concept 1	Concept 2	Concept 3	Concept 4	Concept 1
Train concept	1	100% (100%)	46.5% (47.8%)	34.5% (33.5%)	23.3% (24.0%)	26.3% (27%)
	2	0.0% (0.0%)	53.5% (52.2%)	30.8% (30.4%)	26.7% (26.4%)	25.9% (24.7%)
	3	0.0% (0.0%)	0.0% (0.0%)	34.7% (36.2%)	21.3% (21.9%)	23.8% (24.9%)
	4	0.0% (0.0%)	0.0% (0.0%)	0.0% (0.0%)	28.7% (27.7%)	23.9% (23.4%)
Accuracy		97.2%	97.6%	97.7%	97.4%	97.9%

Table 5
Average accuracy of the Dynse framework in the original SEA Concepts Benchmark (including noise), considering different accuracy estimation window sizes.

Classification eng.	Accuracy estimation window size (M)					
	$M = 1$	$M = 4$	$M = 8$	$M = 16$	$M = 32$	$M = \infty$
K-E $k = 5; l = 0$	87.07%	**87.48%**	87.38%	86.98%	85.95%	84.22%
K-U $k = 5$	85.48%	86.09%	86.46%	86.33%	85.54%	84.09%
NaiveComb						83.71%
Oracle						93.90%

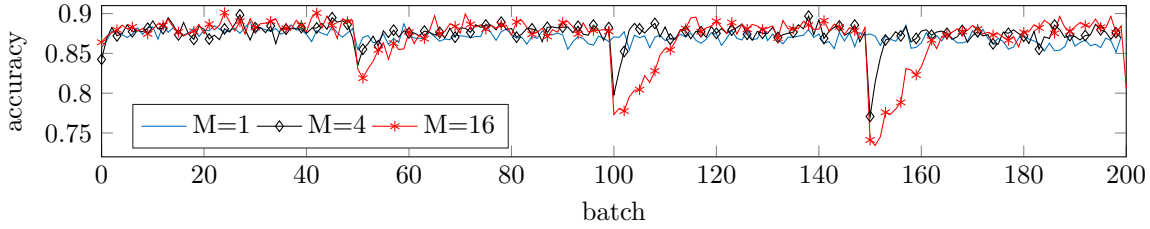


Fig. 6. Accuracy over time plots in the SEA Concepts problem for different M values, considering the K-E as the classification engine and $K = 5; l = 0$.

Table 6
Average accuracy of the Dynse framework in a virtual concept drift scenario in the Digit dataset, considering different accuracy estimation window sizes.

Classification eng.	Accuracy estimation window size (M)					
	$M = 1$	$M = 4$	$M = 8$	$M = 16$	$M = 32$	$M = \infty$
K-E $k = 5; l = 0$	13.78%	36.02%	53.46%	68.56%	76.79%	**77.17%**
K-U $k = 5$	15.02%	38.90%	54.00%	68.70%	74.90%	75.80%
NaiveComb						12.89%
Oracle						86.15%

SEA Concepts benchmark (including noise) is used. The average accuracy achieved for different M sizes considering the K-E and K-U as classification engines, and for the Naive Combination and the Oracle methods, are presented in Table 5. The complete plot containing the accuracy over time for some of the M sizes considering the K-E classification engine is presented in Fig. 6. No pruning strategy was used in any scenario.

Table 5 and Fig. 6 show that bigger M values cause a slower adaptation under concept changes. Notwithstanding, when the value of M is increased from 1 to 4 (and also from 4 to 8 for the K-U), a better average accuracy is achieved. By analyzing the plot in Fig. 6, it becomes clear that, besides the larger value of M generated a slower recover under concept changes, the average accuracy was improved by a more accurate classification when the concept is stable (See the discussion in Section 4.2.1). Since a value of $M = 4$ presented a good trade-off between a fast reaction and a good performance in stable regions, it will be considered as the default value for M in the Dynse framework (i.e. a good starting point when configuring the Dynse for a given real concept drift scenario).

To test the M size impact in a virtual concept drift scenario, the Digit recognition problem (i.e. the *Dig* problem) is used. Table 6 contains the results achieved by varying the M size for the Dynse framework using the K-E and K-U as classification engines, without considering any pruning strategy. In Table 6 the results achieved by the *Naive Combination* and the *Oracle* are also showed. We can observe from Table 6 and Fig. 7 that bigger values for M lead to better results in the virtual concept drift problem. These results corroborate the discussion we presented in Section 4.2.2. Since the dataset used this test is relatively small, it was possible to keep all labeled instances received in the Accuracy Estimation Window ($M = \infty$). Of course, this is impossible in most real-world problems, and thus, an interesting challenge for future works could be how to maintain this window with a fixed size and covering the biggest possible feature space to better adapt to a virtual concept drift.

Note that in the beginning of the test, the Dynse framework configured with larger values for M generated results close to the Oracle upper bound, however after about 10 time steps the Oracle became the leading method, indicating that there is room for improvement. Due to its good results, an accuracy estimation window of size 32 ($M = 32$) will be defined as the default value for virtual concept drift problems, although it is clear that bigger values could lead to better results (however a window that is too wide may be impractical in a real environment). A final observation regarding

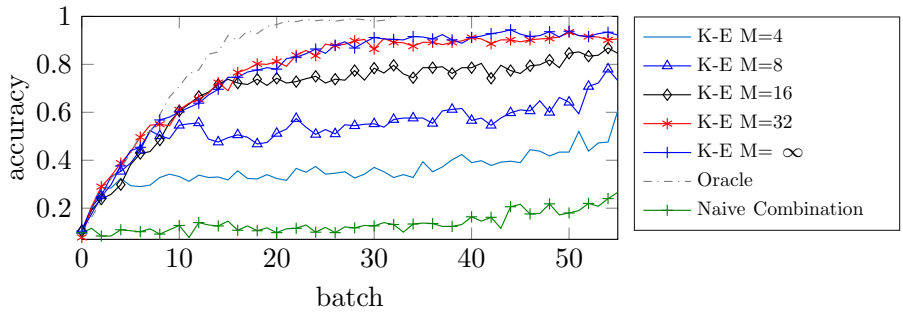


Fig. 7. Accuracy over time plots for the Digit problem using the K-E ($k = 5; l = 0$) as classification engine and different M values.

Table 7
Artificial and real world benchmarks average accuracies (%), using different classification engines. Best results (not considering the Oracle) are in bold. Standard deviation between test batches are shown in parenthesis.

Benchmark	OLA	LCA	Priori	Posteriori	K-U	K-UW	K-E	Oracle	NaiveComb
Real concept drift benchmarks									
Stagger	91.9(17.7)	75.2(13.9)	93.6(12.0)	**93.9**(12.3)	86.1(17.9)	91.8(14.3)	92.0(16.6)	99.7(1.2)	66.3(17.6)
SEA	87.4(1.3)	83.5(3.1)	86.5(1.4)	86.9(1.5)	86.1(1.9)	86.3(1.9)	**87.5**(1.4)	93.9(1.4)	83.7(3.0)
SEARec	**87.1**(1.8)	83.5(3.0)	86.3(1.6)	86.6(1.7)	86.3(2.0)	86.3(1.9)	**87.1**(1.8)	94.1(1.4)	83.8(3.2)
CkrC	85.6 (3.1)	68.7 (7.4)	**86.4** (2.7)	83.9 (2.9)	83.7 (3.5)	86.1 (3.0)	85.6 (3.1)	99.9 (2.5)	51.1 (8.6)
CkrP	86.5(4.9)	68.4(6.9)	**86.8**(4.8)	85.2(5.0)	85.1(5.1)	86.7(4.7)	86.5(4.9)	99.8(2.8)	49.2(8.8)
CkrS	86.2(5.1)	70.4(9.8)	**86.7**(4.6)	84.2(4.7)	84.0(5.3)	86.3(5.0)	86.2(5.1)	99.6(3.6)	63.7(18.9)
CkrE	85.3(3.3)	64.0(6.1)	**86.1**(3.5)	83.9(3.4)	83.2(3.5)	85.8(3.5)	85.3(3.3)	99.6(3.8)	51.7(10.2)
Gauss[a]	89.5(5.7)	85.9(8.1)	**90.1**(5.4)	89.9(6.6)	89.4(7.1)	89.5(7.0)	89.5(5.7)	-	78.4(13.3)
Real world benchmarks									
Nebr	74.5(1.0)	70.0(1.1)	73.7(1.0)	71.9(1.1)	74.4(1.1)	**74.7**(1.0)	74.5(1.0)	97.2(4.3)	70.4(1.1)
For	**78.3**(10.6)	50.2(19.8)	77.5(10.5)	76.3(11.0)	77.5(11.1)	77.8(11.0)	**78.3**(10.6)	98.6(7.4)	68.8(11.9)
Virtual concept drift benchmarks									
Dig	74.9(20.6)	11.2(2.4)	75.4(20.1)	72.6(19.8)	74.9(22.1)	74.5(21.3)	**76.8**(20.5)	86.2(24.4)	12.9(3.7)
Let	66.2(14.9)	23.1(8.4)	64.6(14.3)	58.5(11.6)	67.1(15.7)	**70.0**(16.1)	66.5(14.0)	93.2(18.7)	41.9(16.0)
Avg. Rank	3.1	7.5	2.6	4.4	5.2	3.1	**2.5**	-	7.5

[a] Only the class priors of the test batches are available in Elwell and Polikar (2011), thus it was not possible to compute the Oracle accuracy.

the results in Table 6 and Fig. 7, is that even if the concept drift is only virtual, some classifiers may be highly specialized in a specific region, and thus, it may be suboptimal to use them to classify test instances that are not in the regions where the classifiers are specialized, as shown by the poor performance achieved by the *Naive Combination* method.

5.3. Classification engines tests using common benchmarks

In this Section the DCS methods listed in Table 1 are used as classification engines for the Dynse framework. During the tests no pruning approach was implemented. As estimated in Section 5.2, the accuracy estimation window size will be set to 4 ($M = 4$) for real concept drift problems, and to 32 for virtual concept drift problems ($M = 32$). Since the authors of the DCS methods implemented as classification engines often defined a small neighborhood to be a good starting point (usually a value between 5 and 10) (Hung-Ren Ko, Sabourin, & de Souza Britto, 2007; Ko et al., 2008; Woods et al., 1996), a neighborhood of size 5 ($k = 5$) was defined in the tests for all DCS methods (also $l = 0$ for the K-E method).

The results of the experiments can be seen in Table 7, where the average accuracy achieved in each dataset and the accuracy standard deviation between test batches are shown. As can be observed, the LCA classification engine presented the worst results when considering the datasets in Table 7, indicating that using the *a posteriori* information of the classifiers in order to select the neighbors may lead to a poor estimation of the classifiers' competences under a concept drift scenario. This conclusion is reinforced by the results of the *A Posteriori* method, which also considers the *a posteriori* information of the classifiers', and was only the 5th best performing classification engine in the test.

On average, the weighted version of the Knora-Union method (K-UW) achieved better results than the unweighted (K-U) version. This result indicates that DCS methods that take into consideration the distance between the neighbors and the test instance when estimating the pool competence are more suitable for concept drift scenarios. This result is reinforced by the good performance of the A Priori method, which also uses weights. The weighted DCS used in Tsymbal et al. (2008), corroborate these findings. Note that the A Priori method is "less permissive" than the K-UW approach since only the best classifier is selected. This behavior seems to be beneficial in some scenarios, such as in a gradual always changing real concept drift environments (see the Checkerboard benchmark variants results).

The results in Table 7 also demonstrate that, in general, the DCS approach adapted using the Dynse framework, is able to handle the concept drifts with almost no parameter tuning (only M is adjusted according to the concept drift nature). This can be concluded by the good results achieved by the different DCS methods used as classification engines when compared with the Naive Combination method since the former does not implement any approach to explicitly adapt to a concept drift. When comparing classification engines that select a single classifier versus an ensemble of classifiers, the results do not indicate a clear best approach. However, when considering the best average rank, the K-E approach, which selects an ensemble of classifiers for each test instance, comes ahead.

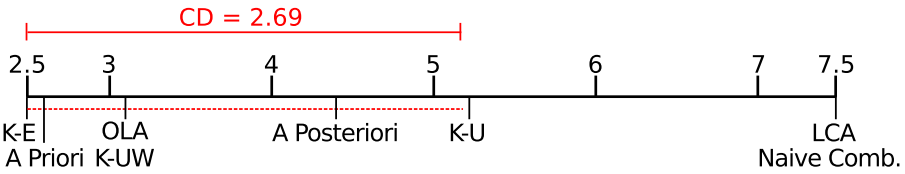


Fig. 8. Bonferroni-Dunn test at a $\alpha = 0.05$ showing the methods that are not significantly different from the K-E Classification Engine.

Table 8
Pairwise comparisons of the top 5 Classification Engines. (a) Comparison with the adjusted p-values using the Bergmann–Hommel procedure. (b) Comparison using the Wilcoxon Signed-Ranks test. No hypothesis were rejected for $\alpha = \{0.1, 0.05, 0.01\}$.

(a)		(b)	
Hypothesis	p_{Berg}	Hypothesis	$p_{Wilcoxon}$
K-E vs Posteriori	0.1182	K-E vs Posteriori	0.0117
Priori vs Posteriori	0.1182	Priori vs Posteriori	0.0146
OLA vs Posteriori	0.3254	OLA vs Posteriori	0.0156
K-UW vs Posteriori	0.3254	K-UW vs Posteriori	0.0532
Priori vs K-UW	1.0000	K-E vs OLA	0.1250
OLA vs Priori	1.0000	Priori vs K-UW	0.4639
K-E vs K-UW	1.0000	K-E vs Priori	0.4653
K-E vs OLA	1.0000	OLA vs Priori	0.8389
K-E vs Priori	1.0000	K-E vs K-UW	0.8340
OLA vs K-UW	1.0000	OLA vs K-UW	0.9775

In Fig. 8 a Bonferroni–Dunn test (Demšar, 2006) with $\alpha = 0.05$ shows the classification engines that are not significantly better than the K-E, where it is possible to verify that most classification engines have no significant difference to the K-E. The five best-ranked classification engines deemed as equivalent by the Bonferroni–Dunn test are further analyzed using pairwise comparisons, considering the hypothesis of equality between each pair of algorithms, using the Bergman–Hommel procedure[2] Bergmann and Hommel (1988); Cruz et al. (2018); Garcia and Herrera (2008), and the Wilcoxon Signed-Ranks test (Demšar, 2006), as suggested in Benavoli, Corani, and Mangili (2016). The comparison results are presented in Table 8, where no hypothesis was rejected at a $\alpha = \{0.1, 0.05, 0.01\}$ for the tests.

Some accuracy over time plots with the K-E as the classification engine for the Dynse framework, the Oracle and Naive Combination methods can be seen in Fig. 9. Note that when the change is abrupt, as in the STAGGER and SEA benchmarks (Figs. 9a and 9 b, respectively), the K-E can quickly recover from the concept drift. When the change is gradual (Fig. 9c) the K-E classification engine was able to keep a good accuracy over all time steps, and it was also able to adapt in the real world scenario presented (Fig. 9d). Besides the good results achieved, the plots in Fig. 9 indicate that some improvement can still be made due to the results of the Oracle method.

5.4. The pruning impact

In Sections 5.1, 5.2 and 5.3 it is showed that the DCS-based approach can adapt to concept drifts using a pool of "infinite size". Nevertheless, an infinite pool size is unfeasible under a real-world scenario, thus, in this section, the classical Age (remove the oldest classifier), Accuracy (remove the less accurate classifier considering the current accuracy estimation window) are used as pruning engines in the Dynse framework. The pruning strategies were configured to keep at most 25 classifiers in the pool during the tests. The tests results are compared with the "infinite pool" of classifiers.

[2] Implementation made by Garcia and Herrera (2008) available at http://sci2s.ugr.es/keel/multipleTest.zip .

Note that the infinite pool can be related to our previous work (Almeida et al., 2016), where the pruning module was not present, and it can be considered the most Concept Diverse (see Section 4.2.4) technique in the tested scenarios since classifiers trained under all past concepts are kept. In the tests the Dynse framework configured with the K-E considering $k = 5$ and $l = 0$ as classification engine is used. The accuracy estimation window size is set to 4 for the real concept drift scenarios ($M = 4$) and to 32 for the virtual concept drift tests ($M = 32$).

Table 9 contains the average accuracies achieved and the maximum memory consumed by the method (i.e. the peak of memory consumed considering all the steps) considering the discussed pruning strategies. As it can be observed in Table 9, the infinite pool generated the best accuracies in the majority of the considered scenarios. In Table 10 the accuracies of the pruning approaches are compared pairwise, considering the hypothesis of equality between each pair of pruning strategies, using the Bergman–Hommel and the Wilcoxon Signed-Ranks test (Bergmann & Hommel, 1988; Demšar, 2006; Garcia & Herrera, 2008). As one can observe, according to both the Bergman–Hommel procedure (Table 10a) and the Wilcoxon Signed-Ranks test (Table 10a), the infinite pool size is significantly better than the age pruned one at a $\alpha = 0.1$.

Besides the pool of infinite size being able to generate better results than the accuracy based pruning in most scenarios, the statistical tests in Table 10 did not encounter a significant difference. As an example of the impact of the pruning, consider the plot in Fig. 10a, where the accuracy over time plots for the infinite pool size, the accuracy and the age based prunings achieved the Digit virtual concept drift benchmark problem are showed. In Fig. 10a, it is demonstrated that the pruning strategies may remove some relevant information (especially the Accuracy Based pruning engine in the presented example), which can lead to a poor performance when the pruning starts.

Besides the good accuracy achieved, the infinite pool can obviously lead to an unpractical amount of resources consumption over time. This can be seen in Fig. 10b, which shows the amount of memory consumed at each time step in the SEA benchmark considering the infinite size pool and the age pruned pool (the accuracy-based pruning generated results close to the age-based pruning and it is not showed in the plot). As one can observe in Fig. 10b, when the pool reaches its maximum and the pruning starts, the amount of memory required stabilizes when using the age-based pruning. The infinite pool, on the other hand, continues to increase the number of classifiers in the pool over time, linearly increasing the amount of memory necessary at each time step, which would lead to an impractical amount of memory consumption in an infinite data stream.

In Table 9 it is possible to observe that, as expected, the amount of memory consumed by the pruned pools configurations are in general much smaller than the infinite pool (note that all benchmarks used are finite, thus it was possible to use the infinite size pool configuration without requiring an unfeasible amount of memory). The pruning approaches memory consumption in Table 9 are similar and, since no significant accuracies difference between the Age- and Accuracy-based pruning strategies

Table 9
Average accuracy achieved and maximum memory requested for different pruning strategies. The numbers in parenthesis indicate the standard deviation between testing batches. Best accuracies in bold.

Benchmark	Pruning engine					
	Infinite		Age prune		Acc. prune	
	Accuracy	Memory	Accuracy	Memory	Accuracy	Memory
STAGGER	**92.0%** (16.6)	0.24 MB	91.4% (17.4)	0.18 MB	91.7% (17.2)	0.18 MB
SEA	87.5% (1.4)	16.23 MB	87.6% (1.5)	2.77 MB	**87.8%** (1.6)	2.80 MB
SEARec	87.1% (1.8)	16.17 MB	**87.2%** (2.2)	2.77 MB	87.1% (2.2)	2.77 MB
CkrC	**85.6%** (3.1)	2.22 MB	83.9% (3.0)	0.29 MB	83.8% (3.1)	0.31 MB
CkrP	**86.5%** (4.9)	3.23 MB	84.6% (4.8)	0.29 MB	84.9% (5.0)	0.31 MB
CkrS	**86.2%** (5.1)	3.20 MB	84.3% (4.7)	0.29 MB	84.3% (4.8)	0.30 MB
CkrE	**85.3%** (3.3)	3.24 MB	83.4% (3.5)	0.30 MB	83.5% (3.6)	0.31 MB
Gauss	89.5% (5.7)	1.77 MB	89.6% (5.8)	0.34 MB	**89.7%** (5.7)	0.35 MB
Nebr	74.5% (1.0)	8.19 MB	74.1% (1.1)	0.53 MB	**74.7%** (0.9)	0.55 MB
For	**78.3%** (10.6)	20.67 MB	78.2% (10.8)	4.89 MB	78.2% (10.8)	4.96 MB
Dig	**76.8%** (20.5)	10.18 MB	74.3% (20.7)	8.21 MB	68.0% (17.3)	8.23 MB
Let	**66.5%** (14.0)	15.85 MB	60.3% (13.5)	4.62 MB	62.0% (12.7)	5.24 MB

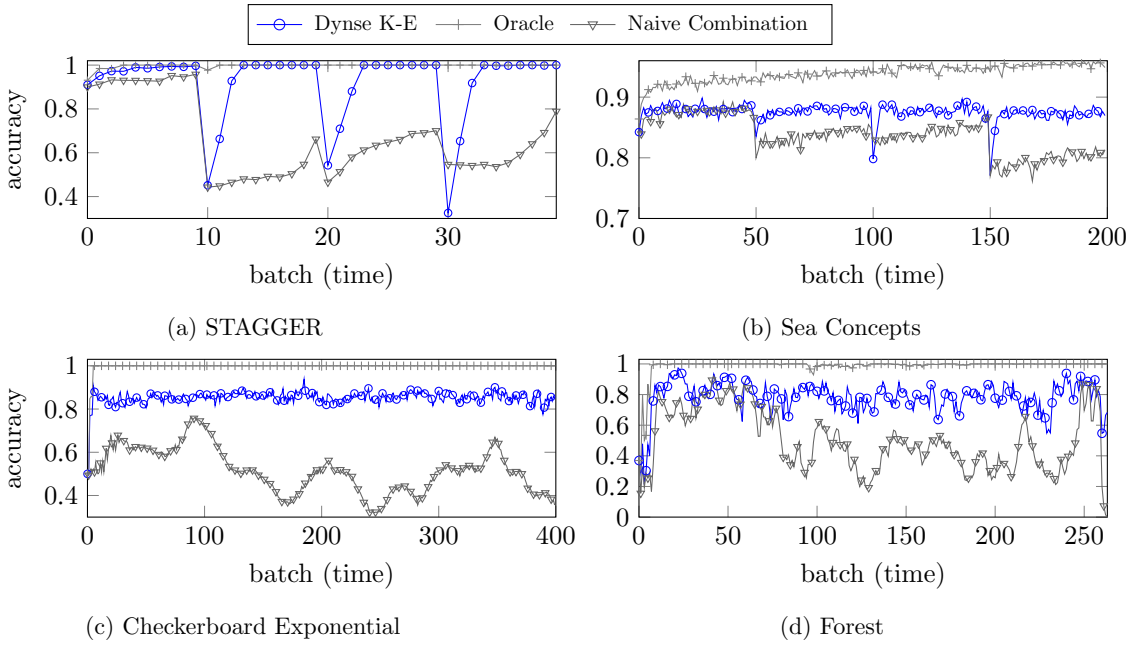
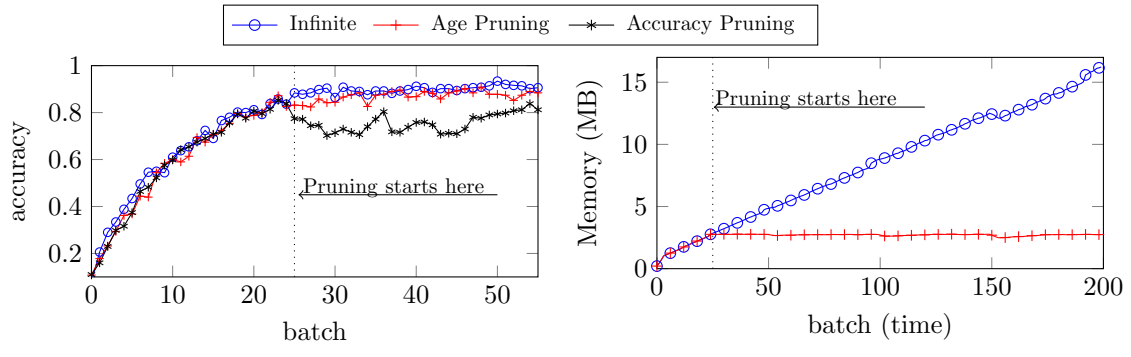


Fig. 9. Accuracy over time plots considering the K-E classification engine, the Oracle and the Naive Combination Methods.



(a) Accuracy over time plot in the Digit benchmark considering the infinite size pool and the pruning strategies.

(b) Memory usage over time in the SEA benchmark. The Accuracy Pruning is not showed since it lead to similar results compared to the Age Pruning.

Fig. 10. The impact of pruning in the accuracy (a) and memory (b).

Table 10
Pairwise comparisons of the prunings methods. (a) Comparison with the adjusted p-values using the Bergmann–Hommel procedure. (b) Comparison using the Wilcoxon Signed-Ranks test. Hypotheses that are rejected at an $\alpha = \{0.1, 0.05, 0.01\}$ are marked with a •, ••, and •••, respectively .

Hypothesis	p_{Berg}	Hypothesis	$p_{Wilcoxon}$
Infinite vs Age	•0.0963	Infinite vs Age	•0.0107
Infinite vs Accuracy	0.2207	Infinite vs Accuracy	0.0332
Age vs Accuracy	0.3583	Age vs Accuracy	0.2168

were found in Table 10, in this work, the Age-based pruning with a pool containing at most 25 classifiers ($D = 25$) will be considered as the default pruning, due to its simplicity and ability to keep the Dynse framework from indefinitely increasing the amount of memory needed.

5.5. Tests comparing to the state-of-the-art results

In this Section the Dynse framework is compared with some important state-of-the-art contributions, listed in Table 2. As specified in the beginning of this Section, the default configuration of the methods available in the MOA framework (Bifet et al., 2010) was used. A default configuration for the Dynse framework, built considering the results presented in Sections 5.2, 5.3 and 5.4, is given in Table 11.

In the tests, the SEA Concepts and the Digit problems were not considered, since these benchmarks were used to validate accuracy estimation window size in Section 5.2. Note that all benchmarks were used in the pruning engines test (Section 5.4), where the Age-based pruning was defined as the default method. We do not consider the tests in Section 5.4 a fine-tuning, since as we discussed, an "infinite pool size" may lead to better results, and the pruning is made only to save computational resources.

Table 12 summarizes the results of the experiments, where it can be observed the default configuration of the Dynse framework generated the best results in the majority of the tested scenarios. It is worth remembering, though, that the accuracy estimation window size was set to 32 in the virtual concept drift scenario, while the parameters in the state-of-the-art methods were kept the same. Albeit it can be considered a tuning, a wider accuracy estimation window size is a specification for the Dynse framework when facing a virtual concept drift problem (see Section 4.2.2), and the tested methods in the state-of-the-art does not specify any modification for this specific scenario.

In Table 12 it is also possible to verify that some state-of-the-art methods perform worse than the Naive Combination (*NaiveComb*) baseline method, which does not take any action to adapt to a concept drift. This may indicate that these methods require a careful tuning to achieve a good performance in some of the tested scenarios. The results in Table 12 indicate that methods that explicitly maintain an ensemble may be more suitable for the presented concept drift scenarios (the DDM and EDDM also used a pool of classifiers as the base learner, nevertheless these methods do not apply any rule in the pool in order to weight the classifiers or to selectively remove them when a concept drift is signaled). The results in Table 12 also indicate that in spite of some state-of-the-art methods surpassed the Dynse framework in some benchmarks, these methods may be more sensitive to the tuning of parameters or to the concept drift properties featured in the datasets. This conclusion becomes clear when the average rank is considered in Table 12 since the Dynse framework is the best-ranked method.

In Fig. 11 it is showed a Bonferroni–Dunn test with $\alpha = 0.05$, where it can be observed that the default configuration of the Dynse framework performs significantly better than seven of the state-of-the-art methods tested (and also better than the Naive Combination). One may argue that the classification engine (K-E) was chosen specifically to get the best results in the tests presented in Table 12 since the same datasets are present in the tests of Section 5.3. Nevertheless, as showed in Section 5.3, no significant difference between most classification engines was fount and, by switching the K-E by the K-U classification engine (which is significantly worse than the K-E), the Dynse would still be the best ranked method in Table 12, tied with the Leveraging Bagging with an average rank of 2.8.

The four best ranked methods, deemed as equivalent by the Bonferroni–Dunn test (Fig. 11), are further analyzed using pairwise comparisons using the Bergman–Hommel procedure and the Wilcoxon Signed-Ranks test (Benavoli et al., 2016; Bergmann & Hommel, 1988; Demšar, 2006; Garcia & Herrera, 2008) in Table 13, considering the hypothesis of equality between each pair of algorithms. As in the previous tests, hypotheses that are rejected at a $\alpha = \{0.1, 0.05, 0.01\}$ are marked with a •, ••, and •••, respectively. As one can observe, according to the Bergman–Hommel procedure (Table 13a) the Dynse generated accuracies significantly better than the AUE and ARF methods for $\alpha = 0.05$. No hypothesis was rejected for $\alpha = 0.01$ and the Wilcoxon Signed-Ranks test (Table 13b) did not found any significant difference for all confidence levels tested.

Since in most real-world scenarios it may be difficult to know *a priori* the exact properties of the concept drift, or to collect a relevant amount of data in order to fine-tune the methods, the results in Table 12 and the statistical tests in Fig. 11 and Table 13, indicate that a DCS-based approach can be considered in these scenarios due to its good performance without the need of a fine-tuning. The method does need to know, however, if the concept drift is real or virtual in order to define a big or small accuracy estimation window, as discussed in this work. To better visualize the behavior of the methods under the benchmarks presented in this Section, the accuracy over time plots for some of them are shown in Fig. 12. Only the default configuration of Dynse framework (the best performing approach), the Leveraging Bagging (the best performing state-of-the-art method) and the Naive Combination (baseline) methods are present in the plots of the Fig. 12.

As one can observe in Fig. 12a, the Dynse framework was able to recover faster than the Leveraging Bagging method under most of the concept drift regions of the SEA Concepts problem, although the Leveraging Bagging method achieved better results under some of the stable regions. In the Checkerboard Constant problem (Fig. 12b), which represents an "always drifting" scenario, the Dynse framework was able to keep its accuracy during the entire test, while the Leveraging Bagging presented several accuracy drops. Similar results were achieved in the Exponential version of the Checkerboard benchmark (Fig. 12c). The Nebraska Weather problem (Fig. 12d), on the other hand, represents a scenario where

Table 11
The proposed default configuraion of the Dynse framework.

Description	Var.	Proposed configuration/values
The *accuracy estimation window* size.	M	$M = 4$ for real concept drift scenarios and $M = 32$ for virtual concept drift scenarios.
The classification engine	CE	The K-E method considering $k = 5; l = 0$.
The pool maximum size	D	$D = 25$.
The pruning engine.	PE	The Age Pruning engine (remove the oldest).

Table 12
Artificial and real world benchmarks average accuracies (%), the default configuration of the Dynse framework and some state-of-the-art methods. Best results are in bold. Standard deviation between test batches are shown in parenthesis.

Bench.	Dynse	AUE	AWE	ADACC	DDM	EDDM	HAT	LevBag	OzaAD	OzaAS	ARFF	NaiveComb
Real concept drift benchmarks												
Stagger	91.4 (17.4)	93.7 (10.3)	97.0 (3.0)	**97.7** (5.2)	93.4 (7.1)	88.7 (16.3)	66.3 (20.1)	85.4 (20.3)	79.8 (20.8)	67.2 (19.1)	91.0 (16.5)	66.3 (17.6)
SEARec	**87.2** (2.2)	86.3 (1.5)	86.7 (1.8)	85.0 (1.7)	84.4 (3.6)	82.5 (4.5)	84.9 (2.6)	87.1 (2.4)	85.7 (2.2)	85.1 (2.2)	86.2 (2.8)	83.8 (3.2)
CkrC	**83.9** (3.0)	58.8 (7.5)	57.3 (7.2)	60.1 (7.9)	51.0 (9.4)	57.3 (8.6)	55.4 (9.4)	75.2 (6.4)	57.3 (7.6)	57.0 (13.1)	69.2 (6.2)	51.1 (8.6)
CkrP	**84.6** (4.8)	54.3 (7.2)	54.2 (7.9)	56.8 (7.7)	51.4 (8.8)	55.0 (6.8)	56.4 (9.4)	82.4 (11.8)	59.5 (12.4)	63.7 (13.6)	79.2 (13.8)	49.2 (8.8)
CkrS	**84.3** (4.7)	59.2 (9.4)	56.2 (7.6)	55.3 (6.7)	46.2 (9.5)	55.7 (7.1)	58.4 (11.6)	79.4 (10.6)	50.7 (8.6)	57.1 (10.9)	75.7 (9.9)	63.7 (18.9)
CkrE	**83.4** (3.5)	58.6 (7.8)	57.8 (7.6)	56.4 (6.9)	57.9 (8.9)	58.5 (7.8)	59.0 (8.7)	77.0 (7.9)	59.0 (7.8)	58.4 (10.4)	75.3 (7.6)	51.7 (10.2)
Gauss	89.61 (5.8)	90.5 (5.2)	90.4 (5.5)	89.2 (6.3)	**91.2** (7.3)	91.0 (7.0)	61.3 (23.6)	90.8 (5.5)	90.2 (5.3)	86.8 (9.0)	88.9 (7.3)	78.4 (13.3)
Real world benchmarks												
Nebr	74.1 (1.1)	73.7 (0.6)	73.8 (0.8)	71.7 (1.1)	71.1 (1.0)	67.7 (2.1)	72.3 (2.2)	77.0 (1.3)	73.4 (1.5)	73.5 (1.6)	**77.6** (2.2)	70.4 (1.1)
For	**78.2** (10.8)	73.7 (11.9)	66.4 (15.8)	64.1 (19.3)	51.9 (24.9)	46.2 (21.9)	67.2 (12.5)	75.1 (13.6)	49.7 (19.8)	74.2 (11.5)	72.1 (13.3)	68.8 (11.9)
Virtual concept drift benchmarks												
Let	**60.3** (13.5)	58.4 (12.1)	23.2 (10.1)	31.4 (12.6)	42.9 (17.4)	42.3 (15.2)	56.7 (11.3)	56.8 (12.2)	58.3 (11.5)	57.2 (11.1)	42.5 (9.8)	41.9 (16.0)
Avg. rank	**2.2**	4.7	7.0	7.7	8.6	8.4	8.1	3.0	7.0	6.8	4.6	10.0

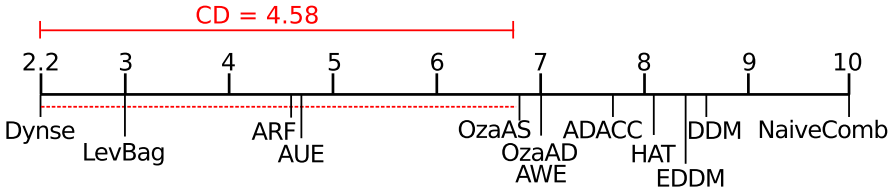


Fig. 11. Bonferroni-Dunn test with at a $\alpha = 0.05$ showing the methods that are not significantly different from the default configuration of the Dynse framework (i.e., the connected methods).

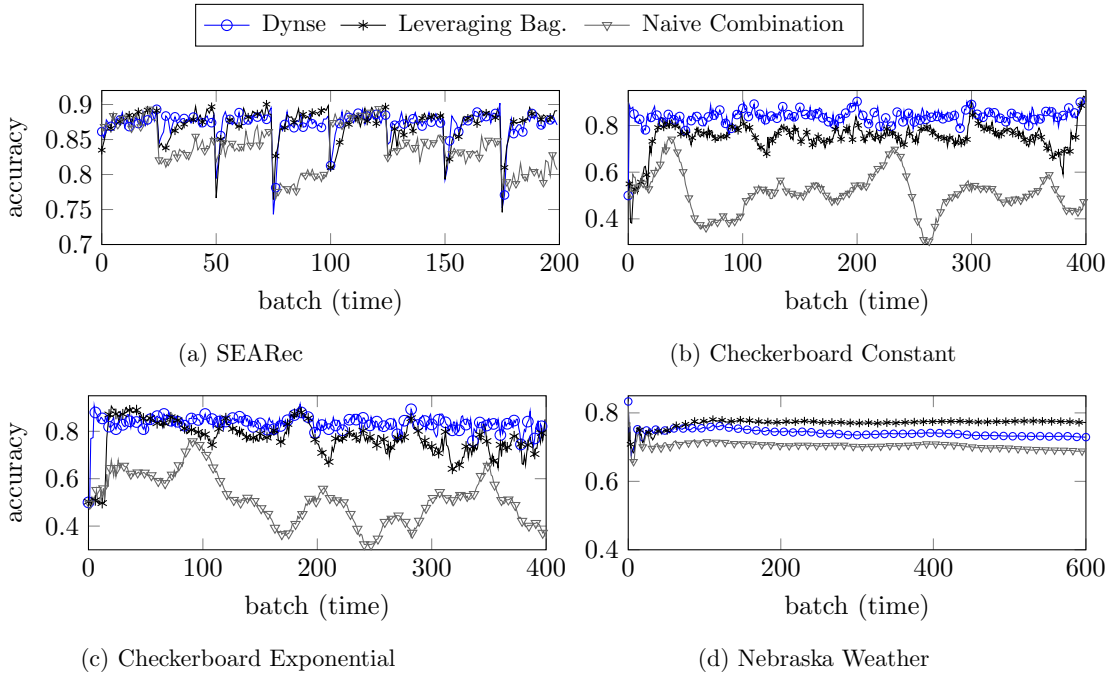


Fig. 12. The accuracy over time plots considering the default Dynse framework configuration, the Leveraging Bagging and Naive Combination methods under some benchmarks.

the Leveraging Bagging was the best performing method. Nevertheless, in the Nebraska Weather problem, the Naive Combination method did not suffer any sudden accuracy drop after the first steps, and its accuracy dropped relatively slowly over time. This behavior may indicate that this benchmark suffers from minor changes over time since even a method that is not able to deal with concept drifts can keep a reasonable accuracy over all time steps.

The amount of memory spent by the default Dynse configuration and by the Leveraging Bagging method is showed in Table 14 (the memory was measured as the maximum amount of memory required by the method considering all time steps), where it can be observed that, often, the Leveraging Bagging uses less memory than the Dynse framework. The Dynse framework requires more memory in some scenarios mainly due to the accuracy estimation window (W), since besides the pool, the Dynse must maintain some labeled batches in W. However, in the worst case scenario, the Dynse framework consumed only 4.96MB of memory (Forest Cover Type problem), which is a feasible amount of memory for most systems. Finally, we would like to stress that the

Table 13
Pairwise comparisons of the Dynse, LevBag, ARF and AUE methods. (a) Comparison using the Bergmann–Hommel procedure. (b) Comparison using the Wilcoxon Signed-Ranks test. Hypotheses that are rejected at a $\alpha = \{0.1, 0.05, 0.01\}$ are marked with a •, ••, and •••, respectively.

Hypothesis	p_{Berg}	Hypothesis	$p_{Wilcoxon}$
Dynse vs AUE	••0.0194	Dynse vs AUE	0.0371
Dynse vs ARF	••0.0194	Dynse vs ARF	0.0137
LevBag vs ARF	0.2498	LevBag vs ARF	0.0645
LevBag vs AUE	0.2498	LevBag vs AUE	0.0840
Dynse vs LevBag	0.4507	Dynse vs LevBag	0.0273
ARF vs AUE	0.8625	ARF vs AUE	0.3086

default version of all methods was used in these experiments. This means that it is possible to increase the performance of all of the presented approaches, by fine-tuning the components when relevant data is available. Nevertheless, our focus was to demonstrate that the DCS-based approach can be designed to deal with a range of concept drift scenarios using a set of default parameters.

5.6. Tests in the PKLot dataset

In this section we present the tests in the PKLot real world dataset (Almeida et al., 2015), which can be modeled as a concept drift scenario where new supervised information arrive in batches. The PKLot dataset contains 695,899 individual parking space images collected at different weather/light conditions in two different parking lots (UFPR and PUCPR). The dataset also contemplates a camera position change in a single parking lot (UFPR04 and UFPR05). In Almeida et al. (2015) it is demonstrated that even a "strong" classifier, trained using 50% of the instances of a given parking lot, may decrease significantly its accuracy when faced with a parking lot change. Other issues that may arise in this problem include luminosity and weather changes, camera movement and sporadic occlusions of individual parking spaces. Thus, in this work, we propose the following experimental protocol to use the PKLot dataset as a concept drift benchmark:

- The problem is defined as classifying each individual parking space as vacant or occupied.
- The LBP uniform (Ojala, Pietikainen, & Maenpaa, 2002) is defined as the feature set.
- Days containing less than 50 samples for each class (vacant or occupied) are not considered.
- The parking lots are presented in the following order: UFPR04, UFPR05 and PUCPR. The images collected in each parking lot are sorted in the chronological order and each day represent a time step. Thus, the time steps are: Day1_UFPR04, ..., Last_UFPR04, Day1_UFPR05, ..., Last_UFPR05, Day1_PUCPR, ..., Last_PUCPR. This configuration generates a camera position change (UFPR04 to UFPR05) and then a parking lot change (UFPR05 to PUCPR).
- At each time step, all instances of the current day must be classified. Also, at each time step (day) 50 samples from each class from the previous day are randomly selected for training. This configuration simulates a scenario where a human supervisor may label a small batch to update the classification system.

Table 15
Average accuracies in the PKLot benchmark. Parenthesis indicate the standard deviation between test batches.

Method	Avg. acc.
Dynse	**92.2% (8.7)**
LevBag	89.7% (13.1)
OzaAD	88.6% (11.1)
OzaAS	88.3% (10.9)
AUE	87.1% (14.9)
ARFF	86.7% (16.2)
DDM	85.5% (10.2)
EDDM	85.4% (10.3)
HAT	84.5% (10.4)
AWE	84.0% (18.2)
NaiveComb	81.4% (11.7)
ADACC	73.9% (27.2)

Table 15 shows the average accuracy of the Dynse framework and the state-of-the-art methods in the PKLot benchmark considering the proposed protocol. As in Section 5.5, we considered the default configuration for all methods, including the Dynse framework (we considered the real concept drift configuration, thus $M = 4$). As one can observe, the Dynse framework achieved the best results when considering the average accuracy, followed by the Leveraging Bagging state-of-the-art method.

Fig. 16 shows the accuracy over time plots for the Dynse framework (the most accurate method), the Leverage Bagging (the best state-of-the-art method) and the Naive Combination (baseline) methods, where it can be observed that both the Dynse and the Leveraging Bagging approaches were able to maintain a relatively stable accuracy over the time steps, and even increase its accuracies after parking lot changes. In Almeida et al. (2015), it was discussed that one of the major challenges in the PKLot dataset is the change between different parking lots. By using methods capable to deal with concept drift scenarios, we showed that this challenge can be overcome. The complete protocol for using the PKLot as a concept drift benchmark, including the files with the LBP features already extracted and correctly ordered, is fully available in Almeida et al. (2018).

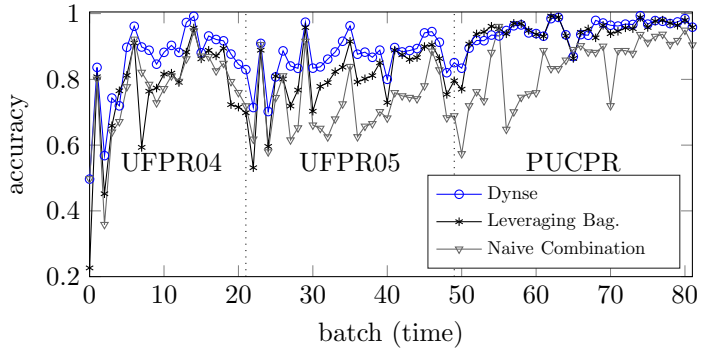
6. Conclusion and future work

In this work, we demonstrate that through some modifications, the DCS approach can be a powerful tool to deal with the concept drift phenomenon, especially under scenarios where some areas do not change between concepts (local changes). Although under a static environment, a DCS approach may be only local-region dependent, we showed that under a concept drift scenario this dependency alone is not sufficient, thus a time dependency must also be incorporated. We modeled this time dependency as a time window that keeps the latest labeled data received in the validation dataset, which is our answer to the question "How do we keep track of changes" when using a DCS-based approach. The size of this window must be defined according to the nature of the concept drift, where a small window should be used in a real concept drift scenario, and a window containing as much labeled data as possible should be defined in a concept drift that affects only $P(\mathbf{x})$.

Table 14
Amount of memory (MB) used by the Dynse framework and the LevBag methods.

	STAGGER	SEARec	CkrC	CkrP	CkrS	CkrE	Gauss	Nebr	For	Let
Dynse	0.18	2.77	**0.29**	**0.31**	**0.30**	**0.31**	0.35	**0.53**	4.96	5.24
LevBag	**0.15**	**0.90**	0.30	0.40	0.42	0.42	**0.22**	1.40	**1.44**	**0.42**

Table 16
PKLot benchmark accuracy over time plots. Only the two most accurate methods and the Naive Combination approach are shown.



A discussion about the neighborhood size is also presented in this work, where we argue that bigger sets of neighbors may impact the classifier competence estimation under a real concept drift negatively. We also discuss that, differently from most approaches used to deal with concept drifts, a DCS-based method may benefit from a pool containing classifiers trained under different concepts, since the neighborhood can help us answer the question "Which learned information (classifiers) is useful in the current scenario?".

The Dynse framework presented in Almeida et al. (2016) was extended in this work in order to accommodate a module responsible for keeping the pool from increasing in size indefinitely, which we called the *Pruning Engine*. Furthermore, we showed that the Dynse framework can take into consideration both the time and local dependency of instances in order to dynamically select the custom ensemble for test instances. We also proposed a novel experimental protocol to use the PKLot dataset as a real world concept drift benchmark, and made both the Dynse framework and the PKLot required files (including the extracted features and ordered according to the proposed protocol) publicly available, which may serve as a basis for researchers who intend to explore the DCS-approach for concept drift environments. Through an extensive range of experiments we validated the discussions presented in this work and also demonstrated that a DCS-based approach (the Dynse framework) is able to maintain a good performance in a range of problems without much tuning of its parameters (it is only necessary to know if the concept drift is real or virtual), which is advantageous in many concept drift scenarios since it may be impossible to know *a priori* all the drift properties present in the environment or to collect relevant data in order to fine tune the method before deploying it.

The results presented in this work indicate that taking into consideration the distance of each neighbor when selecting the custom ensemble for the test instance may lead to better results in some scenarios, as we observed in the results achieved by the A Priori and K-UW methods. As expected, most DCS methods used as classification engines in the Dynse framework were capable to deal with the concept drifts present in the benchmark problems used in this work. Nevertheless, the LCA method achieved a poor performance when compared to the other DCS methods, indicating that taking into consideration the *a posteriori* information of the classifiers in order to select the neighbors may be a suboptimal solution for dealing with the concept drift problem through a DCS-based approach. When compared with the state-of-the-art methods, the default configuration of the Dynse framework was the best-ranked method, and also the most stable, achieving good results in all tested benchmarks.

In the experiments, we assessed an Age and Accuracy-based pruning strategies. When compared to the pruned pools, the pool of "infinite size" generated the best results for most tested benchmarks, indicating that by maintaining classifiers specialized under different feature regions and trained at different concepts may lead to a better result, although no significant difference was found between the infinite size pool and the accuracy-based pruning. This result reinforces the discussion about the concept diversity we presented in this work, and as a future work, we intend to develop pruning strategies that take the concept diversity idea into consideration.

Also as a future work, we intend to develop and test methods of maintaining the accuracy estimation window with a fixed size, covering the biggest and most relevant area of the feature space possible, to deal with virtual concept drifts. Finally, we also plan to explore alternatives to maintaining the accuracy estimation window up to date with the current concept under a real concept drift scenario, such as replacing the sliding window approach by a trigger-based one, which could even remove the need to know the type of concept drift (real or virtual) *a priori*, in order to define this window size.

References

Alippi, C., Boracchi, G., & Roveri, M. (2013). Just-in-time classifiers for recurrent concepts. *IEEE Transactions on Neural Networks and Learning Systems, 24*(4), 620–634. doi:10.1109/TNNLS.2013.2239309.

Almeida, P. R., Oliveira, L. S., Britto, A. S., Silva, E. J., & Koerich, A. L. (2015). PKLot – a robust dataset for parking lot classification. *Expert Systems with Applications, 42*(11), 4937–4949. doi:10.1016/j.eswa.2015.02.009.

Almeida, P. R. L. d., Oliveira, L. S. d., Britto, A. d. S., & Sabourin, R. (2016). Handling concept drifts using dynamic selection of classifiers. In *IEEE international conference on tools with artificial intelligence* (pp. 989–995). doi:10.1109/ICTAI.2016.0153.

Almeida, P. R. L. d., Oliveira, L. S. d., Britto, A. d. S., & Sabourin, R. (2018). *Dynse – dynamic selection based drift handler*. https://web.inf.ufpr.br/vri/software/dynse

Alpaydin, E., & Kaynak, C. (1998). Cascading classifiers. *Kybernetika, 34*, 369–374.

Baena-Garcıa, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavalda, R., & Morales-Bueno, R. (2006). Early drift detection method. In *International workshop on knowledge discovery from data streams: 6* (pp. 77–86).

Benavoli, A., Corani, G., & Mangili, F. (2016). Should we really use post-hoc tests based on mean-ranks? *Journal of Machine Learning Research, 17*(5), 1–10.

Bergmann, B., & Hommel, G. (1988). Improvements of general multiple test procedures for redundant systems of hypotheses. In P. Bauer, G. Hommel, & E. Sonnemann (Eds.), *Multiple Hypothesenprüfung/Multiple Hypotheses Testing: Symposium, November 1987: vol. 6–7* (pp. 100–115). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-52307-6_8.

Bifet, A. (2017). Classifier concept drift detection and the illusion of progress. In L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. A. Zadeh, & J. M. Zurada (Eds.), *Artificial intelligence and soft computing: 16th international conference, ICAISC 2017, Zakopane, Poland, June 11–15, 2017, Proceedings, Part II* (pp. 715–725)). Cham: Springer International Publishing. doi:10.1007/978-3-319-59060-8_64.

Bifet, A., & Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. *SIAM international conference on data mining*.

Bifet, A., & Gavaldà, R. (2009). Adaptive learning from evolving data streams. In N. Adams, C. Robardet, A. Siebes, & J.-F. Boulicaut (Eds.), *Advances in intelligent*

data analysis viii. In *Lecture notes in computer science: 5772* (pp. 249–260). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-03915-7_22.

Bifet, A., Holmes, G., Kirkby, R., & Pfahringer, B. (2010a). MOA: Massive online analysis. *Journal of Machine Learning Research, 11*, 1601–1604.

Bifet, A., Holmes, G., & Pfahringer, B. (2010). Leveraging bagging for evolving data streams. In J. L. Balcázar, F. Bonchi, A. Gionis, & M. Sebag (Eds.), *Machine learning and knowledge discovery in databases*. In *Lecture notes in computer science: 6321* (pp. 135–150). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-15880-3_15.

Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., & Gavaldà, R. (2009). New ensemble methods for evolving data streams. In *International conference on knowledge discovery and data mining, KDD '09* (pp. 139–148). New York, NY, USA: ACM. doi:10.1145/1557019.1557041.

Britto, A. S., Jr., Sabourin, R., & Oliveira, L. E. (2014). Dynamic selection of classifiers—A comprehensive review. *Pattern Recognition, 47*(11), 3665–3680. doi:10.1016/j.patcog.2014.05.003.

Brzeziński, D., & Stefanowski, J. (2011). Accuracy updated ensemble for data streams with concept drift. In E. Corchado, M. Kurzyński, & M. Woźniak (Eds.), *Hybrid artificial intelligent systems*. In *Lecture notes in computer science: 6679* (pp. 155–163). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-21222-2_19.

Cavalcante, C. R., Minku, L. L., & Oliveira, L. I. A. (2016). FEDD: Feature extraction for explicit concept drift detection in time series. In *International joint conference on neural networks* (pp. 740–747).

Chan, P., Zhang, Q.-Q., Ng, W., & Yeung, D. (2011). Dynamic base classifier pool for classifier selection in multiple classifier systems. In *International conference on machine learning and cybernetics: 3* (pp. 1093–1096). doi:10.1109/ICMLC.2011.6016933.

Chen, K., Koh, Y. S., & Riddle, P. (2016). Proactive drift detection: Predicting concept drifts in data streams using probabilistic networks. In *International joint conference on neural networks* (pp. 780–787). doi:10.1109/IJCNN.2016.7727279.

Cruz, R. M., Sabourin, R., & Cavalcanti, G. D. (2018). Dynamic classifier selection: Recent advances and perspectives. *Information Fusion, 41*(Suppl. C), 195–216. doi:10.1016/j.inffus.2017.09.010.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research, 7*, 1–30.

Didaci, L., Giacinto, G., Roli, F., & Marcialis, G. L. (2005). A study on the performances of dynamic classifier selection based on local accuracy estimation. *Pattern Recognition, 38*(11), 2188–2191. doi:10.1016/j.patcog.2005.02.010.

Ditzler, G., Roveri, M., Alippi, C., & Polikar, R. (2015). Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine, 10*(4), 12–25. doi:10.1109/MCI.2015.2471196.

Domingos, P., & Hulten, G. (2000). Mining high-speed data streams. In *International conference on knowledge discovery and data mining, KDD '00* (pp. 71–80). New York, NY, USA: ACM. doi:10.1145/347090.347107.

Elwell, R., & Polikar, R. (2011). Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks, 22*(10), 1517–1531. doi:10.1109/TNN.2011.2160459.

Escovedo, T., Da Cruz, A. V. A., Vellasco, M. M. B. R., & Koshiyama, A. S. (2013). Learning under concept drift using a neuro-evolutionary ensemble. *International Journal of Computational Intelligence and Applications, 12*(04), 1340002. doi:10.1142/S1469026813400026.

Frey, P. W., & Slate, D. J. (1991). Letter recognition using Holland-style adaptive classifiers. *Machine Learning, 6*(2), 161–182. doi:10.1007/BF00114162.

Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with drift detection. In A. C. Bazzan, & S. Labidi (Eds.), *Advances in artificial intelligence*. In *Lecture notes in computer science: 3171* (pp. 286–295). Berlin, Heidelberg: Springer. doi:10.1007/978-3-540-28645-5_29.

Gama, J. a., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys, 46*(4), 44:1–44:37. doi:10.1145/2523813.

Garcia, S., & Herrera, F. (2008). An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research, 9*(December), 2677–2694.

Giacinto, G., & Roli, F. (1999). Methods for dynamic classifier selection. In *International conference on image analysis and processing* (pp. 659–664). doi:10.1109/ICIAP.1999.797670.

Gomes, H. M., Barddal, J. P., Enembreck, F., & Bifet, A. (2017). A survey on ensemble learning for data stream classification. *ACM Computing Surveys, 50*(2), 23:1–23:36. doi:10.1145/3054925.

Gomes, H. M., Bifet, A., Read, J., Barddal, J. P., Enembreck, F., Pfharinger, B., et al. (2017). Adaptive random forests for evolving data stream classification. *Machine Learning, 106*(9), 1469–1495. doi:10.1007/s10994-017-5642-8.

Gonçalves, P. M., de Carvalho Santos, S. G., Barros, R. S., & Vieira, D. C. (2014). A comparative study on concept drift detectors. *Expert Systems with Applications, 41*(18), 8144–8156. doi:10.1016/j.eswa.2014.07.019.

González-Castro, V., Alaiz-Rodríguez, R., & Alegre, E. (2013). Class distribution estimation based on the Hellinger distance. *Information Sciences, 218*(0), 146–164. doi:10.1016/j.ins.2012.05.028.

Hoens, T., Polikar, R., & Chawla, N. (2012). Learning from streaming data with concept drift and imbalance: An overview. *Progress in Artificial Intelligence, 1*(1), 89–101. doi:10.1007/s13748-011-0008-0.

Hung-Ren Ko, A., Sabourin, R., & de Souza Britto, A. (2007). K-nearest oracle for dynamic ensemble selection. In *Ninth international conference on document analysis and recognition, 2007. ICDAR 2007.: 1* (pp. 422–426). doi:10.1109/ICDAR.2007.4378744.

Jaber, G., Cornuéjols, A., & Tarroux, P. (2013). A new on-line learning method for coping with recurring concepts: The ADACC system. In M. Lee, A. Hirose, Z.-G. Hou, & R. M. Kil (Eds.), *Neural information processing* (pp. 595–604). Berlin, Heidelberg: Springer.

Jian-guang, H., Xiao-feng, H., & Jie, S. (2010). Dynamic financial distress prediction modeling based on slip time window and multiple classifiers. In *International conference on management science and engineering* (pp. 148–155). doi:10.1109/ICMSE.2010.5719798.

Jordaney, R., Sharad, K., Dash, S. K., Wang, Z., Papini, D., Nouretdinov, I., & Cavallaro, L. (2017). *Transcend: Detecting concept drift in malware classification models*.

Kapp, M. N., Sabourin, R., & Maupin, P. (2011). A dynamic optimization approach for adaptive incremental learning. *International Journal of Intelligent Systems, 26*(11), 1101–1124. doi:10.1002/int.20501.

Karnick, M., Ahiskali, M., Muhlbaier, M. D., & Polikar, R. (2008). Learning concept drift in nonstationary environments using an ensemble of classifiers based approach. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)* (pp. 3455–3462). doi:10.1109/IJCNN.2008.4634290.

Ko, A. H., Sabourin, R., & Britto, A. S., Jr. (2008). From dynamic classifier selection to dynamic ensemble selection. *Pattern Recognition, 41*(5), 1718–1731. doi:10.1016/j.patcog.2007.10.015.

Kolter, J. Z., & Maloof, M. A. (2007). Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research, 8*, 2755–2790.

Krawczyk, B., Minku, L. L., Gama, J., Stefanowski, J., & Woźniak, M. (2017). Ensemble learning for data stream analysis: A survey. *Information Fusion, 37*, 132–156. doi:10.1016/j.inffus.2017.02.004.

Krawczyk, B., & Wonźiak, M. (2014). One-class classifiers with incremental learning and forgetting for data streams with concept drift. *Soft Computing*, 1–14. doi:10.1007/s00500-014-1492-5.

Kuncheva, L. (2004). Classifier ensembles for changing environments. In F. Roli, J. Kittler, & T. Windeatt (Eds.), *Multiple classifier systems*. In *Lecture notes in computer science: 3077* (pp. 1–15). Berlin, Heidelberg: Springer. doi:10.1007/978-3-540-25966-4_1.

Kuncheva, L. I., & Žliobaitė, I. (2009). On the window size for classification in changing environments. *Intelligent Data Analysis, 13*(6), 861–872.

Lichman, M. (2013). *UCI machine learning repository*. http://archive.ics.uci.edu/ml

Markou, M., & Singh, S. (2003). Novelty detection: A review-part 1: Statistical approaches. *Signal Processing, 83*(12), 2481–2497. doi:10.1016/j.sigpro.2003.07.018.

Minku, L., White, A., & Yao, X. (2010). The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering, 22*(5), 730–742. doi:10.1109/TKDE.2009.156.

Minku, L., & Yao, X. (2012). DDD: A new ensemble approach for dealing with concept drift. *IEEE Transactions on Knowledge and Data Engineering, 24*(4), 619–633. doi:10.1109/TKDE.2011.58.

Ojala, T., Pietikainen, M., & Maenpaa, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 24*(7), 971–987. doi:10.1109/TPAMI.2002.1017623.

Pan, S., Wu, K., Zhang, Y., & Li, X. (2010). Classifier ensemble for uncertain data stream classification. In M. Zaki, J. Yu, B. Ravindran, & V. Pudi (Eds.), *Advances in knowledge discovery and data mining*. In *Lecture notes in computer science: 6118* (pp. 488–495). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-13657-3_52.

Pan, S., Zhang, Y., & Li, X. (2012). Dynamic classifier ensemble for positive unlabeled text stream classification. *Knowledge and Information Systems, 33*(2), 267–287. doi:10.1007/s10115-011-0469-2.

Pérez-Gállego, P., Quevedo, J. R., & del Coz, J. J. (2017). Using ensembles for problems with characterizable changes in data distribution: A case study on quantification. *Information Fusion, 34*, 87–100. doi:10.1016/j.inffus.2016.07.001.

Polikar, R., Krause, S., & Burd, L. (2003). Ensemble of classifiers based incremental learning with dynamic voting weight update. In *International joint conference on neural networks: 4* (pp. 2770–2775). doi:10.1109/IJCNN.2003.1224006.

Radtke, P. V., Granger, E., Sabourin, R., & Gorodnichy, D. O. (2014). Skew-sensitive boolean combination for adaptive ensembles—An application to face recognition in video surveillance. *Information Fusion, 20*(0), 31–48. doi:10.1016/j.inffus.2013.11.001.

Rakitianskaia, A., & Engelbrecht, A. (2012). Training feedforward neural networks with dynamic particle swarm optimisation. *Swarm Intelligence, 6*(3), 233–270. doi:10.1007/s11721-012-0071-6.

Raza, H., Cecotti, H., & Prasad, G. (2016). A combination of transductive and inductive learning for handling non-stationarities in motor imagery classification. In *International joint conference on neural networks* (pp. 763–770).

Schlimmer, J., & Granger, R., Jr. (1986). Incremental learning from noisy data. *Machine Learning, 1*(3), 317–354. doi:10.1007/BF00116895.

Sethi, T. S., Kantardzic, M., & Hu, H. (2016). A grid density based framework for classifying streaming data in the presence of concept drift. *Journal of Intelligent Information Systems, 46*(1), 179–211. doi:10.1007/s10844-015-0358-3.

Siahroudi, S. K., Moodi, P. Z., & Beigy, H. (2018). Detection of evolving concepts in non-stationary data streams: A multiple kernel learning approach. *Expert Systems with Applications, 91*(Suppl. C), 187–197. doi:10.1016/j.eswa.2017.08.033.

Street, W. N., & Kim, Y. (2001). A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the international conference on knowledge discovery and data mining, KDD '01* (pp. 377–382). New York, NY, USA: ACM. doi:10.1145/502512.502568.

Sun, J., & Li, H. (2011). Dynamic financial distress prediction using instance selection for the disposal of concept drift. *Expert Systems with Applications, 38*(3), 2566–2576. doi:10.1016/j.eswa.2010.08.046.

Tsymbal, A., Pechenizkiy, M., Cunningham, P., & Puuronen, S. (2006). Handling local concept drift with dynamic integration of classifiers: Domain of antibiotic resistance in nosocomial infections. In *IEEE international symposium on computer-based medical systems* (pp. 679–684). doi:10.1109/CBMS.2006.94.

Tsymbal, A., Pechenizkiy, M., Cunningham, P., & Puuronen, S. (2008). Dynamic integration of classifiers for handling concept drift. *Information Fusion, 9*(1), 56–68. doi:10.1016/j.inffus.2006.11.002.

Wang, H., Fan, W., Yu, P. S., & Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the international conference on knowledge discovery and data mining, KDD '03* (pp. 226–235). New York, NY, USA: ACM. doi:10.1145/956750.956778.

Widmer, G., & Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning, 23*(1), 69–101. doi:10.1007/BF00116900.

Woods, K., Bowyer, K., & Kegelmeyer, J. W. P. (1996). Combination of multiple classifiers using local accuracy estimates. In *Computer society conference on computer vision and pattern recognition* (pp. 391–396). doi:10.1109/CVPR.1996.517102.

Zhu, X., Wu, X., & Yang, Y. (2004). Dynamic classifier selection for effective mining from noisy data streams. In *IEEE international conference on data mining* (pp. 305–312). doi:10.1109/ICDM.2004.10091.