International Journal on Artificial Intelligence Tools

LEARNING NEGOTIATION POLICIES USING ENSEMBLE-BASED DRIFT DETECTION TECHNIQUES

FABRÍCIO ENEMBRECK
PUCPR - Pontifícia Universidade Católica do Parana
Graduate Program on Computer Science
Rua Imaculada Conceição, 1155
BP 80.215-901 Curitiba - PR - Brazil
fabricio@ppgia.pucpr.br

CESAR AUGUSTO TACLA
UTFPR - Universidade Federal Tecnológica do Paraná
Graduate Program on Electrical Engineering and Industrial Informatics
Curitiba - PR - Brazil
tacla@utfpr.edu.br

JEAN-PAUL BARTHÈS
UTC - Université de Technologie de Compiègne
Centre de Recherches Royallieu
60200 Compiègne - France
barthes@utc.fr

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

In this work we compare drift detection techniques and we show how they can improve the performance of trade agents in multi-issue bilateral dynamic negotiations. In a dynamic negotiation the utility values and functions of trade agents can change on the fly. Intelligent trade agents must identify and take such drift in the competitors into account changing also the offer policies to improve the global utility throughout the negotiation. However, traditional learning mechanisms disregard possible changes in a competitor's offer/counter-offer policy. In that case, the agent performance may decrease drastically. In our approach, a trade agent has a staff of weighted learner agents used to predict interesting offers. The staff uses the Dynamic Weighted Majority (DWM) algorithm to adapt itself creating, deleting and adapting staff members. The results obtained with the IB3 (Instance-based) learners and IB3-DWM learners show that ensemble methods like DWM are suitable for correctly identifying changes in agent negotiations.

Keywords: Dynamic Negotiation; Ensemble-based Drift Detection; Learning Agents

1. Introduction

Electronic negotiation is an active research area, with many models proposed. According to Faratin et al. [5], negotiation is a process whereby two or more agents are

required to come to a decision in order to resolve a conflict. When both agents are involved in transactions to attain a common goal one talks of *Cooperative Negotiation*. Negotiation processes can be used in a variety of domains involving distributed problem solving. Within an environment with limited resources like e-commerce, negotiations between customer and supplier agents can lead to coordinated forms of allocating and saving resources, meeting values that mutually satisfy each agent in the supply chain.

Whenever there are multiple criteria for evaluating offers and counter-offers in a negotiation process, the process is said to be multi-issue. A number of projects have been carried aiming at developing intelligent trading agents for learning their competitors' preferences, functions or utility values [2 3 9 15]. Utility values are usually calculated using private functions that quantify the level of interest an offer can represent to the agent. Therefore, the estimation of such functions by the competitors can maximize the global negotiation utility, optimizing resources and obtaining higher utility values for specific agents. Consider, for instance a product provider who has problems in his supply chain. If this is discovered by clients, then they can ask for a discount because the delivery delay is bound to increase. Of course such information is not public but an adaptive agent can discover it by itself, because throughout the iterations with the supplier it realizes that the *delivery delay* has a higher impact on the supplier utility. In this case, *delivery delay* becomes an important issue in the negotiation. We can observe that such changes can occur dynamically and they have to be detected on the fly.

In a previous paper [4] we proposed the use of a drift detection algorithm to satisfy an important constraint inherent to the problem of discovering offer policies in negotiation: the learning process must be incremental, because besides the limited number of observations, the agent function and utility values may be modified during the negotiation process, and there may be interferences from external factors as well.

In this paper, our main focus is on a methodology to develop adaptive trade agents using ensemble-based drift detection techniques. We believe that such a technique constitutes an innovative solution to the proposed problem. We are not, however, interested in proposing and/or using complex methods of negotiation, since our objective is to propose a generic technique that can be used with any negotiation model. In addition, the use of a specific utility model might introduce biased results, rendering it invalid for other negotiation models. We propose a generic technique allowing an agent to identify changes in a competitor's negotiation policy, classifying an offer as either possibly interesting or non-interesting to its competitor, using a staff of learning agents. We propose an interaction protocol between agents, and compare the technique using IB3 as a base learning algorithm [1], starting from the idea that the negotiation process is bilateral and multi-issue.

In the following section we discuss some ideas related to offer policy learning, as well as the most important issues related to this problem. We characterize the proposed problem as a drift detection approach by discussing its most relevant

theoretic aspects in Section 3. We present our technique in Section 4 and discuss experimental results in Section 5. Our conclusions are given in Section 6.

2. Negotiation and Learning

With the growth of Internet and the democratization of technology, it can be observed that the number of service and product suppliers, and, specially, potential customers grows exponentially every year. Nevertheless, technology must favor the establishment of relationships amongst such players in a way that both customers and suppliers can achieve their goals, like minimizing costs, acquiring goods safely, guaranteeing a large number of customers, or solving problems of demand and stock. One way of favoring the growth of such business alternatives employs trading agents. Trading agents can play the role of customers or suppliers depending on the context. Automatic discovery of offer policies in negotiation is a fundamental issue for the evolution of e-commerce technologies, as trading agents participate in negotiations and business transactions in order to maximize resources in favor of some customer or supplier.

A trading agent must implement at least two main components: an *interaction protocol* and a *negotiation model*. The interaction protocol defines the meaning of each interaction between agents, what can be done and which sequences of actions should be employed [7]. The negotiation model comprises the description of the offers and issues, determining value domains for issues and, more importantly, a reasoning model, usually based on a utility model. The reasoning model allows the agent to reason about offers and counter-offers [5], thereby defining specific priorities and behaviors. In recent years some researchers have proposed intelligent trading agents equipped with an extra module composed of a learning model allowing the agent to learn their competitors' utility function. The module helps the agent to define its own goals and to produce offers that satisfy its needs more quickly. For instance, consider a buyer agent that needs to receive a large quantity of an arbitrary product immediately. If few suppliers are known to be capable of guaranteeing the volume to be delivered, the vendor may guarantee immediate delivery at a high price, provided that it is not higher than the competitors' price. Of course, the offer depends on knowing the customers' limitations, which generally is confidential information. That piece of information can be discovered by observing similar behaviors in previous negotiations using pattern recognition techniques and machine learning. Several projects with that goal have been proposed.

Genetic algorithms have been used by Gerding and Bragt [9] and Lau [15] in an attempt to find the best offer and counter-offer policy. However, such techniques are hardly applicable to e-business due to the complexity of the required calculations. Probabilistic techniques based on Bayes model have also been used with relative success in [27] or [2]. The proposed solutions however, suffer from problems related to the distribution of data, which is rarely known, and the volume of data, which is usually insufficient for the discovery of effective probability functions. Moreover, a

pre-existing dataset is rarely available to estimate probabilities *a priori*. Orthogonal polynomials have been used for estimating the utility functions in trading agents by [18]. Nonetheless, even for limited negotiation spaces, this technique needs a large number of interactions to discover efficient utility functions. Coehoorn and Jennings [3] have proposed a technique based on kernel density estimation to discover the weights of each competitor's issue. Based on such weights, the authors used the algorithm proposed in [6] to estimate trade-offs that satisfy the agents involved in the negotiation.

Although some of the cited projects gave satisfactory results, they do not consider an important aspect of negotiation: there are various cases where the negotiation process cannot be modelled as a stationary function, but as one under the influence of factors that may or may not be related to the negotiation itself. This means that agent utility values and functions may change in time, for example due to its poor performance or because of the occurrence of unexpected events, unrelated to the negotiation environment. As an example, a sudden lack of raw material for a supplier will seriously affect the way its sales agent negotiates with customers. Likewise, a customer who notices that his income is decreasing in time may gradually alter his offer policies in order to reduce the need for immediate capital. Such characteristics make most of the learning methods we mentioned inappropriate, since they are unable to adapt the learned model to changes dynamically . Therefore, the agents must restart the learning process after a long period of time during which their performance is poor and does not reflect the brand new competitor's negotiation model. The identification of changes in concept descriptions is a task known as *drift detection* [19] [10] in Machine Learning [17]. In this work, we focused on detection of changes in negotiation policies by describing how *drift detection* algorithms can contribute to the solution of this problem. Next section discusses the main concepts related to this subject.

3. Drift Detection

In this section, we present the drift detection problem and discuss some algorithms used for solving it.

3.1. *Drift Detection Definition*

In domains where the environment constantly changes or data is continually produced, dynamic learning techniques need to be used as soon as the target-concepts of the algorithm change as time elapses. Such modifications may be abrupt or may occur slowly. Nevertheless, most machine learning and data mining algorithms assume that training sets are generated according to a stationary distribution of a probability function. Those algorithms are, therefore, incapable of handling the problem we want to solve. Gama et al. [8] mention other problems that involve detection of changes in target-concepts like biomedical and industrial processes monitoring, fault detection and diagnostic, or complex security systems.

Detecting changes while learning dynamic class models is the problem known as drift detection [14] [22]. The following paragraphs characterize the concept of drift detection as described by Stanley [22].

For simplicity, we defined as a concept a normal disjunctive formula that relates a set of qualitative variables like Price and Quality ∈ {very_low, low, normal, high, very_high}. Thus, a concept may be described as, for instance, Price ∈ {very_low, low} and Quality ∈ {high}. With such a description one can easily verify whether an instance is covered by an arbitrary concept, by checking if the conditions in the corresponding concept description are satisfied. Drift detection may be defined as a modification in the current concept. In order to illustrate such a modification we need a couple more concepts. Let A and B be two concepts and $i_1 \in i$ an instance. Until reaching instance i_x, the current concept A is stable. After a certain number Δx past i_x, the concept B takes the role of current concept. The change from concept A to concept B is taking place smoothly between instances i_x and $i_{x+\Delta x}$ as shown Figure 1.

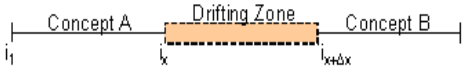

Fig. 1. Drifting from A to B (extracted from [4]).

When $\Delta x = 1$, an abrupt change from A to B is said to have occurred. On the other hand, when $\Delta x > 1$, there exists a drifting zone and the modification proceeds gradually. The drifting zone can be modeled as a probability function a that defines the dominance of concept A over B. Thus, $p(A) = a$ and $p(B) = 1 - a$. In order to measure the probability of concept A in an arbitrary $i_c \in \{i_x, i_{x+1}, ..., i_{x+\Delta x}\}$ one needs to use $a = (c - x)/\Delta x$. The equation establishes that the probability of occurrence of a concept A decreases linearly as the probability of B increases.

3.2. *Considerations for Concept Drift*

A concept drift technique must include three main goals: (i) rapid detection of concept drift, (ii) distinguishing concept drift from noise; and (iii) recognizing and dealing with recurring contexts. Rapidly detecting a concept change in data is essential. Such a feature enables actions to be selected in a way that minimizes the negative impact caused in the system by wrong actions. To put it simply, less instances will be incorrectly classified by the system, thereby less mistakes will be done. On the other hand, some systems may receive noisy data due to faults in the input or monitoring devices, or due to partial representations of the environment. Such changes consist of anomalies that must be identified without, however, modifying concepts acquired by the system during the learning process. The work done by [24] or [22] mentions such a need. Another important aspect of concept drift techniques lies in the ability to identify recurring concepts, i.e., concepts that alternate between

valid and invalid states with some degree of regularity. There are several domains where concepts are naturally recurring, for instance, the cycle of seasons through a year (spring, summer, fall and winter). Recurring concepts directly affect current prediction parameters. They can be stored and reused for future situations. By so acting, the system need not repeat the learning process for a previously learned concept; it only needs to identify the valid recurring concept, saving resources, reducing response time and possibly increasing its utility.

3.3. *Updating Techniques*

In several application domains, data is continuously generated and stored. Moreover, a system for administration of critical situations continuously acts according to input data. Consider for example a network of atmospheric radars that monitor an arbitrary region in order to predict destructive electrical storms. Data transmission frequencies are about hundredths of seconds. Within such an environment, the number of concepts (atmospheric conditions) is limited, while delay and confidence in decision making are critical. Therefore, an issue must be considered: how to guarantee that the prediction model of the system will always be up-to-date? To let a system have its concepts continuously provide a consistent representation of data, the use of techniques for online (incremental) learning is recommended. In this case, concepts are only modified; no concept is reconstructed. Modifications of concepts should permit a new observation to be correctly classified. On the other hand, a procedure for off-line (batch) learning would not be suitable, since it would require that the system store a great number of instances before starting a new execution of the learning algorithm. During the learning process (that may be slow), instances may be incorrectly classified, as the system operates with an obsolete model.

Some systems also need user interaction to support the update process. For instance, when a tool for detection of unsolicited electronic messages (SPAM) intercepts a suspicious message, the system asks the user to indicate whether or not that message is SPAM. Without explicit user feedback, the system is unable to update its knowledge base, which might result in incorrect classification of messages, letting unsolicited messages come through.

3.4. *Detecting Concept Drift*

Numerous methods have been proposed to deal with the drift detection problem in machine learning. They can be categorized according to their training features as either online or batch. An online learning method modifies its models whenever a new example is observed. On the other hand, a batch method processes a whole dataset once, updating its data models periodically. Online learning is particularly interesting to this work, since we need to consider that offers/counter-offers are received and used during negotiation. The most ubiquitously widespread drift detection techniques are (i) windowing; (ii) example weighting, according to their age or utility; and (iii) example selection. According to Gama et al. (2004), techniques

that deal with drift detection can still be split into two other categories: (a) techniques that adapt the classifier at regular intervals without verifying whether any changes have occurred; or (b) techniques that detect changes beforehand and then adapt their corresponding classifiers to them. Examples of the former are windowing and example weighing. The hypothesis whereby the importance of an example decreases with time is the foundation of the example weighing technique. It can also be used as a tool to select the most relevant examples (iii). More sophisticated techniques overcome the drift detection problem by generating and maintaining a set of classifiers [14] [22]. Some of them are described in the following sections.

3.4.1. *Windowing*

Whenever a time window is used, the classifier is reconstructed from the examples within such a window. The complexity of this technique lies on identifying the proper size of the temporal window: excessively small windows ensure rapid adaptation, but cannot identify relationships among variables due to the partitioned view of data; whereas extremely large ones produce good classifiers for stable zones but loose the capacity for identifying changes quickly. If some change is detected at any instant of the monitoring phase, some actions may be triggered in order to let the classifier adapt accordingly. For instance, modifying the window size in order to accommodate the set of examples affected by that change will do that. In addition, the window size may be reduced upon the detection of a change and enlarged when no change is observed.

The windowing technique obeys the following principle: a storage area (or buffer), called Window, is created to serve as a container for the N most recent observed instances. The learning algorithm uses the N instances in the window to learn the current data model. The window size may be fixed or variable. An advantage for the variable one is that it may adjust the number of instances that will be used in the pattern discovery process as a function of the stability of the data distribution. For instance, when the distribution of data undergoes a stable period, i.e., when no concept change has been observed recently, the bigger the window, the better the model generated by the algorithm to represent it. Conversely, when the distribution undergoes a transitional period, the smaller the window, the better the system performance, as instances display significant variations to one another [12].

In [22] a comparative study on windowing is presented along with other strategies for detection of concept drift. In the study, the authors experienced difficulties in their attempt to automatically adjust the window size. All strategies imply a slow change in window size, which requires the algorithm to experiment over relatively long periods with a high error rate in classifications [12].

3.4.2. *Example Selection by Error Estimation*

The strategy for error estimation consists in using the error rate of the algorithm on the instances received by the system to identify concept changes. This strategy employs two thresholds, namely *Warning Level* and *Drift Level*. Such thresholds are used by the concept drift detection algorithm to indicate that the error rate for classification lies within an interval that denotes a possible concept change. Thus, when the error rate overtakes the *Warning Level*, additional instances are stored in a buffer. When the error rate reaches the *Drift Level*, current concepts need to be updated and the learning algorithm starts. All the instances stored from the instant when the *Warning Level* was overtaken until the one when the *Drift Level* was reached become the training dataset for the algorithm [16].

3.4.3. *Instance-Based Concept Drift Detection*

In this work we use a well-known drift detection algorithm: the IB3 algorithm, proposed by Aha (1991). As stated by the author, the algorithm presents a relative robustness against noise and irrelevant attributes. Such characteristics are quite valuable in negotiation, as agents can distinguish real changes from noise besides the possibility of exploring relationships amongst the various offer issues. Updating is a low cost process in that algorithm, which enables responsiveness for decisions and thus detection of changes in an agent offer policies. IB3 is a member of the instance-based algorithms family. IB1 corresponds to the standard nearest neighbor algorithm. Although instance-based learning algorithms are capable of reproducing complex discriminating functions, they usually require massive memory resources and high processing time for the classification process due to the need to compare and contrast the instance to be classified with all available instances. IB2 attempts to reduce the problem by introducing a criterion for the storage of a new instance. When a new instance is classified, its storage in the concept description is only performed if its classification was incorrect, otherwise it is discarded. Although IB2 is saving memory and processing resources, experiments have shown that IB2 is fairly sensitive to noise in data. IB3 has evolved from IB2. Experiments demonstrated that the method is quite robust when applied to data with such characteristics. IB3 maintains a classification record with each and every instance stored. This record quantifies the classification rate of an instance, i.e., the accuracy rate of the instance in cases where it has been used to classify a new one. That information is used to estimate its future precision. The algorithm also uses a significance test to determine whether an instance either should be relevant for future classifications or represents noise. In the latter case, it is discarded. The algorithm stores an instance if its precision is significantly higher than the observed frequency of its class, and removes it if its precision is significantly lower. A detailed description of the algorithm can be found in [1]. The main idea of IB3 is used in the Flora family of algorithms [25]. Flora4 saves a record of classification for any descriptor represented by a pair <Attribute,Value>. The algorithm maintains three sets of descriptors:

ADES (Accepted DEScriptors) matching only positive examples; NDES (NEgative DEScriptors) matching only negative examples and PDES (Potential DEScriptors), matching positive and negative examples but are too general. The statistical tests of IB3 are used to decide in which set a descriptor ought to be stored. The records are updated whenever a new training instance arrives.

3.4.4. *Ensemble-Based Concept Drift Techniques*

The CDC [22] is an alternative for algorithms that use time windows. These algorithms require heuristics to decide when window sizes should be altered, whereas CDC does not. The CDC algorithm relies on a poll to detect the Concept Drift and to update concepts. Consider a committee C, comprising n hypotheses, where each hypothesis is a decision-tree built upon all instances it knows (s_i). An incoming instance is evaluated by each hypothesis; the ones that incorrectly classify the instance have their weights reduced until they are removed from the committee and replaced by new hypotheses. Initially, C is empty (no hypotheses). Whenever a new instance arrives, if the number of hypotheses in the committee is less than a maximum n, a new member is added to it. The newly created member h_1 initializes its set s_1 with the current instance only. Thus, upon the arrival of the second instance, the committee C contains the hypothesis h_1 only, whose training set is s_1, which holds the instance i_1. At the time the instance i_2 arrives, a new member h_2 is added to the committee and its training set s_2 is composed of the instance i_2 only. At this moment, the committee has two hypotheses:

- h_1: with s_1 comprising $\{i_1, i_2\}$;
- h_2: with s_2 comprising $\{i_2\}$.

Members are added to the committee until the maximum number of members (n) is reached. For the committee members (hypotheses) to be evaluated, each of them must vote on test instances that are derived from the current distribution (target concept). The weight of each vote is equal to the performance of the corresponding member over the last m training instances. For this reason, members whose recent performance has been satisfactory participate with votes having higher weight. When a member's performance falls beneath a threshold t, it is replaced by a brand new one that has no knowledge of past instances. Therefore, the committee will always be representing the current distribution of data. In periods of stability, the oldest members are more reliable, though when there are transitions between concepts, members are constantly being replaced. When a new member is added to the committee, there are not enough instances to gain considerable weight for its vote; therefore, it must reach a certain degree of maturity before it can vote and be replaced (if necessary) by a new member. Such a mature age is reached when the member has already observed m instances throughout its existence. Before that, its vote weight is 0 (zero). The work of Stanley [22] describes the CDC algorithm in details. It is possible to consider that CDC uses pseudo time windows, as each member

is trained with a set of instances different from the ones used by the other members. This characteristic allows CDC to exhibit good performances in both gradual and abrupt Concept Drift. It should be emphasized that, in the latter case, almost all members are usually replaced, while in the former case they are replaced gradually. Such behaviors make the algorithm noise tolerant and efficient for the detection of Concept Drift.

SEA (Stream Ensemble Algorithm) [23] induces an ensemble of decision trees from data streams and explicitly addresses concept drift. SEA generates decision trees from small blocks of data read sequentially. The maximal number of classifiers is fixed. Whenever such a number is reached, new classifiers can be generated if they satisfy some quality criterion. Generally a new classifier is added to the ensemble if it improves the performance of the ensemble. On the other hand, the addition of new classifiers causes the deletion of others, keeping the size of the ensemble constant. The quality of a candidate classifier or committee member is improved whenever a validation instance is correctly classified or goes down otherwise. Some heuristics are used to calculate rewards or punishments. If an instance is easy to classify (most members of the ensemble classify it correctly) or impossible to classify (there is no correct predictions), rewards or punishments are minimal. On the other side, if the members have heterogeneous opinions, then rewards and punishments are high. Simple majority vote is used to classify an instance. Experiments have shown that SEA generates ensembles whose performance is comparable to a single decision tree generated from all the training data. Besides, it identifies drift situations correctly. Recent works suggest that ensembles formed of weighted members are preferable in domains with concept drift. The DWM (Dynamic Weighted Majority) [13], AddExp [14] and KBS-Stream (Knowledge-Based Sampling-Stream) [20] [21] systems take into account the weight of the base classifiers for voting, returning the class with the highest accumulated weight.

DWM reduces the weight of an expert (base classifier) by a constant multiplicative whenever it makes a false prediction. The global prediction is determined summing the weights for the classes. If it is incorrect, then a new expert is created with a weight of one. The algorithm normalizes expert weights by scaling them such that the highest weight is 1. The algorithm also removes experts with weights less than a threshold. Finally, the training instance is passed to each expert for individual learning. We can see that DWM generates a variable number of classifiers. Experiments presented in [13] show that for the Stagger concepts [25] DWM reaches 100% of accuracy using Naïve Bayes as the base learning algorithm.

AddExpert is very similar to DWM algorithm but the authors propose two different criteria for pruning (deletion of an expert): *Oldest First* and *Weakest First*. Whenever the maximal number K of experts is reached, *Oldest First* removes the oldest expert before creating a new one. On the other hand, when a new expert is added to the ensemble, if the number of experts is greater than K, *Weakest First* removes the expert with the lowest weight before adding a new member. Unfortunately, the number of classifiers created with such simple approaches can be

prohibitive and can cause the algorithm to be much slower than a single learner. On the other hand, the main idea of KBS-Stream is to use the KBS boosting algorithm [20] for the generation of samples with weighted instances and weighted classifiers. The training set sizes effectively used for each model are determined dynamically by the algorithm. In order to process a new batch of data two ensemble variants have been used. The first one appends the current batch to the cache used for training in the last iteration, refining the last base model. The second one adds a new model, trained using the latest batch of data. Only the ensemble variant performing better on the next batch is kept. The second step is to foresee a re-computation phase in which base model performances are updated with respect to the current data distribution. The authors argue that KBS-Stream adapts very early and quickly to different kinds of concept drift, although a comparative study with other ensemble-based concept drift has not been done.

4. Ensemble-based Drift Detection for Drifting Negotiation

We believe that ensemble-based drift detection techniques satisfy important constraints inherent to the problem of learning offer policies, such as effectiveness, efficiency, complexity and, mainly, changes in the agent behaviors. Such features may be used along with a strategy for exploring the space of possible offers in order to discover an interesting offer and to let the agent better negotiate. Furthermore, the results presented in [13] and [4] motivated us to study the behavior of DWM when IB3 is used as the base learning algorithm.

Figure 2 shows the architecture of an adaptive trade agent proposed in [4]. Upon receiving an offer/counter-offer, the agent classifies it as interesting in case it has been accepted by the corresponding peer and non-interesting otherwise. Based on that offer, a training instance is then generated and transferred to the drift detection module, where it may be stored in the concept description database or rejected. A training instance may be rejected if the learning module (drift module) can predict its class properly. Otherwise, the drift module must change the current concept description such that the new description is able to classify the instance properly. Next, the negotiation module generates repeatedly offers/counter-offers until one of them is classified as *interesting* by the drift detection module. As soon as an offer/counter-offer is classified as *interesting*, the iterative process ends and the offer is sent to the peer agent.

The technique presented in [4] has shown good results but we believe that it can be improved with an ensemble-based drift detection technique. In such an approach, the trade agent has specialized learning agents responsible to learn cooperatively the data distribution and values for interesting/non-interesting offers. The architecture proposed is shown Figure 3.

In the proposed architecture a trade agent has an ensemble of learners that discover patterns from accepted or rejected offers throughout past negotiations. This makes the trade agent simpler because it is just specialized in negotiation and

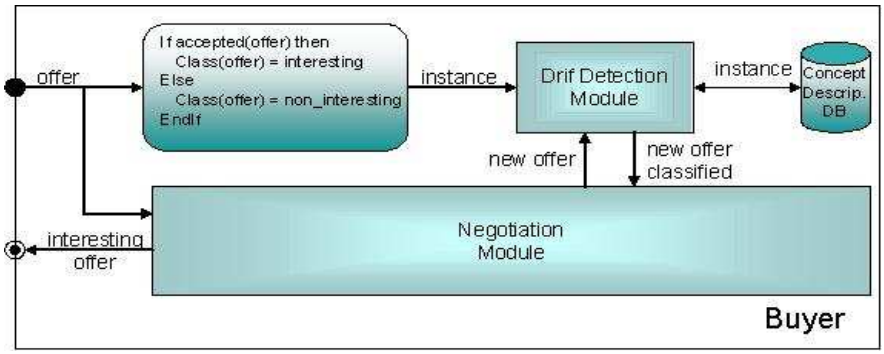

Fig. 2. Architecture of an Adaptive Trading Agent.

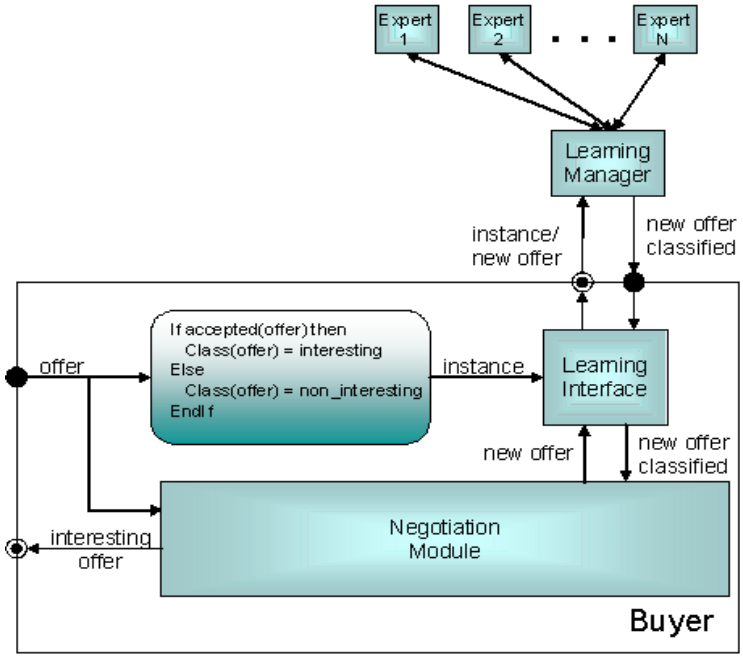

Fig. 3. Architecture for a trade MAS.

generation of offers. The learning manager agent coordinates the learning process and classifies offers as interesting or non-interesting. This information is used by the trade agent for directing its answer policy.

Currently, a learning manager uses the DWM [13] algorithm to coordinate Expert Agents. Such a distributed implementation improves the use of resources lke memory and processing power, because Experts can be distributed in different machines. During the execution of incremental learning algorithms each expert keeps

a database of instances of varying sizes. Besides, the classification can be very time consuming. When implementing a distributed architecture for concept drift, we indirectly reduce scalability problems. However, we need a well structured coordination protocol for the collective learning. The protocol is presented Figure 4.

The protocol developed here is based on the DWM (Dynamic Weighted Majority) algorithm proposed by Kolter and Maloof [13]. There are two use cases: the first one for learning and the second, for classifying. A classification is performed whenever the trade agent must do an offer. It will try to always propose offers that the manager classifies as "interesting" for the opponent agent. If an offer is classified as "non-interesting" it is rejected and the trade agent should generate another candidate offer. The "getGlobalPrediction" method does a weighted voting, returning the class with highest weight. The weight of a class is the sum of the expert weights that voted on it. The most important variables for the learning use case are the following:

- $x[i]$: feature vector of a learning instance i;
- $y[i]$: the class (interesting/non-interesting) of the instance i;
- $Beta$: a constant for decreasing weights of experts ($0 \leq Beta \leq 1$);
- $Teta$: a threshold for deleting experts;
- $e[j]$: an expert j;
- $w[e]$: the weight of expert e;
- $u[c]$: the weight of a class c;
- gp: global prediction (cooperative prediction);
- lp: local prediction (prediction of a specific expert);
- p: a period of time;
- $o[k]$: an offer k.

Whenever a learnt instance arrives the manager initializes the classes weights with 0 and interacts with each expert j asking the class for the instance. If the returned class is not y_i and it is time to learn ($i \ mod \ p = 0$) the weight of expert j is decreased ($w[j] = w[j] * Beta$). Then the classification weight for each class is calculated and the best class (higher weight) is found (global prediction). The weight for each expert is normalized (greater weight must be 1.0) if it is time to learn. Then the experts with a weight less than $Teta$ are deleted. If the global prediction is not correct a new expert is created with weight 1. Finally, the manager sends the instance to the experts for learning.

The interaction protocol of Figure 4 generates a reputation model for a set of experts e. The reputation of an expert $e[j]$ is represented by a weight $w[j]$. The size of e varies depending on the performance of the experts. When $w[j] < Teta$ then $e[j]$ is deleted. In fact, all the experts are evaluated when a new training instance arrives. If an expert $e[j]$ does a wrong prediction $w[j]$ is decreased. When a period of time is reached, the expert weights are normalized and weak experts are removed. In this case, if the global prediction is wrong, there are not enough current experts and a new expert $e[j+1]$ is created and $w[j+1] = 0$. Furthermore, the remaining

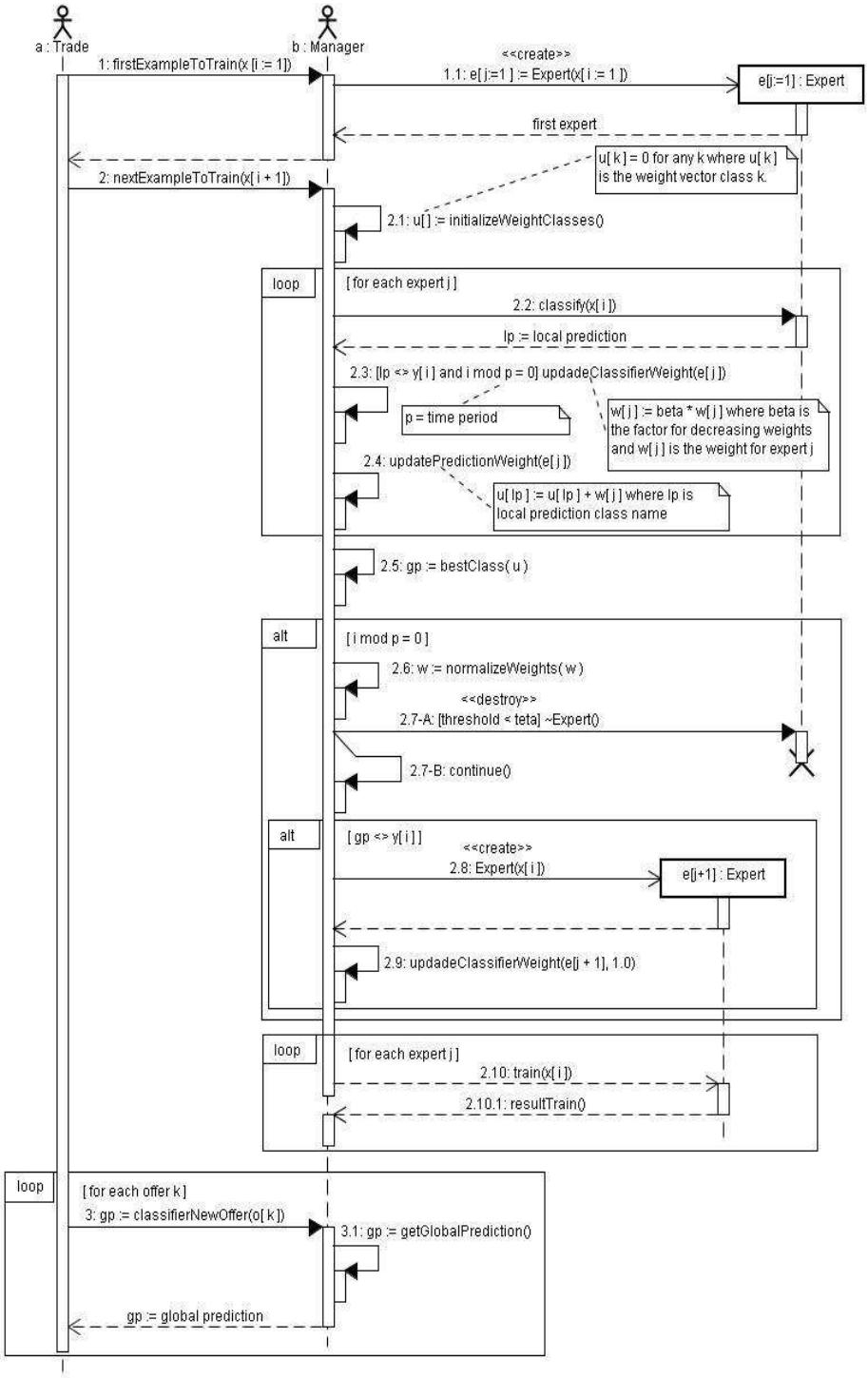

Fig. 4. Interaction Protocol for ensemble-based concept drift problem.

experts update local models with the current training instance.

We can observe that such a protocol is generic and different learning algorithms can be used by the experts. In this work we decided to implement homogeneous experts (all of them use IB3) to have a fair comparison with the simple IB3. Otherwise, we could not have analyzed the effectiveness of our approach.

The last loop of the protocol corresponds to the interaction between the *Negotiation Model* and the *Learning Interface*. The Learning Manager uses the reputation model generated previously to classify an offer generated by the Negotiation Model. They interact repeatedly until an offer is classified as *interesting*. The negotiation model then uses such an offer to answer the competitor trade agent. The number of interactions between Negotiation Model and Learning Interface depends on the negotiation tactics. Well-known tactics can be *imitative*, *time dependent* or *resource dependent* [5]. However, in this paper we do not study the use of such techniques, focusing only on the learning issue. Again, the introduction of specific tactics for the generation of candidate offers could produce biased results. We assume that results free of bias can be generated through random offers.

4.1. *Offer Space*

To illustrate the technique we need to define the offer space. For the sake of simplicity, we defined issues as qualitative variables. They may or may not be ordered. In this section we use the offer definitions proposed in [4]. Table 1 presents each of the issues used in this work.

Table 1. Description of Issues.

Issue	Domain
Color	[black, blue, cyan, brown, red, green, yellow, magenta]
Price	[very_low, low, normal, high, very_high, quite_high, enormous, non_salable]
Payment	[0, 30, 60, 90, 120, 150, 180, 210, 240]
Amount	[very_low, low, normal, high, very_high, quite_high, enormous, non_ensured]
Delivery_Delay	[very_low, low, normal, high, very_high]

4.2. *Offer Policies*

To contextualize the definition of concepts we employed a bilateral negotiation between a buyer and a sales agent. The former will always attempt to predict whether an arbitrary instance constructed upon an offer will be either *interesting* or *non-interesting* to the latter.

Definition(Concept). Henceforward, in this paper a **concept** is the description of an *interesting offer*. Whenever such a description is not valid for an offer, this offer is classified as *non-interesting*.

4.3. *Simulating Negotiation*

We consider that a vendor agent starts a negotiation with a specific definition for an interesting offer (concept). On the other hand, the buyer agent tries to discover such a definition. As the time goes by, the vendor drifts to another concept and the previous definition learned by the buyer is no longer valid. Thus the buyer is supposed to adapt the current definition to the new one.

Considering the definitions of concepts in the previous section, we used a procedure for generating instances similar to the Stagger system [25] and used in [22], [13], [8], [14] and [4]. We determined that a training or test instance can have its class labelled as *interesting* or *non-interesting*. First, a set of test instances is generated. They are used to assess the system accuracy. The Equation 1 is used to calculate the system accuracy. *TP* (True Positives) and *FN* (False Negatives) are the number of *interesting* and *non-interesting* offers classified properly respectively and T is the instances test set. In the experiments the size of T is 100.

$$accuracy(c, i) = \frac{TP + FN}{|T|} \times 100 \tag{1}$$

For each concept (c), an arbitrary number of training instances x (e.g. 50) is generated. The drift detection model then receives one training instance at a time (i) and is assessed on the test set. The procedure described in the last paragraphs is illustrated in Figure 5.

```
Foreach concept c ∈ [1,...,C] do
        generate test instances according to concept c
         Foreach instance i ∈ [1,...,x] do
                learn i with the drift detection approach
                evaluate the performance over test instances
        endForeach
endForeach
```

Fig. 5. An drift detection evaluation algorithm.

We have defined some hypothetical, disjunct concepts that a vendor can use and whose definitions the buyer must discover. Table 2 illustrates such a sequence of concepts, that represents an abrupt drift.

Concept drifts can also occur moderately. In this case, each successor concept maintains some degree of similarity to the previous one. To illustrate this situation we defined eight concepts described in Table 3. For the definition of such concepts we used the attributes *Delivery_Delay* that is constant, and *Amount* only. This sequence of concepts can illustrate, for example, the constant increase in the vendor's capacity of production. Consequently, the vendor is capable of increasing the amount of products delivered with short delay.

We also used two concepts A and B to simulate a gradual concept change in the sales agent. Both concepts are displayed in Table 4. The procedure for gradual

Table 2. Conceptual Description of Interesting Offers for Abrupt Drift.

Concept ID	Description
1	$(Price = normal\ and\ Amount = large)$ *or* $(Color = brown\ and\ Price = very\_low\ and\ Delivery\_Delay = long)$
2	$(Price = high\ and\ Amount = very\_large)$ *and* $(Delivery\_Delay = very\_short)$
3	$(Price = very\_low\ and\ Payment = 0\ and\ Amount = large)$ *or* $(Color = red\ and\ Price = low\ and\ Payment = 30)$
4	$(Color = black\ and\ Payment = 90\ and\ Delivery\_Delay = very\_short)$ *or* $(Color = magenta\ and\ Price = high\ and\ Delivery\_Delay = very\_short)$
5	$(Color = blue\ and\ Payment = 60\ and\ Amount = little\ and\ Delivery\_Delay = normal)$ *or* $(Color = cyan\ and\ Amount = little\ and\ Delivery\_Delay = normal)$

Table 3. Conceptual Description of Interesting Offers for Moderate Drift.

Concept ID	Description of an interesting offer
1	$(Delivery\_Delay = very\_low\ and\ Amount = very\_little)$
2	$(Delivery\_Delay = very\_low\ and\ Amount = little)$
3	$(Delivery\_Delay = very\_low\ and\ Amount = normal)$
4	$(Delivery\_Delay = very\_low\ and\ Amount = large)$
5	$(Delivery\_Delay = very\_low\ and\ Amount = quite\_large)$
6	$(Delivery\_Delay = very\_low\ and\ Amount = very\_large)$
7	$(Delivery\_Delay = very\_low\ and\ Amount = enormous)$
8	$(Delivery\_Delay = very\_low\ and\ Amount = non\_ensured)$

drifting generation was described in Section 3.1.

Table 4. Conceptual Description of Interesting Offers for Gradual Drift.

Concept ID	Description of an interesting offer
A	$(Color = blue\ and\ Delivery\_Delay = very\_short)$
B	$(Color = black\ and\ Price = high)$ *or* $(Color = magenta\ and\ Payment = 0)$

5. Experiments

The experiments we performed show the accuracy of the IB3 algorithm and the IB3-DWM algorithms applied to the concepts described in the previous section. The average results obtained with 20 iterations are discussed in this section. Initially, the systems were evaluated on the existence of abrupt drifts. We fixed the number of instances $x = 50$ to use with each concept. Figure 6 illustrates the obtained results.

It is also noticeable that the detection of drifts occurs quite rapidly in both systems, usually in less the 10 iterations for each new concept. Another remarkable

feature is the performance of the first concept that was just moderate; probably a greater number of iterations would have been necessary so that the IB3 algorithm could have constructed an adequate representation for concept 1. Such a behavior tends to be exhibited by IB3 due to its conservative model, adopted to prevent the algorithm from learning inaccurate relationships from noisy data. However we can clearly observe the improvement of performance thanks to the DWM coordination protocol. DWM improves the performance of IB3 for any concept significantly. On the other hand, such an improvement has a computational cost, since a number of experts is generated. Figure 7 shows the evolution in terms of number of experts.

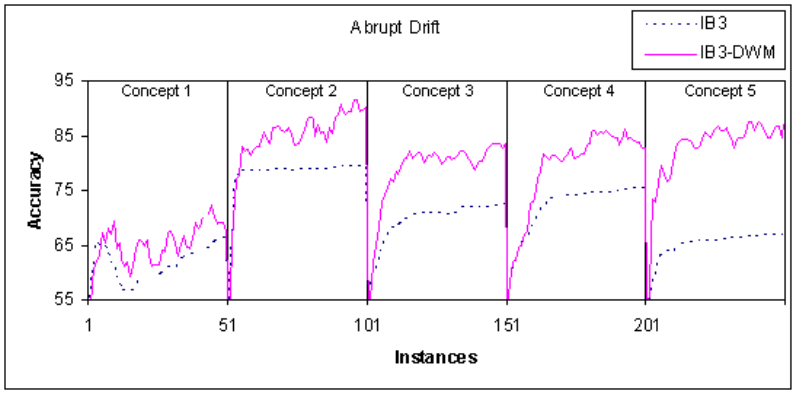

Fig. 6. Results of abrupt drift concepts.

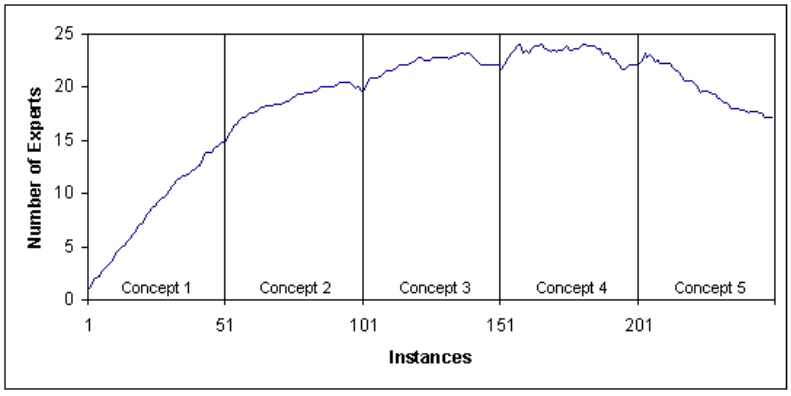

Fig. 7. Number of experts generated with abrupt concepts.

Results obtained from data with moderate drift shown Figure 8 are quite encouraging. One can clearly notice that falls in performance caused by changes are

much less than those identified Figure 6 (less than 10% on average). This behavior results from similarities among the produced concepts. One can also notice an increase in precision for each concept in comparison with its previous peer. However, the increase in precision occurs more slowly than it did when changes were abrupt. As concept descriptions learnt by the algorithm present a reasonable performance the algorithm reacts more smoothly in order not to remove instances whose absence might decrease the precision of the current model, once the stored instances present good classification accuracy. On the other hand, when the performance is deficient (in case of abrupt changes), performance in the classification of stored concepts tends to be rather poor and the algorithm executes frequent modifications in the set of instances that form the concept description. We can observe a great improvement with DWM experts. For most concepts, the prediction accuracy is in between 90% and 95%.

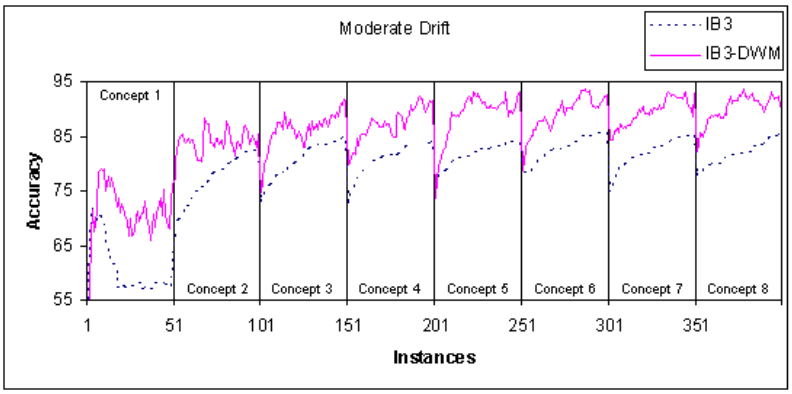

Fig. 8. Results of Moderate Drift.

Figure 9 presents the number of experts generated throughout learning. We can observe that the number of experts goes down after the concept 5, starting about 22 experts and finishing with 15 experts. We conducted experiments to assess the impact of gradual drift between a couple of concepts A and B in a number of instants during the learning process of the buyer agent. Both A and B were randomly generated. The first experiment aimed at measuring the influence of drift at the beginning of the agent learning process.

Figure 10 shows the results obtained with a drift zone ranging from 50 to 150. In this case, the agent performance is minimally affected by changes, maintaining an ascending learning bias. DWM outperforms again IB3 for any concept, however we can observe in the drift zone that DWM detects drifts more slowly. This happens because changes in the committee of experts are motivated by repetitive rewards and punishments stimulations. Thus, they can take a long period to be done.

Another observation is the stability of the number of experts for concept B. In

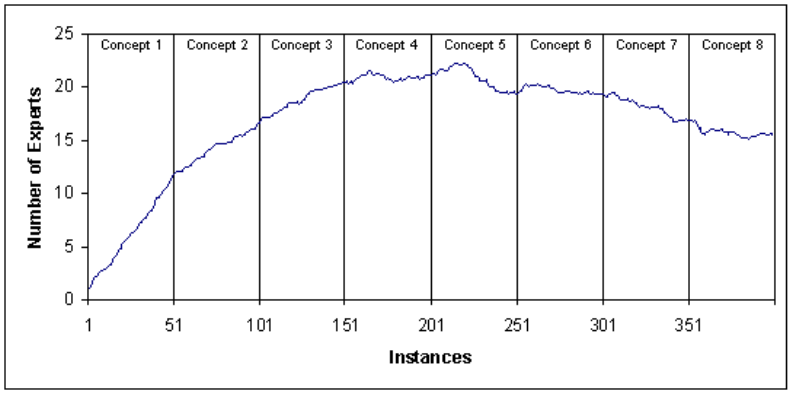

Fig. 9. Number of experts with moderate drift.

spite of the stabilization of the number of experts (around 18 in Figure 11), the accuracy is improved, showing that the system is learning the concept B properly. Comparing the results from the three types of drift (abrupt, moderate and gradual) we observe that the difference between the two algorithms is minimal for gradual drift. Theoretically, this type of drift is the easiest to learn because data distribution suffers a tiny change each time and no advanced techniques for detecting changes are necessary.

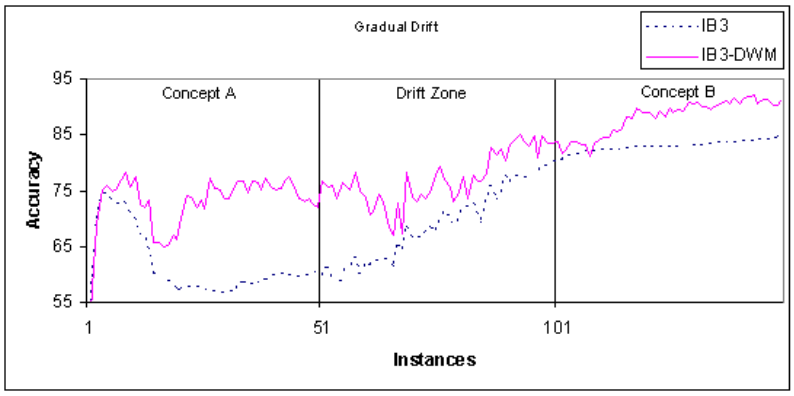

Fig. 10. Gradual Drift with drift zone $\in]50, 100]$.

We can state that considering the experiments that have been conducted, algorithms with drift detection capacities could be successfully used to learn offer policies, since results of all experiments show that the learning curve may suffer interferences, though it always maintains an ascending bias. In most cases, detection of changes is quite fast and the agent performance increases quickly. Such a

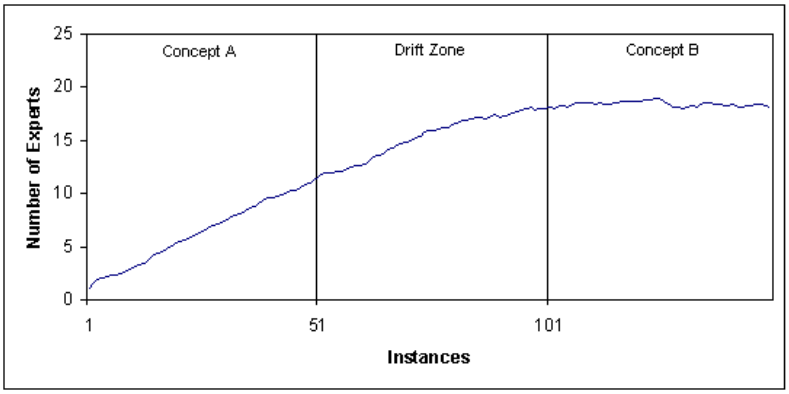

Fig. 11. Number of experts with gradual drift.

behavior makes the technique useful in most real negotiations scenarios involving human, automated or hybrid negotiations. We believe this is the stronger point of our technique.

Ensemble-based drift detection techniques are supposed to raise the accuracy of simpler drift detection techniques. In the experiments carried out, the DWM technique outperformed IB3 in all cases. This is to be expected because ensemble-based techniques use much more information (a number of experts) to make decisions and are less sensitive to the bias of a single classifier.

An interesting behavior observed in the experiments (Figures 7 and 9) is that the number of experts tends to decrease after a maximum value. This happens because ensemble-based algorithms use the diversity of classifiers to cover different regions of the tuple space. To profit of maximal performance, the experts have to be complementary. Thus, a number of experts is needed to cover a large region of the tuple space. When enough region is covered, some classifiers become useless, because they represent regions already covered by other experts. This explains why the number of experts decreases.

The number of needed experts is strongly related to the tuple space size and to the data distribution. For large tuple space, more experts are needed. However, depending on the hidden relationships between variables this rule might not be true in all cases and even small tuples spaces might require a lot of experts. Since the data distribution generation algorithm is the same for all the experiments, the curves displaying the number of experts tend to be similar. In the case of Figure 11 the number of experts would probably start to decrease around the 200^{th} training instance. Therefore different data sets can require a different number of classifiers, even if the same ensemble-based algorithm is used.

We could observe quite good results regarding the moderate and gradual drift modes. This is an important point because a negotiator rarely changes his mind drastically. On the other hand, it can make small changes in his goals very often.

Consider, for instance, a student who wants to buy a car. He is probably interested on a small or cheap car. However, as his earnings increases, the requirements for a new car changes slowly.

6. Conclusions

In this work, we demonstrated that ensemble-based algorithms developed for concept drift detection can be successfully used in bilateral and multi-issue negotiations. Recall that our goal was not to propose an integrated model of negotiation (see Section 4), but to show that ensemble drift detection techniques satisfy important constraints inherent to the problem of learning offer policies, such as effectiveness, efficiency, complexity and, mainly, changes in the agent behaviors. When building trading agents for dynamic environments, developers should consider ensemble-based drift detection techniques using a staff of learner agents.

Experiments performed on simulated concepts of negotiation showed that the interaction protocol developed based on DWM outperforms the IB3 algorithm and detects changes rapidly. It can be used to update an agent concept description in order to improve its performance even with a few iterations only. In that case, learning is carried out by an ensemble of distributed learning agents. We started from the idea that a negotiation model could exploit the drift model to generate offers that are interesting for both agents involved in the negotiation process, and, as a result, the agent performance in the negotiation process should be proportional to the one obtained by the drift detection model.

In future work, we intend to employ techniques such as the ones described in this work to improve existing negotiation models, similarly to what has been proposed by Coehoorn and Jennings [3]. Once a complete negotiation model is built, we will be capable of assessing the utility values obtained by agents that use different drift detection algorithms. Finally, in spite of a higher computation cost, ensemble-based techniques can produce fairly satisfactory results for the detection of different types of concept drifts.

Acknowledgements

This research was funded by Pontifical Catholic University of Paraná. We thank Diogo Domanski and Richardson Ribeiro for comments and time spent with the discussions.

References

1. D. W. Aha, D. Kibler and M. K. Albert, Instance-based Learning Algorithms, *Machine Learning*, n. 6, v. 1, pp. 37-66, 1991.
2. S. Buffet and B. Spencer, *Learning Opponents' Preference in Multi-Object Automated Negotiation* Proc. of International Conference on Electronic Commerce'05, pp. 300-305, Xi'an, China, 2005.

3. R. M. Coehoorn and N. R. Jennings, *Learning an Opponent's Preferences to Make Effective Multi-Issue Negotiation Trade-Offs*, Proc. of the Sixth International Conference on Electronic Commerce, pp. 59-68, 2004.
4. F. Enembreck, F., B. C. Avila, E. E. Scalabrin and J-P. A. Barthès, Drifting Negotiations, *Applied Artificial Intelligence*, Taylor & Francis, pp. 861-881, v. 21, n. 9, 2007. ISSN: 1087-6545
5. T. Faratin, C. Sierra and N. R. Jennings, Negotiation decision functions for autonomous agents, *Robotics and Autonomous Systems*, v. 24, n. 3-4, pp. 159-182, 1998.
6. T. Faratin, C. Sierra and N. R. Jennings, Using similarity criteria to make issue tradeoffs in automated negotiations, *Artificial Intelligence*, n. 142, v. 2, pp. 205-237, 2002.
7. S. Fatima, M. Woodridge and N. Jennings, An agenda-based framework for multi-issue negotiation, *Artifcial Intelligence*, n. 152, v. 1, 2004.
8. L. Gama, P. Medas, G. Castillo and P. Rodrigues,*Learning with Drift Detection*, Proc. of the 17th Brazilian Symposium on Artificial Intelligence - SBIA'04, LNAI 3171, Ana L.C. Bazzan and Sofiane Labidi (eds.), 2004. ISBN 3-540-23237-0
9. E. Gerding and D. van Bragt, Multi-issue negotiation processes by evolutionary simulation, validation and social extensions,*Computation Economics*, n. 22, v. 1 pp. 39-63, 2003.
10. D. P. Helmbold and P. M. Long, Tracking drifting concepts by minimizing disagreements, *Machine Learning*, n. 14, v. 1, pp. 27-46, 1994.
11. S. Huang and F. Lin,*Designing Intelligent Sales-agent for Online Selling*, Proc. of the International Conference on Eletronic Commerce, pp. 279-286, Xi'an, China, 2005.
12. R. Klinkenberg and I. Renz, *Adaptive Information Filtering: Learning in the Presence of Concept Drifts*, ICML-98, pp. 33-40, 1998.
13. J. Kolter and M. A. Maloof, *Dynamic Weighted Majority: A New Ensemble Method for Tracking Concept Drift*, Proc. of 3th International IEEE Conference on Data Mining, pp. 123-130. IEEE Press, 2003.
14. J. Kolter and M. A. Maloof, *Using additive expert Ensembles to Cope with Concept Drift*, 22th. International Conference on Machine Learning, ACM Press, pp. 449-456, Germany, 2005. ISBN:1-59593-180-5
15. R. Y. K. Lau, *Adaptive Negotiation Agents for E-business*, International Conference on Eletronic Commerce, pp. 271-278, Xi'an, China, 2005.
16. M. Maloof, Incremental Learning with Partial Instance Memory, *Artificial Intelligence*, v. 154, n. 1-2, pp. 95-126, 2004. ISSN:0004-3702.
17. T. Mitchell, Machine learning, *McGraw Hill*, 1997.
18. S. Saha, A. Biswas S. and Sen, *Modeling Opponent Decision in Repeated One-shot Negotiations*, Int. Conference on Autonomous Agents and Multi-Agent Systems - AAMAS'05, pp. 397-403, 2005.
19. J. Schlimmer and R. Granger, Beyond incremental processing: Tracking concept drift, Proceedings of the 5th. AAAI Conference, *AAAI Press*, pp. 502-507, 1986.
20. M. Scholz, and R. Klinkenberg, *An Ensemble Classifier for Drifting Concepts*, Proceedings of the Second International Workshop on Knowledge Discovery in Data Streams, J. Gama and L. S. Aguilar-Ruiz (eds), pp. 53-64, Porto, Portugal, 2005.
21. M. Scholz, and R. Klinkenberg, Boosting Classifiers for Drifting Concepts, Intelligent Data Analysis (IDA), *Special Issue on Knowledge Discovery from Data Streams*, pp. 3-28, IOS Press, v. 11, n. 1, 2007.
22. K. O. Stanley, *Learning Concept Drift with a Committee of Decision Trees*, Department of Computer Sciences, University of Texas at Austin, Technical Report AI-03-302, September, 2003.
23. W. N. Street and Y. Kim, *Streaming Ensemble Algorithm (SEA) for Large-Scale*

Classification, Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM Press, pp. 377-382, 2001. ISBN:1-58113-391-X

24. A. Tsymbal, *The problem of concept drift: definitions and related work*, Technical Report TCD-CS-2004-15, Department of Computer Science, Trinity College Dublin, Ireland, April 2004. http://citeseer.ist.psu.edu/tsymbal04problem.html.
25. G. Widmer and M. Kubat, Learning in the presence of concept drift and hidden contexts, *Machine Learning*, n. 23, v. 1, pp. 69-101, 1996.
26. R. Kohavi, *Bottom-Up Induction of Oblivious Read-Once Decision Graphs: Strengths and Limitations*, Proceedings of the European conference on Machine Learning, pp. 154-169, 1994. ISBN:3-540-57868-4
27. D. Zeng and K. Sycara, Bayesian learning in negotiation, *International Journal on Human-Computer Studies*, v. 48, n. 1, pp. 125-141, 1998. ISSN:1071-5819