

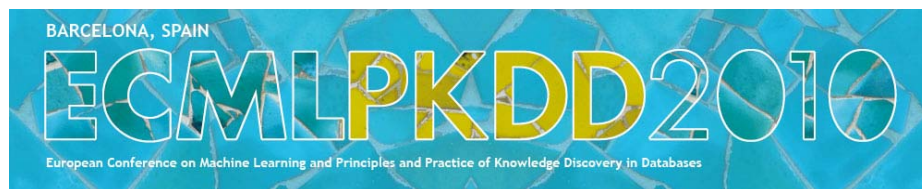
Proceedings of the First International Workshop on

Handling Concept Drift
in Adaptive Information Systems:
Importance, Challenges and Solutions

HaCDAIS 2010

Held in conjunction with
ECML/PKDD 2010

September 24, 2010
Barcelona, Spain



Editors

Mykola Pechenizkiy

Eindhoven University of Technology, the Netherlands

Indrė Žliobaitė

Eindhoven University of Technology, the Netherlands

Preface

In the real world data is often non stationary. In predictive analytics, machine learning and data mining the phenomenon of unexpected change in underlying data over time is known as concept drift. Changes in underlying data might occur due to changing personal interests, changes in population, adversary activities or they can be attributed to a complex nature of the environment.

When there is a shift in data, the predictions might become less accurate as the time passes or opportunities to improve the accuracy might be missed. Thus the learning models need to be adaptive to the changes.

The problem of concept drift is of increasing importance to machine learning and data mining as more and more data is organized in the form of data streams rather than static databases, and it is rather unusual that concepts and data distributions stay stable over a long period of time. It is not surprising that the problem of concept drift has been studied in several research communities including but not limited to machine learning and data mining, data streams, information retrieval, and recommender systems. Different approaches for detecting and handling concept drift have been proposed in the literature, and many of them have already proved their potential in a wide range of application domains, e.g. fraud detection, adaptive system control, user modeling, information retrieval, text mining, biomedicine.

HaCDAIS 2010 workshop¹ organized in conjunction with ECML/PKDD 2010 and held on 24 September 2010 in Barcelona, Spain provides a focused international forum for researchers to discuss new, we aim to attract researchers with an interest in handling concept drift and recurring contexts in adaptive information systems.

Topics discussed at the workshop include classification and clustering on data streams and evolving data, change and novelty detection in online, semi-online and offline settings, adaptive ensembles, adaptive sampling and instance selection, incremental learning and model adaptivity, delayed labeling in data streams, dynamic feature selection, handling local and complex concept drift, qualitative and quantitative evaluation of concept drift handling performance, reoccurring contexts and context-aware approaches, application-specific and domain driven approaches within the areas of information retrieval, recommender systems, pattern recognition, user modeling, decision support and adaptive information systems

These proceedings include abstract of the invited talk, invited software report and six peer-reviewed papers accepted to the workshop, two as full papers and four as short papers. We would like to thank all the authors for their interest in the workshop and for submitting their contributions.

We would like to thank PC member for publicizing the workshop and providing timely and constructive reviews for the submitted papers.

We are thankful to Prof. João Gama who kindly agreed to give an invited talk “Reasoning about the Learning Process” and Dr. Albert Bifet for presenting the “MOA: Massive Online Analysis” framework and software demonstration.

¹ For further information about workshop-related information, scientific program overview and link to the online proceedings please refer to <http://www.wis.win.tue.nl/hacdais2010/>

We are looking forward for an interesting workshop and to meeting you in Barcelona!

27 July 2010
Eindhoven, the Netherlands

Mykola Pechenizkiy
Indrė Žliobaitė

Co-Chairs of
HaCDAIS 2010 Workshop

Workshop Organization

Workshop Chairs

Mykola Pechenizkiy Eindhoven University of Technology, the Netherlands
Indrė Žliobaitė Eindhoven University of Technology, the Netherlands

Workshop Program Committee

Albert Bifet University of Waikato, New Zealand
Sarah Jane Delany Digital Media Centre, Ireland
Anton Dries Katholieke Universiteit Leuven, Belgium
Bogdan Gabrys Bournemouth University, UK
João Gama University of Porto, Portugal
Ioannis Katakis Aristotle University of Thessaloniki, Greece
Yehuda Koren Yahoo! Research, Israel
Ludmila Kuncheva Bangor University, UK
Matthijs van Leeuwen Universiteit Utrecht, the Netherlands
Ernestina Menasalvas Universidad Politecnica de Madrid, Spain
Robi Polikar Rowan University, USA
Myra Spiliopoulou Otto-von-Guericke-University Magdeburg, Germany
Alexey Tsymbal Siemens AG, Germany
Athena Vakali Aristotle University, Thessaloniki, Greece

External Reviewers

Elena Ikonomovska Jozef Stefan Institute, Slovenia

Table of Contents

Invited Talk

Reasoning about the Learning Process	1
<i>João Gama</i>	

Invited Software Report

MOA: Massive Online Analysis, a framework for stream classification and clustering	3
<i>Albert Bifet, Geo Holmes, Bernhard Pfahringer, Philipp Kranen, Hardy Kremer, Timm Jansen, and Thomas Seidl</i>	

Full Papers

Incremental option trees for handling gradual concept drift	17
<i>Elena Ikonomovska, João Gama, and Sašo Džeroski</i>	
Handling concept drift in preference learning for interactive decision making.....	29
<i>Paolo Campigotto, Andrea Passerini, and Roberto Battiti</i>	

Short Papers

Cost-sensitive boosting for concept drift.....	41
<i>Ashok Venkatesan, Narayanan C. Krishnan, Sethuraman Panchanathan</i>	
An adaptive hybrid recommender system that learns domain dynamics.....	49
<i>Fatih Aksel and Ayşenur Birtürk</i>	
Modeling the example life-cycle in an online classification learner.....	57
<i>Gary R. Marrs, Ray J. Hickey, and Michaela M. Black</i>	
An incremental text segmentation by clustering cohesion.....	65
<i>Raúl Abella Pérez and José Eladio Medina Pagola</i>	

Reasoning about the Learning Process

João Gama

LIAAD-INESC Porto, FEP-University of Porto,
Rua de Ceuta 118, Porto 4050-190, Portugal
jgama@fep.up.pt

Abstract. Data Mining is faced with new challenges. In emerging applications (like financial data, traffic TCP/IP, sensor networks, etc) data continuously flow eventually at high speed. The processes generating data evolve over time, and the concepts we are learning change. Evolving data requires that learning algorithms must be able to monitor the learning process and the ability of predictive self-diagnosis. A significant and useful characteristic is diagnostics - not only after failure has occurred, but also predictive (before failure). These aspects require monitoring the evolution of the learning process, taking into account the available resources, and the ability of reasoning and learning about it. In this work we present a one-pass classification algorithm able to detect and react to changes in the process that generates data. Instead of forgetting the previous learned models, they are stored for possible reuse if the context where they were learned re-appears. The system uses meta-learning techniques that characterize the domain of applicability of previous learned models. The meta-learner can detect re-occurrence of contexts and take pro-active actions by activating previous learned models. The main benefit of this approach is that the proposed meta-learner is capable of selecting similar historical concept, if there is one, without the knowledge of the true classes of examples.

Data streams; concept drift; meta-learning; recurrent concepts.
Join work with Petr Kosina.

MOA: Massive Online Analysis, a Framework for Stream Classification and Clustering.

Albert Bifet¹, Geoff Holmes¹, Bernhard Pfahringer¹,
Philipp Kranen², Hardy Kremer², Timm Jansen², and Thomas Seidl²

¹ Department of Computer Science, University of Waikato, Hamilton, New Zealand
{abifet, geoff, bernhard}@cs.waikato.ac.nz

² Data Management and Exploration Group, RWTH Aachen University, Germany
{kranen, kremer, jansen, seidl}@cs.rwth-aachen.de

Abstract. In today's applications, massive, evolving data streams are ubiquitous. **Massive Online Analysis (MOA)** is a software environment for implementing algorithms and running experiments for online learning from evolving data streams. MOA is designed to deal with the challenging problems of scaling up the implementation of state of the art algorithms to real world dataset sizes and of making algorithms comparable in benchmark streaming settings. It contains a collection of offline and online algorithms for both classification and clustering as well as tools for evaluation. Researchers benefit from MOA by getting insights into workings and problems of different approaches, practitioners can easily compare several algorithms and apply them to real world data sets and settings. MOA supports bi-directional interaction with WEKA, the Waikato Environment for Knowledge Analysis, and is released under the GNU GPL license. Besides providing algorithms and measures for evaluation and comparison, MOA is easily extensible with new contributions and allows the creation of benchmark scenarios through storing and sharing setting files.

1 Introduction

Nowadays data is generated at an increasing rate from sensor applications, measurements in network monitoring and traffic management, log records or click-streams in web exploring, manufacturing processes, call detail records, email, blogging, twitter posts and others. In fact, all data generated can be considered as streaming data or as a snapshot of streaming data, since it is obtained from an interval of time.

In data stream scenarios data arrives at high speed strictly constraining processing algorithms in space and time. To adhere to these constraints, specific requirements have to be fulfilled by the stream processing algorithms that are different from traditional batch processing settings. The most significant requirements are the following:

Requirement 1 Process an example at a time, and inspect it at most once

Requirement 2 Use a limited amount of memory

Requirement 3 Work in a limited amount of time

Requirement 4 Be ready to predict at any time

Stream learning algorithms are an important type of stream processing algorithms: In a repeated cycle, the learned model is constantly updated to reflect the incoming examples from the stream. They do so without exceeding their memory and time bounds. After processing an incoming example, the algorithms are always able to output a model. Typical learning tasks in stream scenarios are classification, outlier analysis, and clustering.

Since a multitude of algorithms exist for stream learning scenarios, a thorough comparison by experimental evaluation is crucial. In most publications, newly proposed algorithms are only compared to a small subset or even none of the competing solutions, making the assessment of their actual effectiveness tough. Moreover, the majority of experimental evaluations use only small amounts of data. In the context of data streams this is disappointing, because to be truly useful the algorithms need to be capable of handling very large (potentially infinite) streams of examples. Demonstrating systems only on small amounts of data does not build a convincing case for capacity to solve more demanding data stream applications [27].

In traditional batch learning scenarios, evaluation frameworks were introduced to cope with the comparison issue. One of these frameworks is the well-known WEKA Data Mining Software that supports adding new algorithms and evaluation measures in a plug-and-play fashion [21, 32, 31]. As data stream learning is a relatively new field, the evaluation practices are not nearly as well researched and established as they are in the traditional batch setting.

For this purpose we introduce **Massive Online Analysis (MOA)** [8], a framework for stream learning evaluation that builds on the work in WEKA. MOA contains state-of-the-art algorithms and measures for both stream classification and stream clustering and permits evaluation of data stream mining algorithms on large streams, in the order of tens of millions of examples where possible, and under explicit memory limits. The main contributions and benefits of the MOA framework are:

- Analysis and comparison both for different approaches (new and state-of-the-art algorithms) and for different (large) streaming setting
- Creation and usage of benchmark settings for comparable and repeatable evaluation of stream mining algorithms
- Open source framework that is easily extensible for data feeds, algorithms and evaluation measures

In the following we first introduce the general architecture of MOA before describing how to use it for classification and clustering on evolving data streams. Section 5 points to additional material including source codes and tutorials and Section 6 concludes the paper.

2 System architecture

A simplified system architecture is illustrated in Figure 1. It shows at the same time the work flow of MOA and its extension points, since all aspects follow the same principle. First a data feed is chosen, then a learning algorithm is configured, i.e. a stream classification or stream clustering algorithm and finally an evaluation method is chosen to analyze the desired scenario. The choice of streams, algorithms and especially evaluation methods differs between the classification and clustering parts and is therefore described separately in the following sections. For both tasks, users can extend the framework in all three aspects to add novel data generators, algorithms or evaluation measures. To run experiments using MOA users can chose between the command line or a graphical user interface.

Generally, MOA permits to define three environments that are simulated using memory limits, since memory limits cannot be ignored and can significantly limit capacity to learn from data streams. Potential practical deployment of data stream learning algorithms has been divided into scenarios of increasing memory utilization, from the restrictive sensor environment, to a typical consumer grade handheld PDA environment, to the least restrictive environment of a dedicated server.

Sensor Network This environment represents the most restrictive case, learning in 100 kilobytes of memory. Because this limit is so restrictive, it is an interesting test case for algorithm efficiency.

Handheld Computer In this case the algorithm is allowed 32 megabytes of memory. This simulates the capacity of lightweight consumer devices designed to be carried around by users and fit into a shirt pocket.

Server This environment simulates either a modern laptop/desktop computer or server dedicated to processing a data stream. The memory limit assigned in this environment is 400 megabytes. Considering that several algorithms have difficulty in fully utilizing this much working space, it seems sufficiently realistic to impose this limit.

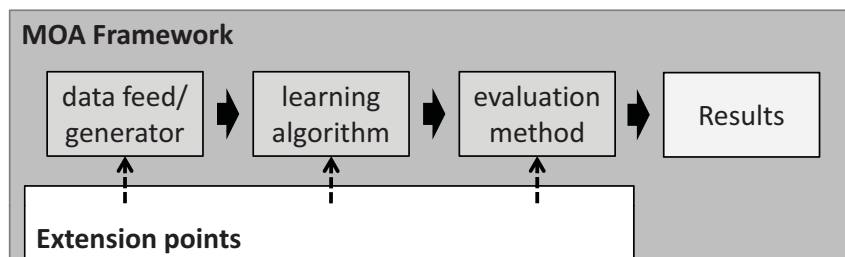


Fig. 1. Architecture, extension points and work flow of the MOA framework.

3 Classification

In this section we detail the features and the usage of stream classification in MOA.

3.1 Data streams generators for stream classification

Considering data streams as data generated from pure distributions, MOA models a concept drift event as a weighted combination of two pure distributions that characterizes the target concepts before and after the drift. Within the framework, it is possible to define the probability that instances of the stream belong to the new concept after the drift. It uses the sigmoid function, as an elegant and practical solution [9, 10].

MOA contains the data generators most commonly found in the literature. MOA streams can be built using generators, reading ARFF files, joining several streams, or filtering streams. They allow for the simulation of a potentially infinite sequence of data. The following generators are currently available:

SEA Concepts Generator This artificial dataset contains abrupt concept drift, first introduced in [42]. It is generated using three attributes, where only the two first attributes are relevant. All three attributes have values between 0 and 10. The points of the dataset are divided into 4 blocks with different concepts. In each block, the classification is done using $f_1 + f_2 \leq \theta$, where f_1 and f_2 represent the first two attributes and θ is a threshold value. The most frequent values are 9, 8, 7 and 9.5 for the data blocks.

STAGGER Concepts Generator They were introduced by Schlimmer and Granger in [39]. The STAGGER Concepts are boolean functions of three attributes encoding objects: size (small, medium, and large), shape (circle, triangle, and rectangle), and colour (red, blue, and green). A concept description covering either green rectangles or red triangles is represented by (shape=rectangle and colour=green) or (shape=triangle and colour=red).

Rotating Hyperplane It was used as testbed for CVFDT versus VFDT in [25]. A hyperplane in d -dimensional space is the set of points x that satisfy

$$\sum_{i=1}^d w_i x_i = w_0 = \sum_{i=1}^d w_i$$

where x_i , is the i th coordinate of x . Examples for which $\sum_{i=1}^d w_i x_i \geq w_0$ are labeled positive, and examples for which $\sum_{i=1}^d w_i x_i < w_0$ are labeled negative. Hyperplanes are useful for detecting time-changing concepts, because we can change the orientation and position of the hyperplane in a smooth manner by changing the relative size of the weights. We introduce change to this dataset adding drift to each weight attribute $w_i = w_i + d\sigma$, where σ is the probability that the direction of change is reversed and d is the change applied to every example.

Random RBF Generator This generator was devised to offer an alternate complex concept type that is not straightforward to approximate with a decision tree model. The RBF (Radial Basis Function) generator works as follows: A fixed number of random centroids are generated. Each center has a random position, a single standard deviation, class label and weight. New examples are generated by selecting a center at random, taking weights into consideration so that centers with higher weight are more likely to be chosen. A random direction is chosen to offset the attribute values from the central point. The length of the displacement is randomly drawn from a Gaussian distribution with standard deviation determined by the chosen centroid. The chosen centroid also determines the class label of the example. This effectively creates a normally distributed hypersphere of examples surrounding each central point with varying densities. Only numeric attributes are generated. Drift is introduced by moving the centroids with constant speed. This speed is initialized by a drift parameter.

LED Generator This data source originates from the CART book [11]. An implementation in C was donated to the UCI [4] machine learning repository by David Aha. The goal is to predict the digit displayed on a seven-segment LED display, where each attribute has a 10% chance of being inverted. It has an optimal Bayes classification rate of 74%. The particular configuration of the generator used for experiments (led) produces 24 binary attributes, 17 of which are irrelevant.

Waveform Generator The goal of the task is to differentiate between three different classes of waveform, each of which is generated from a combination of two or three base waves. The optimal Bayes classification rate is known to be 86%.

Function Generator It was introduced by Agrawal et al. in [3], and was a common source of data for early work on scaling up decision tree learners. The generator produces a stream containing nine attributes, six numeric and three categorical. Although not explicitly stated by the authors, a sensible conclusion is that these attributes describe hypothetical loan applications. There are ten functions defined for generating binary class labels from the attributes. Presumably these determine whether the loan should be approved.

3.2 Classifiers methods

MOA contains several classifier methods such as: Naive Bayes, Decision Stump, Hoeffding Tree, Hoeffding Option Tree, Bagging, Boosting, Bagging using ADWIN, and Bagging using Adaptive-Size Hoeffding Trees.

A *Hoeffding tree* [15] is an incremental, anytime decision tree induction algorithm that is capable of learning from massive data streams, assuming that the distribution generating examples does not change over time. Hoeffding trees exploit the fact that a small sample can often be enough to choose an optimal splitting attribute. This idea is supported mathematically by the Hoeffding bound, which quantifies the number of observations (in our case, examples) needed to estimate some statistics within a prescribed precision (in our case, the

goodness of an attribute). More precisely, the Hoeffding bound states that with probability $1 - \delta$, the true mean of a random variable of range R will not differ from the estimated mean after n independent observations by more than:

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}.$$

A theoretically appealing feature of Hoeffding Trees not shared by other incremental decision tree learners is that it has sound guarantees of performance. Using the Hoeffding bound one can show that its output is asymptotically nearly identical to that of a non-incremental learner using infinitely many examples.

Hoeffding Option Trees [34] are regular Hoeffding trees containing additional option nodes that allow several tests to be applied, leading to multiple Hoeffding trees as separate paths. They consist of a single structure that efficiently represents multiple trees. A particular example can travel down multiple paths of the tree, contributing, in different ways, to different options.

ADWIN [7] is a change detector and estimator that solves in a well-specified way the problem of tracking the average of a stream of bits or real-valued numbers. ADWIN keeps a variable-length window of recently seen items, with the property that the window has the maximal length statistically consistent with the hypothesis “there has been no change in the average value inside the window”.

Bagging using ADWIN [10] is based on the online bagging method of Oza and Rusell [33] with the addition of the ADWIN algorithm as a change detector. When a change is detected, the worst classifier of the ensemble of classifiers is removed and a new classifier is added to the ensemble.

Adaptive-Size Hoeffding Trees (ASHT) [10] are derived from the Hoeffding Tree algorithm with the following differences: they have a value for the maximum number of split nodes, or *size*, and after one node splits, they delete some nodes to reduce its size if it is necessary. The intuition behind this method is as follows: smaller trees adapt more quickly to changes, and larger trees perform better during periods with little or no change, simply because they were built on more data.

3.3 Evaluation methods for stream classification

In traditional batch learning the problem of limited data is overcome by analyzing and averaging multiple models produced with different random arrangements of training and test data. In the stream setting the problem of (effectively) unlimited data poses different challenges. One solution involves taking snapshots at different times during the induction of a model to see how much the model improves.

The evaluation procedure of a learning algorithm determines which examples are used for training the algorithm, and which are used to test the model output by the algorithm. When considering what procedure to use in the data stream setting, one of the unique concerns is how to build a picture of accuracy over time. Two main approaches arise:

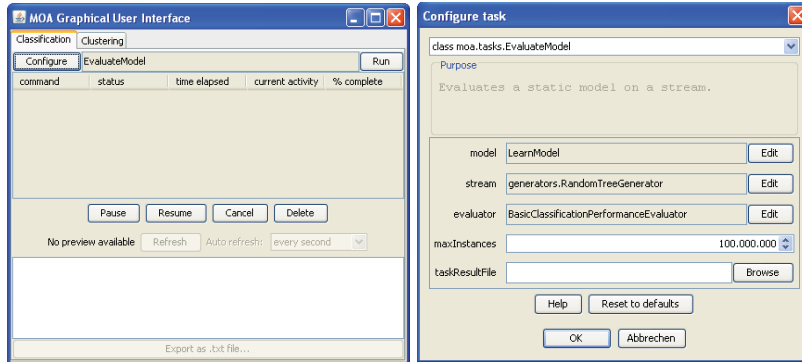


Fig. 2. MOA Graphical User Interface

- **Holdout:** When traditional batch learning reaches a scale where cross-validation is too time consuming, it is often accepted to instead measure performance on a single holdout set. This is most useful when the division between train and test sets has been pre-defined, so that results from different studies can be directly compared.
- **Interleaved Test-Then-Train or Prequential:** Each individual example can be used to test the model before it is used for training, and from this the accuracy can be incrementally updated. When intentionally performed in this order, the model is always being tested on examples it has not seen. This scheme has the advantage that no holdout set is needed for testing, making maximum use of the available data. It also ensures a smooth plot of accuracy over time, as each individual example will become increasingly less significant to the overall average [20].

MOA contains the above mentioned stream generators, classifiers and evaluation methods. Figure 2 shows the MOA graphical user interface. However, a command line interface is also available.

A non-trivial example of the EvaluateInterleavedTestThenTrain task creating a comma separated values file, training the HoeffdingTree classifier on the WaveformGenerator data, training and testing on a total of 100 million examples, and testing every one million examples, is encapsulated by the following commandline:

```
java -cp .:moa.jar:weka.jar -javaagent:sizeofag.jar moa.DoTask \
  "EvaluateInterleavedTestThenTrain -l HoeffdingTree \
  -s generators.WaveformGenerator \
  -i 100000000 -f 1000000" > htresult.csv
```

MOA is easy to use and extend. A simple approach to writing a new classifier is to extend `moa.classifiers.AbstractClassifier`, which will take care of certain details to ease the task.

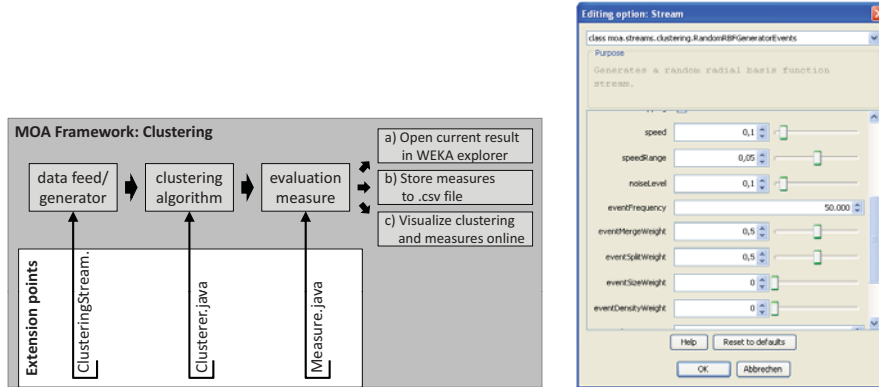


Fig. 3. Left: Extension points and work flow of the MOA stream clustering framework. Right: Option dialog for the RBF data generator (by storing and loading settings benchmark streaming data sets can be shared for repeatability and comparison).

4 Clustering

The stream clustering component of MOA has the following main features:

- data generators for stream clustering on evolving streams (including events like novelty, merge, etc. [41]),
- a set of state-of-the-art stream clustering algorithms,
- evaluation measures for stream clustering,
- visualization tools for analyzing results and comparing different settings.

The left part of figure 3 shows the extension points of MOA stream clustering and thereby illustrates the architecture as well as the usage of the clustering component. First a data feed is chosen and configured, then a stream clustering algorithm and its settings are fixed, then a set of evaluation measures is selected and finally the experiment is run to obtain and analyze the result. We detail these four aspects in the following subsections.

4.1 Data feeds and data generators

For stream clustering we added new data generators that support the simulation of cluster evolution events such as merging or disappearing of clusters [41].

The right part of figure 3 shows a screenshot of the configuration dialog for our RBF data generator with events. Generally the dimensionality, number and size of clusters can be set as well as the drift speed, decay horizon (aging) and noise rate etc. Events constitute changes in the underlying data model such as growing of clusters, merging of clusters or creation of new clusters [41]. Using the event frequency and the individual event weights, one can study the behaviour and performance of different approaches on various settings. Finally, the settings

for the data generators can be stored and loaded, which offers the opportunity of sharing settings and thereby providing benchmark streaming data sets for repeatability and comparison. New data feeds and generators can be added to the MOA framework by implementing the `ClusteringStream` interface (further description and source code can be found on the MOA website, cf. Section 5).

4.2 Stream clustering algorithms

Currently MOA contains several stream clustering methods including:

- `StreamKM++` [1]: It computes a small weighted sample of the data stream and it uses the `k-means++` algorithm as a randomized seeding technique to choose the first values for the clusters. To compute the small sample, it employs coreset constructions using a coreset tree for speed up.
- `CluStream` [2]: It maintains statistical information about the data using micro-clusters. These micro-clusters are temporal extensions of cluster feature vectors. The micro-clusters are stored at snapshots in time following a pyramidal pattern. This pattern allows to recall summary statistics from different time horizons.
- `ClusTree` [28]: It is a parameter free algorithm automatically adapting to the speed of the stream and it is capable of detecting concept drift, novelty, and outliers in the stream. It uses a compact and self-adaptive index structure for maintaining stream summaries.
- `Den-Stream` [13]: It uses dense micro-clusters (named core-micro-cluster) to summarize clusters. To maintain and distinguish the potential clusters and outliers, this method presents core-micro-cluster and outlier micro-cluster structures.
- `D-Stream` [43]: This method maps each input data record into a grid and it computes the grid density. The grids are clustered based on the density. This algorithm adopts a density decaying technique to capture the dynamic changes of a data stream.
- `CobWeb` [18]. One of the first incremental methods for clustering data. It uses a classification tree. Each node in a classification tree represents a class (concept) and is labeled by a probabilistic concept that summarizes the attribute-value distributions of objects classified under the node.

The set of algorithms is extensible through classes that implement the interface `Clusterer.java`. These are added to the framework via reflections on start up. The three main methods of this interface are

- `void resetLearningImpl()`: a method for initializing a clusterer learner
- `void trainOnInstanceImpl(Instance)`: a method to train a new instance
- `Clustering getClusteringResult()`: a method to obtain the current clustering result for evaluation or visualization

Internal measures	External measures
Gamma [5]	Rand statistic [35]
C Index [24]	Jaccard coefficient [19]
Point-Biserial [30]	Folkes and Mallow Index [19]
Log Likelihood [22]	Hubert T statistics [23]
Dunn's Index [17]	Minkowski score [12]
Tau [37]	Purity [44]
Tau \underline{A} [24]	van Dongen criterion [16]
Tau \underline{C} [24]	V-measure [38]
Somer's Gamma [24]	Completeness [38]
Ratio of Repetition [24]	Homogeneity [38]
Modified Ratio of Repetition [24]	Variation of information [29]
Adjusted Ratio of Clustering [24]	Mutual information [14]
Fagan's Index [24]	Class-based entropy [40]
Deviation Index [24]	Cluster-based entropy [44]
Z-Score Index [24]	Precision [36]
\underline{D} Index [24]	Recall [36]
Silhouette coefficient [26]	F-measure [36]

Table 1. Internal and external clustering evaluation measures.

4.3 Stream clustering evaluation measures

For cluster evaluation various measures have been developed and proposed over the last decades. A common classification of these measures is the separation in so called internal measures and external measures. Internal measures only consider the cluster properties, e.g. distances between points within one cluster or between two different clusters. External evaluation measures compare a given clusterings to a ground truth. Table 1 shows a selection of popular measures from the literature. MOA contains an extensible set of both internal and external measures that can be applied to both micro and macro clusterings. Moreover, specialized measures that take the peculiarities of an evolving data stream into account to fairly evaluate the performance of a stream clustering algorithm will be included in our set of measures.

To extend the available collection with additional or novel evaluation measures one has to implement the Measure interface. The main methods are:

- `void evaluateClustering(Clustering clustering, Clustering trueClustering)`: uses the implemented measure to evaluate the given clustering w.r.t. to the provided ground truth.
- `double getLastValue()`: a method that outputs the last result of the evaluation measure.
- `double getMaxValue(), getMinValue(), getMean()`: methods that provide more statistics about the measure's distribution.

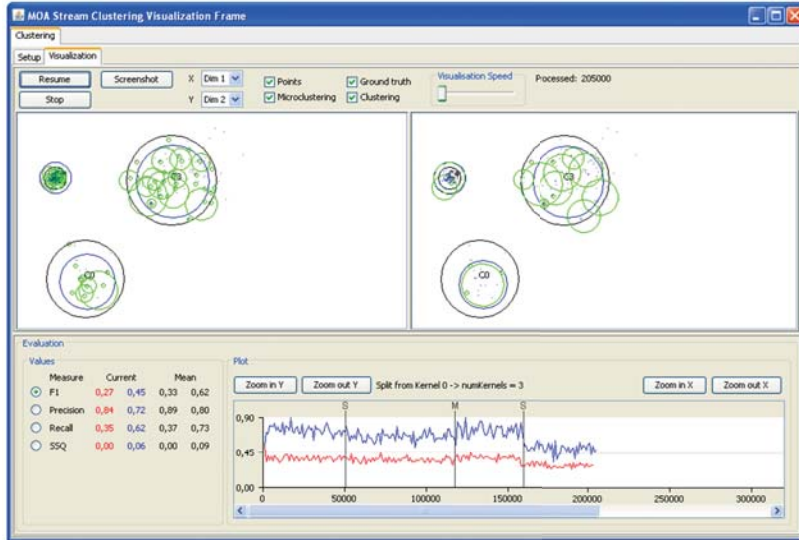


Fig. 4. Visualization tab of the clustering MOA graphical user interface.

4.4 Visualization and analysis

After the evaluation process is started, several options for analyzing the outputs are given: a) the stream can be stopped and the current (micro) clustering result can be passed as a data set to the WEKA explorer for further analysis or mining; b) the evaluation measures, which are taken at configurable time intervals, can be stored as a .csv file to obtain graphs and charts offline using a program of choice; c) last but not least both the clustering results and the corresponding measures can be visualized online within our framework.

Our framework allows the simultaneous configuration and evaluation of two different setups for direct comparison, e.g. of two different algorithms on the same stream or the same algorithm on streams with different noise levels etc.

The visualization component allows to visualize the stream as well as the clustering results, choose dimensions for multi dimensional settings, and compare experiments with different settings in parallel. Figure 4 shows a screenshot of our visualization tab. For this screen shot two different settings of the CluStream algorithm [2] were compared on the same stream setting (including merge/split events every 50000 examples) and four measures were chosen for online evaluation (F1, Precision, Recall, and SSQ). The upper part of the GUI offers options to pause and resume the stream, adjust the visualization speed, choose the dimensions for x and y as well as the components to be displayed (points, micro- and macro clustering and ground truth). The lower part of the GUI displays the measured values for both settings as numbers (left side, including mean values) and the currently selected measure as a plot over the arrived

examples (right, F1 measure in this example). For the given setting one can see a clear drop in the performance after the split event at roughly 160000 examples (event details are shown when choosing the corresponding vertical line in the plot). While this holds for both settings, the left configuration (red, CluStream with 100 micro clusters) is constantly outperformed by the right configuration (blue, CluStream with 20 micro clusters). A video containing an online demo of our system can be found at our website along with more screenshot and explanations. (cf. Section 5).

5 Website, Tutorials, and Documentation

MOA is open source and released under the GNU GPL License. It can be downloaded at:

<http://moa.cs.waikato.ac.nz/>

The website includes a tutorial, an API reference, a user manual, and a manual about mining data streams. Several examples of how the software can be used are available. Additional material regarding the extension of MOA to stream clustering can be found at

<http://dme.rwth-aachen.de/moa-datastream/>

The material includes a live video of the software as well as screenshots and explanations for the most important interfaces that are needed for extending our framework through novel data feeds, algorithms or measures.

6 Conclusions

Our goal is to build an experimental framework for classification and clustering on data streams similar to the WEKA framework. Our stream learning framework provides a set of data generators, algorithms and evaluation measures. Practitioners can benefit from this by comparing several algorithms in real world scenarios and choosing the best fitting solution. For researchers our framework yields insights into advantages and disadvantages of different approaches and allows the the creation of benchmark streaming data sets through stored, shared and repeatable settings for the data feeds. The sources are publicly available and are released under the GNU GPL license. Although the current focus in MOA is on classification and clustering, we plan to extend the framework to include regression, and frequent pattern learning [6].

7 Acknowledgments

This work has been supported by the UMIC Research Centre, RWTH Aachen University, Germany.

References

1. M. R. Ackermann, C. Lammersen, M. Märtens, C. Raupach, C. Sohler, and K. Swierkot. StreamKM++: A clustering algorithm for data streams. In *SIAM ALENEX*, 2010.
2. C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *VLDB*, pages 81–92, 2003.
3. R. Agrawal, S. P. Ghosh, T. Imielinski, B. R. Iyer, and A. N. Swami. An interval classifier for database mining applications. In *VLDB '92*, pages 560–573, 1992.
4. A. Asuncion and D. Newman. UCI machine learning repository, 2007.
5. F. B. Baker and L. J. Hubert. Measuring the power of hierarchical cluster analysis. *Journal of the American Statistical Association*, 70(349):31–38, 1975.
6. A. Bifet. *Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams*. IOS Press, 2010.
7. A. Bifet and R. Gavaldà. Learning from time-changing data with adaptive windowing. In *SIAM International Conference on Data Mining*, 2007.
8. A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA: Massive Online Analysis <http://sourceforge.net/projects/moa-datastream/>. *Journal of Machine Learning Research (JMLR)*, 2010.
9. A. Bifet, G. Holmes, B. Pfahringer, and R. Gavaldà. Improving adaptive bagging methods for evolving data streams.
10. A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà. New ensemble methods for evolving data streams. In *15th ACM SIGKDD*, 2009.
11. L. Breiman et al. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.
12. M. Brun, C. Sima, J. Hua, J. Lowey, B. Carroll, E. Suh, and E. R. Dougherty. Model-based evaluation of clustering validation measures. *Pattern Recognition*, 40(3):807–824, 2007.
13. F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *SDM*, 2006.
14. T. Cover and J. Thomas. *Elements of Information Theory (2nd Edition)*. Wiley-Interscience, 2006.
15. P. Domingos and G. Hulten. Mining high-speed data streams. In *Knowledge Discovery and Data Mining*, pages 71–80, 2000.
16. S. Dongen. Performance criteria for graph clustering and markov cluster experiments. Technical report, Amsterdam, The Netherlands, The Netherlands, 2000.
17. J. Dunn. Well separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4:95–104, 1974.
18. D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, 1987.
19. E. Folkes and C. Mallows. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78:553–569, 1983.
20. J. Gama, R. Sebastião, and P. P. Rodrigues. Issues in evaluation of stream learning algorithms. In *15th ACM SIGKDD*, 2009.
21. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.
22. J. A. Hartigan. *Clustering Algorithms*. Wiley, New York, 1975.
23. L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2:193–218, 1985.

24. L. J. Hubert and J. R. Levin. A general statistical framework for assessing categorical clustering in free recall. *Psychological Bulletin*, 83(6):1072–1080, 1976.
25. G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *KDD'01*, pages 97–106, San Francisco, CA, 2001. ACM Press.
26. L. Kaufmann and P. Rousseeuw. *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley & Sons, 1990.
27. R. Kirkby. *Improving Hoeffding Trees*. PhD thesis, University of Waikato, November 2007.
28. P. Kranen, I. Assent, C. Baldauf, and T. Seidl. Self-adaptive anytime stream clustering. In *IEEE ICDM*, pages 249–258, 2009.
29. M. Meila. Comparing clusterings: an axiomatic view. In *ICML*, pages 577–584, 2005.
30. G. W. Milligan. An examination of the effect of six types of error perturbation on fifteen clustering algorithms. *Psychometrika*, 45(3):325–342, 1980.
31. E. Müller, I. Assent, S. Günemann, T. Jansen, and T. Seidl. OpenSubspace: An open source framework for evaluation and exploration of subspace clustering algorithms in weka. In *OSDM in conjunction with PAKDD*, pages 2–13, 2009.
32. E. Müller, I. Assent, R. Krieger, T. Jansen, and T. Seidl. Morpheus: Interactive exploration of subspace clustering. In *ACM KDD*, pages 1089–1092, 2008.
33. N. Oza and S. Russell. Online bagging and boosting. In *Artificial Intelligence and Statistics 2001*, pages 105–112. Morgan Kaufmann, 2001.
34. B. Pfahringer, G. Holmes, and R. Kirkby. New options for hoeffding trees. In *Australian Conference on Artificial Intelligence*, pages 90–99, 2007.
35. W. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66:846–850, 1971.
36. C. Rijsbergen. *Information Retrieval (2nd Edition)*. Butterworths, London, 1979.
37. F. J. Rohlf. Methods for comparing classifications. *Annual Review of Ecology and Systematics*, 5:101–113, 1974.
38. A. Rosenberg and J. Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *EMNLP*, pages 410–420, 2007.
39. J. C. Schlimmer and R. H. Granger. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986.
40. M. J. Song and L. Zhang. Comparison of cluster representations from partial second- to full fourth-order cross moments for data stream clustering. In *ICDM*, pages 560–569, 2008.
41. M. Spiliopoulou, I. Ntoutsi, Y. Theodoridis, and R. Schult. MONIC: modeling and monitoring cluster transitions. In *ACM KDD*, pages 706–711, 2006.
42. W. N. Street and Y. Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *KDD '01*, pages 377–382, New York, NY, USA, 2001. ACM Press.
43. L. Tu and Y. Chen. Stream data clustering based on grid density and attraction. *ACM Trans. Knowl. Discov. Data*, 3(3):1–27, 2009.
44. Y. Zhao and G. Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Machine Learning*, 55(3):311–331, 2004.

Incremental Option Trees for Handling Gradual Concept Drift

Elena Ikononovska¹, João Gama^{2,3}, and Sašo Džeroski¹

¹ Jozef Stefan Institute, Department of Knowledge Technologies
Jamova cesta 39, SI-1000 Ljubljana, Slovenia

² LIAAD/INESC - University of Porto
Rua da Ceuta, 118 - 6, 4050-190 Porto, Portugal

³ Faculty of Economics - University of Porto
Rua Roberto Frias, 4050-190 Porto, Portugal

Abstract. Data streams are inherently time-varying and exhibit various types of dynamics. The presence of concept drift in the data significantly influences the accuracy of the learner, thus efficient handling of non-stationarity is an important problem. In this paper, we address the problem of modeling the transition phases between consecutive concepts in gradual concept drift. In those transition phases, examples correspond to at least two different concepts. Learning a single model cannot be an efficient solution because it cannot represent all the examples. In addition, the growth of an incrementally induced tree will be severely affected due to the ambiguity in the data. Our hypothesis is that incremental option trees are a solution to these problems. Option trees compress several different models in one structure which enables representing several different concepts at the same time. To avoid excessive growth, option nodes are introduced only when splitting decisions are ambiguous. When the ambiguity has been resolved, the redundant options can be removed from the tree. To the best of our knowledge, no other work has explored option trees for regression in the streaming setting. We performed experimental evaluation of the proposed approach on real-world data and several simulated scenarios of concept drift. The experimental results support our hypothesis and motivate further research.

1 Introduction

In almost all real-life situations, the concepts we are trying to model may change during the course of learning, especially when data are analyzed over longer periods of time. This is an important issue when learning from data streams due to their dynamical nature. The changes in the hidden variables or context induce changes in the target concept which is known as concept drift [17]. In this paper we are mostly interested in the problem of learning under gradual concept drift, where the current concept is being gradually replaced with a new and different concept. In the transition phase between the old and the new concept, the examples are mixed and pertain with a variable probability to the interleaved concepts. Thus, a single model will not be able to achieve satisfactory accuracy.

While ensemble models are one of the most popular methods for efficient learning on non-stationary data [1][11][13][16], they are not able to give an explicit interpretation of the changes made to the model. Another disadvantage is that they are very prone to local concept drift, as noted in [16]. In this paper, we offer an alternative solution which possesses some of the characteristics of both ensembles and interpretable models like regression trees.

We propose option trees for regression on fast non-stationary data streams. Option trees are introduced to solve the ambiguity problem in the splitting decisions, especially in the transition phases between the shifting concepts. In this paper, we focus only on the regression task, though the method can also be extended to classification problems. We describe an algorithm named FIOT (Fast and Incremental Option Trees) that induces regression trees with option nodes from data streams.

The proposed algorithm can offer at any-time a set of trees compressed in an option tree. From the set of trees, a single 'best' tree can be composed using the locally 'best' sub-models. The predictions can be obtained by averaging or by using the best predictor. To the best of our knowledge, there does not exist any work in the literature that explores option trees for regression neither in the context of batch nor in the context of incremental learning.

FIOT uses an informed change detection method which triggers a re-growing process of the sub-models with erroneous predictions. The resulting model is easily interpretable and provides, in real-time, a description of the changes to the model. Although more work is necessary in realistic domains, and additional comparisons are needed with other drift tracking algorithms, these early results show the potential of using option trees for more efficient learning under gradual concept drift.

The rest of the paper is organized as follows. Section 2 surveys the related work. The algorithm is presented in Section 3. Section 4 describes the evaluation methodology, and Section 5 presents the experimental results. The last section concludes and gives some directions for future work.

2 Related Work

2.1 Option trees (for classification)

Option trees are similar to regular decision trees, except that they can contain *option* nodes in addition to regular decision and leaf nodes. The work of Kohavi [10] is the first which discusses option trees in their own right.

An option node implements an OR function, implying that an example can follow multiple paths at an *option* node and thus arrive at multiple leaves. In Figure 1 we give an example of a partially constructed option tree for the Winequality dataset from the UCI repository. The triangular nodes represent the option nodes. Each branch of an option tree leads to a different sub-tree with a split on a different attribute in its root node. The leaf nodes hold the prediction for the target attribute and the corresponding standard deviation. An example which

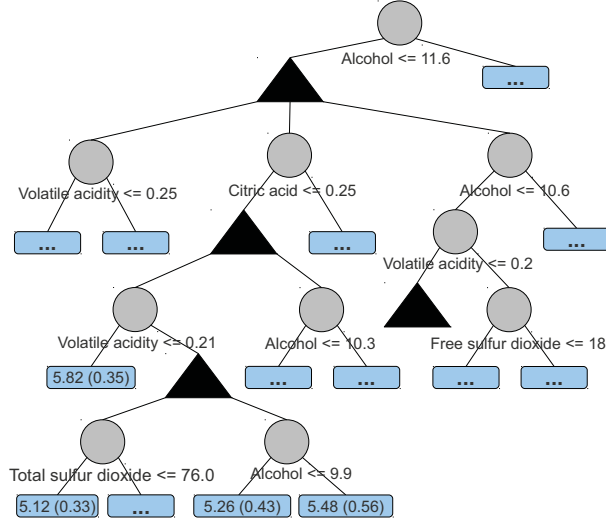


Fig. 1. An example of an option tree for the Winequality problem.

arrives to an option node can follow all of its branches, ending at multiple leaves. The usual strategy for combining these different predictions is averaging.

2.2 Option trees (for classification) on streams

Option nodes have been also used in the context of learning from data streams, as an extension of Hoeffding trees [4] for classification [14]. Although Hoeffding trees are more stable than batch tree learners, decisions are still subject to limited lookahead. This is the main motivation of Pfahringer et al. [14]. Their approach is somewhat different from the proposed batch ones, mainly because option nodes are not introduced in the split selection process, but only after a node has been transformed into an internal (decision) node. A new option can be introduced in an internal node if the Gain index of the best unused attribute is greater than the Gain index of the current one, for at least the value of the Hoeffding bound. The prediction strategy used is *weighted voting* as in [2], where the individual probability predictions of each class are summed.

Another paper [12] proposing a variant of option trees on streams uses an approach based on CVFDT [7] in which alternating trees are grown every time a new splitting test is being found as better than the existing one. The main difference is that the alternating trees are included as possible predictors for the succeeding examples. The prediction rule is to use the most recent option that offers the highest accuracy. An option node is thus reflecting the most recent and best performing knowledge.

The disadvantage is in the way option nodes are introduced. As in [14], this action is triggered after a re-evaluation step on predefined time intervals. To be

able to perform a re-evaluation, the sufficient statistics must be stored in all the internal nodes. The alternate trees will be invoked whenever several attributes evaluated in isolation appear as equally good or when a concept drift occurs. For this reason, a user cannot be sure whether the existence of several sub-models is due to ambiguity or a change in the concept. An additional disadvantage is the delayed response to any of these situations. Because the Hoeffding bound is conservative the new sub-models will be introduced with some delay, which will at least temporarily decrease the accuracy.

Our proposal is to use an explicit change detection method which will point out to the exact option or sub-tree whose accuracy has degraded due to a concept drift. A model that is not performing well will be removed or deactivated. In addition, the location of the change can be described by conjugating the tests in all the predecessor nodes. To deal with ambiguous splits, all the options should be introduced immediately. However, the most interesting use of the option nodes is for simultaneous learning of multiple concepts which is necessary when gradual concept drift occurs.

2.3 Regression trees on streams

In our previous work, we proposed an algorithm for inducing incremental regression trees from data streams with drift detection [9]. The algorithm FIRT-DD [9] is able to incrementally induce regression trees by looking at each example only once, in the order of their arrival. Splitting decisions are supported by using a bound on the probability of exceeding an error threshold. The only memory required is for storing the sufficient statistics in the leaves.

The drift detection method is based on the Page-Hinckley test employed in every node of the tree. The Page-Hinckley test uses two parameters (α and λ) which determine the level of sensitivity and robustness on false alarms. The parameter α corresponds to the minimal absolute value of the amplitude of the jump to be detected. It should be adjusted according to the standard deviation of the target attribute. Larger deviations will require larger values for α . On the other hand, large values can produce larger delays in detection. The optimal performance is achieved in combination with the value of λ . Increasing λ entails fewer false alarms, but might miss some changes. The drift is localized using the tree structure itself, thus the adaptation is local and only for the sub-trees that do not correspond to the current concept.

The FIRT-DD algorithm is the basis for the FIOT algorithm, which is described in the next section.

3 Fast and Incremental Option trees (FIOT)

We begin by a brief description of our induction algorithm and then discuss the relevant points in more detail. FIOT uses an incremental top-down method for building decision trees, which employs the Hoeffding bound [6] in both the splitting criterion and the criterion for introducing option nodes. This is discussed in more detail in sub-section 3.1.

```

1: procedure FIOT( $k, N_{min}$ )
2:   Begin with an empty Leaf (Root)
3:   repeat
4:     Read next example
5:     Traverse the example through the tree to a Leaf
6:     Update statistics in the Leaf
7:     if (Examples seen in a Leaf  $\bmod N_{min} = 0$ ) then
8:       Find best split per attribute
9:       Rank attributes using the split evaluation measure
10:      if (Splitting criterion is satisfied) then
11:        Make a split on the best attribute
12:        Make two new branches leading to (empty) Leaves
13:      else
14:        Introduce an Option node
15:        Make splits on all the attributes that rank highly enough
16:        if (The number of possible trees is greater than  $k$ ) then
17:           $T^* = FindWorstSubtree()$ 
18:          Remove  $T^*$  from the option node where it is rooted
19:          Reconstruct the tree if necessary
20:   until End of stream.

```

Fig. 2. Pseudo code for the FIOT algorithm.

The pseudo code of the algorithm is given in Figure 2. The algorithm starts with an empty leaf and reads examples from the stream in the order of their arrival. Each example is traversed to a leaf where the necessary statistics are updated. Given the first portion of instances ¹, the algorithm finds the best split for each attribute, then ranks the attributes according to the split evaluation measure given. If the splitting criterion is satisfied, a standard split is made on the best attribute, creating two new leaves, one for each branch of the split. Otherwise, an option node is introduced with splits on all the attributes that appear equally good as the best one. If the maximum number of trees k is reached, the worst option is being removed.

Newly arrived instances are passed down the branch of a decision node corresponding to the outcome of the test. In the case of an option node, newly arrived instances are passed down along all the options, until they reach a leaf where they are used to update the necessary statistics. The necessary statistics maintained in the leaves are blocks of $\sum y_k, \sum (y_k)^2$ per attribute value. For each numerical attributes we use an extended binary search tree, which enables sorting unique values on the fly and efficient one-pass computation of the variance reduction for every possible split point. More details can be found in [8].

There are three crucial parts of the algorithm: 1) the splitting criterion which integrates the criterion for introducing option nodes, 2) the prediction strategy and 3) the management of the tree, which includes maintaining a fixed number of

¹ Since a major difference is not expected to be observed after each consecutive example, attributes are ranked after consecutive chunks of N_{min} data points.

possible trees and memory management features . Each of them will be addressed in more detail in the following subsections.

3.1 Splitting Criterion

Let A_1, A_2, A_3, \dots be the ranked attributes obtained at step 9 in Procedure FIOT in Figure 2. Ranking of attributes is made using the variance reduction as a measure (as an alternative one may use the standard deviation reduction).

Let us further consider the ratio of the variance reduction $VR(\cdot)$ values of any two attributes (e.g. $VR(A_2)/VR(A_1)$) as a real-valued random variable r . Having a predefined range $R = 1$ for the values of the random variables, the Hoeffding bound can be used to obtain high confidence intervals for the true average (r_{true}) of the sequence of random variables $r_1, r_2, r_3, \dots, r_N$.

Thus, let now $\bar{r} = \frac{1}{N} \sum_{i=1}^N r_i$, and ε be the value of the Hoeffding bound, calculated using the following formula:

$$\varepsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2N}}. \quad (1)$$

where δ is a user predefined parameter which specifies the confidence in the calculation of the probability that $r_{true} - \varepsilon \leq \bar{r} \leq r_{true} + \varepsilon$.

If after observing N_{min} examples the following inequality holds

$$\bar{r} < 1 - \varepsilon \quad (2)$$

then $r_{true} < 1$, meaning that the best attribute observed over a portion of the stream is truly the best over the whole stream.

Due to the use of a probability bound in the split selection process incremental trees are less unstable than batch trees. However, experience has shown that this approach introduces a significant delay in the growth of the tree which in effect affects the any-time accuracy and on-time adaptation.

While lowering the confidence will enable faster splitting, it is not the most appropriate solution because it introduces erroneous decisions. As a solution we propose incremental option trees. Instead of waiting for more statistical evidence to be collected, an option node will enable splitting on all the competing attributes. In the same time, this will act as a several-steps lookahead strategy which will replace the greedy component in the search process ².

The procedure is the following: after observing N_{min} examples the splitting criterion is being examined. If inequality (2) is satisfied a normal split is performed. Otherwise, an option node is introduced with splits on all the attributes (A_i) for which the following inequality holds:

$$VR(A_i)/VR(A_1) > 1 - \varepsilon. \quad (3)$$

² Tree-building methods use one level of lookahead which prefers attributes that score high in isolation and may overlook combinations which may lead to better solutions globally.

The values of $VR(\cdot)$ can change with observing more examples, and some of the splits might be overrated at this point. Therefore, bad split decisions will be removed with the subsequent examples using an adaptive option tree management method.

3.2 Option Tree Management

An obvious shortcoming of the use of option trees is a possible excessive growth of the tree. For this reason, it is crucial to have some form of tree management, which will control the tree growth and the memory allocation. We propose an adaptive option tree management method that tracks the performance of all the sub-trees³ corresponding to the options in the tree. The base idea is to keep only the best performing sub-models in the tree evaluated over the most recent data.

Option nodes are introduced whenever there exist ambiguity in the splitting decision (correlated attributes, or existence of multiple concepts). This ambiguity can be always considered as temporal due to the dynamic nature of the streaming data. Thus, one should expect that the sub-trees rooted at the same option node will exhibit different performance over the time. For the specific case of gradual concept drift, as long as the drift is present all the sub-trees reflecting the different concepts will perform equally well. With the end of the transition phase between one group of concepts and another, those sub-trees which were modeling past concepts will start to degrade in performance. This observation enables to determine the moment and the method of removing redundant and bad-performing options.

However, removing the sub-trees that show significant degradation in performance relative to the other options will not constrain the size of the option tree in case of many correlated features. Thus, we additionally place a constraint on the maximum number of possible sub-trees in the option tree (k). The non-weighted sequential error [3] which is commonly used in an on-line evaluation is a pessimistic estimate because it accumulates all the errors from the beginning of the learning and the evaluation. The most efficient way to get a precise measure of the recent accuracy of the sub-trees is by using fading factors in the computation of the sum as proposed in [5], where it has been shown that the fading factor mimics the sliding window computation of any statistic.

The procedure is the following: when the maximum number of options k is reached, every time a new option node is introduced the algorithm evaluates all the existing sub-trees, and removes the worst-performing one. The evaluation criterion is a weighted prequential mean-squared error $wPreqMSE(T_i, t)$ computed locally for the sub-tree T_i at a time point t , by dividing the weighted prequential squared error $wPreqSE(T_i, t)$ by the weighted number of examples $wNum(T_i, t)$:

³ A sub-tree is a tree rooted at an internal decision node of the main tree. As such it corresponds to a region of the hyper-space of possible attribute values.

<pre> 1: procedure FINDBESTTREE(T) 2: if (T is option node) then 3: $o_1 \leftarrow \text{FirstOption}(T)$ 4: $T_{min} \leftarrow \text{FindBestTree}(o_1)$ 5: $min \leftarrow \text{ComputeTotalVar}(T^*)$ 6: for all $o \in \{\text{options}\}$ do 7: $T^* \leftarrow \text{FindBestTree}(o)$ 8: $err \leftarrow \text{ComputeTotalVar}(T^*)$ 9: if $err < min$ then 10: Update min and T_{min} 11: else 12: for all children of T do 13: $\text{FindBestTree}(\text{children})$ 14: return T_{min} </pre>	<pre> 1: procedure FINDWORSTSUBTREE(T) 2: $max \leftarrow 0$ 3: if (T is option node) then 4: for all $o \in \{\text{Options}(T)\}$ do 5: $T^* \leftarrow \text{SubtreeAtOption}(o)$ 6: $err \leftarrow w\text{PreqMSE}(T^*)$ 7: if ($err > max$) then 8: Update max and T_{max} 9: $\text{FindWorstSubtree}(T^*)$ 10: else 11: for all children of T do 12: $\text{FindWorstSubtree}(\text{children})$ 13: return T_{max} </pre>
---	--

Fig. 3. Pseudo codes of the procedures: a) FindBestTree(), b) FindWorstSubtree().

$$w\text{PreqMSE}(T_i, t) = \frac{w\text{PreqSE}(T_i, t)}{w\text{Num}(T_i, t)} \quad (4)$$

The weighted prequential squared error ($w\text{PreqSE}(T_i, t)$) is computed incrementally, by cumulating the squared prediction error of every example seen at the root of the sub-tree:

$$w\text{PreqSE}(T_i, t) = f \cdot w\text{PreqSE}(T_i, t - 1) + (y_t - r(t))^2 \quad (5)$$

where f is a fading factor, $r(t)$ denotes the prediction and y_t the true value for the example seen at time point t . Each example is first used for testing and then for training. The prediction from the tree T_i can be obtained by averaging or by using the best predictor if T_i contains option nodes.

The weighted number of examples $w\text{Num}(T_i, t)$ seen till the moment t in the tree T_i is calculated using:

$$w\text{Num}(T_i, t) = f \cdot w\text{Num}(T_i, t - 1) + 1 \quad (6)$$

The term weighted is used here due to the use of a fading factor f . Namely, if we recursively substitute the right-side member of the equation using the same formula, we will get sum of weighted squared errors. The weight for the oldest one will be f^t . At the beginning of the evaluation $w\text{PreqSE}(T_i, 0) = 0$ and $w\text{Num}(T_i, 0) = 0$. The value of f should be around 0.9997 (< 1).

The procedure *FindWorstSubtree* given in Figure 3 starting from the root recursively evaluates the $w\text{PreqMSE}$ of all the sub-trees in the tree (in a depth-first manner) and returns the worst option.

3.3 The Prediction Strategy

When an example arrives to an option node it may follow multiple ways and thus will fall in multiple leaves, each offering a different prediction. For those

situations we have considered two general options as prediction strategies: the prediction from the *best tree*, or an average of the possible predictions which is the most straightforward solution.

The first option requires a definition of the notion of *best tree*. We define as *best tree* the tree with the smallest sum of variances of all the leaves. Since the tree is built incrementally, the leaves are corresponding to the most recent examples from the stream. Thus the variance is representing the most recent confidence in the predictions. The procedure *FindBestTree* given in Figure 3 recursively evaluates all the possible sub-trees by comparing their total variance averaged over all of the leaves.

To compute the total variance of a tree a recursive procedure *ComputeTotalVar* is used, that gathers all the leaves of the sub-tree and sums their variances $Var(I_{leaf})$. To account for the size of the sub-tree T^* , the summed variance is then divided by the total number of leaves, as given below:

$$TotalVar(T^*) = \frac{1}{|T^*|} \sum_{leaf \in T^*} Var(I_{leaf}) \quad (7)$$

where $|T^*|$ denotes the size of the tree in terms of number of leaves.

If the sub-tree T^* contains option nodes, then in formula (7) we only use the leaves from the option with the smallest total variance. The variance in the leaves can be computed using the sufficient statistics maintained in each leaf, thus no storing of examples is necessary.

4 Evaluation Methodology

4.1 Datasets

In the evaluation we have used the ten biggest benchmark datasets from the UCI Machine Learning Repository. All of these datasets were used for the evaluation on stationary problems, because none of them has been collected with the aim of on-line learning or incremental analysis. To the best of our knowledge, there does not exist a benchmark real-world dataset suitable for evaluation of on-line regression learners designed for learning on massive streaming data.

For evaluation on non-stationary problems, we have simulated several different types of drift in the same way as in our previous work [9]. Using the Fried dataset, we simulate and study three scenarios of concept drift:

1. *Local expanding abrupt drift* (LA) - The concept drift appears in two distinct regions of the instance space. There are three points of abrupt change in the training dataset. At every consecutive change the region of drift is expanded.
2. *Global reoccurring abrupt drift* (GA) - The concept drift appears over the whole instance space. There are two points of concept drift. At the second point of drift the old concept reoccurs.
3. *Global and slow gradual drift* (GG) - The concept drift appears over the whole instance space. There are two points of concept drift. At each point of concept drift, examples from the new concept are being gradually introduced among

the examples from the old concept. After 100k examples the data represents only the new concept.

4.2 Experimental Setup

The incremental algorithm FIOT offers two options for predictions. We will use the following acronyms for the different predicting strategies: FIOT-A (Option Tree with Averaging), and FIOT-BT (Option Tree with Best Tree). For FIOT we set parameter δ to 0.000001, and N_{min} to 200. We set the maximum number of trees k to 50. For the change detection we set parameter α to 0.005 and λ to 50. In the comparison, we also used the incremental algorithm OnlineRD for learning model trees from [15], the incremental algorithm FIRT for learning regression trees from [8], and the incremental algorithm FIRT-DD with drift detection from [9].

4.3 Experimental Results

The experimental evaluation starts with a comparison of the predictive accuracy of an option tree with an incremental regression tree built using the FIRT algorithm. Table 1 gives the results of a 10-fold cross-validation procedure over ten stationary real-world datasets.

The results show that the option tree manages to improve the accuracy of the regression tree due to its ability for faster learning. The results prove that the criterion for introducing options is effective and includes only the relevant attributes. The FIOT-A algorithm has slightly better accuracy than FIOT-BT. Note that the number of leaves for the FIOT-A algorithm is the total number of leaves in all the options, while for FIOT-BT it is the number of leaves in the best tree. The Friedman statistical test with the Bonferroni-Dunn post-hoc test (at $p=0.10$) shows a statistically significant improvement of the accuracy of FIRT by both versions of the option tree.

The second part of the evaluation uses the artificial datasets with simulated concept drift. Comparison is performed with the incremental algorithm FIRT-DD with drift detection and alternating trees as an adaptation strategy and the incremental algorithm OnlineRD [15]. The results are obtained over the last window of examples of size 10000 (used for learning) and are given in Table 2.

The results in Table 2 show that the option tree improves the accuracy of the model on the last window of examples only for the global slow and gradual concept drift. However, the improvement is substantial and supports our claims. The averaging of predictions gives again slightly better results. For the local abrupt concept drift, it seems that the adaptation of the model by regrowing with option nodes is too severe, considering the size of the trees produced. Namely, for the option trees we use only pruning, instead of growing alternate trees as in FIRT-DD, thus there are possible reactions to false alarms. We plan to further investigate the reasons for the lower accuracy of the FIOT algorithm. For the global abrupt concept drift the OnlineRD algorithm is substantially

Table 1. Results from 10-fold cross-validation on stationary real-world datasets. Comparison on RRMSE and size (number of leaves/rules).

Dataset	FIRT		FIOT-A		FIOT-BT	
	RRMSE	Size	RRMSE	Size	RRMSE	Size
Abalone	0.775 ± 0.029	9.3	0.746 ± 0.031	83.5	0.751 ± 0.029	13.6
Ailerons	0.564 ± 0.012	21.8	0.524 ± 0.013	77.6	0.533 ± 0.025	45.1
Cal Housing	0.621 ± 0.022	50.1	0.599 ± 0.035	83.2	0.603 ± 0.031	68.2
Elevators	0.682 ± 0.023	34.1	0.729 ± 0.019	53.2	0.735 ± 0.018	26.1
House 8L	0.670 ± 0.019	53.2	0.659 ± 0.016	77.4	0.661 ± 0.017	63.5
House 16H	0.770 ± 0.016	49.4	0.755 ± 0.017	88.0	0.760 ± 0.021	72.3
Mv Delve	0.091 ± 0.005	97.2	0.069 ± 0.002	147.7	0.069 ± 0.002	133.7
Pol	0.324 ± 0.015	34.0	0.288 ± 0.023	66.7	0.287 ± 0.019	36.6
Wind	0.604 ± 0.017	11.8	0.562 ± 0.019	42.6	0.573 ± 0.017	22.4
Winequality	0.865 ± 0.016	11.4	0.855 ± 0.019	48.5	0.864 ± 0.019	14.8

Table 2. Results over the last 10000 examples used for learning on the non-stationary artificial datasets. Comparison on the sequential MSE and size (number of leaves).

	FIRT-DD		FIOT-A		FIOT-BT		OnlineRD	
	MSE	Leaves	MSE	Size	MSE	Size	MSE	Size
LA	3.17 ± 0.07	1840.7	3.94 ± 0.34	721.2	4.03 ± 0.32	692.7	5.95 ± 1.09	173.6
GA	3.58 ± 0.06	629.2	3.69 ± 0.12	606.5	3.71 ± 0.13	556.3	1.21 ± 0.002	177.9
GG	3.56 ± 0.06	528.3	2.84 ± 0.08	461.8	2.86 ± 0.08	408.2	4.35 ± 3.53	241.8

better. However, it should be noted that OnlineRD is a model tree learner and the other algorithms are regression tree learners.

5 Conclusions

In this paper, we propose incremental option trees for regression on fast non-stationary data streams. The option nodes are introduced in order to improve the bias management in incremental learners, introduce ambiguous splits for learning under gradual concept drift and enable faster growth without an instability in the splitting decisions. We have shown that the option tree is able to achieve better accuracy faster than a regular regression tree. This is especially pronounced for the data with gradual concept drift. Option nodes act as an improved look-ahead strategy and present an interpretable version of ensemble methods.

As future work, we would like to investigate more strategies for combining predictions or choosing the best predictor. Another interesting problem is to employ the change detection mechanism over the different prediction strategies and in that way enable dynamic real-time decision-making mechanism that can suggest which predictions should be used at a given point in time. We also plan to evaluate our ideas on non-stationary real-world streaming problems, where an interpretable model can be useful.

References

1. C. Alippi, G. Boracchi, and M. Roveri. Just in time classifiers: Managing the slow drift case. *Neural Networks, IEEE - INNS - ENNS International Joint Conference on*, 0:114–120, 2009.
2. Wray Buntine. Learning classification trees. *Statistics and Computing*, 2:63–73, 1992.
3. Philip Dawid. Statistical theory: The prequential approach. with discussion. *Journal of Royal Statistical Society A*, 147:278–292, 1984.
4. Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *KDD*, pages 71–80, 2000.
5. João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. Issues in evaluation of stream learning algorithms. In John F. Elder IV, Françoise Fogelman-Soulié, Peter A. Flach, and Mohammed Javeed Zaki, editors, *KDD*, pages 329–338. ACM, 2009.
6. Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.
7. Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106, New York, NY, USA, 2001. ACM.
8. Elena Ikonomovska and João Gama. Learning model trees from data streams. In Jean-François Boulicaut, Michael R. Berthold, and Tamás Horváth, editors, *Discovery Science*, volume 5255 of *Lecture Notes in Computer Science*, pages 52–63. Springer, 2008.
9. Elena Ikonomovska, João Gama, Raquel Sebastião, and Dejan Gjorgjevik. Regression trees from data streams with drift detection. In *DS '09: Proceedings of the 12th International Conference on Discovery Science*, pages 121–135, Berlin, Heidelberg, 2009. Springer-Verlag.
10. Ron Kohavi and Clayton Kunz. Option decision trees with majority votes. In *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*, pages 161–169, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.
11. J. Zico Kolter and Marcus A. Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *J. Mach. Learn. Res.*, 8:2755–2790, 2007.
12. Jing Liu, Xue Li, and Weicai Zhong. Ambiguous decision trees for mining concept-drifting data streams. *Pattern Recogn. Lett.*, 30(15):1347–1355, 2009.
13. Michael D. Muhlbaier and Robi Polikar. An ensemble approach for incremental learning in nonstationary environments. In *MCS'07: Proceedings of the 7th international conference on Multiple classifier systems*, pages 490–500, Berlin, Heidelberg, 2007. Springer-Verlag.
14. Bernhard Pfahringer, Geoffrey Holmes, and Richard Kirkby. New options for hoeffding trees. In Mehmet A. Orgun and John Thornton, editors, *Australian Conference on Artificial Intelligence*, volume 4830 of *Lecture Notes in Computer Science*, pages 90–99. Springer, 2007.
15. Duncan Potts and Claude Sammut. Incremental learning of linear model trees. *Mach. Learn.*, 61(1-3):5–48, 2005.
16. Alexey Tsymbal, Mykola Pechenizkiy, Pádraig Cunningham, and Seppo Puuronen. Dynamic integration of classifiers for handling concept drift. *Inf. Fusion*, 9(1):56–68, 2008.
17. Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Mach. Learn.*, 23(1):69–101, 1996.

Handling concept drift in preference learning for interactive decision making

Paolo Campigotto, Andrea Passerini, and Roberto Battiti

DISI - Dipartimento di Ingegneria e Scienza dell'Informazione,
Università di Trento, Italy
{campigotto, passerini, battiti}@disi.unitn.it

Abstract. Interactive decision making methods use preference information from the decision maker during the optimization task to guide the search towards favourite solutions. In real-life applications, unforeseen changes in the preferences of the decision maker have to be considered. To the best of our knowledge, no interactive decision making technique has been explicitly designed to recognize and handle preference drift. This paper aims at covering this gap, by extending the Brain-Computer Evolutionary Multi-Objective Optimization (BC-EMO) algorithm to handle preference drift. BC-EMO is a recent multi-objective genetic algorithm. It exploits user judgments of couples of solutions to build incremental models of the user value function. The learnt model is used to refine the genetic population, generating the new individuals in the region of the Pareto front surrounding the favourite solution of the decision maker. The proposed extension of BC-EMO detects the changes of the user preferences by observing the decrease of prediction accuracy of the learnt model. The preference drift is jointly tackled by the BC-EMO learning phase, by a discounting policy for outdated training examples, and by the BC-EMO search phase, by encouraging diversification in the genetic population. Experimental results for a representative preference drift scenario are presented.

1 Introduction

Modeling real-world problems often generates optimization tasks involving multiple and conflicting objectives. Because the objectives are in conflict, a solution simultaneously optimizing all of them does not exist. The typical approach to multi-objective optimization problems (MOOPs) consists of searching for a set of trade-off solutions, called *Pareto-optimal set*, for which any single objective cannot be improved without compromising at least one of the other objectives.

Usually, the size of the Pareto-optimal set is large or infinite and the decision maker (DM) cannot tackle the overflow of information generated when analyzing it entirely. In this scenario, *interactive* decision making (IDM) techniques come to the rescue. They assume that the optimization expert (or the optimization software) cooperates with the DM. Through the interaction, the search process can be directed towards the DM preferred Pareto-optimal solutions and only a fraction of the Pareto-optimal set needs to be generated.

To the best of our knowledge, current IDM techniques consider a static preference model for the DM. This is rather unrealistic in many applications, where the DM has limited initial knowledge of the problem at hand. Only when the DM see the actual tentative solutions, she becomes aware of “what is possible”. Confronted with this new knowledge, her preferences may evolve. Typical scenarios involve a DM introducing new objectives in her preference model during the search, changing the relations between the different objectives or adjusting her preference model according to the observed limitations of the feasible set. Furthermore, the DM may not be aware of her preference changes and may not explicitly alert the optimization component. From a learning perspective, interactive multi-objective optimization should thus be seen as a joint learning process involving the model and the DM herself [2].

In the machine learning (ML) community, the problem of learning in these changing conditions is known as learning under *concept drift* [13]. The problem has received increasing attention in last years, and a number of solutions have been proposed to tackle it. For a review of the recent approaches in this area, see [16]. In this work we consider concept drift in the specific setting of interactive optimization. We call *preference drift* the tendency of the decision maker to change her preferences during the interactive optimization stage.

To the best of our knowledge, no IDM technique has been explicitly designed to handle preference drift. Among the plethora of IDM algorithms, reference point methods [10, 12], which iteratively minimize the distance to ideal reference points provided by the DM, could in principle naturally handle preference drifts. However, the cognitive demands required to the DM can easily become prohibitive, especially when dealing with non-linear preference models and an increasing number of objective.

Machine learning techniques [14, 15, 8] have been employed in IDM by learning the user preferences in an interactive fashion, and can be easily adapted to deal with preference drifts. Most existing approaches are limited either by not guaranteeing the generation of Pareto optimal solutions, or by assuming a linear set of weights, one for each objective. We recently developed the Brain-Computer Evolutionary Multi-Objective Optimization (BC-EMO) algorithm [1] in order to overcome these limitations.

BC-EMO is a genetic algorithm that learns the preference information of the decision maker (formalized as a value function) by the feedback received when the DM evaluates tentative solutions. Based on this feedback, the predicted value function is refined, and it is used to modify the fitness measure of the genetic algorithm. The algorithm was shown [1] to early converge to the desired solution on both combinatorial and continuous problems with linear and non-linear value functions. The learning stage is based on a support vector ranking algorithm which provides robustness to inaccurate and contradictory DM feedback [3]. We thus selected BC-EMO as a natural candidate to be extended for managing preference drift.

The extension of BC-EMO for preference drift recovery is based on the approach of instance weighting [9], a popular strategy in the concept drift literature.

The instance weighting technique consists of reweighting the examples according to their predicted relevance for the current concept. We include this reweighting scheme in the learning component of the BC-EMO algorithm. A change detection monitor is responsible for activating the mechanism. In order to deal with concept drift in the specific setting of interactive optimization, we also introduce a diversification strategy aimed at escaping from minima which could become suboptimal for the changed preference of the DM.

The remainder of the paper is organized as follows. Section 2 introduces IDM and discusses the limitations of current techniques in regard to preference drift handling. Section 3 briefly reviews the BC-EMO algorithm, while Section 4 extends it to automatically handle preference drift. An experimental evaluation of the proposed extension is reported in Section 5. Section 6 draws some conclusions and proposes possible directions for future research.

2 Interactive decision making techniques

A MOOP can be stated as:

$$\begin{aligned} & \text{minimize } \mathbf{f}(\mathbf{x}) = \{f_1(\mathbf{x}), \dots, f_m(\mathbf{x})\} & (1) \\ & \text{subject to } \mathbf{x} \in \Omega \end{aligned}$$

where $\mathbf{x} \in \mathbb{R}^n$ is a vector of n decision variables; $\Omega \subset \mathbb{R}^n$ is the *feasible region* and is typically specified as a set of constraints on the decision variables; $\mathbf{f} : \Omega \rightarrow \mathbb{R}^m$ is made of m objective functions which need to be jointly minimized. Objective vectors are images of decision vectors and can be written as $\mathbf{z} = \mathbf{f}(\mathbf{x}) = \{f_1(\mathbf{x}), \dots, f_m(\mathbf{x})\}$. Problem 1 is ill-posed whenever objective functions are conflicting, a situation which typically occurs in real-world applications. In these cases, an objective vector is considered optimal if none of its components can be improved without worsening at least one of the others. An objective vector \mathbf{z} is said to *dominate* \mathbf{z}' , denoted as $\mathbf{z} \succ \mathbf{z}'$, if $z_k \leq z'_k$ for all k and there exist at least one h such that $z_h < z'_h$. A point $\hat{\mathbf{x}}$ is Pareto-optimal if there is no other $\mathbf{x} \in \Omega$ such that $\mathbf{f}(\mathbf{x})$ dominates $\mathbf{f}(\hat{\mathbf{x}})$. The set of Pareto-optimal points is called *Pareto set* (PS). The corresponding set of Pareto-optimal objective vectors is called *Pareto front* (PF).

Several IDM approaches have been developed to aid the DM in identifying her preferred solution [11], including evolutionary multi-objective algorithms (see for example [5] and contained references). IDM procedures exploit the preference feedback from the DM to refine a preference model, usually expressed as a value function.

In the popular family of reference point methods [10, 12] the value function is interpreted as an achievement scalarizing function, which measures the distance from a selected objective vector $\bar{\mathbf{z}}$, called *reference point*. The reference point specifies the desirable values of the objectives and it is usually provided by the DM. The distance from the reference point has a preferential meaning: the

tentative solution $\mathbf{x}^* \in \Omega$ showed to the DM is the solution minimizing the deviation from the reference point. In detail, the solution \mathbf{x}^* is obtained by solving the following program:

$$\begin{aligned} \mathbf{x}^* = \min & \max_{k=1 \dots m} [w_k(f_k(\mathbf{x}) - \bar{z}_k)] \\ & \text{subject to } \mathbf{x} \in \Omega \end{aligned} \quad (2)$$

with weight $w_k > 0, k = 1 \dots m$. The achievement scalarizing function to minimize in Eq. 2 is the weighted Tchebychev distance from the reference point. The DM can express her bias for the k -th objective by assigning a value to weight w_k . After the DM has specified her desirable solution as a reference point, she can see what was feasible (the solution \mathbf{x}^*) and in case provide a new reference point. Many refinements and extensions of this approach exist [12]. They consider different ways of interaction with the DM (e.g., by showing a set of solution in the neighborhood of \mathbf{x}^*) and different refinements of the achievement scalarizing function, designed to obtain Pareto-optimal solutions with particular properties.

In principle, reference points approaches could be considered a natural way of accounting for preference drift: the DM is free to modify the reference point, exploring new regions of the Pareto front in response to a change in her preferences. However, the effort of the decision maker to modify the reference point when her preference model includes non-linear relations between the objectives may be prohibitive. The cognitive demands become unrealistic when the dimensionality of the problem increases, providing a large set of candidate directions to shift the reference point.

In the past, a number of works [14, 15, 8] introduced ML-based approaches to learn the user preferences in an interactive fashion. However, they have several limitations [1]. The approach in [14] does not guarantee the generation of Pareto optimal solutions, while the strategies developed in [15, 8] generate a linear local approximation of the user preferences and do not use directly the learned preference model to drive the search. Furthermore, in all these works the feedback from the DM is expressed in terms of quantitative scores.

The BC-EMO algorithm [1] overcomes these limitations, by learning the preference model with pairwise preference supervision, a much more affordable task for the DM, and by directly using the preference model to drive the search over the Pareto front. The algorithm does not make any assumption about the preference structure of the DM, possibly accounting for highly non-linear relations between the different objectives. This work extends BC-EMO to handle preference drift.

3 The BC-EMO algorithm

The goal of the BC-EMO algorithm consists of identifying the non-dominated solution preferred by the decision maker. To fulfill this scope, BC-EMO learns a value function from the preference information provided by the DM by using the support vector ranking [4], a supervised machine learning technique that learns

Algorithm 1 Training procedure at the generic i -th EMO iteration

- 1: **procedure** TRAIN(P_i, U_{i-1}, exa)
 - 2: $P_{tr} \leftarrow$ PREFORDER(P_i, U_{i-1}, exa)
 - 3: obtain pairwise preferences for P_{tr} from the DM
 - 4: sort P_{tr} according to user preferences and add it to training instances
 - 5: Choose best kernel K and regularization C by k-fold cross validation
 - 6: $U_i \leftarrow$ function trained on full training set with K and C
 - 7: $res_i \leftarrow$ k-fold cv estimate of function performance
 - 8: **return** U_i, res_i
 - 9: **end procedure**
-

to rank the input data. Training examples consist of pairwise comparisons of non-dominated solutions which are turned into ranking constraints for the learning algorithm. No specific assumptions are made about the form of the DM value function: BC-EMO has a tuning phase selecting the most appropriate kernel (i.e., similarity measure) in order to best approximate the targets, allowing it to learn an *arbitrary* value function provided enough data are available. Furthermore, support vector ranking allows to effectively deal with noisy training observations thanks to a regularization parameter C trading-off data fitting with complexity of the learned model.

The learned value function is used to rank the current population during the *selection* phase of the BC-EMO algorithm, where a sub-population is selected for reproduction on the basis of fitness (i.e., quality of the solutions). In particular, the BC-EMO selection procedure, which we will refer to as PREFORDER, consists of:

1. collecting the subset of non-dominated individuals in the population;
2. sorting them according to the learned value function;
3. appending to the sorted set the result of repeating the procedure on the remaining dominated individuals, until the desired number of individuals is reached.

The procedure is guaranteed to retain Pareto-optimality regardless of the form of the learned value function. Any evolutionary multi-objective algorithm (EMOA) that needs comparisons between candidate individuals can be equipped with the BC-EMO selection procedure (replacing or integrating the original selection procedure). Algorithm 1 describes the procedure of the generic i -th training iteration, in which: 1) the *exa* best individuals from the current population P_i are selected according to PREFORDER with current value function U_{i-1} ; 2) DM feedback is collected for these examples; 3) parameter selection, training and evaluation are conducted on the training data enriched with P_{tr} . This procedure will be modified in the next section in order to account for preference drifts.

The overall BC-EMO approach consists of three steps:

1. *initial search phase*: the *plain* EMOA selected is run for a given number of generations and produces a final population P_1 ;

2. *training phase*: using P_1 as initial population, a specific number of training iterations are executed to learn the value function V by interacting with the DM. The final population obtained (P_2) is collected;
3. *final search phase*: the selected EMOA equipped with the BC-EMO selection procedure is run for a given number of generations, using P_2 as initial population and producing the final ordered population.

Each training iteration alternates a refinement phase, where the DM is queried for feedback on candidate solutions and the value function is updated according to such feedback, with a search phase, where the EMOA equipped with the BC-EMO selection procedure is run for a given number of iterations. The training phase is executed until the maximum number of training iterations or the desired accuracy level are reached.

The parameters of the BC-EMO algorithm are: the number of allowed training iterations ($maxit$), the number of training individuals for iteration (exa), the number of generations before the first training iteration (gen_1) and between two successive training iterations (gen_s). Algorithm 2 contains the pseudocode of the BC-EMO approach applied on top of a generic EMO algorithm. Further details on the algorithm can be found in [1].

Algorithm 2 The BC-EMO algorithm

```

1: procedure BC-EMO( $maxit, exa, gen_1, gen_s$ )
2:    $res \leftarrow 0, it \leftarrow 0, U \leftarrow \text{RAND}$ 
3:   run the EMOA for  $gen_1$  generations
4:   collect last population  $P$ 
5:   while  $it \leq maxit$  do
6:      $U, res \leftarrow \text{TRAIN}(P, U, exa)$ 
7:     run the EMOA for  $gen_s$  generations guided PREFORDER with  $U$ 
8:     collect last population  $P$ 
9:   end while
10:  run the EMOA for the remaining generations guided PREFORDER with  $U$ 
11:  return the final population  $P$ 
12: end procedure

```

4 Handling preference drift with BC-EMO

The effect of preference drift is a decrease of the accuracy of the learnt model over time. In the original version of BC-EMO, training data arrives in batches over time and the model is re-trained every gen_s generations, when a new batch of training examples is available. The extension to handle preference drift consists of a mechanism for drift detection and of a reweighting of the past training examples inversely proportional to the observed decrease in the performance accuracy.

First, a cost in the range $[0, 1]$ is associated with each training example, initialized to the value one and defining the relevance of the example for the concept to predict. The detection of a drift in the preferences of the decision maker is based on the prediction accuracy of the learnt model. Let b_i and U_{i-1} the new batch of observable data and the current model at the generic i -th training iteration, respectively. The performance of the current model is the prediction accuracy $0 \leq res_{i-1} \leq 1$ over batch b_{i-1} . Furthermore, let $0 \leq res'_{i-1} \leq 1$ the prediction accuracy of the current model over batch b_i . If the difference between res_{i-1} and res'_{i-1} is bigger than a fixed threshold t_d , with $t_d > 0$, a drift in the preferences of the decision maker is assumed. In this case, the cost of the training examples collected so far (i.e., the training examples of batches b_1, b_2, \dots, b_{i-1}) is decreased as a function of the value $res_{i-1} - res'_{i-1}$. In detail, the cost is updated by a multiplicative factor $d = c(res_{i-1} - res'_{i-1})$, where the function c is defined as follows:

$$c(x) = \begin{cases} 1 & \text{if } x \leq t_d \\ 1 - x & \text{if } t_d < x < 0.5 \\ 0 & \text{if } x \geq 0.5 \end{cases} \quad (3)$$

Let us comment. If the value $res_{i-1} - res'_{i-1}$ is bigger than the threshold and smaller than 0.5, the cost of the training examples is decreased by the normalized value of the difference $res'_{i-1} - res_{i-1}$. When the decrease of the performance accuracy over the last batch of observable data is bigger than value 0.5, the training examples of the previous batches are discarded (i.e., their cost becomes zero). The rationale for this choice is that a large decrease in the accuracy of the learnt model is seen as symptom of a radical change in the preferences of the DM, outdated training examples collected in previous iterations.

If a drift in the preferences of the user has been detected, the model selection phase is executed using only the data in the i -th batch rather than using all the collected examples, as in the original version of BC-EMO (algorithm 1, line 6). Furthermore, the genetic population of the EMOA is reinitialized.

Let res_i the prediction accuracy of the selected model over batch b_i . If res_i is smaller than threshold t_r , the selected model is discarded as it does not satisfy the minimal performance requirement, and all training examples of of batches $b_1 \dots b_{i-1}$ are discarded as well. The plain EMOA underlying BC-EMO will then be executed starting with a random population, until the next training iteration is reached. The rationale for this choice is the assumption that the poor performance of the selected model is caused by the collected training examples, localized in a region of the Pareto front that does not provide informative examples to learn the drift of the user preferences. The plain EMOA algorithm is executed to generate a population representing the whole Pareto front (without considering the preferences of the decision maker), in order to create more informative training examples at the next training iteration.

Algorithm 3 describes the modification of the training procedure at the generic i -th training iteration of BC-EMO to handle preference drift.

Algorithm 3 Training procedure to handle preference drift

```
1: procedure TRAIN( $P_i, U_{i-1}, \text{exa}, \text{res}_{i-1}, t_d, t_r$ )
2:    $P_{tr} \leftarrow \text{PREFORDER}(P_i, U_{i-1}, \text{exa})$ 
3:   obtain pairwise preferences for  $P_{tr}$  from the DM
4:    $b_i \leftarrow \text{sort } P_{tr}$  according to user preferences
5:   Add  $b_i$  to training instances
6:    $\text{res}'_{i-1} \leftarrow \text{test } U_{i-1}$  using  $b_i$ 
7:   if  $\text{res}_{i-1} - \text{res}'_{i-1} > t_d$  then
8:     Decrease costs of examples in  $b_1 \dots b_{i-1}$  according to (3) using  $t_d$ 
9:     Re-initialize  $P_i$  randomly
10:  end if
11:  Choose best kernel  $K$  and regularization  $C$  by k-fold cross validation
12:   $\text{res}_i \leftarrow$  k-fold cv estimate of function performance
13:  if  $\text{res}_i \geq t_r$  then
14:     $U_i \leftarrow$  function trained on full training set with  $K$  and  $C$ 
15:  else
16:     $\text{res}_i \leftarrow 0, U_i \leftarrow \text{RAND}$ 
17:  end if
18:  return  $U_i, \text{res}_i$ 
19: end procedure
```

5 Experimental results

The experimental evaluation is focused on demonstrating the effectiveness of the extension of BC-EMO to handle decision maker preference drift for a selected case study. Given this focus, we did not attempt to fine-tune non-critical parameters which were fixed for the experiment.

Following [1], BC-EMO has been applied on top of NSGA-II [6] EMOA. We chose a population size of 100, 2000 generations, probability of crossover equal to one and probability of mutation equal to the inverse of the number of decision variables. Concerning the learning task of BC-EMO, the number of initial generations (gen_1) was set to 200, while the number of generations between two training iterations (gen_s) was set to 100. Both 5 and 10 examples per training iteration are tested. The minimum performance requirement threshold t_r was set to 0.5, while a decrease of the performance greater than 10% ($t_d = 0.1$) triggers the procedure handling the preference drift.

The case study consists of the bi-objective version of DTLZ6 problem, taken from popular DTLZ suite [7]:

$$\begin{aligned} & \min_{x \in \Omega} (x) \\ & \Omega = \{x \mid 0 \leq x_i \leq 1 \forall i = 1, \dots, n\} \\ & f_1(x) = x_1, \dots, f_{m-1}(x) = x_{m-1}, \\ & f_m(x) = (1 + g(x_m))h(f_1, f_2, \dots, f_{m-1}, g) \\ & g(x_m) = 1 + (9/|x_m|) \sum_{x_i \in x_m} x_i \\ & h = m - \sum_{i=1}^{m-1} [(f_i/(1+g))(1 + \sin(3\pi f_i))] \end{aligned}$$

This problem is characterized by an highly disconnected Pareto front, with both convex and concave regions (Fig. 1 (left)).

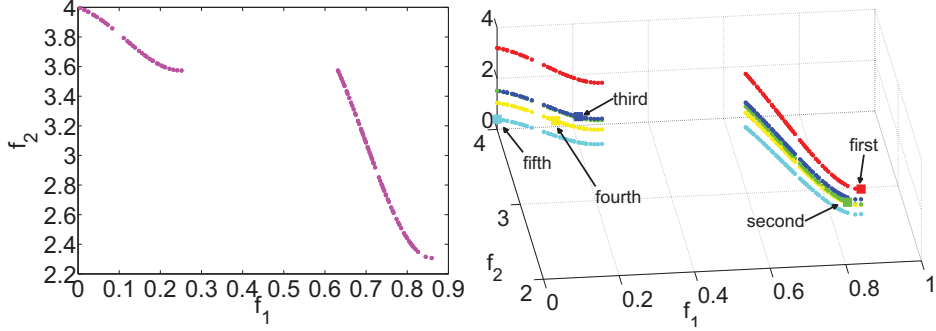


Fig. 1: Problem DTLZ6 with two objectives: (left) Pareto front for a sample run of plain NSGA-II without user preference; (right) preference values of the Pareto front according to the different values functions simulating the preference drift.

The drift in the preferences of the user is simulated by a sequence of five value functions:

1. $0.2 * f_1 + 0.8 * f_2$
2. $0.05 * f_2 * f_1 + 0.6 * f_1^2 + 0.38 * f_2$
3. $0.05 * f_2 * f_1 + 0.6 * f_1^2 + 0.38 * f_2 + 0.23 * f_1$
4. $0.05 * f_2 * f_1 + 0.68 * f_1^2 + 0.26 * f_2 + 0.23 * f_1$
5. $0.05 * f_2 * f_1 + 0.68 * f_1^2 + 0.1 * f_2 + 0.23 * f_1$

The sequence is generated by increasing the importance of the first objective (f_1) w.r.t. the second objective (f_2), assuming a non-linear formulation of the user preferences. This experimental setting simulates a decision maker that gradually becomes aware of the relation between her objectives to be optimized. Note that designing value functions which are non-monotonic in the Pareto front while retaining Pareto dominance properties is a non-trivial task. See [1] for a description of the generation process. Fig.1 (right) shows the different value functions considered, and their global minima over the Pareto front (square marked points). Tracking the shift of the global minimum between disconnected regions of the Pareto front is a challenging task for the optimization algorithm. The changes between the different value functions were fixed at generations 300, 600, 900 and 1200.

Fig. 2 and 3 report the results for the plain BC-EMO algorithm, for a baseline algorithm and for our BC-EMO extension, respectively, over the considered case study. The performance of the algorithms is measured in terms of percent approximation error w.r.t. the *gold standard* solution (y -axis) in function of the generation of the genetic population (x -axis). The gold standard solution is obtained by guiding the algorithm with the true value function. Each graph reports

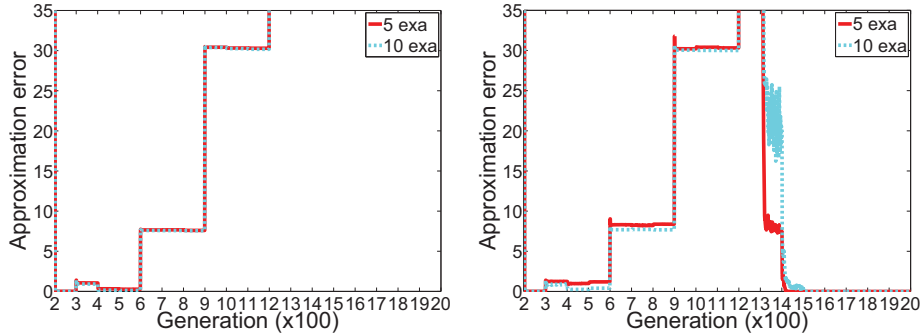


Fig. 2: Performance of the BC-EMO algorithm (left) and of the baseline algorithm (right).

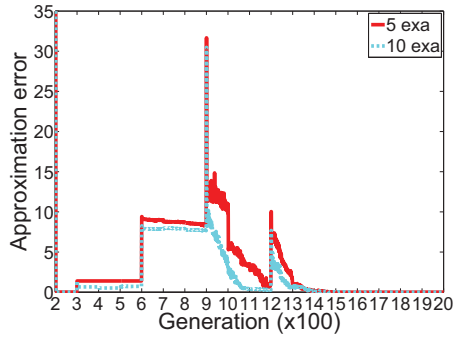


Fig. 3: Performance of the extension of BC-EMO to handle preference drift.

two learning curves for an increasing number of training examples per iteration (*exa*). Results are the medians over 500 runs with different random seeds for the search of the evolutionary algorithm.

At the generic i -th training iteration, the baseline algorithm retrains the learnt model using only the i -th batch of observable data. This is the only difference with the plain BC-EMO. Experimental results not reported here due to space limitations show the better performance obtained by discarding the previous training examples rather than discounting their cost by a fixed multiplicative factor in $(0, 1)$.

Fig. 2 (left) shows that the original version of BC-EMO cannot handle preference drift. The algorithm cannot track the changes of the user preferences: with the exception of the first drift, the performance of the algorithm keeps degrading each time the value function changes. After the last drift of the user preferences, the percent approximation error exceeds value 35%. This sub-optimal performance is caused by the lack of diversification during the search phase of the algorithm: the genetic population “gets trapped” in the region surrounding the global minima of the first and the second value functions.

A better performance is showed by the baseline algorithm (Fig. 2 (right)). Like BC-EMO, with 10 examples per iteration, a successful recover from the first drift is shown. The baseline algorithm fails to detect the second and the third changes of the value function (at generation 600 and 900, respectively): between generations 900 and 1200, the curves for both 5 and 10 training examples show a constant percentage deviation from the gold solution greater than 30%. When the fourth concept drift happens, the worst performance is observed. However, after generation 1400 the approximation error rapidly becomes zero, with both 5 and 10 training examples per iteration. Three iterations are required for perfect recovery from the fourth concept drift.

As expected, the best results are observed for the extension of BC-EMO designed for handling preference drift. Even if three training iteration are not enough for the perfect recovery from the second concept drift, the favourite solution of the decision maker generated by her third preference drift is perfectly identified. In the case of 10 training examples per iteration, an approximation error smaller than 1% is obtained at generation 1100. An even faster recovery is observed from fourth concept drift: two training iteration are required to approximate the new gold solution within an 1% approximation error. Note that, with the exception of the peak at generation 900 (corresponding to the third preference drift), the results tend to remain within 10% of the gold solution when 10 examples per iteration are provided.

6 Conclusion

This work addresses the problem of handling evolving preferences in interactive decision making. We modify BC-EMO, a recent multi-objective genetic algorithm based on pairwise preferences, by adapting its learning stage to learn under a concept drift. Our solution relies on the popular approach of instance weighting, in which the relative importance of examples is adjusted according to their predicted relevance for the current concept. We integrate these modifications with a diversification strategy favouring exploration as a response to changing DM preferences. Experimental results on a benchmark MOO problem with non-linear user preferences show the ability of the approach to early adapt to concept drifts.

Our promising preliminary results leave much room for future work. First, additional benchmark problems with evolving non-linear user preferences will be generated, possibly derived from real-world applications. Both sudden and gradual preference drift will be considered. Furthermore, active learning approaches could be devised in order to reduce the number of answers to the DM. This requires a shift of paradigm with respect to standard active learning strategies, in order to model the relevant areas of the optimization surface rather than reconstruct it entirely, and early detect and adapt to a changing surface.

References

1. Battiti, R., Passerini, A.: Brain-computer evolutionary multi-objective optimization (BC-EMO): a genetic algorithm adapting to the decision maker. *IEEE Transactions on Evolutionary Computation* (2010, to appear)
2. Belton, V., Branke, J., Eskelinen, P., Greco, S., Molina, J., Ruiz, F., Słowiński, R.: Interactive multiobjective optimization from a learning perspective. In: *Multiobjective Optimization: Interactive and Evolutionary Approaches*, pp. 405–433. Springer-Verlag, Berlin, Heidelberg (2008)
3. Campigotto, P., Passerini, A.: Adapting to a realistic decision maker: experiments towards a reactive multi-objective optimizer. In: *LION IV: Learning and Intelligent Optimization Conference*, Venice, Italy, Jan 18–22, 2010. *Lecture Notes in Computer Science*, Springer Verlag (2010, to appear)
4. Collins, M., Duffy, N.: Convolution kernels for natural language. In: *Advances in Neural Information Processing Systems 14*. pp. 625–632. MIT Press (2001)
5. Deb, K.: *Multi-objective optimization using evolutionary algorithms*. Wiley (2001)
6. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 182–197 (2000)
7. Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable multi-objective optimization test problems. In: *Congress on Evolutionary Computation (CEC2002)*. pp. 825–830 (2002)
8. Huang, H.Z., Tian, Z.G., Zuo, M.J.: Intelligent interactive multiobjective optimization method and its application to reliability optimization. *IIE Transactions* 37(11), 983–993 (2005)
9. Klinkenberg, R., Rüping, S.: Concept drift and the importance of examples. In: *Text Mining Theoretical Aspects and Applications*. pp. 55–77. Physica-Verlag (2002)
10. Miettinen, K.: *Nonlinear Multiobjective Optimization*, International Series in Operations Research and Management Science, vol. 12. Kluwer Academic Publishers, Dordrecht (1999)
11. Miettinen, K., Ruiz, F., Wierzbicki, A.: Introduction to Multiobjective Optimization: Interactive Approaches. In: *Multiobjective Optimization: Interactive and Evolutionary Approaches*, pp. 27–57. Springer-Verlag Berlin, Heidelberg (2008)
12. Miettinen, K., Mäkelä, M.M.: On scalarizing functions in multiobjective optimization. *OR Spectrum* 24, 193–213 (2002)
13. Schlimmer, J., Granger, R.: Incremental learning from noisy data. *Mach. Learn.* 1(3), 317–354 (1986)
14. Sun, M., Stam, A., Steuer, R.: Solving multiple objective programming problems using feed-forward artificial neural networks: the interactive fann procedure. *Manage. Sci.* 42(6), 835–849 (1996)
15. Sun, M., Stam, A., Steuer, R.: Interactive multiple objective programming using tchebycheff programs and artificial neural networks. *Comput. Oper. Res.* 27(7-8), 601–620 (2000)
16. Žliobaitė, I.: Learning under concept drift: an overview. Tech. rep., Vilnius University, Faculty of Mathematics and Informatics (2009)

Cost-sensitive Boosting for Concept Drift

Ashok Venkatesan, Narayanan C. Krishnan, Sethuraman Panchanathan

Center for Cognitive Ubiquitous Computing,
School of Computing, Informatics and Decision Systems Engineering,
Arizona State University, Tempe 85281, Arizona, USA,
{ashok.venkatesan, narayanan.ck, panch}@asu.edu

Abstract. Concept drift is a phenomenon typically experienced when data distributions change continuously over a period of time. In this paper we propose a cost-sensitive boosting approach for learning under concept drift. The proposed methodology estimates relevance costs of ‘old’ data samples w.r.t. to ‘newer’ samples and integrates it into the boosting process. We experiment this methodology on usenet1 and accelerometer based activity gesture datasets. The results demonstrate that the cost-sensitive boosting approach significantly improves classification performance over existing algorithms.

Keywords: Cost-sensitive Boosting, Adaptive Boosting, Concept Drift

1 Introduction

Concept drift is a phenomenon typically experienced when data distributions change continuously over a period of time. In these scenarios, traditional machine learning frameworks could result in poor performance as they assume training and test data to be sampled from the same distribution. Re-training classifier models for adapting to changes in the data requires sufficient amount of labeled new data, which is often expensive. An alternative approach is to selectively use the labeled old instances along with the few labeled new instances for effectively classifying unseen new examples, termed as *instance-transfer*[1]. We henceforth refer to the outdated training data as *diff-distribution* data and the small amount of labeled new data as *same-distribution* data[4].

AdaBoost, as proposed by Freund and Schapire[2], is a well established algorithm that iteratively weighs samples to reduce the cumulative error of the strong classifier, in the process, allowing identification of the important examples in the training data. This principle of the boosting algorithm makes it suitable for use with instance-transfer methods. In this paper, we present such an approach that integrates cost items with AdaBoost, based on the work done in [3], for learning under concept drift. We experiment our algorithm on two different real world datasets, where concept drift is evident and compare the performance with that of AdaBoost and TrAdaBoost[4].

The rest of the paper is organized in the following manner. The next section discusses some of the related work done in this area. Section 3 elaborates on

the proposed algorithm. The experiment setup and evaluation procedures are described in Section 4 and the results obtained is analyzed in Section 5. Following that, the paper is concluded in Section 6.

2 Related Work

Our learning objectives and methodologies are similar to some boosting based algorithms that have been recently proposed. Dai et al.’s TrAdaBoost[4] treats the diff-distribution and same-distribution data separately by applying different re-weighting schemes. Regular AdaBoost is applied on the newer examples, with the weight update factor computed from the training error over the same examples. In contrast, the weights of misclassified old examples are, decreased by a constant factor in order to weaken their impact on the model. Such a weighting scheme, however, reduces the usefulness of the diff-distribution examples that may be of support for learning the harder same-distribution examples. This was also noted by Eaton et al.[5], who proposed a set-based boosting algorithm called TransferBoost.

TransferBoost[5] uses a hierarchical weight updating scheme and breaks instances into task based sets. Similar to TrAdaBoost, standard AdaBoost is run on the newer examples and weighted accordingly. The older examples are, however, weighted by augmenting the update factor with a heuristic measure called transferability, depending on the set each old example belongs to. Taking a cue from both these algorithms, the method proposed in this paper adapts AdaBoost for use with the same-distribution data. However, a separate instance level cost estimation procedure supported by a cost-sensitive boosting framework, based on Sun et al.’s work [3], is used for the actual knowledge transfer.

In general, cost-sensitive algorithms attempt at reducing the costs associated with classifying samples belonging to one class as another and find extensive use in handling class imbalance problems. However, at an algorithmic level, the provision of implementing adaptive instance-level re-weighting schemes make them attractive for use in handling concept drift.

3 Methodology

Formally, let $T_d = \{(x_i^d, y_i^d)\}_{i=1}^n$ and $T_s = \{(x_j^s, y_j^s)\}_{j=1}^m$ represent the labeled different and same-distribution data respectively. The objective is to learn a model that classifies unseen same-distribution data with minimum error, by training on T_s supplemented by relevant instances belonging to T_d . Our solution to this problem is centered around two heuristics namely (1) attaching weights or relevance costs C to T_d based on the estimated relevance w.r.t T_s and (2) applying separate boosting schemes on T_d and T_s .

3.1 Cost Estimation

The relevance of samples in T_d can be characterized based on the similarity of the instance distributions as well as the classification functions between the diff-

distribution and same-distribution datasets. Keeping this in mind, two straightforward techniques that intuitively integrate the similarities on the two fronts, were used for attaching costs to samples in T_d . The first approach computes a *relevance ratio* r_i between the euclidean distances of the i^{th} instance in T_d from T_s samples belonging to a different class and the same class respectively.

$$r_i = \frac{\sum_{j, y_i^d \neq y_j^s} \text{dist}(x_i^d, x_j^s)}{\sum_{j, y_i^d = y_j^s} \text{dist}(x_i^d, x_j^s)}$$

The resulting ratio obtained is then normalized to fit in the interval $[0, 1]$ for use in the boosting phase. As it can be observed, the weights computed gives increased importance to the data points in T_d that are close to samples in T_s . The second method is based on an approach proposed by Jiang et al.[6] for removing misleading source domain instances. It involves determining the probability p_i of classifying samples in T_d using a model θ_{T_s} trained only on T_s .

$$p_i = p_{T_s}(y = y_i^d | x_i^d; \theta_{T_s})$$

Thus, correctly classified samples in T_d will have high costs attached to them while the misclassified samples will be assigned lower costs. Being probability estimates, the cost computed is already normalized between $[0, 1]$.

3.2 Cost-sensitive Boosting

The boosting algorithm presented here is founded on the dual objectives of minimizing training error over T_s and reducing net misclassification costs over T_d . Retaining the basic AdaBoost algorithm for minimizing the training error over T_s , the weight update factor α_i^s is determined so as to focus on learning the harder same-distribution examples. On the other hand, to reduce the misclassification costs over T_d , the cost-sensitive boosting framework from [3] is applied with the relevance costs computed earlier. The weight update factor α_i^d is derived using one of the three algorithm schemes namely *AdaC1*, *AdaC2* and *AdaC3* given in Table 1. These re-weighting schemes prove to be a factor in differentiating this algorithm from others by continuing to use the error computed over T_d as the

Table 1. Weight update equations for the different Boosting schemes[3]

Algorithm	α_i^d	$D^{t+1}(x_i^d)$
<i>AdaC1</i>	$\frac{1}{2} \log \frac{1 + \sum_i y_i^d h_t(x_i^d) C_i D^t(x_i^d)}{1 - \sum_i y_i^d h_t(x_i^d) C_i D^t(x_i^d)}$	$\frac{D^t(x_i^d) \exp(-\alpha_i^d C_i h_t(x_i^d) y_i^d)}{Z_t}$
<i>AdaC2</i>	$\frac{1}{2} \log \frac{\sum_{i, y_i^d = h_t(x_i^d)} C_i D^t(x_i^d)}{\sum_{i, y_i^d \neq h_t(x_i^d)} C_i D^t(x_i^d)}$	$\frac{C_i D^t(x_i^d) \exp(-\alpha_i^d h_t(x_i^d) y_i^d)}{Z_t}$
<i>AdaC3</i>	$\frac{1}{2} \log \frac{\sum_i C_i D^t(x_i^d) + \sum_i y_i^d h_t(x_i^d) C_i^2 D^t(x_i^d)}{\sum_i C_i D^t(x_i^d) - \sum_i y_i^d h_t(x_i^d) C_i^2 D^t(x_i^d)}$	$\frac{C_i D^t(x_i^d) \exp(-\alpha_i^d C_i h_t(x_i^d) y_i^d)}{Z_t}$

Algorithm 1 Cost-sensitive Boosting for Concept Drift

Given: Labeled datasets T_d and T_s and the number of iterations T .

1. Compute a cost item $C_i \in [0, 1]$ for each instance $(x_i^d, y_i^d) \in T_d$.
2. Initialize weight vector $D^1(x^{s,d}) = 1/(n+m)$.
3. For $t = 1, \dots, T$:
 4. Train base learner using distribution D^t .
 5. Obtain hypothesis $h_t : X \rightarrow Y; Y \in \{-1, 1\}$.
 6. Calculate weighted errors, ϵ_t^s and ϵ_t^d on T_s and T_d respectively.
 7. Choose α_t^d (refer Table 1).
 8. Update weight vectors $D^{t+1}(x^d)$ augmented by the cost items C_i (refer Table 1).
 9. Set $\alpha_t^s = \frac{1}{2} \log\left(\frac{1-\epsilon_t^s}{\epsilon_t^s}\right)$.
 10. Update weight vectors $D^{t+1}(x_j^s) = \frac{D^t(x_j^s) \exp(-\alpha_t^s h_t(x_j^s) y_j^s)}{Z_t}$,
where Z_t normalizes D^{t+1} to a distribution.

Output: the hypothesis $H(x) = \text{sign}(\sum_{t=1}^T \alpha_t^s h_t(x))$.

base for determining α_t^d . This also alters the algorithm’s criteria for convergence by looking not only to classify samples in T_s correctly but also the important instances in T_d . In contrast to the cost-sensitive boosting approach described in [3], the proposed boosting framework applies cost-sensitivity selectively to samples in T_d alone.

4 Evaluation Design

Two datasets experiencing concept drift were used for evaluating the proposed algorithms. The first dataset consists of 5 cooking and eating activity gestures[8] captured from multiple users with the help of 2 tri-axial accelerometers strapped on the right hand wrist and elbow of each user. The data was collected under two different settings: laboratory and real-life. While the laboratory dataset contains samples recorded from repeated hand movements, the real-life dataset contains annotated samples of users making a glass of powdered drink and drinking it. For the experiments, the real-life dataset was split into four sets each carrying samples originating from one user. Four trials were run, trained on the laboratory data T_d and little real-life data T_s ($\approx 2\%$ of $|T_d|$), on each of these sets and the mean accuracy measured. Further details on the dataset can be found in [8].

The second dataset, usenet1¹ is based on the 20 Newsgroups collection and consists of documents labeled as *interesting* or *junk* according to users’ personal interests over time. To test on this dataset, the samples were split into five batches each containing 300 instances as specified in [7]. These batches were further paired as T_d and T_s and results obtained were averaged over five trials of random sampling of T_s . For example, ‘batch 1 vs 2’ refers to the case where samples in ‘batch 1’ form T_d and a random subset of ‘batch 2’ forms T_s . The trained model is tested on the remaining samples of ‘batch 2’.

¹ http://mlkd.csd.auth.gr/concept_drift.html

Table 2. Classification accuracy of the different boosting schemes over activity gesture and usenet1 datasets. For the usenet1 data, 1% of the new data is used for training

Data Set	AdaBoost	TrAdaBoost	Cost $C_i = r_i$			Cost $C_i = p_i$		
			AdaC1	AdaC2	AdaC3	AdaC1	AdaC2	AdaC3
User 1	75.46	78.28	77.46	86.13	76.31	76.35	88.09	79.16
User 2	83.33	85.42	87.50	87.50	87.50	84.38	87.50	86.46
User 3	61.90	61.90	61.90	66.67	64.29	61.90	66.67	63.10
User 4	63.73	60.85	65.77	80.42	70.62	64.81	75.54	65.69
batch 1 vs 2	27.97	28.04	36.76	34.73	27.70	47.43	33.31	29.46
batch 2 vs 3	20.88	48.51	66.55	59.12	59.46	66.62	58.72	58.72
batch 3 vs 4	25.34	46.69	55.68	46.89	50.34	55.07	55.14	57.09
batch 4 vs 5	21.96	29.19	34.05	33.38	34.32	41.55	25.54	31.89

The experiments compare the performance of the cost-sensitive boosting algorithms using the relevance ratio r_i and the classification probabilities p_i separately as costs against that of AdaBoost and TrAdaBoost. Linear SVM with the parameters $c = 1$ and $\epsilon = 0.001$ was used as the base classifier in all the boosting algorithms. In each case the maximum number of iterations was set to 100. Linear SVM was also used for computing the probability based cost p_i , with the LIBSVM[9] option of obtaining probability estimates.

5 Results and Discussion

Table 2 presents the classification accuracies obtained using the different boosting schemes to learn the above described datasets. It can be observed that the performance of AdaBoost is consistently poor compared to the other boosting algorithms indicating the merit of using separate weight updates for samples belonging to T_d and T_s . Even though all the cost-sensitive schemes in general show a better performance in comparison to AdaBoost and TrAdaBoost, the current results do not clearly indicate which modified boosting scheme works best. Taking into account both the costs, computed using relevance ratio and the probability estimates, AdaC1 seems to perform better among the three schemes on the usenet1 dataset, while AdaC2 works better for the activity gesture dataset. Also, due to the complexity of the cost factor introduced in the boosting schemes, it is hard to conclude the effectiveness of one scheme over another at this point and would require further investigations using well calibrated datasets.

Analysis of the performance of the different techniques with increasing number of samples in T_s (1%, 2%, 3%, 4%, 5%, 10%, 20% and 50%) was also conducted with the usenet1 dataset. Figure 1a and b illustrate the classification accuracy curve obtained for ‘batch 1 vs 2’ and ‘batch 2 vs 3’. The cost-sensitive algorithms referred to in the image used cost factors computed on probability estimates. It is interesting to note that for one scenario, the cost-sensitive boosting schemes perform significantly better than TrAdaBoost, while the other scenario reflects just the opposite. The ratio between the number of samples of both the

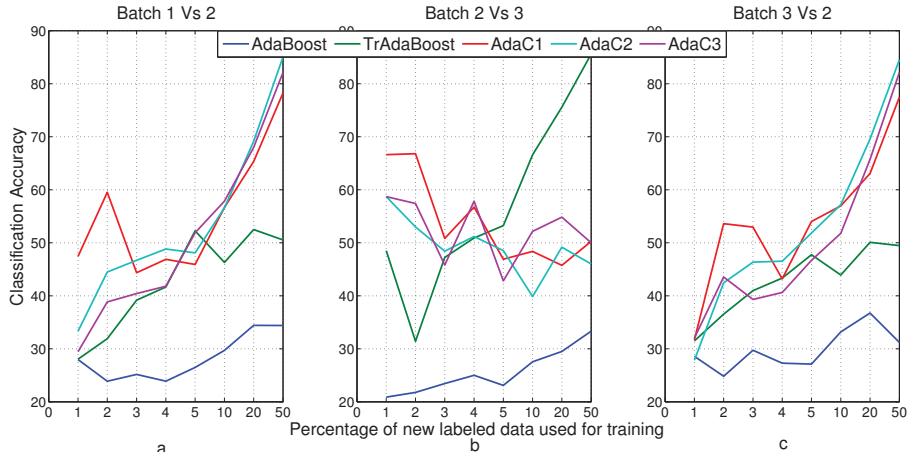


Fig. 1. Classification accuracy of the different boosting schemes using different number of new training samples on usenet1 data

classes reverses as we move from ‘batch 1’ to ‘batch 2’ and this class imbalance could have an adverse effect on the estimation of relevance cost as computed in the paper. The results for the different techniques for varying number of data samples in ‘batch 3 vs 2’ is presented in Figure 1c. Quite surprisingly, the performance of cost-sensitive boosting schemes improved significantly over ‘batch 2 vs 3’. We attribute this reversal in the trends of performance of TrAdaBoost and cost-sensitive boosting schemes to the reversal in the class imbalance in the usenet1 dataset.

6 Conclusion

This paper proposes cost-sensitive boosting techniques for learning under concept drift, where relevance cost factors associated with samples belonging to old data can be integrated into the training process. Two methods based on computing a relevance ratio and estimating classification probabilities were presented for computing cost factors. The proposed technique was evaluated against AdaBoost and TrAdaBoost approaches on usenet1 and accelerometer based activity gesture datasets. The results indicate in general a superior performance of the cost-sensitive boosting schemes over AdaBoost and TrAdaBoost. Experiments on calibrated datasets will be further conducted to understand the merits of the proposed approach for learning under concept drift. In addition, there also lies scope for exploring other methods of computing cost factors for use with the algorithm.

References

1. Pan, S.J. and Yang, Q.: A Survey on Transfer learning. IEEE Transactions on Knowledge and Data Engineering (2009)

2. Freund, Y. and Schapire, R.: A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Computational learning theory*, pp. 23–37. Springer (1995)
3. Sun, Y. and Kamel, M.S. and Wong, A.K.C. and Wang, Y.: Cost-sensitive Boosting for Classification of Imbalanced Data. *Pattern Recognition 12*, vol. 40, pp. 3358–3378. Elsevier (2007)
4. Dai, W. and Yang, Q. and Xue, G.R. and Yu, Y.: Boosting for Transfer Learning. *Proceedings of the 24th international conference on Machine learning*, pp. 200. ACM (2007)
5. Eaton, E. and desJardins, M.: Set-Based Boosting for Instance-Level Transfer. *IEEE International Conference on Data Mining Workshops*, pp. 422–428. IEEE (2009)
6. Jiang, J. and Zhai, C.X.: Instance weighting for domain adaptation in NLP. *Annual Meeting-Association For Computational Linguistics 1*, vol. 45, pp. 264. (2007)
7. Katakis, I. and Tsoumakas, G. and Vlahavas, I.: An Ensemble of Classifiers for coping with Recurring Contexts in Data Streams. *Proceeding of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, pp. 763–764. IOS Pres (2008)
8. Krishnan, N.C. and Juillard, C. and Colbry, D. and Panchanathan, S.: Recognition of hand movements using wearable accelerometers. *Journal of Ambient Intelligence and Smart Environments 2*, vol. 1, pp. 143–155. IOS Press (2009)
9. Chang, C.C. and Lin, C.J.: LIBSVM: a library for support vector machines (2001), Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>

An Adaptive Hybrid Recommender System that Learns Domain Dynamics

Fatih Aksel and Ayşenur Birtürk

Department of Computer Engineering, METU
{fatih.aksel,birturk}@ceng.metu.edu.tr

Abstract. Traditional hybrid recommender systems typically follow a manually created fixed prediction strategy in their decision making process. Experts usually design these static strategies as fixed combinations of different techniques. However, people’s tastes and desires are temporary and the gradually evolve. Moreover, each domain has unique characteristics, trends and unique user interests. Recent research has mostly focused on static hybridization schemes which do not change at runtime. In this paper, we describe an adaptive hybrid recommender system, called AdaRec that modifies its attached prediction strategy at runtime according to the performance of prediction techniques. Our approach to this problem is to use adaptive prediction strategies. Experiment results with datasets show that our system outperforms naive hybrid recommender.

1 Introduction

With the rapid development of the recommender system technology, a number of recommender engines have been developed for various application domains. Content-based and collaborative filtering are the two major recommendation techniques that have come to dominate the current recommender system. Content-based recommender system uses the descriptions about the content of the items (such as meta-data of the item), whereas collaborative filtering system tries to identify users whose tastes are similar and recommends items they like. Every recommendation approach has its own strengths and weaknesses. Hybrid recommender systems have been proposed to gain better results with fewer drawbacks [3].

Most of the recommender system implementations focuses on hybrid systems that use mixture of recommendation approaches [3]. This helps to avoid certain limitations of content-based and collaborative filtering systems. Previous research on hybrid recommender system has mostly focused on static hybridization approaches (strategy) that do not change their hybridization behavior at runtime. Fixed strategy may be suboptimal for dynamic domains&user behaviors. Moreover they are unable to adapt to domain drifts. Since people’s tastes and desires are transient and subject to change, a good recommender engine should deal with changing consumer preferences.

In this paper, we describe an *Adaptive Hybrid Recommender System*, called AdaRec, that modifies its switching strategy according to the performance of

prediction techniques. Our hybrid recommender approach uses adaptive *prediction strategies* that determine which *prediction techniques* (algorithms) should be used at the moment an actual prediction is required. Initially we used manually created rule-based strategies which are static. These static hybridization schemes have drawbacks. They require expert knowledge and they are unable to adapt to emerging trends in the domain. We now focus on prediction strategies that learn by themselves.

The paper is organized as follows. Related work is described in Sec. 2. In Sec. 3, we present the adaptive prediction strategy model for hybrid recommenders. We then describe our experimental recommender systems' architecture & learning module that dynamically adjusts recommendation strategy in response to the changes in domain. An initial evaluation of our approach, based on MovieLens dataset, is presented in Sec. 5.

2 Related Work

There have been several research efforts to combine different recommendation technologies. The BellKor system [2], statically combines weighted linear combination of more than a hundred collaborative filtering engines. The system uses the model based approach that first learns a statistical model in an offline fashion, and then uses it to make predictions and generate recommendations. The weights are learned by using a linear regression on outputs of the engine. The STREAM [1] recommender system, which can be thought of as a special case of the BellKor system, classifies the recommender engines in two levels: called level-1 and level-2 predictors. The hybrid STREAM system uses runtime metrics to learn next level predictors by linear regression. However combining many engines in to levels cause performance problems at run-time. Our approach combines different algorithms on a single hybrid engine with an adaptive strategy.

In our system, we chose the Duine Framework¹, which is an open-source hybrid recommendation system [6]. The Duine framework allows users to develop their own prediction engines. The framework contains a set of recommendation techniques, ways to combine these techniques into recommendation strategies, a profile manager, and it allows users to add their own recommender algorithm to the system. It uses switching hybridization method in the selection of prediction techniques.

3 Our Approach

Hybrid recommendation systems combine multiple algorithms and define a *switching behavior* (strategy) among them. This strategy decides which technique to choose under what circumstances for a given prediction request. Recommender system's behavior is directly influenced by the prediction strategy.

¹ <http://duineframework.org/>

Most of the currently available personalized information systems and research into these systems focus on the use of a single selection technique or a fixed combination of techniques [6],[4]. However, domains are dynamic environments. Users are continuously interacting with domain, new concepts and trends emerge each day. Therefore, user interests change dynamically over time. It does not seem possible to adapt trends by using a static approach. Instead of using pre-defined static methods for hybridization, it may be more effective to use adaptive methods in dynamic domains.

In our approach we apply an instance based, supervised learning scheme, which learns through predictions performance results. The main idea of our system is to re-design the hybrid system’s switching behavior at runtime according to the performance results of the algorithms. In AdaRec System, we implement prediction strategy learning module, which produces a strategy by using its attached learning technique. Learning module initializes prediction engine according to the specified Machine Learning (ML) technique. Different instance-based ML algorithms could be attached to our experimental design.

A prediction strategy uses attribute-value pairs called *threshold values* in order to make decisions about which algorithm (case-based reasoning, collaborative filtering, content-based etc.) to use and how to combine them. However in naive hybrid recommender systems, the supplied threshold values are static and do not change during the execution of recommender engine. For example, as shown in the Fig. 1, in the Duine Framework the knowledge that designs the selection strategy is provided manually by experts. The combination of techniques should not be fixed within a system and that the combination ought to be based on knowledge about strengths and weaknesses of each technique and that the choice of techniques should be made at the moment a prediction is required.

Our system, AdaRec, employs switching hybridization by deciding which algorithm is most suitable to provide a recommendation. The decision is based on the most up-to-date knowledge about the current user, other users, the information for which a prediction is requested, other information items and the system itself. In AdaRec we propose the use of ML techniques to analyze and learn which user and item features in a personalized recommender system are more adequate for correct recommendations.

The main purpose of a prediction strategy is to use the most appropriate prediction technique in a particular context. Domain experts usually design the static decision rules in a generic way. But each domain has unique characteristics including user behaviors, emerging trends etc. Adaptive prediction strategy frequently re-design the selection rules of prediction techniques according to the domain drifts. Fig. 1 shows a sample prediction strategy that decides when to use which predictor (gray nodes) by using the threshold values (arrows).

To make decisions about which algorithm is suitable for the current context, threshold values, algorithm’s state and users’ feedbacks are used by the adaptive prediction strategy. In the AdaRec, initial strategy is defined by using the expert knowledge like the naive hybrid recommender systems. System starts with the

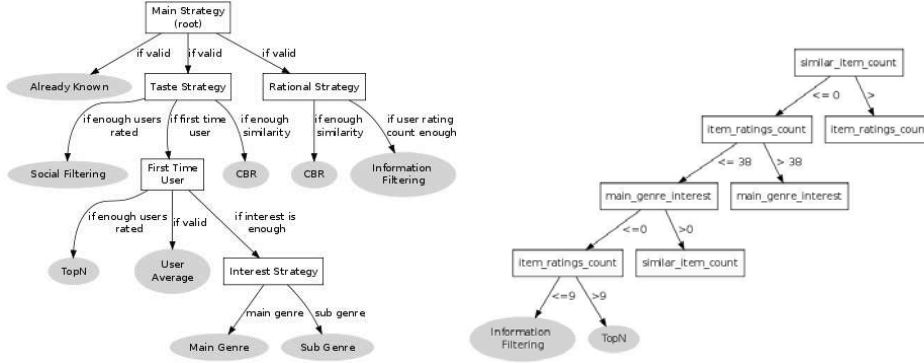


Fig. 1. The default Duine Framework’s static prediction strategy is shown on the left. A subset of a sample adaptive prediction strategy that uses domain attributes is shown on the right.

defined initial prediction strategy. Later on the Learning Module adjusts the prediction strategy to the current systems domain.

Depending on the nature of the domains (movie, music, book etc.) different attribute-value combinations can be used for prediction strategy design. In our proposed system, since we tested on MovieLens dataset, we chose these specific attributes that have meaningful correlations between movie domain and prediction techniques. We believe that, by measuring the changes on these attributes, we can capture the domain drifts and trends.

1. *item ratings count*, the number of ratings that the current item has.
2. *item similar user count*, similar users count that have already rated the current item.
3. *similar item count*, the number of similar items count according to similarity measures.
4. *main genre interest*, main genre interest certainty² of the current item among the users items.
5. *sub genre interest*, sub genre interest certainty² of the current item among the users items.

Decision trees/decision rules are constructed using the combination of the above five attributes. As shown in the Fig. 1 at each node of the decision tree an attribute is compared with a value, which is called *threshold value*. These five attributes are used to classify the prediction techniques. In our system, we used and tested different prediction techniques. These are; *topNDeviation*, *userAverage*, *socialFiltering*, *CBR*, *mainGenreLMS*, *subGenreLMS*, *informationFiltering*. Because of the dynamic nature of the domain, these attributes create different forms of decision trees.

² Sub-genre and main-genre are domain-dependent attributes. Genre certainty is measured by using the movies metadata information which is interpreted according to MovieLens and IMDB (Internet Movie Database) datasets

4 Overview of the AdaRec System

Fig. 2 depicts the architectural overview of the proposed AdaRec system. Our experimental framework is an extension of the open-source Duine Framework. System consists of two core parts, *Recommender Engine* and *Learning Module*. Recommender Engine is responsible for generating the predictions of items based

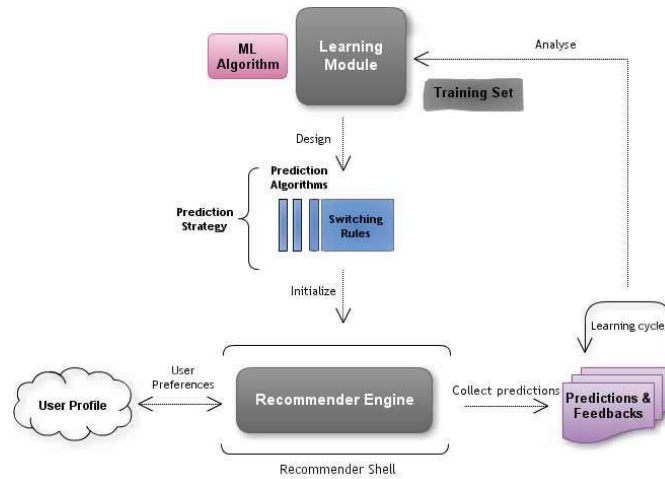


Fig. 2. Overall architecture of the AdaRec System.

on the previous user profiles and item contents. It attempts to recommend information items, such as movies, music, books, that are likely to be of interest to the user. Learning Module handles the new prediction strategy creation upon the previous instances and performance results of the prediction techniques on each learning cycle. It allows the building of new decision trees/decision rules based on the previous recorded instances. *Learning cycle* is a logical concept that represents the re-design frequency of the prediction strategy. A learning cycle indicates the instance count. Each instance, based on the indicated count, and prediction algorithms performance results are collected between two learning cycles.

The learning module first tests the accuracy of each algorithm in the system. Then the prediction strategy is re-designed by the learning module in order to improve proper use of algorithms. Adaptive prediction strategy improves its prediction accuracy by learning when to use which algorithm. The learning module adapts the hybrid recommender system to the current dynamics of the domain.

A learning algorithm is attached to the framework that analysis previously stored instances. The algorithm determines the attribute-value pairs and builds a model with respect to previously recorded instances called *training set*. Previous predictions and user feedbacks are fed to the training set of the next

learning cycle. User feedbacks and MAE (Mean Absolute Error) are the main concepts, which describe the trends in the domain. Adaptive prediction strategy learns its domain trends over time via unobtrusive monitoring and relevance feedback. The prediction strategy is adapted to the current context by analyzing the previous performance results of the techniques. Different ML algorithms that induce decision trees or decision rule sets could be attached to our experimental design. The architecture is open and flexible enough to attach different ML algorithms. Learning module adjusts the prediction strategy and re-initializes the recommender system with modified strategy.

5 Experiments

In our experiments MovieLens³ one million ratings dataset is used, with 6040 users and 3900 movies pertaining to 19 genres [5]. MovieLens dataset contains explicit ratings about movies and has a very high density. In order to train the recommender system, the MovieLens dataset is divided in to different logical time periods (subsets) based on their distribution in time (*time-stamp*).

5.1 Experimental Setup

For our experiments, we use a widely popular statistical accuracy metric named MAE, which is a measure of the deviation of recommendations from their true user-specified values. For each ratings-prediction pair $\langle p_i, q_i \rangle$, this metric treats the absolute error between them i.e., $|p_i, q_i|$ equally. The MAE is computed by first summing over these absolute errors of the N corresponding ratings-prediction pairs and then computing the average. Formally;

$$MAE = \frac{\sum_{i=1}^N |p_i, q_i|}{N} \quad (1)$$

The Duine generates prediction ratings in unipolar range goes from 0 (not interesting) to +1 (interesting) with 0.5 representing a neutral interest. This original state of the framework is referred as *baseline*. We want to compare the prediction quality obtained from the framework’s baseline (non adaptive) to the quality obtained by AdaRec (adaptive). The approach will be considered useful if the prediction accuracy is equal or better than the baseline.

The validation process is handled using the following procedure: The ratings provided by the users are fed to the system one by one, in the logical (time-stamp) order of the system; when a rating is provided during validation, prediction strategy is invoked to provide a prediction for the current user and the current item. The adaptive system collects the feedback and current attributes of the system as instances. After the MAE has been calculated, the collected instances with the best performed algorithms are fed as feedback to the learning module. Whenever the collected instances reached the learning cycle’s instance count (1K

³ <http://www.movielens.umn.edu>

for example), the prediction strategy of the system will be redesigned by the adaptive system according to the instances. This way, when the next learning cycle is processed, the adaptive system knows and has learned from all the previously processed ratings.

5.2 Results and Discussion

In the experiments, different number of instances, such as 1K, 2K and 3K, are used. The purpose of different number of instances was to compare the influence of the instance size on algorithms at the same domain. In the adaptive system, C4.5, BF-Tree and Conjunctive Rules classifier algorithms are attached to the learning module and its results are recorded. We tuned the algorithms to optimize and configured to deliver the highest quality prediction without concern for performance. We also plot the result of the best MAE (less is better) of the hybrid recommender in the current context at each run. The best MAE is referred to *best* in the charts. Therefore it is possible to compare the performance of the algorithms and the best possible result. The left part of the Fig. 3 shows

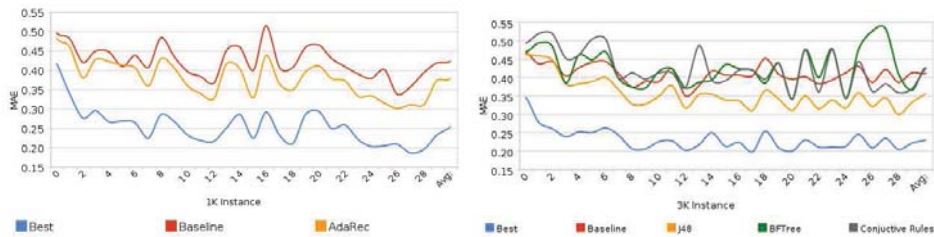


Fig. 3. Quality of prediction (MAE) using AdaRec with J48 (pruned C4.5, BF-Tree and Conjunctive Rules) vs Baseline & Best.

the prediction quality (average MAE) results of our experiments, which used 1K instances as learning cycle count, for the AdaRec (with attached J48 ML algorithm) as well as the original system referred as baseline. In this chart, average MAE is plotted for each of the runs. On each run adaptive system re-designs its prediction strategy according to the previous runs instances. We make two important observations from this chart. First, the prediction quality of adaptive system performs better than the baseline. It can also be observed from the chart that the the adaptive systems MAE that changes overtime shows parallel variation. The adaptive system shows similar trends with the best results. It can be inferred that the adaptive system adapts itself to the current trends. Second, it can also be observed from the chart that as we increase the instance size of algorithms the quality tends to be superior (decreased MAE). The same trend is observed in the case of 2K and 3K instances. The right part of the Fig. 3 presents the performance results of our experiments for both of the learning algorithms

and baseline. From the plot we see that using the $J48$ as the plugged ML, the MAE is substantially better than the baseline. This is due to the reason that with the adaptive approach the recommender system adapts itself to the current context.

6 Conclusion & Future Work

In this paper, we introduced an adaptive hybrid recommender system that re-designs its prediction (switching) strategy according to the performance of prediction algorithms. The learning module adjusts and re-designs the switching rules based on the feedbacks gathered from users. As a result, the system becomes adaptive to the application domain, and the accuracy of recommendation increases as more data are accumulated. In the MovieLens dataset, the adaptive system adapts an adequate to good prediction strategy as it is sometimes capable of providing more accurate predictions than the baseline (naive hybrid system).

Initial experimental results show its potential impacts. Therefore, for the next step this learning module will be deeply tested with various heterogeneous datasets. Also, our future work will explore the effectiveness of other ML techniques for use in the learning module. We believe that with this adaptive learning module, a naive hybrid recommender should have higher chance to allow its users to efficiently obtain an accurate and confident decision.

References

1. Bao, X., Bergman, L., Thompson, R.: Stacking recommendation engines with additional meta-features. In: Proceedings of the third ACM conference on Recommender systems. pp. 109–116. ACM (2009)
2. Bell, R., Koren, Y., Volinsky, C.: The bellkor solution to the netflix prize. KorBell Team's Report to Netflix (2007)
3. Burke, R.: Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction* 12(4), 331–370 (2002)
4. Middleton, S.E.: Capturing knowledge of user preferences with recommender systems. Ph.D. thesis, University of Southampton (May 2003)
5. Movielens dataset. <http://www.grouplens.org/node/73>
6. Setten, M.V.: Supporting People in Finding Information- Hybrid Recommender Systems and Goal Based Structuring. Telematica instituut fundamental research series no:016, Telematica Instituut (November 2005)

Modeling the Example Life-Cycle in an Online Classification Learner

Gary R. Marrs, Ray J. Hickey, Michaela M. Black,

School of Computing and Engineering, University of Ulster, Coleraine, County Londonderry, N. Ireland
marrs-g@email.ulster.ac.uk,
{mm.black, rj.hickey}@ulster.ac.uk

Abstract. An online classification system maintained by a learner can be subject to latency and filtering of training examples which can impact on its classification accuracy especially under concept drift. A life-cycle model is developed to provide a framework for studying this problem. Meta data emerges from this model which it is proposed can enhance online learning systems. In particular, the definition of the time-stamp of an example, as currently used in the literature, is shown to be problematic and an alternative is proposed.

Keywords: Classification, Online Learning, Example Life-Cycle, Latency, Filtering, Concept Drift.

1 Introduction

Online learning for classification to date largely involves sourcing data descriptions with classifications for use in creating an initial classifier and for subsequent updates: an excellent introduction to the current state of research has been written by Kolter and Maloof [1]. The current approaches mainly deploy machine learning algorithms for use in determining a concept representation, such as rules, for that environment. In doing so they inherit all the issues and difficulties of machine learning approaches [2]. Even the collecting and preparation of training examples for such algorithms remains an area open to difficulty, both in balancing the data sets [3] and in the actual sources and sourcing of such data [4]. Furthermore, as a participating factor within a concept environment, the classifier may induce a concept drift of its own thus making any classifying success a short-term success, as might be reasonably concluded from Fama's Efficient Market Hypothesis [5], e.g. in email spam filtering the update of a classifier to filter out spam emails immediately results in the spammers altering their behaviour to avoid detection in this latest version.

Concept learning raises so many issues and criticisms; it is desirable that a common understanding and models of the components of concept learning exist to aid in targeting research efforts [6]. In this paper, we explore and present a model of the life cycle such learners might face; a rationale for latency, as previously summarised

by Marrs, Hickey and Black [7]; and the impact of example filtering in order that more informed approaches might be developed towards online learning.

2 The Online Learner Life-Cycle (OLLC)

According to Domingos and Hulten [8], the online data-stream learner must incorporate certain design criteria, in particular: “When concept drift is present, the model at any time should be up-to-date but also include all information from the past that has not become outdated.” While this could be viewed as specifically an algorithmic issue, it could also be seen as being deeply influenced by the example life-cycle. The learner may only be as up-to-date as the quality of examples it contains in its training base.

They also raise the question of just how much data is enough to learn an accurate model [9]. Alternatively we might consider how much quality data is required to ensure a good model is produced, i.e. what circumstances could occur during the example life-cycle that may directly impact the usefulness of the next training data and just how might this be improved upon to aid our learner.

2.1 Describing the Online Learner Process

The first stage in the online learning cycle commences with the supervised collection of data to comprise the initial training set. Various sources for example data may be used and measures taken to preserve an accurate representation of the current universe: a concept universe being the collective environment represented by a contemporary true rule-base. For example, for a loan granting system the training samples may consist of historic bank data of customers granted loans and the outcomes of such decisions. The supervising authority selects and pre-processes the training data and forwards it on as the initial learning data. Receiving the training data, the learner induces the first classifier and releases it to make predictions with real world input.

The domain presents a descriptive sample as input for classification, e.g. a description of a loan applicant's current circumstances. The classifier predicts the likely class and releases the now classified sample back into the domain. Dependent upon the nature of the classification, our sample may have its true classification determined at a future date. Some classes may not lend themselves to being trackable, rejection classes [10], e.g. a choice to reject a loan applicant cannot be verified as the loan applicant is no longer with the lending authority. Others will give you a determinable verification of your initial classification, acceptance classes, e.g. at the end of the period given for a successful loan application we should be able to verify the validity of our initial classification and should the applicant fail in paying back the loan prior to the end of the period we will have determined a failed prediction and the true class that the applicant should have been rejected. With the domain sample now verified it might find itself returned at a future date to the training base for use in

improving the accuracy of subsequent classifiers. These may be composed of both supervised data and returned verified classification examples.

2.2 Overview of the Stages in the OLLC

The progress of examples through an online learning system can be modeled in a series of stages.

At the first stage, initial supervised example collection takes place using external sources and is placed into the training base (figure 1).

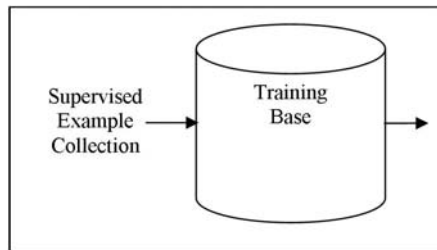


Figure 1: Supervised Example Collection

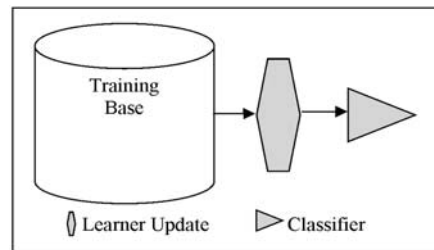


Figure 2: Classifier Induction

Next, the learning algorithm, upon receiving data from the training base, generates the first instance classifier, as in figure 2. This classifier is then placed online for use in the domain. Training data may not necessarily arrive at a constant rate.

At the third stage, a real-world unclassified example, *description*, arrives at time t_c for classification. Once again the rate of arrival is unlikely to be constant. It is given a predicted class, *pclass*, it is sent back out to the domain (figure 3).

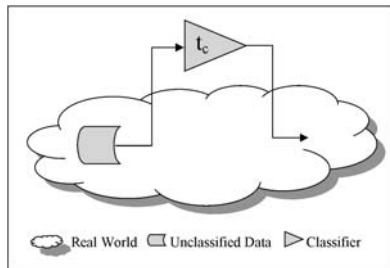


Figure 3: Classification

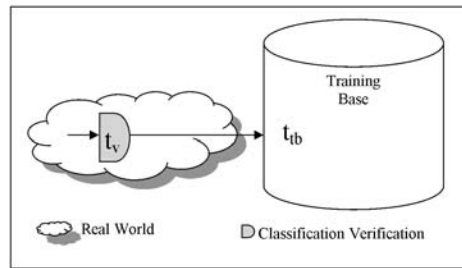


Figure 4: Verification and Feedback

Finally, at time, $t_v > t_c$, the true class, *vclass*, may be discovered from the domain. The lag time, referred to as *verification latency*, between classification and verification can vary from example to example. After a further period of *feedback latency*, the example may be fed back to the training base at time t_{ib} (figure 4).

For some existing online learner approaches a time-stamp may be used [11]. In these learners, the time of arrival at the training base, t_{ib} , is most often used as the timestamp, i.e. examples are time-stamped as they arrive for use, (t_{ib} , *description*,

vclass). However, we propose that any time-stamp other than that at which the data was first classified, t_c , may give the learner an erroneous representation of the current concept universe. This point is discussed further below.

Upon completion of the verification and feedback stage the cycle repeats with the learner algorithm updating the classifier using newly available training examples. In addition to previously classified examples returning, new examples may be obtained from sources external to the cycle.

2.3 Latency

Concepts can be in a state of drift and change, the act of learning the latest concept universe is a competition against the rate of drift. During any period of latency if the concept universe drifts beyond that represented by our most current data then the data is now in a state of lag. We have determined the following key latency periods:

- External source training example collection
- Training base to learner
- Classification to verification
- Verification to training base

During the external example collecting period, the time taken, latency, might lead to examples being in a state of lagging behind the drift in the current concept universe. By minimizing the latency between the example and its inclusion in the training base the concept drift rate is less likely to render an example irrelevant.

As the example batch is released to the learner, the random latency of the examples risks them being in a state of lag with the current concept universe. The latency of the training base to the learner impacts on the relevance of the learned classifier to the current concept universe.

Within the OLLC, as noted above, an example is subject to verification latency $t_1^{lat} = t_v - t_c$ and feedback latency $t_2^{lat} = t_{fb} - t_v$. In many applications these latencies are random and therefore may bias the availability of training examples in the next round of learning. In a real estate domain, a 25 year mortgage may be granted to several similar clients. For each example granted that fails, we discover the true class at some point prior to the closure of the 25 year period. Sending failed examples back to the training base immediately at the time of failure will result in a disproportionate number of negative cases available for the next round of learning. In this domain it could be argued that examples corresponding to failed mortgages should be withheld from the training base until the end of the 25 year loan period and, at that point, returned with the successful loan repayment cases. Unfortunately, the lag factor for the return of these training examples against the rate of concept drift, could ultimately eliminate any true value.

2.4 Filtering

During the life cycle some examples may be filtered out and so never return to the training base. The incidence of such filtering may depend on the example description and on class. Thus there is the potential for bias in the training base: examples

returning may not be a genuine random sample. This would present a problem for the learner. In effect such bias could be perceived as false concept drift. Two types of filtering occur during the OLLC: *trackability* and *selection*.

Trackability filtering is an example of unsupervised filtering. It is first necessary to group classification values under two headings: trackable classes and non-trackable classes: sometimes referred to as positive and negative classes [10].

In a bank loan granting domain the classifier might be required to predict whether to grant a loan application or reject a loan application. If the loan is granted then the example remains trackable and therefore verifiable, i.e. the bank will be able to monitor the application's progress over the loan period. If the prediction proves to be wrong then the bank will be able to determine this fact and use to update the training base for the next learning cycle with a new domain example.

If loan is rejected, however, the applicant returns to the real world again and is no longer able to be tracked by the bank. In this instance the bank will never be able to verify the prediction. In this case, the example is said to be non-trackable.

Therefore, there is a natural tendency for some examples to filter themselves out. Only certain trackable examples are likely to return to the training base for future updates leading to a bias in the next classifier concept representation. In this example, where the trackability factor is dependent upon class value, we could say that an example with a grant loan value has a probability of 1 to remain trackable and 0 to be non-trackable, whereas, a reject loan value has a probability of 0 to be trackable and 1 to be non-trackable.

Other domains may be less exact in the probability values of an example's trackability. Cancer diagnosis may have a trackable probability of 1 for those who are diagnosed as having cancer and 0 probability of being non-trackable. However, those who are diagnosed as not having cancer may not have entirely clear trackability probabilities. If there is a 0.1 probability that a healthy diagnosis is wrong then there remains a 0.1 chance that the example will return and become trackable and a 0.9 probability that it becomes non-trackable.

Selection filtering is a managed filtering process where examples may be selected for verification. The example's true class may not become obvious without some supervised examination of the classifications, e.g. spam filtering. An email may be accepted as being non-spam by an online spam detection software system. It would require managed interrogation of these examples in order to determine whether they were classified correctly. It is highly unlikely that every example would be interrogated in a large system and therefore a selection process is likely to occur. The practices undertaken for selecting examples for further verification could establish a bias leading to an inaccurate concept representation. The rest of the examples are not verified then these are never returned to the training base for subsequent learning.

Examples may be verified incorrectly, e.g. human error, or deliberately bogus, e.g. malicious attempts to break the classifier [12]. There are two potential types of malicious attempts to break the classifier. Firstly, we have gaming the classifier, i.e. disguising the data so as to defeat the current classifier. The second approach is to game the learner, i.e. introducing enough malicious examples into the training base that the learner updates the classifier with a bogus concept representation. Externally sourced examples remain a substantial problem. There may be a lack of control and knowledge of what filtering bias has occurred prior to their arrival.

Ideally, training data is a random sample of the world. However, as a result of the filtering processes this may not be a possibility since certain example types may be less likely to become available. Rare classes may not be represented at all. For many domains it is the rare classes that are the most important target for learning, e.g. credit card fraud detection. In a credit card system, the most important concept class to capture is the fraudulent transaction. However, the ratio of fraudulent to non-fraudulent transactions ensures that our fraudulent class will have a low representation. This may require steps to improve the representation in the training base through a balancing process, e.g. duplicating existing examples. However it may come at the price of altering the class distribution. The value of such balancing measures is still in question [3].

The now verified examples, whether correctly or incorrectly verified, are now available in the training base for possible inclusion in the next learning cycle.

2.5 The Complete OLLC and the Meta Example

The complete OLLC with latencies and filters as developed above is summarised in Figure 5.

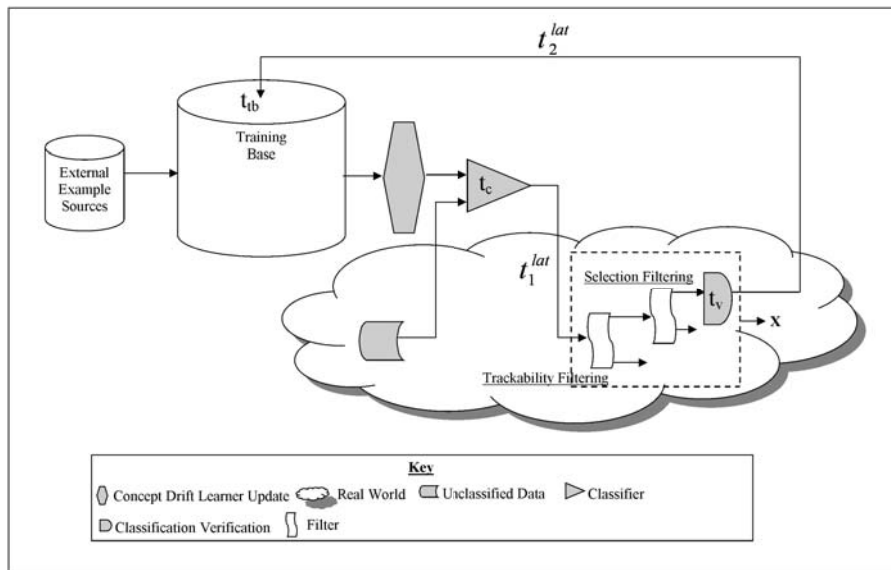


Figure 5: Online Learner Cycle

The traditional training example for a classification learner has, in the notation used here, the form $e = (description, vclass)$. From the OLLC model this can be enhanced with meta attributes to produce the meta example

$$e = (t_c, \text{description}, pclass, t_v, vclass, t_{ib})$$

The training base may also receive examples other than those which have been through the life cycle. This can be flagged by an additional meta attribute. It is likely that for external attributes some or all of the meta attribute values will be missing.

4 Discussion

The development of the OLLC was driven by the awareness that latency and filtering as defined above do occur in the real world and can present a serious hazard to an on-line learner when concept drift occurs.

Wang et al [13] raise issues about online learning. They state that using only historical sources may lead to learning obsolete models whereas using only current data can lead to biased classifiers. The benefit of using internal examples, over external examples, includes the ability to have a greater understanding of the training samples. This is not to say that external examples have no value in the system. Trackability filtering demonstrates how certain classes could end up being poorly represented in the training base causing either up-sampling or down-sampling to occur in the training base. The impact that a highly-skewed class distribution can have upon a learner's performance is clearly demonstrated by Weiss and Provost [14], McCarthy et Al [15], and, Chawla [16].

It has been demonstrated that latency, and the nature of, has a significant impact on the overall performance of an online classifier [7]. The experiments investigated the impact of four latency models, zero, constant, normal distribution-based and negative exponential-based latency, upon an online learner, CD3 [17], using a variety of drift simulations. Proving that latency can hamper both the time taken and overall ability of an online learner to recover accuracy: even to the extent of never recovering.

Hitherto, the only meta attribute routinely used in online learners has been t_{ib} . This has been implicitly assumed to be the appropriate time-stamp of the example. The correct time-stamp, however is t_c . An online learner is attempting to discover the true concept definitions operating at a particular time. A training example is an instance of the true definition and so its time-stamp is that when the rules were applied to produce the example not t_{ib} which is just a result of latencies. Put another way, using t_{ib} as the time-stamp amounts to assuming zero latency.

The value of meta attributes lies in their potential to assist in the maintenance of an online learner when concept drift occurs and there is latency. In the absence of drift, latency will not impact upon a learner.

When latency is random, examples will typically not return in chronological order. Therefore, having an appropriate time-stamp for the example is essential if we are to preserve the grouping of each example within its relevant concept universe.

By including accurate time-stamp, t_c , the internal example gives greater accuracy and understanding of the arrival order of examples and the relationship they have with both current and historic concepts. Current experimentation in progress has already shown that the use of t_c as a meta attribute brings substantial improvements.

Furthermore, by comparing the historic frequency of concept drift sequences against the latency time recorded in the example meta data it is possible to estimate

the lag factor occurring in the system, i.e. just how accurate is the classifier produced in representing the current concepts and as such the domain's suitability for host an online learner.

Finally, internal examples and the feedback they provide allow for a post-mortem evaluation of the current classifier performance.

References

1. Kolter, J.Z., Maloof, M.A.: Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research* 8 (2007) 2755-2790
2. Ben-David, A., Frank, E.: Accuracy of Machine Learning Models Versus "hand Crafted" Expert Systems – A Credit Scoring Case Study. *Expert Syst. Appl.*, 36 (2009) 5264-5271
3. Provost, F.: Learning with Imbalanced Data Sets 101. *AAAI'2000 Workshop on Imbalanced Data Sets* (2000)
4. Gao, J., Fan, W., and Han, J.: On appropriate assumptions to mine data streams: Analysis and practice. In: *Proc. ICDM' 07*, (2007), 143-152.
5. Fama, E.F. Efficient capital markets: A review of theory and empirical work. *Journal of Finance*, May (1970), 383–417.
6. Gama, J., Pedersen, R., U.: Predictive Learning in Sensor Networks. In: *Learning from Data Streams*, pp. 143-164. Springer, Heidelberg (2007)
7. Marrs, G.R., Hickey R.J., Black M.M.: Impact of Latency on Online Classification Learning with Concept Drift, In: *Proceedings of 4th International Conference on Knowledge Science, Engineering & Management, LNAI 2010*, Springer
8. Domingos, P, Hulten, G.: Catching up with the data: research issues in mining data streams. In: *Proc. of Workshop on Research Issues in Data Mining and Knowledge Discovery*, (2001)
9. Hickey, R.J. (1996) Noise modelling and evaluating learning from examples, *Artificial Intelligence* 82: 157–179
10. Sculley, D.: Practical learning from one-sided feed-back. In *KDD '07: Proc. of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining* (2007)
11. Hickey, R.J., Black, M.M.: Refined Time Stamps for Concept Drift Detection During Mining for Classification Rules. In: *Proceedings of the First International Workshop on Temporal, Spatial, and Spatio-Temporal Data Mining, LNCS 2007*, Springer-Verlag, London, pp. 20-30. (2001)
12. Sculley D., Cormack G. V.: Filtering spam in the presence of noisy user feedback. Tufts University, 2008.
13. Wang, H., Yin, J., Pei, J., Yu, P., Yu, J.: Suppressing model over-fitting in mining concept-drifting data streams. In: *Proc. KDD 2006*, Philadelphia, August 20–23, pp.736–741.
14. Weiss, G.M., & Provost, F.: Learning when training data are costly: The effect of class distribution on tree induction. *Journal of Artificial Intelligence Research*, 19, (2003), 315–354.
15. McCarthy, K., Zabar, B., Weiss G.: Does Cost-sensitive Learning Beat Sampling for Classifying Rare Classes? In: *Proceedings of the 1st International Workshop on Utility based Data Mining*, ACM Press, New York, NY, USA, pp. 69–77, 2005.
16. Chawla, N.V.: C4.5 and imbalanced data sets: Investigating the effect of sampling method, probabilistic estimate and decision tree structure. In: *Workshop on Learning from Imbalanced Datasets II, ICML*, Washington DC, 2003.
17. Black, M. and Hickey, R.J.: Maintaining the performance of a learned classifier under concept drift. *Intelligent Data Analysis* 3 (1999), pp. 453-474.

An Incremental Text Segmentation by Clustering Cohesion

Raúl Abella Pérez and José Eladio Medina Pagola

Advanced Technologies Application Centre (CENATAV), 7a #21812 e/ 218 y 222, Rpto.
Siboney, Playa, C.P. 12200, Ciudad de la Habana, Cuba
{rabella, jmedina} @cenatav.co.cu

Abstract. This paper describes a new method, called IClustSeg, for linear text segmentation by topic using an incremental overlapped clustering algorithm. Incremental algorithms are able to process new objects as they are added to the collection and, according to the changes, to update the results using previous information. In our approach, we maintain a structure to get an incremental overlapped clustering. The results of the clustering algorithm, when processing a stream, are used any time text segmentation is required, using the clustering cohesion as the criteria for segmenting by topic. We compare our proposal against the best known methods, outperforming significantly these algorithms.

1 Introduction

Topic segmentation intends to identify the boundaries in a document with goal of capturing the latent topical structure. The automatic detection of appropriate subtopic boundaries in a document is a very useful task in text processing. For example, in information retrieval and in passages retrieval, to return documents, segments or passages closer to the user's queries. Another application of topic segmentation is in summarization, where it can be used to select segments of texts containing the main ideas for the summary requested [6].

Many text segmentation methods by topics have been proposed recently. Usually, they obtain linear segmentations, where the output is a document divided into sequences of adjacent segments [7], [9]. Another approach is a hierarchical segmentation; the outputs of these methods try to identify the document structure, usually chapters and multiple levels of sub-chapters.

Analyzing the methods of text segmentation by topic [7], [8], [9], we have observed some drawbacks as, for instance, wrong interruptions of segments, leaving out sentences or paragraphs which belong to segments, and generating some with incomplete information. When these situations happen, spurious segments are obtained. Another problem we have observed is that those methods are not able to identify the true cohesion amongst paragraphs of each segment considering the main topics. Finally, those methods require a full pre-processing of all the paragraphs in order to decide a valid segmentation; so, they are not conceived to work

incrementally, and cannot be used in several applications as, for example, when segmenting news of broadcast stories transcription.

In this paper we propose an algorithm of linear text segmentation of multi-paragraphs based on topics, called IClustSeg, defined from a solution of the aforementioned difficulties. This method is based on a window approach to identify boundaries of topics. Each paragraph is represented using the vector space model, similar to other methods [6], [7], [9], considering a paragraph to be the minimum text unit. We assumed that paragraph cohesion is obtained by an incremental and overlapped clustering method; it allows obtaining topic boundaries pre-processing only the new paragraphs, considering the paragraph indexes from the clustering outputs to produce the linear segmentation at any time it is required.

We have structured the present work as follows. In Section 2 we briefly explain some previous works to solve the segmentation problem and their drawbacks. In Section 3 we describe the proposed method. In the last section we present the experimental results by using a textual corpus which we prepared with articles selected from the ICPR'2006 proceedings.

2 Methods of Segmentation by Topic

Many segmentation methods by topic use linguistic signs to identify the changes of topics between textual units. In order to select an adequate sign, it is very important to take into account the text types which are going to be segmented. Lexical cohesion is one of the linguistic signs more used in segmentation methods. The term Lexical Cohesion was defined by Halliday and Hasan in 1976 as a sense relationship that exists among the textual units in a text.

Hearst's work is an example that uses lexical cohesion as a mechanism to identify the topic boundaries. Hearst proposed a method which tries to split texts into discourse units of multiple paragraphs, called TextTiling [7]. This algorithm uses a sliding window approach and, for each position, two blocks are built; one preceding and the second succeeding each position. To determine the lexical punctuation between these two blocks, it uses the term repetition as a lexical cohesion mechanism. These blocks are formed by a specified amount of pseudo-sentences which are represented by the vector space model and the cosine as the similarity measure. Considering the lexical values calculated, this method splits the text from the valleys, or points with low lexical scores.

Unlike Hearst's, Heinone proposed a method which uses a sliding window to determine, for each paragraph, which is the most similar paragraph inside the window [8]. The sliding window is formed by several paragraphs on both sides (above and below) of every processed paragraph. This segmentation method is especially useful when it is important to control the segment length. The author uses a dynamic programming technique which guarantees getting segments of minimum cost. The segment cost is obtained by a lexical cohesion curve among paragraphs, a preferential segment size specified by the user, and a defined parametric cost function.

Another approach of linear text segmentation by topic is TextLec, proposed in 2007 [9]. This method uses word repetition as a lexical cohesion mechanism. Each paragraph is represented by the vector space model. The authors of this work assume that all the sentences which belong to a paragraph are about a same topic. This method also uses a sliding window approach but, unlike Hearst's, it only uses a window of paragraphs which are below each position. This method consists of two stages; the first finds for each paragraph the farthest cohesive one within the window, using the cosine measure and a threshold. Finally, it searches the segment boundaries in a sequential process, in which a paragraph is included in a segment if it is a farthest cohesive one related to any other paragraph previously included in the segment.

A C99 algorithm proposed by Choi [3] is another example that uses lexical cohesion as a mechanism to identify the topic boundaries. This method uses the vector space model to projected words; sentences are then compared using the cosine similarity measure. Similarity values are used to build a similarity matrix. More recently, Choi improved C99 by using the Latent Semantic Analysis (LSA) achievements to reduce the size of the word vector space [4]. Once the similarity matrix is calculated, an image ranking procedure is applied to obtain a rank matrix, which is a proportion of neighbours with lower values. The hypothesis in this paper is that LSA similarity values are more accurate than cosine ones.

3 IClustSeg: A method for text segmentation

In this new approach we propose a linear segmentation method by topic that can be used in an incremental way. We consider paragraphs to be the minimum text units. But, in spite of other methods, we assume that sentences belonging to a paragraph, and each paragraph *per se*, could be on several topics. The paragraph representation is based on the vector space model and the topic cohesion is calculated using the cosine measure. Nevertheless, the segment boundaries are defined from the results of an incremental overlapped clustering algorithm, as we can see below.

The IClustSeg algorithm, after preprocessing the text where the stop-words (prepositions, conjunctions, articles, and pronouns) are eliminated, has three main stages: Searching for topic cohesion, Detection of topic segment boundaries, and Detection of document segment boundaries.

3.1 Searching for topic cohesion

The main drawback that all those methods present is the assumption that each paragraph or textual unit is about a unique topic, and the cosine (or any other) similarity amongst paragraphs can link them, creating segments based on a topic-driven process. Also, we should point out that the output of these methods is only obtained over the whole document in a static way; they do not work on text streams.

In this paper, we have considered that a paragraph could be on several topics, and the significant cohesion amongst them depends on the related topic and on its

significance to the stream processed. This algorithm has the possibility of obtaining text segmentations in a sequential and in an incremental way.

Taking into consideration those drawbacks and hypotheses, we decided to use an incremental overlapped clustering algorithm. These algorithms are able to process new objects as these are added to the collection and, in consequence with the changes, to update the set of groups using the previous clustering. It allows obtaining boundary topics pre-processing only the new paragraphs, and considering the paragraph indexes from the clustering outputs to produce a linear segmentation at any time it is required. Besides, because of the hypothesis of multi-topic paragraphs, a convenient (or maybe necessary) restriction is that the clustering algorithm could produce overlapped clusters.

The algorithm that we used for our segmentation proposal is the ICSD algorithm [12]. The novelty of this algorithm is its ability to obtain a set of dense and overlapped clusters using a new graph cover heuristic while reducing the amount of computation by maintaining clusters structured incrementally. The experiments show that ICSD outperforms other graph-based algorithms and achieves better time performance. It is applied over a threshold similarity graph formed by objects (paragraphs represented by the vector space model) as the vertices and the similarity values (using the cosine measure) amongst them. The threshold is defined by the user as a quality measure.

When applying the ICSD algorithm in each batch of the stream, we can obtain a set of clusters, where each cluster represents a set of cohesive paragraphs belonging (presumably) to an independent topic. This set of clusters is updated for each batch taking into account the new paragraphs added. These paragraphs are the only ones to be preprocessed, producing a new clustering.

Any time a new segmentation is required, the last batch is processed by the ICSD algorithm and its output, containing the indexes of cohesive paragraphs, is used by the following stages.

3.2 Detecting the topic segment boundaries

After obtaining the clustering with the last batch, we process every cluster in order to obtain all the segments which can be formed from each cluster. As in other methods, we use a window (W) to define if two adjacent paragraphs in a cluster are close. Each segment is obtained linking adjacent paragraphs according to the predefined window.

The result of this stage is a set of segments defined by a set of pairs $\langle I, F \rangle$, where I and F represent the indexes of the initial and final paragraphs of the segment. The algorithm of this stage is shown in fig. 1.

Algorithm: Topic Segmentation	
Input: C - Clustering of paragraph indexes; W - A predefined window;	
Output: SI - Set of $\langle I, F \rangle$ obtained from C ;	
1)	for each $c \in C$ do begin
2)	$j = 1$;

3)	$c' = \{p_1, \dots, p_k\}$ by sorting c in ascending order;
4)	while $j < k$ do begin
5)	$I = \text{First } p_i \in c' / p_{i+1} - p_i \leq W$ and $i \geq j$;
6)	if does not exist I then $j = k$
7)	else begin
8)	$F = p_{i+1}$;
9)	$j = i+2$;
10)	while $p_j - p_{j-1} \leq W$ do begin
11)	$F = p_j$;
12)	$j = j+1$;
13)	end
14)	$SI = SI \cup \{\langle I, F \rangle\}$;
15)	end
16)	end
17)	end

Fig. 1. Topic segment boundaries detection algorithm

3.3 Detecting the document segment boundaries

As a result of the previous stage, we have obtained a set of topic segment boundaries in the stream. As these segments were obtained from different clusters, and they are overlapped, these segments could be also overlapped.

In this stage, we concatenate all the segments that have at least one common paragraph. Observe that, with this consideration, we obtain a linear segmentation where a document segment could be made up by a concatenation of a set of topic segments from different topics only if any topic segment has a non null intersection (at least one paragraphs) with another one. The Fig. 2 shows this stage.

Algorithm: Document Segmentation	
Input: SI - Set of $\langle I, F \rangle$ from <i>Topic Segmentation</i> ;	
Output: SF - Set of $\langle I, F \rangle$ obtained from SI ;	
1)	for each $\langle I, F \rangle = \text{MinArg} \{I' / \langle I', F' \rangle \in SI\}$ do begin
2)	$\langle In, Fn \rangle = \langle I, F \rangle$;
3)	$SI = SI \setminus \{\langle I, F \rangle\}$;
4)	while exist $\langle I1, F1 \rangle \in SI$ and $I1 \geq I$ and $I1 \leq Fn$ do begin
5)	$Fn = \max(Fn, F1)$;
6)	$SI = SI \setminus \{\langle I1, F1 \rangle\}$;
7)	end
8)	$SF = SF \cup \{\langle In, Fn \rangle\}$;
9)	end

Fig. 2. Document segment boundaries detection algorithm

Finishing this stage, we obtain the boundary topics in the stream after processing the last batch. This process is repeated for each batch any time it is required.

Although it has not been included in this paper, the segmentation output after each batch processing can be analyzed with regard to previous segmentations in order to detect variations and trends.

4 Evaluation

There are two main problems concerning the evaluation of text segmentation algorithms. The first one is given by the subjective nature when detecting the right boundaries of topics and sub-topics into texts; it turns the selection of reference segmentation for a fair and objective comparison into a very difficult task [13]. In order to solve this problem, usually artificial documents are created, concatenating different real documents, on the assumption that the limits between these documents are good breaking points [6], [9], [13]. Another way is to compare the results against a manual segmentation based on human judgments, which makes a “gold standard”. The second problem is the selection of a measure to use in the evaluation; because, for different applications of text segmentation, different types of mistakes become more or less important.

In our proposal, the accuracy of the boundaries is not important, because we are trying to automatically discover topic segmentations. One of the best measures to evaluate this task is WindowDiff, a measure proposed by Pevzner and Hearst in 2000 [13].

The WindowDiff measure uses a sliding window of length k to find disagreements between the reference and the algorithm segmentation. In this work we take k as the half of the average true segment size in the reference segmentation.

The amount of boundaries inside the window of both segmentations is determined for each window position; it is penalized if the amount of boundaries disagrees. Later, all penalizations found are added. This value is normalized and the metric takes a value between 0 and 1. WindowDiff takes a score of 0 if all boundaries are correctly assigned and it takes a score of 1 if there is a complete difference between the automatic segmentation and the reference one. The WindowDiff formal expression and other details of this measure can be seen in Pevzner and Hearst [13].

In order to check the performance of our algorithm we made two types of experiments. In the first experiment we took the segmentation output of our algorithm using the whole stream as a unique document, and we compared it with the results shown in Table 1 of Hernandez&Medina’s work [9]. Besides, we included the C99 algorithm results and our proposal with the same corpus.

The corpus that we used in the experimentation is the same as Hernandez&Medina’s work [9]. This corpus was built joining 14 different papers taken from the ICPR’2006 proceedings. The resultant corpus has 305 paragraphs and an average of 22 paragraphs approximately for each paper.

The results of the first experiment can be seen in Table 1. It was done with a window (W) equal to 9 and a threshold of 0.31 for deciding a significant cosine value between two paragraphs. We can notice a significantly better performance of IClustSeg. We also made other evaluations, varying the window size and the threshold. In Table 2 we can see the results of IClustSeg with W equal to 9, 10 and 11 respectively, with different thresholds.

Table 1. WindowDiff values

Algorithms	IClustSeg	TextLec	TextTiling	Heinone's	C99
WindowDiff	0.11	0.21	0.33	0.26	0.21

Table 2. WindowDiff values of IClustSeg with different W and threshold values

WindowDiff\Threshold	0.30	0.31	0.32	0.33	0.34	0.35
$W = 9$	0.15	0.11	0.14	0.17	0.17	0.15
$W = 10$	0.15	0.13	0.13	0.13	0.13	0.12
$W = 11$	0.15	0.13	0.13	0.13	0.13	0.12

The second experimentation was made to evaluate the performance of IClustSeg algorithm for several batch in a stream. In Table 3 we show the average of the segmentation outputs using the WindowDiff measure. Our method was run with W equal to 9, a threshold equal to 0.31 and batches of 50 and 100 paragraphs.

Table 3. WindowDiff averages with batches of 50 and 100 paragraphs

Algorithms\WindowDiff	Average with 50	Average with 100
C99	0.20	0.21
IClustSeg	0.10	0.11

To evaluate the performance of our algorithm we did not accomplish any experimentation in terms of execution time because none of the algorithms analyzed makes batch segmentation. Nevertheless, we ran our algorithm and measured the time consumption with batches of 100 paragraphs, the execution times were: 156, 344, 578 and 109 ms. Running C99, the execution times were: 194, 469, 872 and 909 ms.

5 Conclusion

In this paper we propose a new segmentation method by topic in an incremental process. It allows us to obtain topic boundaries when needed. Although we use the vector space model and the cosine measure, we consider that the paragraph cohesion can be better obtained from an incremental clustering method, generating segments containing paragraphs from one topic or a mixture of them.

The IClustSeg algorithm was compared with four methods in two experimentations, obtaining more cohesive segments and increasing significantly its performance.

As future work, we propose to evaluate other clustering methods and to achieve a better integration of both strategies. Besides, although we ran our method obtaining a reasonable efficiency, it should be evaluated in larger data sets.

Reference

1. Aslam, J., Pelekhev, E., Rus, D.: The star clustering algorithm for static and dynamic information organization. *Journal of Graph Algorithms and Applications*, 8(1), 95–129 (2004).
2. Beeferman, D., Berger, A., and Lafferty, J.: Statistical Models for Text Segmentation. In: *Second Conference on Empirical Methods in Natural Language Processing*, p. 35–46 (1997).
3. Choi, F.Y.Y.: Advances in domain independent linear text segmentation. In: *NAACL-00*, 26–33 (2000).
4. Choi, F.Y.Y., Wiemer-Hastings, and Moore, J.: Latent semantic analysis for text segmentation. In: *EMNLP*, 109–117 (2001).
5. Filippova, K. and Strube, M.: Using Linguistically Motivated Features for Paragraph Boundary Identification. In: *The 2006 Conference on Empirical Methods in Natural Language Processing (EMNLP 06)*, 267–274 (2006).
6. Ken, R. and Granitzer, M.: Efficient Linear Text Segmentation Based on Information Retrieval Techniques. In: *MEDES 2009, Lyon, France (2009)*.
7. Hearst, M.: TextTiling: Segmenting Text into Multi-paragraph Subtopic Passages. *Computational Linguistics*, Vol. 23(1) (1997).
8. Heinonen, O.: Optimal Multi-Paragraph Text Segmentation by Dynamic Programming. In: *COLING-ACL 1998, Montreal, Canada*, 1484–1486 (1998).
9. Hernández, L. and Medina, J.: TextLec: A Novel Method of Segmentation by Topic Using Lower Windows and Lexical Cohesion. In: *CIARP*, 724–733 (2007).
10. Labadié, A. and Prince, V.: Finding text boundaries and finding topic boundaries: two different tasks. In: *GoTal'08 (2008)*.
11. Misra, H., et al.: Text Segmentation via Topic Modeling: An Analytical Study. Hong Kong, China (2009).
12. Pérez-Suárez, A., Martínez, J.F., Carrasco-Ochoa, Medina-Pagola, J.E.: A New Incremental Algorithm for Overlapped Clustering. In: *CIARP 2009, LNCS 5856*, 497–504 (2009).
13. Pevzner, L. and M. Hearst. A Critique and Improvement of an Evaluation Metric for Text Segmentation. In: *Computational Linguistics*, vol. 16(1) (2000).
14. Pons-Porrata, A., Ruiz-Shulcloper, J., Berlanga-Llavori, R., Santiesteban-Alganza, Y.: Un algoritmo incremental para la obtención de cubrimientos con datos mezclados. In: *CIARP 2002*, 405–416 (2002).
15. Schmid, H. Probabilistic part-of-speech tagging using decision trees. In: *International Conference on New Methods in Language Processing (1994)*.
16. Schmid, H. Improvements in part-of-speech tagging with an application to german. In: *ACL SIGDAT-Workshop (1995)*.