Engineering Applications of Artificial Intelligence 65 (2017) 336–345

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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

Adaptive learning for dynamic environments: A comparative approach

Joana Costa [a,b,*], Catarina Silva [a,b], Mário Antunes [a,c], Bernardete Ribeiro [b]

[a] School of Technology and Management, Polytechnic Institute of Leiria, Portugal
[b] Center for Informatics and Systems (CISUC), Department of Informatics Engineering, University of Coimbra, Portugal
[c] Center for Research in Advanced Computing Systems (CRACS), INESC-TEC University of Porto, Portugal

ARTICLE INFO

Keywords:
Dynamic environments
Ensembles
Learn++.NSE
Twitter

ABSTRACT

Nowadays most learning problems demand adaptive solutions. Current challenges include temporal data streams, drift and non-stationary scenarios, often with text data, whether in social networks or in business systems. Various efforts have been pursued in machine learning settings to learn in such environments, specially because of their non-trivial nature, since changes occur between the distribution data used to define the model and the current environment.

In this work we present the Drift Adaptive Retain Knowledge (DARK) framework to tackle adaptive learning in dynamic environments based on recent and retained knowledge. DARK handles an ensemble of multiple Support Vector Machine (SVM) models that are dynamically weighted and have distinct training window sizes. A comparative study with benchmark solutions in the field, namely the Learn++.NSE algorithm, is also presented. Experimental results revealed that DARK outperforms Learn++.NSE with two different base classifiers, an SVM and a Classification and Regression Tree (CART).

1. Introduction

Streaming sources are becoming ubiquitous. Ranging from data generated by sensors on the Internet of Things (IoT) to social media platforms increasingly accessed with mobile devices, such deluge of data streams is becoming one of the greatest challenges in terms of learning and information extraction (Huijse et al., 2014). Hence, nowadays most learning problems demand dynamic models, adaptive to new circumstances as they emerge. Paradigmatic to this setting are social networks as Twitter, where new information appears all the time. Albeit we can undoubtedly benefit from all these data, one major drawback of such overflow is the inability to easily perceive important, significant and accurate information. This challenge arises not only because the amount of data is overwhelming to process, but also because time plays an important role by fast out-dating information (Costa et al., 2016).

To handle such challenges of dynamic environments we have to address some innovative models that are able to deal with models ageing as, so far, the deployed models performance is reduced because they are not able to deal with dynamic environments.

Additionally, drifts can have different patterns and thus must be treated differently. The most significant types of drifts are depicted in Fig. 1, namely sudden, gradual, incremental and reoccurring (Zliobaite, 2010). Sudden drift is present when the occurring rate of the drift is high and a concept appears or disappears abruptly. Although it is mostly stated as sudden or abrupt drift, it can also be referred as concept change. Gradual drift is characterized by a low drift rate and occurs when the probability of a given context to be associated with a concept increases or decreases during a certain period of time. Additionally, the probability to be associated with another context increases proportionally. Incremental drift can be considered as a subgroup of gradual drift, through the main difference is that the change between the two concepts is much slower and only perceived when looking to what is occurring during longer periods of time. Reoccurring drift occurs when a previously active concept reappears after a period of time. It is important to refer that although it appears seasonally its periodicity must be unknown, otherwise the core assumption of the uncertainty about the future could be compromised.

Different approaches have been pursued with the above goals, like ensemble systems for classification problems (Kuncheva, 2004), proposed and discussed in this work. We present the DARK framework, Drift Adaptive Retain Knowledge framework, that uses an ensemble of Support Vector Machines with dynamic weighting schemes and variable training window sizes for text classification scenarios. A comparative

* Corresponding author.
E-mail addresses: joana.costa@ipleiria.pt, joanamc@dei.uc.pt (J. Costa), catarina@ipleiria.pt, catarina@dei.uc.pt (C. Silva), mario.antunes@ipleiria.pt, mantunes@dcc.fc.up.pt (M. Antunes), bribeiro@dei.uc.pt (B. Ribeiro).

http://dx.doi.org/10.1016/j.engappai.2017.08.004
Received 5 March 2017; Received in revised form 13 June 2017; Accepted 7 August 2017
Available online 1 September 2017
0952-1976/

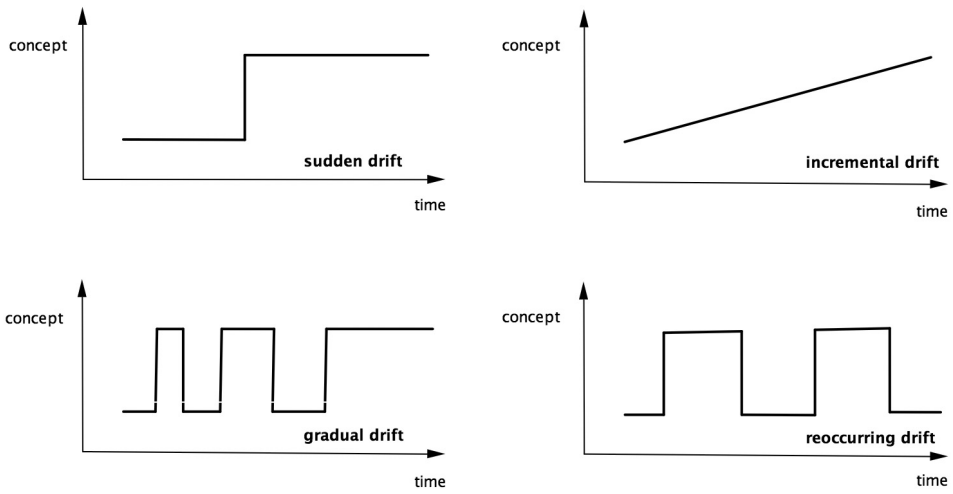

Fig. 1. Different types of drift.

study with benchmark solution in the field is also put forward and the experimental results attest the potential of DARK, as it outperforms both Learn++.NSE with two different base classifiers, an SVM and a Classification and Regression Tree (CART).

There are tree main contributions in this paper: to infer about the influence of recent examples for the overall learning and classification performances; to validate the DARK framework with text classification scenarios, by applying it to text datasets based on Twitter social network public stream and present a comparative study with benchmark solutions in the field, namely Learn++.NSE algorithm.

The rest of the paper is organized as follows. Section 2 presents active and passive approaches for handling drift in dynamic environments. Section 3 defines and details the proposed DARK framework. In Section 4 we introduce the experimental setup, including the Twitter case study and a description of Learn++.NSE. Section 5 presents and discusses the obtained results. Finally, we address conclusions and future lines of research in Section 6.

2. Approaches for drift detection, adaptation and learning

Different approaches exist for learning in nonstationary environments that can be casted as active or passive approaches, that are described and summarized in Table 1.

2.1. Active approaches

Active approaches for learning in nonstationary and dynamic environments are used to detect changes in the environment and react adaptively, updating or building a new classifier. Features are extracted for change detection and, once a change is detected, the classifier model is updated or rebuilt by discarding the obsolete knowledge and adapting to the new environment. The whole process involves change detection and adaptation methods (Ditzler et al., 2015).

Change detection approaches inspect extracted features and variations in the underlying distribution data using theoretically-grounded statistical techniques and include (Ditzler et al., 2015):

1. *Hypothesis Tests* assess the validity of a hypothesis by controlling the false positive rate in change detection based on predetermined confidence calculations and using statistical techniques. The confidence threshold can be based on the mean value with which a set of samples has been drawn from a specific distribution as in Patist (2007) and Nishida and Yamauchi (2007);
2. *Change-Point Methods* use a fixed data sequence to verify if a given sequence contains a change-point, by analysing all possible partitions of the data stream. This statistical technique is highly computational bounded, nevertheless it has the ability to detect the presence of a change and estimate the instant where the change occurred, as in Ross et al. (2011);
3. *Sequential Hypothesis Tests* inspect sequentially incoming examples, one at a time, until there are enough examples to determine the presence of a change or not. Some examples of this technique are probability ratio test (Wald, 1992) and repeated significance test (Armitage, 1960);
4. *Change Detection Tests* overcome limitations of the previous technique by sequentially analysing the statistical behaviour of data streams. This method consists on a change detection based on a threshold as in Harel et al. (2014); Haque et al. (2015). The limitation of this method is the difficulty to set the threshold to an optimal value with which we may have a reasonable classification performance.

The **Adaptation** phase occurs after a change in environment is observed and detected. It consists on adapting the classifier to the change by learning from the new available information and discarding the obsolete (Gama et al., 2014). Adaptation mechanisms can be grouped into the following three main categories (Ditzler et al., 2015):

- *Windowing* is the most used and easiest mechanism. It is based on a sliding window that includes, at each given moment, the most recent and up-to-date examples, while the obsolete ones are discarded. With this mechanism the up-to-date examples are used to retrain the classifier and thus enhance its performance for the next batch(es). The choice of the appropriate window length is a critical issue and can itself be adaptively calculated (Alippi et al., 2013, 2012; Bifet and Gavalda, 2007) or determined by the expected change ratio (Alippi and Roveri, 2008; Cohen et al., 2008b). Just-In-Time (JIT) adaptive classifier, a new generation of adaptive classifiers that are able to operate in nonstationary environments is proposed in Alippi and Roveri (2008).
- *Weighting*, unlike windowing mechanisms, takes into account all the examples weighted according to some rule, like their age or relevancy with respect to the recent classification accuracy performance (Koychev, 2000). Several approaches can be found in the literature regarding the weighting mechanisms used: gradual forgetting (Koychev, 2000); time-based weighting (Datar and Motwani, 2016), change index which measures the variation of data processing over time (Alippi et al., 2009); and based on the accuracy/error calculated in the last batch of supervised data (Klinkenberg, 2004);

Table 1
Summary of comparison between active and passive approaches.

Category	Type	Approaches	References
Active	Change detection	Based on theoretically-grounded statistical techniques: hypothesis tests, change-point methods, sequential hypothesis tests, and change detection tests.	Patist (2007), Nishida and Yamauchi (2007), Ross et al. (2011), Wald (1992), Armitage (1960), Harel et al. (2014), Haque et al. (2015).
	Adaptation	Adapting the classifier by learning from the newly avai lable information and discarding the obsolete one. Main mechanisms: windowing, weighting and random sampling.	Alippi et al. (2013), Alippi et al. (2012), Bifet and Gavalda (2007), Alippi and Roveri (2008), Cohen et al. (2008b), Koychev (2000), Datar and Motwani (2016), Alippi et al. (2009), Klinkenberg (2004), Vitter (1985).
Passive	Single models	Applied in big data scenarios. Paradigmatic models include decision trees (DT), very fast decision trees (VFDT). Other approaches include extreme learning machine (ELM).	Domingos and Hulton (2000), Hulten et al. (2001), Liu et al. (2009); Cohen et al. (2008a), Ye et al. (0000).
	Ensemble models	Extremely appropriate for learning in dynamic environments. Reduction in the variance of the error and attain the flexibility. Approaches include boosting, bagging or random forests.	Freund and Schapire (1997), Breiman (1996), Breiman (2001), Bagul and Phulpagar (2016), Ditzler and Polikar (2013), Tabassum and Ahmed (2016), Ren et al. (2016), Jr. (2011), Elwell and Polikar (2011), Karnick et al. (2008).

- *Sampling*, more precisely reservoir sampling (Vitter, 1985), uses randomization and is able to select a subset of unique examples from the data stream.

Active approaches in dynamic and nonstationary environments, like those related with change detection in temporal data streams, can be easily observed in some real-world applications like network intrusion (Kim et al., 2007; Haque and Alkharobi, 2015) and spam detection (Ahsan et al., 2016; Lughofer and Mouchaweh, 2015).

2.2. Passive approaches

Passive approaches, unlike active approaches, do not aim at detecting the presence of changes or drifts in the environment, but assures a natural path of continuous adaptation of the model parameters every time new data arrive (Ditzler et al., 2015). The complexity of such adaptation methods varies, but the main goal is to keep the final model as close as the state of reality brought by current data. Passive approaches can be divided in single models and ensemble models:

Single models are constituted by only one model, presenting a lower computational burden that is often appropriate for massive data streams. As a consequence, less complex models, e.g. decision trees (DT) can be used, in fact, decision trees are the mostly common classifiers used for data stream mining with the very-fast decision tree (VFDT) learner being one of the most popular (Domingos and Hulton, 2000). A sliding-window approach was also proposed to take different options into consideration at each tree node split (Hulten et al., 2001; Liu et al., 2009).

Other examples of single models include a fuzzy-logic based approach that also exploits a sliding window over the training data stream (Cohen et al., 2008a) and an online extreme learning machine (ELM) combined with a time-varying neural network for learning from nonstationary data (Ye et al., 0000).

Ensemble models are one of the best researched methods for adaptive learning in dynamic environments. Ensembles of classifiers integrate multiple classifiers and are based on the idea that, given a task that requires expert knowledge, k experts (baseline classifiers) may perform better than one, if their individual judgements are appropriately combined. A classifier committee is then characterized by (i) a choice of k classifiers, and (ii) a choice of a combination function, sometimes denominated a voting algorithm. The classifiers should be as independent as possible to guarantee a large number of inductions on the data. Using different classifiers to exploit diverse patterns of errors to make the ensemble better than just the sum (or average) of the parts, we can obtain a gain from synergies between the ensemble classifiers.

The simplest combination function is just a majority voting mechanism with an odd number of baseline classifiers. However, other more advanced strategies have been pursued. In Kuncheva (2002), a theoretical study on six classifier fusion strategies is presented. The authors analyse the classification error for different fusion methods: average, minimum, maximum, median, majority vote, and oracle.

There are many approaches for ensemble of classifiers, such as boosting (Freund and Schapire, 1997), bagging (Breiman, 1996), or random forests (Breiman, 2001), but their original form is usually applied in static environments. However, ensembles are specially adequate to tackle with dynamic evolving settings, given their modular nature. Ensembles are cutting-edge solutions to many different learning challenges and different researchers have been studying ensembles and their applications in various fields (Tabassum and Ahmed, 2016; Ditzler and Polikar, 2013; Ren et al., 2016; Jr., 2011; Bagul and Phulpagar, 2016). In Elwell and Polikar (2011) and Karnick et al. (2008) two approaches of incremental learning of concept drift in nonstationary environments are presented. The authors describe ensemble-based approaches of classifiers for incrementally learning from new data drawn from a distribution that changes in time and generate a new classifier using each additional dataset that becomes available from the changing environment. A popular batch-based learning algorithm for non-stationary environments is Learn++.NSE (NSE for NonStationary Environments) (Polikar et al., 2001) that will be detailed further later in this paper.

3. DARK framework

This section describes the Drift Adaptive Retain Knowledge (DARK) framework. We will firstly describe its aim, and then detail its characteristics.

We are focusing on dynamic environments in text classification scenarios, where the model must adapt to deal with changes usually dependent of hidden contexts. One of the major challenges is the amount of data, specially when dealing with streams. It is sometimes infeasible to store all the previously seen data, although it may carry substantial information for future use.

The goal of DARK framework is to build an ensemble of Support Vector Machines (SVM) with dynamic weighting schemes and variable train size windows for model adaptation in incremental learning, and therefore effectively learn in dynamic environments in text classification scenarios.

The rationale of DARK is to use ensembles of classifiers to integrate multiple experts with different characteristics and thus benefit from their multitude, specially as they are created in different moments. Due to its modular structure, which enables temporal adaptation to new incoming examples on the basis of the data sampling real distribution over time, we have a built-in memory mechanism that is inherited from the (recent) past, and thus we can achieve an improvement in classification performance that otherwise would be dependent on more examples and computational burden.

Fig. 2 depicts the DARK framework. The framework can be divided in three parts, from top to bottom: (i) models' construction; (ii) learning process and (iii) models' combination.

The construction of the models (i) is carried out by defining time-windows and learning models for each time-window. Different settings

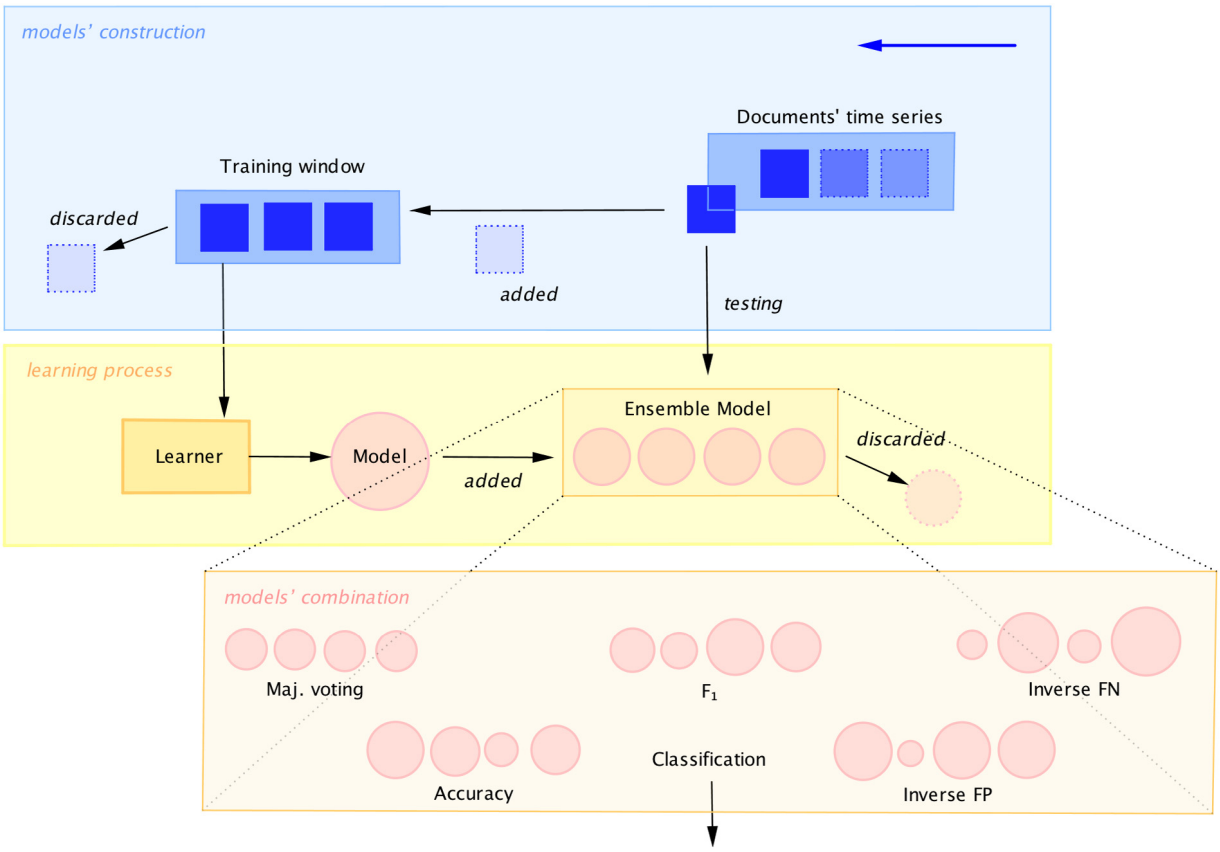

Fig. 2. DARK framework (training window size = 3 and size of ensemble = 4).

can be constructed, i.e., the examples that are considered in each time-window depend on the specific approach. The timestamp of the example can be used, but other approaches may consider its relevance though the choice of the best samples (Costa et al., 2016).

In order to perceive the importance of past examples in the classification process we use a batch learning strategy that retains previously seen examples during a prefixed period. So, we aim to evaluate for how long it is relevant to keep information according to the different types of drift. This is the first memory mechanism that is present in our framework.

Algorithm 1 defines the basic steps to create one single base classifier. For each collection of documents $\mathcal{T}$ in a time-window t, $\mathcal{T}^t = \{x_1, \ldots, x_{|\mathcal{T}^t|}\}$ with labels $\{y_1, \ldots, y_{|\mathcal{T}^t|}\} \rightarrow -1, 1$, and by considering the training window size j, the dataset $\mathcal{D}^t$ is updated incrementally until the training window size is complete. By updating the documents collection $\mathcal{D}^t$ based on a time window we retain the information during a defined amount of time, discarding the examples that occur before that moment. A base classifier $\mathcal{C}^t$ is then trained with $\mathcal{D}^t$ and can be used for classification purposes.

The learning process (ii) focuses on the definition of the k baseline classifiers according to this algorithm. Notice that, in dynamic environments, the ensemble must adapt to deal with changes usually dependent on hidden contexts, and it is usually infeasible to store all previously seen data, although it may carry substantial information for future use. Hence, not all previously constructed models are kept in the ensemble and, the learning process determines which should be kept (or added) and which should be discarded (Costa et al., 2015b, 2014).

Algorithm 2 presents the DARK framework learning process. Besides the base classifier model creation that is detailed in Algorithm 1, we also have the combination of models in the ensemble. Firstly, for each time-window t we add the base classifier $\mathcal{C}^t$ to the ensemble $\mathcal{E}^t$. Secondly, as with examples, we also consider that models can be outdated, so the

Algorithm 1: Base classifier model creation.

Input:
For each collection of documents $\mathcal{T}$ in a time window t,
$\mathcal{T}^t = \{x_1, \ldots, x_{|\mathcal{T}^t|}\}$ with labels $\{y_1, \ldots, y_{|\mathcal{T}^t|}\} \rightarrow -1, 1 \quad t = 1, 2, \ldots$

Training window size j

```
for i = 1, 2, ... j do
    D^t ← D^t ∪ T^{t−j}
    i++
end
Classifier C^t : Learn (D^t), obtain: h^t: X → Y
```

ensemble $\mathcal{E}^t$ is pruned so that all classifiers $\mathcal{C}^{t-k}$ that match the condition $t - k > 0$ are discarded and not included in the ensemble final decision. Finally, we use the ensemble $\mathcal{E}^t$ to classify the document collection $\mathcal{T}^{t+1}$. The purpose of the classification is to define the unknown mapping function $e^t : \mathcal{X} \rightarrow \mathcal{Y}$, that predicts the class label y_i, according to x_i, the document message. In a timeline perspective, e^t uses the historical data $\{x_1, \ldots, x_t\}$ to predict x_{t+1} by combining the unknown prediction function $h^t : \mathcal{X} \rightarrow \mathcal{Y}$ provided by each base classifier model that composes the ensemble. The prediction function e^t can use different combining strategies for the output h^t of each classifier in the model's combining phase (iii), e.g. a majority voting strategy, where $e^t = \frac{\sum_t h^t(\mathcal{T}^{t+1})}{|\sum_t h^t(\mathcal{T}^{t+1})|}$. Other combining schemes use performance metrics as we will mention further, which account for each individual contribution of the classifier in previous time windows.

Algorithm 2: DARK Framework learning process.

Input:
For each collection of documents $\mathcal{T}$ in a time window t,
$\mathcal{T}^t = \{x_1, \ldots, x_{|T^t|}\}$ with labels
$\{y_1, \ldots, y_{|T^t|}\} \rightarrow \{-1, 1\}$ $\quad t = 1, 2, \ldots$

Training window size j

Ensemble size k

for $t = 1, 2, \ldots T$ **do**
  for $i = 1, 2, \ldots j$ **do**
    $\mathcal{D}^t \leftarrow \mathcal{D}^t \cup \mathcal{T}^{t-j}$
    $i++$
  end
  Classifier C^t : Learn ($\mathcal{D}^t$), obtain: h^t: $\mathcal{X} \rightarrow \mathcal{Y}$
  Ensemble $\mathcal{E}^t \leftarrow C^t$
  if $t - k > 0$ **then**
    $\mathcal{E}^t \leftarrow \mathcal{E}^t \setminus C^{t-k}$
  end
  Ensemble $\mathcal{E}^t$: Classify ($\mathcal{T}^{t+1}$) using: e^t: $\mathcal{X} \rightarrow \mathcal{Y}$
end

4. Experimental setup

This section describes the experimental setup used in the comparative study. We will firstly present the Twitter case study and then proceed by detailing the Drift Oriented Tool System (DOTS), a framework developed to create drift text-based datasets. We then describe the dataset created to evaluate the framework along with the representation and preprocessing and detail the classifiers used in this comparative study. Finally, the performance metrics used to evaluate the framework are described.

4.1. Social networks: twitter case study

Twitter stream is a paradigmatic example of a text-based application where drift phenomena commonly occur. *Twitter* is a micro-blogging service where users post text-based messages up to 140 characters, also known as *tweets*. It is also considered one of the most relevant social networks, along with *Facebook*, as millions of users are connected to each other by a following mechanism that allows them to read each others posts.

Twitter is also responsible for the popularization of the concept of *hashtag*. An *hashtag* is a single word started by the symbol "#" that is used to classify the message content and to improve search capabilities. Besides improving search capabilities, *hashtags* have been identified as having multiple and relevant potentialities, like promoting the phenomenon described in Huang et al. (2010) as *micro-meme*, i.e., an idea, behaviour or style that spreads from person to person within a culture (Dic, 2012). By tagging a message with a trending topic identified by a certain *hashtag*, a user expands the audience of the message, compelling more users to express themselves about the subject (Zappavigna, 2011).

Considering the importance of the *hashtag* in Twitter, it is relevant to study the possibility of evaluating message contents to predict its *hashtag*. If we can classify a message based on a set of *tweets-hashtags*, we are able to suggest an *hashtag* for a given *tweet*, bringing a wider audience into discussion (Johnson, 2009), spreading an idea (Tsur and Rappoport, 2012), get affiliated with a community (Yang et al., 2012), or bringing together other Internet resources (Chang, 2010).

The classification of Twitter messages can be described as a multi-class problem that can be cast as a time series of tweets. It consists of a continuous sequence of instances, in this case, Twitter messages, represented as $\mathcal{X} = x_1, \ldots, x_t$, where x_1 is the first occurring instance and x_t the latest. Each instance occurs at a time, not necessarily in equally spaced time intervals, and is characterized by a set of features, usually words, $\mathcal{W} = w_1, w_2, \ldots, w_{|\mathcal{W}|}$. Consequently, the instance x_i is represented by the feature vector $w_{i1}, w_{i2}, \ldots, w_{i|\mathcal{W}|}$.

If x_i is a labelled instance it can be represented by the pair (x_i, y_i), where $y_i \in \mathcal{Y} = \{y_1, y_2, \ldots, y_{|\mathcal{Y}|}\}$ is the class label for instance x_i.

In Costa et al. (2013) we defined semantic *hashtags*, where the Twitter message *hashtag* is used to label the content of the message. In other words, it means that y_i represents the *hashtag* that labels the Twitter message x_i.

4.2. DOTS: drift oriented tool system

DOTS is a drift oriented framework (Costa et al., 2015a) developed to dynamically create textual datasets with drift.

The main purpose of DOTS framework is to represent drift patterns in a text-based dataset. Therefore, the input of the DOTS framework is two-fold: text documents and a frequency table, as depicted in Fig. 3. Each text document file represents the documents of the same class and the frequency table is used to define the drift patterns. A major characteristic of DOTS is the possibility of defining the exact time, more precisely the exact time window, where each document appears, being thus possible to define time drifts. This is done by the frequency table that is a mandatory input of the DOTS framework (see Fig. 3). The main idea is to use the frequency to reproduce drifts.

The frequency table must be in the CSV format and each row corresponds to a time instance. As stated above, it is not important whether a time instance represents a minute, an hour, or a day, but it is assumed that all of them correspond to the same amount of time. The first row contains the identification of the class, and each cell of all the other rows contain the number of documents of a given class that occur in a given time instance. Fig. 4 depicts a task being added to the framework, in which the first time-window has, respectively, one, three and four documents of the classes `nfl`, `jobs` and `android`.

The DOTS framework includes preprocessing methods commonly used in text processing, such as stopwords removal and stemming, that will be described further in the paper. The framework also includes two important stemming algorithms: the Porter algorithm (Willett, 2006) and Krovetz algorithm (Krovetz, 1993).

It is also possible to define the weighting scheme used to represent each word of a document, that is the weight of each feature of a document. Two weighting schemes were defined, namely term frequency (*tf*) and term frequency-inverse document frequency, commonly known as *tf-idf*. Considering the setup configuration, DOTS will create a word index, the INDRI index, provided by INDRI API, from the Lemur Project.[1] It allows users to use different strategies of filtering and analysing data. By outputting an INDRI index, DOTS provides all the features presented by the INDRI project, a powerful query interface, that provides state-of-the-art text search, field retrieval and text annotation. It is also possible to define multiple training window sizes and multiple export file formats. The training window size will define in each time-window how many previous time-windows will be taken into account, as this is important to test learning models with memory capabilities. For instance, to perceive for how long it is relevant to keep previously gathered information and how that can affect the learning model capabilities. By exporting in multiple file formats, DOTS allows for creating datasets that can be used in different classification frameworks, like `SVM Light` and `Weka`. Three output formats were implemented: Comma-Separated Values (CSV) file format, the Attribute-Relation File Format (ARFF) used in the widely used `WEKA` software, and the `SVM Light` file

[1] http://www.lemurproject.org/.

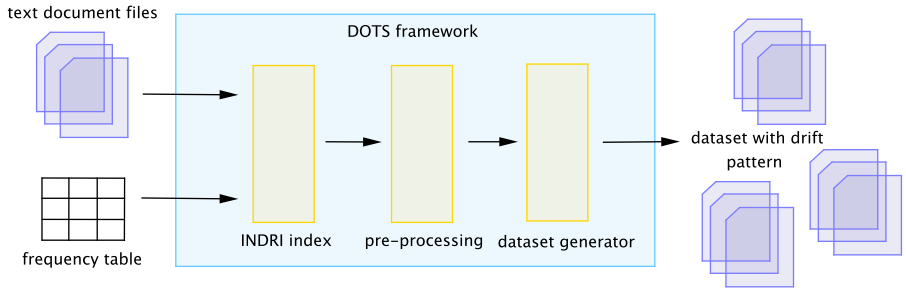

Fig. 3. The DOTS framework.

Table 2
Example of frequency count of considered drifts: Sudden#1, Sudden#2, Gradual#1, Gradual#2, Incremental#1, Incremental#2, Reoccurring, Normal#1, Normal#2, Normal#3.

Timewindow	Sudden#1	Sudden#2	Gradual#1	Gradual#2	Incremental#1	Incremental#2	Reoccurring	Normal#1	Normal#2	Normal#3
1	0	0	0	60	0	60	0	20	50	20
2	0	0	10	60	0	60	0	20	50	50
3	0	0	10	0	0	60	0	20	50	20
4	0	0	0	0	0	60	0	20	50	50
5	0	200	0	50	0	60	0	20	50	20
6	0	200	20	50	4	56	50	20	50	50
7	0	200	20	0	8	52	50	20	50	20
8	0	200	0	0	12	48	50	20	50	50
9	0	200	0	40	16	44	0	20	50	20
10	0	200	30	40	20	40	0	20	50	50
11	0	0	30	0	24	36	0	20	50	20
12	0	0	0	0	28	32	0	20	50	50
13	0	0	0	30	32	28	0	20	50	20
14	0	0	40	30	36	24	50	20	50	50
15	500	0	40	0	40	20	50	20	50	20
16	500	0	0	0	44	16	50	20	50	50
17	500	0	0	20	48	12	0	20	50	20
18	0	0	50	20	52	8	0	20	50	50
19	0	0	50	0	56	4	0	20	50	20
20	0	0	0	0	60	0	0	20	50	50
21	0	0	0	10	60	0	0	20	50	20
22	0	0	60	10	60	0	50	20	50	50
23	0	0	60	0	60	0	50	20	50	20
24	0	0	0	0	60	0	50	20	50	50

format. The versatility of using the concept of training windows extends the framework, as users can define the previous amount of data for each time-window. By defining different training window sizes, memory characteristics can be cast into the training models, thus inducing a storage mechanism. Additionally, one can also test the performance of a given learning model if the examples of more than one time window are used as training set, which can be seen as the importance of previously seen examples in future classifications. As it is often relevant to define various testing settings, DOTS permits adding tasks using INI files. These are structured files with `''key=value''` pairs, that contain the definition of multiple tasks. By using an INI file as input, users are able to define more than one task at a single time, thus optimizing the time spent on task setup.

Finally, DOTS is a simple and easy to use freeware application with a friendly interface as shown in Fig. 4. The application and a complete tutorial can also be download from http://dotspt.sourceforge.net/.

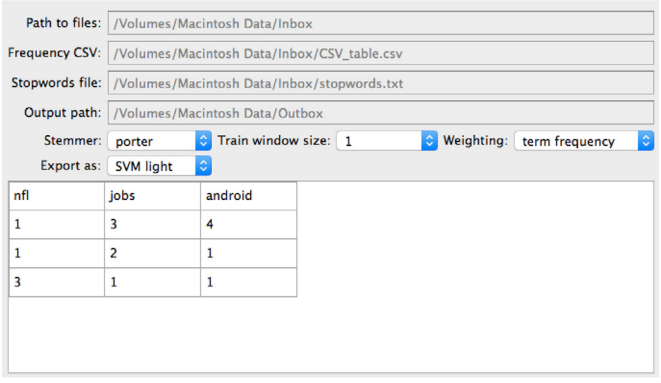

Fig. 4. DOTS interface.

4.3. Dataset

The dataset uses 10 different *hashtags* that represent the different drifts, based on the assumption that they correspond mutually exclusive concepts, like *#realmadrid* and *#android*. By trying to use mutually exclusive concepts we intent to avoid misleading a classifier, as two different *tweets* could represent the same concept. To obtain a relevant number of tweets, and consequently diversity, we have chosen trending *hashtags* like *#syrisa* and *#airasia*. Table 3 shows the chosen *hashtags* and the corresponding drift they represent. This correspondence was arbitrarily done and does not correspond to any possible occurrence in the real Twitter environment, since as stated above, no information is known about the occurrence of drifts in Twitter.

The Twitter API[2] was then used to request public *tweets* that contain the defined *hashtags*. The requests have been carried out between 28 December 2014 and 21 January 2015 and *tweets* were only considered if the user language was English. We have requested more than 75 000 *tweets* for the given *hashtags*, even though some of them were discarded, like for instance those containing no message content besides the *hashtag*. The *hashtag* was then removed from the message content to be exclusively

[2] https://dev.Twitter.com/.

Table 3
Mapping between type of drift and hashtag.

Drift	Hashtag
Sudden #1	#syrisa
Sudden #2	#airasia
Gradual #1	#isis
Gradual #2	#bieber
Incremental #1	#android
Incremental #2	#ferrari
Reoccurring	#realmadrid
Normal #1	#jobs
Normal #2	#sex
Normal #3	#nfl

used as the document label. The *tweets* matching these constraints were considered labelled and suited for classification purposes, and were used by their appearing order in the public feed.

We have simulated the different types of drift by artificially defining timestamps to the previously gathered *tweets*. Time is represented as 100 continuous time windows, in which the frequency of each *hashtag* is altered in order to represent the defined drifts. Each *tweet* is then timestamped so it belongs to one of the defined time windows. Due to space constraints we refrain to present herein the complete table with the frequency of each *hashtag* in each time window but, for better understanding, we have included an example in Table 2 so we can explain how can we use it to simulate drift patterns. It contains 24 continuous time-windows instead of the used 100. In the given example Sudden #1 is represented by the appearance of 500 *tweets* with the *hashtag #syrisa* in each time window from 15 to 17, and in any of the other time windows this *hashtag* appear. Differently from *Sudden #1*, *Sudden #2* is represented with only 200 *tweets* with the *hashtag #airasia* in each time windows from 5 to 10, thus simulating a softer occurring drift, but with a more long-standing appearance. By making both concepts disappear, in time windows, 18 and 11, respectively, we can also simulate the opposite way of the proposed sudden drift (Zliobaite, 2010). Our final dataset contains 34 240 *tweets*.

4.4. Representation and preprocessing

A *tweet* is represented as one of the most commonly used document representation, which is the vector space model, also known as *Bag of Words*. The collection of features is built as the dictionary of unique terms present in the documents collections. Each tweet of the document collection is indexed with the *bag* of the terms occurring in it, i.e., a vector with one element for each term occurring in the whole collection. The weighting scheme used to represent each term is the *term frequency-inverse document frequency*, also know as *tf-idf*.

The use of vector space model can cause computational problems due to high dimensional space, and overfitting can also occur, which can prevent the classifier to generalize and thus the prediction ability becomes poor. In order to reduce feature space preprocessing methods were applied. These techniques aim at reducing the size of the document representation and prevent the mislead classification as some words, such as articles, prepositions and conjunctions, called *stopwords*, are non-informative words, and occur more frequently than informative ones. An english-based stopword dictionary was used, but *Twitter* related words like "*r* t" or "*http*" were also considered as they can be seen as *stopwords* in the *Twitter* context. *Stopword removal* was then applied, preventing those non informative words from misleading the classification.

Stemming method was also applied. This method consists in removing case and inflection information of each word, reducing it to the word stem. Stemming does not alter significantly the information included, but it does avoid feature expansion.

4.5. Classifiers

We are focusing on learning models that can cope with dynamic environments, where models must adapt to deal with changes usually dependent of hidden contexts. We will present two classifiers that are used in our experiments: Support Vector Machines and Learn++.NSE. Support Vector Machines constitute currently the best of breed kernel-based technique, exhibiting state-of-the-art performance in diverse application areas, such as text classification (Joachims, 2002; Tong and Koller, 2002), and are used as the base classifier of both DARK and Learn++.NSE. To perform the comparative approach of DARK with other frameworks, we will use Learn++.NSE (Elwell and Polikar, 2011), which will be detailed further.

Support vector machines

Support Vector Machines (SVM) is a machine learning method introduced by Vapnik (1999), based on his Statistical learning Theory and Structural Risk Minimization Principle. The underlying idea behind the use of SVM for classification, consists on finding the optimal separating hyperplane between the positive and negative examples. The optimal hyperplane is defined as the one giving the maximum margin between the training examples that are closest to it. Support vectors are the examples that lie closest to the separating hyperplane. Once this hyperplane is found, new examples can be classified simply by determining on which side of the hyperplane they are.

The output of a linear SVM is $u = \mathbf{w} \times \mathbf{x} - b$, where $\mathbf{w}$ is the normal weight vector to the hyperplane and $\mathbf{x}$ is the input vector. Maximizing the margin can be seen as an optimization problem:

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\|\mathbf{w}\|^2, \\ &\text{subjected to} && y_i(\mathbf{w}.\mathbf{x} + b) \geq 1, \forall i, \end{aligned} \tag{1}$$

where $\mathbf{x}$ is the training example and y_i is the correct output for the ith training example. Intuitively the classifier with the largest margin will give low expected risk, and hence better generalization.

To deal with the constrained optimization problem in (1) Lagrange multipliers $\alpha_i \geq 0$ and the Lagrangian (2) can be introduced:

$$L_p \equiv \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{l} \alpha_i (y_i(\mathbf{w}.\mathbf{x} + b) - 1). \tag{2}$$

Learn++.NSE

Learn++.NSE (NSE stands for Non Stationary Environments) (Elwell and Polikar, 2011) is one of the most relevant passive solutions for learning in the presence of drift. In Elwell and Polikar (2011) it is stated that it is not only able to learn in the presence of drift without explicit knowledge, but also that Learn++.NSE can accommodate a wide variety of drift settings, regardless of the drift nature, which is not common in most learning algorithms that try to deal with dynamic environments.

Learn++.NSE is then a popular batch-based algorithm that trains one new classifier for each batch of data it receives, and combines these classifiers using a dynamically weighted majority voting. According to its authors, the novelty of the approach is based on determining the voting weights, for each classifier's time-adjusted accuracy on current and past environments. It aims to act accordingly to the changes in underlying data distributions, as well as to a possible reoccurrence of an earlier distribution. In Learn++.NSE all classifiers are maintained and reweighted on the most recent training data, and thus the learning time increases linearly.

Learn++.NSE software implementation for *Matlab*, with a classification and regression tree (CART) base classifier, was made available by authors at the website http://github.com/gditzler/IncrementalLearning.

Table 4
Contingency table for binary classification.

	Class positive	Class negative
Assigned positive	a (True positives)	b (False positives)
Assigned negative	c (False negatives)	d (True negatives)

4.6. Performance metrics

The evaluation of our approach was carried out with the previously described dataset and using the Support Vector Machine (SVM). Notwithstanding it is a multi-class problem in its essence, it can be decomposed into multiple binary tasks in a one-against-all binary classification strategy. In this case, a classifier h^t is composed by $|Y|$ binary classifiers. In order to evaluate a binary decision task we first define a contingency table representing the possible outcomes of the classification, as shown in Table 4.

Several measures have been defined based on this table, such as, error rate ($\frac{b+c}{a+b+c+d}$), recall ($R = \frac{a}{a+c}$), and precision ($P = \frac{a}{a+b}$), as well as combined measures, such as, the van Rijsbergen F_β measure (van Rijsbergen, 1979), which combines recall and precision in a single score:

$$F_\beta = \frac{(\beta^2 + 1)P \times R}{\beta^2 P + R}. \quad (3)$$

F_β is one of the best suited measures for text classification used with $\beta = 1$, i.e. F_1, a harmonic average between precision and recall (4), since it evaluates well unbalanced scenarios that usually occur in text classification settings and particularly in text classification in the Twitter environment.

$$F_1 = \frac{2 \times P \times R}{P + R}. \quad (4)$$

Considering the proposed approach, the fact that we are working with a time series, and that we use a one-against all strategy, we will have a classifier for each batch of the time series that is composed by $|Y|$ binary classifiers, being $|Y|$ the collection of possible labels. To perceive the performance of the classification for each drift pattern, we will consider all the binary classifiers that were created in all the time series batches. To evaluate the performance obtained across time, we will average the obtained results. Two conventional methods are widely used, specially in multi-label settings, namely macro-averaging and micro-averaging. Macro-averaged performance is obtained by computing the scores for each learning model in each batch of the time series and then averaging these scores to obtain the global means. Differently, micro-averaged performance is computed by summing all the previously introduced contingency matrix values (*a,b,c and d*), and then use the sum of these values to compute a single micro-averaged performance that represents the global performance.

As mentioned above, performance metrics can also be used as the combining strategies for the output of each classifier in an ensemble, which account for each individual contribution of the classifier in previous time windows.

5. Experimental results and analysis

In this section we evaluate the performance obtained with the Twitter data set using the approach described in Section 3 in comparison with Learn++.NSE benchmark algorithms.

We have tested both DARK framework and Learn++.NSE with different settings. Firstly, we tested the classifier used by Learn++.NSE authors as base classifier, a classification and regression tree (CART), namely Learn++.CART, and we have implemented the SVM as an alternative base classifier, so we can have similar base classifiers in Learn++.NSE and DARK. As the implementation provided by Learn++.NSE is in *Matlab* code, and the base classifier must report a single output, we have used a multiclass SVM strategy provided by Matlab called *ClassificationECOC*, which is an error-correcting output codes (ECOC) classifier for multiclass learning by reduction to multiple, binary classifiers like SVM. We have named this approach as Learn++.SVM. A one-against-all strategy is used both in this approach and in DARK.

Secondly, we have set up DARK with two different configurations: an ensemble composed by 4 models, namely DARK.4.InverseFN and a solution that retains all models previously created, namely DARK.All.InverseFN. The performance metric used to combine the ensemble in DARK was the Inverse False Negative ($\frac{1}{FN}$).

All SVM models processed in our experiments have linear kernels, and DARK implements the SVM Light application provided by Joachims (2002) and available at http://svmlight.joachims.org/ with cost-factor equal to 2. We have tested both algorithms with two distinct setups: using a training window of size 1 and a training window of size 4. The rationale of using training window size equal to 4 is detailed in Costa et al. (2015b), where we have studied the importance of past examples in the classification process by presenting a batch learning model that retained previously seen examples during a defined period. By retaining examples during different periods we aimed to evaluate for how long it is relevant to keep information according to the different types of drift, and thus best tailoring the memory mechanism needed for classification purposes. Training window size equal to 4 was revealed to have the best cost benefit relation between performance and computational burden. Table 5 summarizes the performance results obtained by classifying the dataset. Three performance metrics of each model are presented, namely precision, recall, and F_1, and all presented values are in percentage. We have highlighted (in bold) the micro-averaged values.

Considering a training window of size 1 means that the models are trained using exclusively the documents seen in the previous time window. All documents that have been seen prior to that are discarded.

By inspecting the results obtained and presented in Table 5, Learn++.NSE outperforms DARK, with both SVM as base classifiers and CART and considering the micro-averaged F_1. DARK scores 74.86% and 75.62%, with an ensemble size of 4 and without pruning, respectively, while Learn++.NSE scores 77.69% and 79.01% with SVM as base classifier and CART, respectively. As it can be stated, using CART as the base classifiers is better than using an SVM. Even thought the performance of DARK is worst than the performance of Learn++.NSE in this scenario, one must notice that DARK precision is far superior, 77.69% and 79.01% against 99.14% and 99.12%. It means that even though we miss to classify more positive examples than Learn++.NSE, we have fewer false positives. It is also important to observe that the implementation of Learn++.NSE is set to achieve the micro-averaged break-even point between precision and recall, as in all settings micro-averaged precision is equal to micro-averaged recall.

In the second setting, as stated above, we have used a training window of size 4. Table 6 summarizes the performance results obtained by classifying the dataset, considering the micro-averaged measures in this new setting.

The results revealed that DARK, performing 86.13% and 86.53%, micro-averaged F_1, outstands both Learn++.NSE with an SVM as base classifier, with 85.47%, as also with CART as base classifier, with 84.62%. It is important to note that the training dataset is exactly the same for all the algorithms, i.e., we have shown to them all examples from the 4 previously occurring time windows. The major difference seems to be that DARK makes a better use of the extra information that is provided when more examples are available, specially longstanding examples. It is also important to note that DARK.4.InverseFN is a lightweighted solution, as it discards all the information more than 4 time windows old. It discards not only the examples but also models created more than 4 time windows old.

Table 5
Performance results for training window size = 1.

	Learn++.NSE.SVM			Learn++.NSE.CART			DARK.4.InverseFN			DARK.All.InverseFN		
Drift	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall	F_1
Sudden #1	70.64%	87.48%	78.16%	90.57%	80.65%	85,32%	99.78%	78.80%	88.06%	99.78%	78.80%	88.06%
Sudden #2	64.85%	87.78%	74.59%	92.79%	86.81%	89.70%	99.83%	83.75%	91.09%	99.83%	83.75%	91.09%
Gradual #1	79.35%	65.96%	72.04%	72.64%	61.63%	66.68%	99.90%	40.08%	57.21%	99.90%	40.08%	57.21%
Gradual #2	86.67%	75.29%	80.58%	92.99%	72.46%	81.45%	99.87%	62.88%	77.17%	99.87%	63.17%	77.39%
Incremental #1	85.91%	84.00%	84.95%	94.12%	87.66%	90.77%	99.29%	76.88%	86.66%	99.14%	79.03%	87.95%
Incremental #2	65.24%	80.20%	71.95%	60.57%	84.92%	70.70%	97.03%	49.09%	65.20%	97.27%	54.30%	69.69%
Reoccurring	79.39%	57.27%	66.54%	73.50%	52.33%	61.14%	99.29%	46.73%	63.55%	99.29%	46.73%	63.55%
Normal #1	98.07%	55.35%	70.76%	95.52%	68.08%	79.50%	100.00%	16.69%	28.60%	100.00%	16.69%	28.60%
Normal #2	90.36%	88.85%	89.60%	95.88%	87.45%	91.47%	98.50%	71.66%	82.96%	98.50%	71.72%	83.00%
Normal #3	83.00%	64.11%	72.34%	51.37%	74.31%	60.75%	99.20%	40.11%	57.12%	99.20%	40.11%	57.12%
Micro-averaged F_1	**77.69%**	**77.69%**	**77.69%**	**79.01%**	**79.01%**	**79.01%**	**99.14%**	**60.13%**	**74.86%**	**99.12%**	**61.13%**	**75.62%**

Table 6
Performance results for training window size = 4.

	Learn++.NSE.SVM			Learn++.NSE.Cart			DARK.4.InverseFN			DARK.All.InverseFN		
Drift	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall	F_1
Sudden #1	84.12%	84.88%	84.49%	90.87%	84.10%	87.35%	99.82%	81.73%	89.87%	99.82%	81.73%	89.87%
Sudden #2	75.28%	89.08%	81.60%	97.14%	85.75%	91.09%	99.81%	87.83%	93.44%	99.81%	87.86%	93.46%
Gradual #1	92.07%	70.13%	79.61%	80.19%	70.33%	74.94%	99.70%	54.79%	70.72%	99.70%	55.29%	71.13%
Gradual #2	94.78%	84.04%	89.09%	97.41%	83.00%	89.63%	99.21%	67.71%	80.49%	99.16%	68.83%	81.26%
Incremental #1	93.13%	85.87%	89.35%	97.39%	89.35%	93.20%	99.47%	87.35%	93.02%	99.38%	87.59%	93.11%
Incremental #2	73.73%	92.66%	82.12%	72.04%	89.47%	79.82%	97.86%	71.78%	82.82%	97.89%	74.99%	84.93%
Reoccuring	88.00%	51.80%	65.21%	75.94%	53.67%	62.89%	97.95%	54.07%	69.67%	97.95%	54.07%	69.67%
Normal #1	97.93%	84.04%	90.45%	92.47%	78.87%	85.13%	99.86%	59.27%	74.39%	99.86%	59.47%	74.55%
Normal #2	86.88%	94.44%	90.50%	96.07%	91.76%	93.86%	99.48%	85.70%	92.08%	99.09%	86.06%	92.12%
Normal #3	89.48%	86.95%	88.20%	61.59%	89.58%	73.00%	99.23%	76.08%	86.13%	99.23%	76.43%	86.35%
Micro-averaged F_1	**85.47%**	**85.47%**	**85.47%**	**84.62%**	**84.62%**	**84.62%**	**99.30%**	**76.04%**	**86.13%**	**99.21%**	**76.72%**	**86.53%**

6. Conclusions and future work

In this paper we have presented the DARK framework to tackle adaptive learning with drifts in dynamic environments based on recent and retained knowledge. DARK supports distinct training window sizes and processes an ensemble of multiple SVM models that are dynamically weighted.

We have achieved a three-fold contribution with this paper: (i) to infer about the influence for the overall learning and classification performances, of having recent examples retained during more than one time window. That is, to analyse the impact of using longstanding examples by the classifiers models and how they can contribute positively to the ensemble classification; (ii) to validate the DARK framework with text classification scenarios, by applying it to text datasets based on Twitter social network public stream and (iii) to present a comparative study with benchmark solutions in the field, namely Learn++.NSE algorithm.

The evaluation of DARK framework reinforced the emerging need to invest on ensemble frameworks deployment for learning, that may cope with flexible and adjustable time windows of recent knowledge. We were able to contribute on that through DARK, as we may test datasets processing with different training window sizes of recent examples in order to infer about its impact on ensemble classification. We were also able to reinforce the effectiveness of using DOTS (Costa et al., 2015a) to create multiple realistic text classification datasets with drift patterns, corresponding to others such well known drift types.

Regarding experimental results, we have processed the same dataset with DARK framework and Learn++.NSE algorithm, being the later processed with both SVM and CART as base classifiers. For both processing we have used two distinct window sizes: 1 and 4. The results calculated for micro-averaged F_1 revealed that as we increased the window size, DARK framework became more precise and accurate.

Regarding computationally demands we observed that DARK.4. InverseFN is a lightweighted solution and competitive with the Learn++.NSE algorithm.

Future research directions will be carried out in the investigation of different learning machines and heterogeneous ensembles along with refine DARK to have a dynamic and JIT window size adaptation for NSE text-related applications.

References

Ahsan, M.I., Nahian, T., Kafi, A.A., Hossain, M.I., Shah, F.M., 2016. Review spam detection using active learning. In: Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2016 IEEE 7th Annual. IEEE, pp. 1–7.

Alippi, C., Boracchi, G., Roveri, M., 2009. Just in time classifiers: managing the slow drift case. In: 2009 International Joint Conference on Neural Networks. IEEE, pp. 114–120.

Alippi, C., Boracchi, G., Roveri, M., 2012. Just-in-time ensemble of classifiers. In: 2012 International Joint Conference on Neural Networks (IJCNN). IEEE, pp. 1–8.

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.

Alippi, C., Roveri, M., 2008. Just-in-time adaptive classifiers - Part II: Designing the classifier. IEEE Transactions on Neural Networks 19 (12), 2053–2064.

Armitage, P., 1960. Sequential medical trials. Blackwell Scientific Publications.

Bagul, R.D., Phulpagar, B.D., 2016. Survey on approaches, problems and applications of ensemble of classifiers. International Journal of Emerging Trends & Technology in Computer Science 5 (1), 28–30.

Bifet, A., Gavalda, R., 2007. Learning from time-changing data with adaptive windowing. In: International Conference on Data Mining.

Breiman, L., 1996. Bagging predictors. Machine Learning 24 (2), 123–140.

Breiman, L., 2001. Random forests. Machine Learning 45 (1), 5–32.

Chang, H.-C., 2010. A new perspective on twitter hashtag use: diffusion of innovation theory. In: Proceedings of the 73rd Annual Meeting on Navigating Streams in an Information Ecosystem. pp. 85:1–85:4.

Cohen, L., Avrahami, G., Last, M., Kandel, A., 2008. Info-fuzzy algorithms for mining dynamic data streams. Applied Soft Computing 8 (4), 1283–1294.

Cohen, L., Avrahami-Bakish, G., Last, M., Kandel, A., Kipersztok, O., 2008. Real-time data mining of non-stationary data streams from sensor networks. Information Fusion 9 (3), 344–353.

Costa, J., Silva, C., Antunes, M., Ribeiro, B., 2013. Defining semantic meta-hashtags for twitter Classification. In: Proceedings of the 11th International Conference on Adaptive and Natural Computing Algorithms. pp. 226–235.

Costa, J., Silva, C., Antunes, M., Ribeiro, B., 2014. Concept drift awareness in twitter Streams. In: Proceedings of the 13th International Conference on Machine Learning and Applications. pp. 294–299.

Costa, J., Silva, C., Antunes, M., Ribeiro, B., 2015. DOTS: Drift Oriented Tool System. In: Proceedings of the 22nd International Conference on Neural Information Processing (ICONIP). pp. 615–623.

Costa, J., Silva, C., Antunes, M., Ribeiro, B., 2015. The impact of longstanding messages in micro-blogging classification. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN). pp. 1–8.

Costa, J., Silva, C., Antunes, M., Ribeiro, B., 2016. Choice of best samples for building ensembles in dynamic environments. In: Proceedings of the 17th International Conference on Engineering Applications of Neural Networks (EANN). pp. 35–47.

Datar, M., Motwani, R., 2016. The sliding-window computation model and results. In: Data Stream Management. Springer, pp. 149–165.

Ditzler, G., Polikar, R., 2013. Incremental learning of concept drift from streaming imbalanced data. IEEE Transactions on Knowledge and Data Engineering 25 (10), 2283–2301.

Ditzler, G., Roveri, M., Alippi, C., Polikar, R., 2015. Learning in nonstationary environments: a survey. IEEE Computational Intelligence Magazine 10 (4), 12–25.

Domingos, P., Hulton, G., 2000. Mining high-speed data streams. In: Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery Data Mining. pp. 71–80.

Elwell, R., Polikar, R., 2011. Incremental learning of concept drift in nonstationary environments. IEEE Transactions on Neural Networks 1517–1531.

Freund, Y., Schapire, R.E., 1997. A decision-theoretic generalization of on-line learning and an application to boosting. Journal Computer and Systems Science 55 (1), 119–139.

Gama, J.,Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A., 2014. A survey on concept drift adaptation. ACM Computing Surveys (CSUR) 46 (4), 44.

Haque, A., Khan, L., Baron, M., 2015. Semi supervised adaptive framework for classifying evolving data stream. In: Conference on Knowledge Discovery and Data Mining. Springer, pp. 383–394.

Haque, M.E., Alkharobi, T.M., 2015. Adaptive hybrid model for network intrusion detection and comparison among machine learning algorithms. International Journal of Machine Learning and Computing 5 (1), 17.

Harel, M., Mannor, S., El-Yaniv, R., Crammer, K., 2014. Concept drift detection through resampling. In: Proceedings of the 31st International Conference on Machine Learning. pp. 1009–1017.

Huang, J., Thornton, K.M., Efthimiadis, E.N., 2010. Conversational tagging in twitter. In: Proceedings of the 21st ACM Conference on Hypertext and Hypermedia. pp. 173–178.

Huijse, P., Estevez, P.A., Protopapas, P., Principe, J.C., Zegers, P., 2014. Computational intelligence challenges and applications on large-scale astronomical time series databases. IEEE Computational Intelligence Magazine 9 (3), 27–39.

Hulten, G., Spencer, L., Domingos, P., 2001. Mining time-changing data streams. In: Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 97–106.

Joachims, T., 2002. Learning Text Classifiers with Support Vector Machines.

Johnson, S., 2009. How twitter will change the way we live. Time Magazine 173, 23–32.

Jr., M.P.P., 2011. Combining classifiers: from the creation of ensembles to the decision fusion. In: 24th Conference on Graphics, Patterns and Images. pp. 1–10.

Karnick, M., Muhlbaier, M.D., Polikar, R., 2008. Incremental learning in non-stationary environments with concept drift using a multiple classifier based approach. In: International Conference on Pattern Recognition. pp. 1–4.

Kim, J., Bentley, P.J., Aickelin, U., Greensmith, J., Tedesco, G., Twycross, J., 2007. Immune system approaches to intrusion detection–a review. Natural Computing 6 (4), 413–466.

Klinkenberg, R., 2004. Learning drifting concepts: example selection vs. example weighting. Intelligent Data Analysis 8 (3), 281–300.

Koychev, I., 2000. Gradual forgetting for adaptation to concept drift. Proceedings of ECAI 2000 Workshop on Current Issues in Spatio-Temporal Reasoning,.

Krovetz, R., 1993. Viewing morphology as an inference process. In: Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 191–202.

Kuncheva, L., 2002. A theoretical study on six classifier fusion strategies. IEEE Transactions on Pattern Analysis and Machine Intelligence 24 (2), 281–286.

Kuncheva, L., 2004. Combining pattern classifiers, methods and algorithms. Wiley.

Liu, J., Li, X., Zhong, W., 2009. Ambiguous decision trees for mining concept-drifting data streams. Pattern Recognition Letters 30 (15), 1347–1355.

Lughofer, E., Mouchaweh, M.S., 2015. Adaptive and on-line learning in non-stationary environments.. Evolving Systems 6 (2), 75–77.

Merriam-webster's dictionary (2012).

Nishida, K., Yamauchi, K., 2007. Detecting concept drift using statistical testing. In: Proceedings of the International Conference on Discovery Science. Springer, pp. 264–269.

Patist, J.P., 2007. Optimal window change detection. In: Proceedings of the 7th IEEE International Conference on Data Mining Workshop (ICDMW 2007). IEEE, pp. 557–562.

Polikar, R., Upda, L., Upda, S.S., Honavar, V., 2001. Learn++: an incremental learning algorithm for supervised neural networks. IEEE Transactions on Systems, Man and Cybernetics (4), 497–508.

Ren, Y., Zhang, L., Suganthan, P.N., 2016. Ensemble classification and regression - recent developments, applications and future directions. IEEE Computational Intelligence Magazine 1 (1), 41–43.

van Rijsbergen, C., 1979. Information retrieval.

Ross, G.J., Tasoulis, D.K., Adams, N.M., 2011. Nonparametric monitoring of data streams for changes in location and scale. Technometrics 53 (4), 379–389.

Tabassum, N., Ahmed, T., 2016. A theoretical study on classifier ensemble methods and its applications. In: Proceedings of the 3rd International Conference on Computing for Sustainable Global Development. pp. 67–78.

Tong, S., Koller, D., 2002. Support vector machine active learning with applications to text classification. Journal of Machine Learning Research (JMLR) (ISSN: 1532-4435) 2, 45–66.

Tsur, O., Rappoport, A., 2012. What's in a hashtag?: content based prediction of the spread of ideas in microblogging communities. In: Proceedings of the 5th International Conference on Web Search and Data Mining. pp. 643–652.

Vapnik, V., 1999. The nature of statistical learning theory.

Vitter, J.S., 1985. Random sampling with a reservoir. ACM Transactions on Mathematical Software (TOMS) 11 (1), 37–57.

Wald, A., 1992. Sequential tests of statistical hypotheses. In: Breakthroughs in Statistics: Foundations and Basic Theory. Springer New York, pp. 256–298.

Willett, P., 2006. The porter stemming algorithm: then and now. Program 40 (3), 219–223.

Yang, L., Sun, T., Zhang, M., Mei, Q., 2012. We know what @you #tag: does the dual role affect hashtag adoption? In: Proceedings of the 21st International Conference on World Wide Web. pp. 261–270.

Ye, Y., Squartini, S., Piazza, F., Online sequential extreme learning machine in nonstationary environments, Neurocomputing 116.

Zappavigna, M., 2011. Ambient affiliation: A linguistic perspective on Twitter. New Media & Society 13 (5), 788–806.

Zliobaite, I., 2010. Learning under concept drift: an overview. Faculty of Mathematics and Informatic, Vilnius University, Latvia.