UNIVERSITAT ROVIRA i VIRGILI

Introduction to Multi-Agent Systems

Agent-based Decision Support System (A-DSS) for Fraud Classification

Team 04

Anne Schreiber
Joaquim Marset
Shivani Patel
Ruben Vera
Genevieve Masioni

Contents

1. Introduction

This project presents the idea of developing a Multi-Agent System that can classify if a particular firm is fraudulent or not, based on present and past risk factors. The agent-based decision support system (A-DSS) would be used by audit companies that want an automatic mechanism to detect fraudulent activities in a firm.

A Multi-Agent System (MAS) is defined by being a system that includes multiple agents that interact with each other to fulfill a certain task. An agent can be defined as a computer system that is situated in an environment and that is capable of taking actions and decisions by itself to fulfill its tasks. (Wooldridge, 2002, p.3)

In order to create the system, we will be using the UCI's Audit dataset, which contains different risk factors and tries to predict if a particular instance (a firm) is fraudulent or not.

We will also be using Weka, open-source software that provides different tools for data processing, machine learning algorithms and visualization tools (Tutorialspoint). This Multi-Agent System has the necessity to train a certain number of classifiers that can perform the aforementioned prediction. In particular, each classifier will use the same J48 algorithm, an open-source Java class for generating a pruned or unpruned C4.5 decision tree (Frank). Once we have a prediction for each classifier, we will need a voting mechanism to produce the final prediction that the user will receive.

2. Final Design

The overall architecture of the agent-based decision support system (A-DSS) is in Fig. 1. As it can be observed, there are 4 agents that will perform different tasks to be able to detect if a firm is fraudulent or not. Only one of them will communicate with the user, and the communication between the different agents and the user is bidirectional. A particular explanation of each agent, with its architecture, type, and properties will be given below.

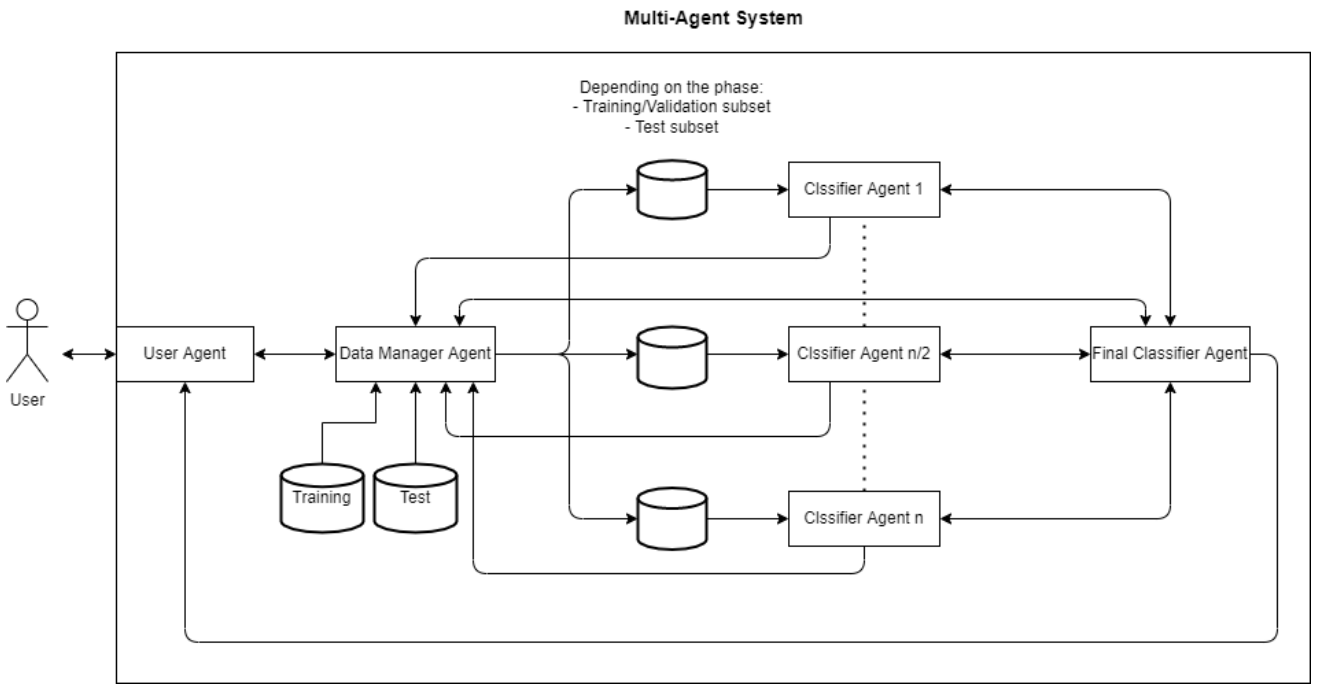


Fig. 1 Design of the Multi-Agent System

2.1 User Agent

As the name implies, this is the agent that will communicate with the user who wants to predict if a particular firm is fraudulent or not. It will be the only agent that will communicate with the user, and will be the starting and end point of the entire execution flow.

The user agent will be the one asking the user to introduce an XML file containing the configuration the system will use, as well as asking about the test instances to predict (and which attributes these test instances will have). The testing phase will start either with another XML file containing the data, or letting the system to randomly decide.

Once it has the configuration to initiate the system, it will send it to the Data Manager Agent, who will be the one in charge of keeping the training and test sets used during the system execution. The configuration received will contain parameters like the paths where the datasets are stored, the number of instances each classifier will use, or the validation percentage to use. In the section explaining the implementation details, it is properly explained.

Then, once the system is ready to accept predictions (i.e. after the classifiers have been trained), it will inform the Data Manager Agent about the instances the user wants to predict, and which attributes should be used. This will delegate all the work and wait for the final prediction.

2.1.1 Architecture

This agent will present a **reactive** architecture, given that we only want this agent to act the first time when the system initiates, and when the user asks the system to perform a query. Moreover, it will also act when the final predictions for each test instance comes from the Final Classifier Agent.
We do not need this agent to have long-term goals or create any kind of plan, as it mainly delegates work or performs simple tasks like reading user input and parsing files.

2.1.2 Properties

There are some properties that will be shared among the 4 agents, so we are going to explain them once, and simply mention them in the next agents.

- *Reactive*: We are considering a static environment where we only have our 4 agents and the human agent that wants to use the system. The different software agents will be continuously interacting between them, sending and receiving messages about performing tasks, or simply passing information. Therefore, we want the agents to be always aware of their environment to ensure that no communication is lost, and we also want the agents to respond in a fast time.

- *Social Ability*: All the agents interact with other agents of the system, sending and receiving FIPA-ACL messages to correctly coordinate and cooperate to perform the system's task. Given that all the agents are benevolent, there is no negotiation involved in their communications.

- *Rational*: All the agents will only act to fulfill their goals, never acting in a way that may make the achievement of these goals difficult.

- *Autonomous*: All the agents will perform their tasks without any user interaction or guidance. It is true that the choice of system configuration, as well as test queries will restrict the agents, as it will change how the system executes.

- *Temporal Continuity*: All the agents will be running indefinitely until the system stops. However, it is not the same as running a script that produces an output, and it ends its execution. In this case, we will have the system running all the time, predicting new test instances that some users may need to predict.

2.1.3 Type

We consider it an **Interface Agent**, given that this agent will be the one communicating with the user, receiving the system configuration, test queries, as well as displaying the final prediction. It will neither learn nor be proactive, as it will not perform any recommendation or learn user preferences, but only serve queries.

Also, and this will be shared among all 4 agents, we can consider the agents to be collaborative, given that all of them want to fulfill the A-DSS goal of being able to predict fraudulent firms. Given that all these agents are created by the same group, they are benevolent, and they always work towards the same goal, without competing or negotiating.

2.2 Data Manager Agent

This agent will be the one storing the datasets that will be used for training and testing. That is, out of the 767 instances the complete dataset contains, it will store 717 for training, and 50 for testing. It will also randomly decide which subset of 300 instances each classifier will train with, as well as which 6 attributes (out of the initial 25) those 300 instances will contain. Also, it will further separate the 300 instances, leaving 25% as a validation set. The number of training instances and attributes, as well as the validation percentage, can be modified when setting the initial configuration.

Once it has created the training/validation subsets for each classifier, this agent will be the one that will send them to each Classifier Agent. Also, this Data Manager Agent will contain information about the attributes each classifier uses, when later performing test queries.

Once the system has been trained, and the user performs a query with some instances to predict, this agent will decide which classifiers will receive each instance. The decision will depend on the subset of 20 features each test instance has, as well as the subset of 6 instances each classifier has been trained with. A classifier receives an instance to predict if the instance includes all the 6 features the classifier works with.

2.2.1 Architecture

The agent presents a **reactive** architecture, as it will perform a simple task that will not need any kind of planning. There is a small set of static rules that would be used for dividing and allocating the data into chunks of (300×6) for training, and the same for testing. However, the agent does not need to learn or demonstrate any kind of "intelligent" behavior to do so. The agent has a specific simple task that would not need any kind of planning module.

2.2.2 Properties

This agent will present the 5 basic properties that have been explained before:

- *Reactive*
- *Social Interaction*
- *Rational*
- *Autonomous*
- *Temporal Continuity*

2.2.3 Type

This agent is a **Facilitator Agent**, and being more specific, it will act as a broker agent. This agent serves as an intermediary between the User Agent, Classifier Agents, and the Final Classifier Agent. These last 2 will be the ones that will end up generating the predictions the human agents want to obtain. We consider it to be a Broker, given that it will store the datasets that the different agents will use during all the execution flow.

2.3 Classifier Agent

This agent is the one that will embed the Weka J48 decision tree that will classify the different firm instances. We will have at least 10 instances of this agent, each one embedding its own classification model. Each classifier will receive a training set of 300 instances and 6 attributes, leaving some of those instances as a validation set to assess the model performance.

Each classifier will generate some performance metrics using the validation set to assess the model performance. Some of them will only be printed for information purposes, but others will be used on the Final Classifier Agent to combine the predictions each Classifier Agent generates for each test instance. In particular, the F1-score will be used to weigh the importance of each classifier in the combined predictions. In a later section, we explain why we decided to use this metric.

Once the classifiers have been trained, the classifiers will receive test instances, and if those instances have all the attributes the classifier needs, it will generate a prediction for each one. These predictions will be also handed to the coordination mechanism to generate the final prediction.

2.3.1 Architecture

This agent will follow a **hybrid** architecture, given that it will follow a plan to build the decision tree model (i.e. the Weka algorithm to create the decision tree). Nevertheless, once it has been trained it will act reactively, without complex reasoning, waiting for instances to classify, generating the predictions, and communicating the results to the Final Classifier Agent.

2.3.2 Properties

This agent will have the following properties, including the 5 basic ones:

- *Reactive*
- *Social Interaction*
- *Rational*
- *Autonomous*
- *Temporal Continuity*

- *Learning*: The agent learns a model that can distinguish at some extent normal and altered data from a firm. Selecting the best feature at each step, the cutting point for each

feature, the post-pruning it does with the validation set, can result in different trees with different performances. However, once the model has been built, it will not learn more if it does not receive more data outside the initial dataset to refine the resulting trees.

- *Proactive*: As a classifier, these agents have a goal-directed behavior (computing predictions for a subset of data). While learning, it takes initiative, when necessary, to ensure the best classification performance.

- *Reasoning*: We can consider that this agent reasons, given that it uses its knowledge-base (i.e. the model) to infer the predictions for each queried instance.

2.3.3 Type

This agent could be considered to be a combination between a **Bridge Agent** and a **Wrapper Agent**. We cannot consider it as a pure bridge given that it is embedding the J48 model, but we cannot consider it as a wrapper either. The reason for this is that we are not modifying the internal code of the J48 algorithm to communicate with the other agents. Also, this agent interacts with the others, receiving training, validation, and test sets, as well as returning predictions and performance measures.

2.4 Final Classifier Agent

This agent's purpose is to compute the ultimate prediction, which corresponds to the A-DSS' response for the asked test instances to predict. To compute the predictions, it combines the predictions each Classifier Agent has been able to generate for each test instance (if the agent can classify those instances), using a voting scheme. In particular, we are using a type of Plurality Voting scheme, but instead of each classifier voting 1, it votes its F1-score. That is, when computing the final predictions, the F1-score is used to weigh the prediction of that classifier, and a weighted average is computed. Then, the result is normalized and rounded to be either 0 or 1. In a later section, the voting scheme is properly explained.

2.4.1 Architecture

This agent has a **reactive** architecture, given that it is simply computing a weighted average of the predictions it will receive from the different Classifier Agents, using the F1-score that those same agents will have sent in a previous step.

2.4.2 Properties

This agent will have the 5 basic properties:

- *Reactive*
- *Social Interaction*
- *Rational*

- *Autonomous*
- *Temporal Continuity*

2.4.3 Type

We are simply going to consider this agent to be a **Collaborative Agent**, given that it cooperates with the others to fulfill the system goal, but it does anything special to be categorized as another type of agent.

3. Implementation details

3.1 Main Functionality

In this first section, we are going to briefly explain the main functionalities of our system, which later are more specific when describing the different agents.
First, when the system is started, a file explorer is prompt, and the user has to select an XML file containing the configuration to initialize the system. When selected, if the configuration was correct, the datasets will be read, the classifiers will be created and later trained. If not, the same window will appear to select a configuration file again.
The policy used to select the training instances and attributes for each classifier is completely random, only ensuring that we are following the asked system configuration.
Each time a classifier ends training its model, some statistics obtained from the confusion matrix will be printed.

Once the system is trained, it will be ready to perform test queries. For this reason, a message will appear in the terminal, asking to select between the 2 testing modes. The user will need to type either '1' or '2' to run the desired mode.
The first mode will open again a file explorer to select another XML file, now containing the indices of the test instances (from the same test dataset the user has specified in the initial configuration), as well as the attributes to use in those test instances.
If there is some invalid data in the test query, the window will appear again to select another file.
If the second mode is selected, the system will randomly decide the test instances, as well as the attributes, again from the same test dataset.
When a classifier is able to predict those instances, the results will be displayed, printing the prediction together with the ground-truth label, if available.

We want to mention that in case the XML files contain invalid data, the reason will be prompt to the user. Also, the specific format of the XML files, as well as how the data should be specified, can be found in the README of the GitHub project.

3.2 User Agent

3.2.1 States

The UserAgent has been designed to be in one of the sequential 6 states:

1. **InitSystem**
   When the A-DSS first starts, the UserAgent being the point of entry to the system, begins with this state. At this point, the system prompts the user to provide it with an XML file from which it will load the system configuration. The contents of the XML are:
   1. No. of classifiers
   2. Location of the training dataset
   3. Location of the testing dataset
   4. No. of training instances
   5. No. of attributes to be considered for training
   6. Percentage of training data to be considered for validation

   This config is then passed on to the DataManagerAgent as a *FIPA-REQUEST*. The agent then changes its state to WaitForTraining.

2. **WaitForTraining**
   While the ClassifierAgents are training, the UserAgent is blocked with this state. It awaits further confirmation from the DataManagerAgent about the state of the config it had earlier passed:
   1. An ***AGREE*** message states that the configuration was successful, and the classifiers are training now. This message does not change the current state.
   2. A ***REFUSE*** type of message would mean the training cannot be done with the system configuration that was passed, and so the current state reverts to InitSystem and the user would have to load the config again.
   3. An ***INFORM*** message means that the classifiers have successfully trained the model and querying phase can now begin. The state changes to PerformTestQueries.

3. **PerformTestQueries**
   With the model trained and ready to be tested, the UserAgent now prompts the user to enter the mode to test:
   1. Whether to use a query via an XML - which states the instances and attributes to be used, or
   2. Let the system randomly pick out 15 instances and 20 attributes, from the pool of 50 instances set aside in the very beginning.

   Once the test query is set by either of the above 2 modes, it is sent to DataManagerAgent and the UserAgents is blocked again with a WaitForQueriesAcceptance state.

4. **WaitForQueriesAcceptance**
   After sending the test query, the UserAgent can expect 2 types of responses from DataManagerAgent:
   1. ***REFUSE*** - the system has to go back to PerformTestQueries state to be able to get the test query again.
   2. ***AGREE*** - The DataManagerAgent has checked that there are no problems with the passed test query, and at least one classifier will be able to predict the instances. The UserAgent now goes to a blocking mode with the next state while waiting for the results.

5. **WaitForQueriesResults**
   In this state, the UserAgent only expects an INFORM response from DataManagerAgent - indicating that the classification has been done on the test data, the results have been published and one complete round of the A-DSS is completed.
   The state now changes back to wait for new test queries to be introduced.

3.3 Data Manager Agent

3.3.1 States

The DataManagerAgent has been designed to be in one of the following sequential states:

1. **WaitingForSystemConfiguration**
   The first state this agent will be in when initiated. In this state, the agent is waiting for a REQUEST from the User Agent, containing the system configuration this agent needs. When trying to retrieve the configuration, if some error happens, a REFUSE will be sent back. If not, the configuration will be retrieved and the parameters will be checked. A REFUSE can also be sent back if the configuration contains some invalid parameter (e.g. asking to train a Classifier Agent with more instances the training dataset contains).
   This state ends with the DataManagerAgent sending to the Final Classifier the number of Classifier Agents that will be created, inside an INFORM message. This is needed for the Final Classifier to know how many messages it should expect to receive in some step of the execution flow.
   Once finished, the agent will transition to the WaitingForFinalClassifierAck.

2. **WaitingForFinalClassifierAck**
   As the name implies, the Data Manager Agent is blocked waiting for the acknowledgment (i.e. INFORM) from the FinalClassifier, telling that it has correctly received either the number of created classifiers or the number of classifiers that will predict a test query.

   When receiving the first INFORM, the state will simply change to CreateClassifiers.
   Once it receives the second ACK, it will send the preprocessed test instances (i.e. selecting the instances and attributes) of the test query that the user will do, starting a FIPA-REQUEST protocol with each Classifier Agent that can classify those instances. Later, it will transition to WaitingForClassifiersToPredict.

   This is a pattern that will be observed in many steps of this and other agents. When we are sending simple information (i.e. using an INFORM), we want to receive back some type of ACK (also as an INFORM message) to know that everything has gone as expected, and the agent can safely transition to the next state.

3. **CreateClassifiers**
   In this step, the agent will dynamically create the number of Classifier Agents specified in the received configuration. With the creation of each ClassifierAgent, a FIPA-REQUEST protocol will start between the Data Manager Agent and each ClassifierAgent, starting with the REQUEST containing the train and validation

instances (randomly selected) it will use to train the J48 decision tree.
Once all the classifiers have been created, and the train and validation instances have been sent, this agent will transition to the WaitingForClassifiersToTrain state.

4. **WaitingForClassifiersToTrain**
   The agent is blocked waiting for the INFORM messages as part of the initiated FIPA-REQUEST, informing about having finished its training and validation phase.
   Once all the INFORM have been received, this agent will answer the REQUEST initiated by the User Agent asking to initiate the system, with another INFORM. This will enable the testing phase.

5. **WaitingForQueries**
   Similar to the first state, the agent is waiting for a REQUEST from the User Agent containing the test query that should be processed. Again, if some problem happens when retrieving it, or it contains some invalid data (e.g. wrong instance index, or wrong attribute name), a REFUSE will be sent back.
   Once the test query is retrieved, the desired test instances, with the desired attributes, will be gathered. Also, it will be determined which Classifier Agents will be able to predict these test instances.

   If no classifier can classify those instances, a REFUSE will be sent back to the UserAgent telling the test query cannot be processed.
   If not, an INFORM will be sent to the Final Classifier telling how many messages to expect from the ClassifierAgents (i.e. it will only wait for those being able to process the instances). Also, an AGREE will be sent to the User Agent as a way of indicating that the test query will be processed, and it has to wait for the results. We want to note that the results will be sent to the UserAgent by the Final Classifier, the FIPA-REQUEST protocol the User Agent initiated will be answered by 2 different agents.

   Once it is done, it will transition to WatingForFinalClassifierAck, to wait again for the acknowledgement from the Final Classifier Agent of having received that number.

6. **WaitingForClassifiersToPredict**
   This last state is used to block the agent until all the possible classifiers predict the test instances. The idea is that it will wait for an INFORM for the FIPA-REQUEST it has initiated with each Classifier Agent that can predict the test instances. This INFORM will not contain the results of the predictions, but simply inform about everything going correct. Once it receives all of them, it has no more responsibility in this loop of predicting the test instances. Then, it will transition to WaitingForQueries.

3.4 Classifier Agent

3.4.1 J48 decision trees

Before discussing the states of the Classifier Agent, it is necessary to further develop how the machine learning algorithm is used to create a model that classifies our test instances. The selected method is J48 decision trees, an open-source Java class for generating a pruned or

unpruned optimized implementation of a C4.5 decision tree (Frank). A decision tree has a tree structure having root, intermediate, and leaf nodes. As can be extracted from the name, each node consists of a decision leading to a result that can be another intermediate node or a final and leaf one.

It is important to notice that in each decision that is made, the input space is divided into exclusive areas. That is why, the root node is the attribute with the best splitting criterion and the nodes near the root are the following ones with best splitting criterion. The last intermediate nodes usually do not have a good splitting criterion, but as there are fewer samples to split, the error is minimized.

3.4.2 F1-score

The F1-score is a metric used to assess the performance of a supervised learning model, computed using the information of the confusion matrix. In particular, it is computed as a harmonic mean of the Precision and Recall metrics.
The idea of those two metrics is to try to assess the performance when we are interested in being able to classify a particular class. For example, in this project we are interested in detecting fraudulent firms by looking at their data (i.e. the altered class).
Precision is defined as the ratio of instances we have predicted as positive that were in fact positive. Recall is defined as the ratio of real positive instances that we have been able to predict. Using one or the other typically depends on the type of error (between false positives and false negatives) we want to minimize. Also, minimizing one metric usually implies increasing the other.
In this case, we have considered that both types of errors are equally important, and it is in those cases that we use the F1-score.

3.4.3 States

The Classifier Agent has been designed to be in one of the following sequential states:

1. **PendingToTrain**
   In this first step, the classifier agent waits for the training and validation sets the Data Manager Agent will send. Once it receives them, it can train the J48, perform validation, and generate the F1-score.

   The resultant F1 Score will be sent to the **Final Classifier Agent** with a FIPA INFORM message. After this, the Classifier Agent changes its state to **WaitingForMetricsAck.**

2. **WaitingForMetricsAck**
   In this second step, the Classifier Agent will be waiting for an INFORM response from the **final Classifier Agent,** so this agent will know that its previous message has arrived correctly and without errors.

   When the ACK is received, the Classifier Agent will change the state to **Trained.**

3. **Trained**
   Thirdly, when the agent has finished training and received the ACK from the final Classifier Agent, this agent will send an INFORM message to the Data Manager Agent telling that it has finished the training. Then, the agent will change to its fourth state, **WaitingForQueries.**

4. **WaitingForQueries**
   Fourthly, the agent is ready to receive test queries to predict, and it waits for a REQUEST message from the Data Manager Agent. Once it has received the test instances, in the remote case it cannot retrieve it, a REFUSE will be sent back.

   Nevertheless, if the test instances were received correctly, the Classifier Agent will start testing and classifying these instances. Immediately, the agent will send a REQUEST message with the testing predictions to the Final Classifier Agent, and change its state to **WaitingForPredictionsAck.**

5. **WaitingForPredictionsAck**
   Finally, the agent will wait to receive the INFORM message from the Final Classifier Agent, confirming the reception of the classified test instances.
   It will send an INFORM to the Data Manager Agent, ending the FIPA-REQUEST protocol this last agent started, telling that the instances have been predicted.
   Lastly, the classifier agent will reverse its state to **WaitingForQueries** and wait until a new set of test instances that need to be predicted.

3.5 Final Classifier Agent

3.5.1 Voting Policy

Before diving into the code and the decisions made in it, the necessity to talk about voting protocols is necessary. The present agent uses a simple voting protocol to select the best predicted outcome among the Classifier Agents, which will be given to the final user.

The classifier uses the Plurality Voting Protocol, which selects the option that was most voted by the agents. The agent's votes for a particular test instance are weighted using their respective F1-score. That is, if one Classifier Agent predicts a positive class for a test instance, but their performance was very bad, this vote will have a low weight in the final decision. To generate the combined prediction for each test instance, a weighted average using this F1-score will be used, normalizing it by the sum of F1-scores. Then, the result will be rounded to generate either a 0 or a 1. These combined predictions will be the final predictions that will be sent back to the User Agent, and displayed to the final user.

3.5.2 States

The Final Classifier Agent has been designed to be in one of the following sequential states:

1. **ReceiveNumOfClassifier**
   In this state, the Final Classifier Agent will be waiting to receive from the Data Manager Agent the number of created Classifier Agent. Once it receives it, it will send an INFORM to the Data Manager Agent, as an ACK message, informing that has

correctly received that information. Then, the agent will change to the next state, which is *ReceiveMetrics*.

2. **ReceiveMetrics**
   When the Final Classifier Agent has this state, the agent will receive from each Classifier Agent the corresponding validation metric of that agent. Then, it will send an INFORM as an ACK to the corresponding agent, informing that the F1-score metric was received successfully. While receiving the metrics from different classifier agents, the Final Classifier Agent will store them for later use.

   This state will be concurrent as there are many Classifier Agents.
   After receiving, storing and informing, the agent will change its state to the next one.

3. **ReceiveNumOfClassifying**
   In the third state, the agent will receive from the Data Manager Agent the actual number of classifiers that will participate in the prediction of the testing subset. Once it receives that number, it will send an INFORM as an ACK to the Data Manager Agent, about having received the information. Then it will proceed to transit to the next state, which is *ReceivePredictions*, and will create an ArrayList to store the receiving predictions.

4. **ReceivePredictions**
   In this state, the Final Classifier Agent is blocked waiting to receive the predictions of the different Classifier Agents, through an INFORM message. Once it has received the predictions of each agent, it will send back another INFORM as ACK, informing about having received the data.
   Once it has the predictions of all the agents that can classify those instances, it will combine the predictions using the explained voting mechanism.
   After having generated the final predictions, it will transit to the *ReturnCombinedPredictions* state.

5. **ReturnCombinedPredictions**
   In the last state, the Agent will send the final combined predictions to the User Agent, through an INFORM ACLMessage. After sending the prediction, the Final Classifier Agent state will return to the third step, to serve another test query.

3.6 Agent Communication

In this section, we are going to explain the communication mechanisms that we have used in our system. More or less they have been mentioned in the details of the implementation of each agent, but we are going to properly explain them here.

In particular, we have been using the FIPA-REQUEST protocol, with some modifications depending on the situation, as well as simple Inform-ACK messages to communicate simple information.

As we have studied in class, the FIPA-REQUEST is used when one agent asks another to do a certain task. It is summarized in the diagram below:

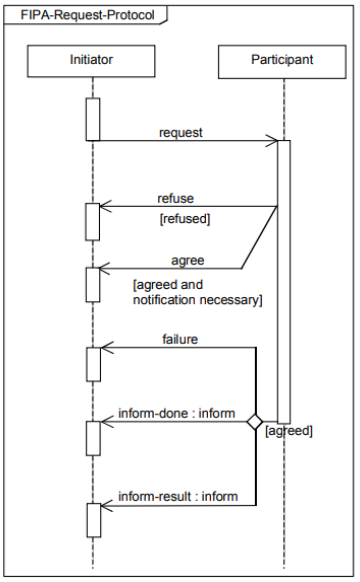


Fig. 2 FIPA-REQUEST protocol flow chart

It starts with the Initiator sending a REQUEST message asking for something. Then the Participant can refuse to serve that request, sending a REFUSE message, or accepting it by sending an AGREE message. Typically, the step of sending the agreement is omitted.
If the Participant accepts, then it will perform the task, and either answer with a FAILURE message if it could not accomplish the task, or answer with an INFORM that the task has been completed.

This protocol has been used in 4 situations:

1. The User Agent asks the Data Manager Agent to initiate the system with the decided configuration
2. The Data Manager Agent asks each Classifier Agent to train the J48 with the corresponding train and validation sets
3. The User Agent asks the Data Manager Agent to predict the desired test query
4. The Data Manager Agent asks each Classifier Agent that can predict the test query, to do it.

Except in the third case, we are simply following the original version, but omitting the AGREE message. The reason for this is that it will always agree to do the task unless there are some problems when reading the system configuration, or the train and validation instances.

In the third case, we have modified a little the way the protocol works, given that the AGREE message and the INFORM message are sent back from different agents. In particular, the AGREE will be sent by the Data Manager Agent once the test instances have been received, checked that it contains valid data, and checked that at least one Classifier Agent can predict them.
If the test instances have been received, but there is some issue, the Data Manager Agent will send a REFUSE back to the User Agent.
If no issue is detected, the test instances are sent to the correct Classifier Agents. These agents predict those instances, and they send the predictions to the Final Classifier. Once the Final

Classifier has combined the received predictions using the F1-score, it will send the INFORM message back to the User Agent with the final predictions to be displayed.

As it can be noticed, we have not mentioned the possibility of sending a FAILURE because the task could not be completed. This is because we are assuming a more or less controlled and closed environment, where we own all the agents, so we do not expect them to fail to produce the task, once they have agreed to do it. The main reason for not expecting them to fail is the simplicity of the tasks they are doing, or in the case of the Classifier Agents, because we are using the Weka algorithm.

The other type of communication we have used is the mentioned Inform-ACK messages. That is, there are some cases where we need to send some piece of information from one agent to another. For example, when the Data Manager Agent needs to send the number of created classifiers to the Final Classifier Agent, or when the Classifier Agent sends the F1-score to the Final Classifier Agent. Given that we want the first one to be sure that the second has received the information, an INFORM message is sent back in the form of an ACK to end that communication. Doing it like this allows us to change the state of the first agent without worries.

In the flow charts below, we have summarized the communications involved in both the train and the testing phase:

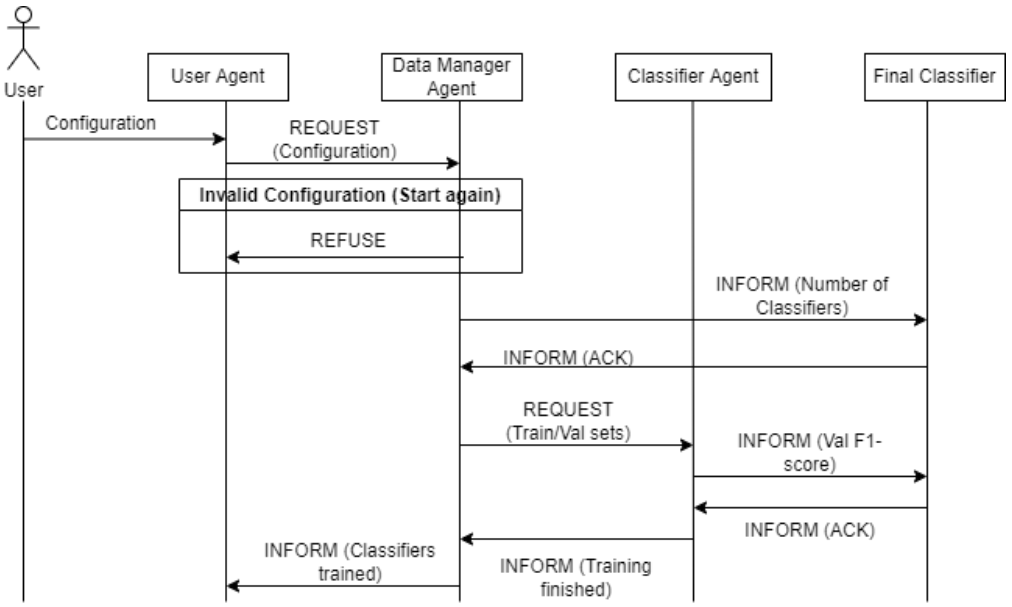


Fig. 3 Training phase communication flow chart

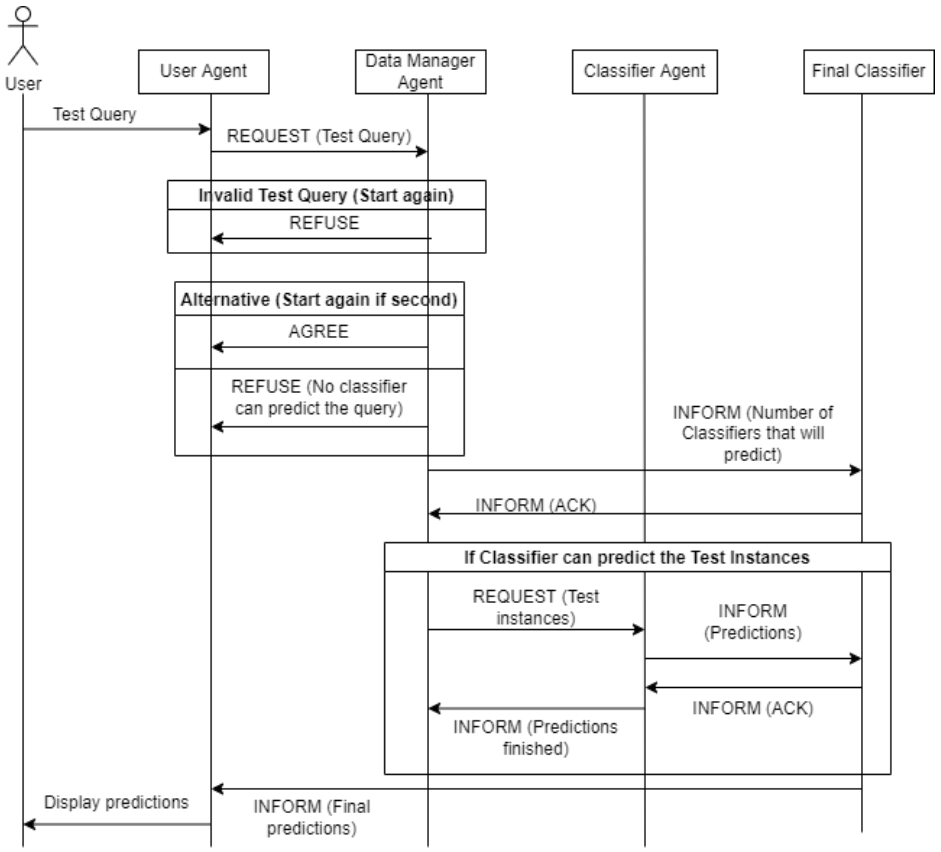


Fig. 3 Testing phase communication flow chart

3.7 Agent Cooperation and Coordination

In the MAS, we are considering a set of completely benevolent agents, given that they have been programmed by the same people, us, and we have not given them any goal that could conflict with the goals of another agent. For this reason, we have a set of agents that will always be ready to help others. In fact, the agents mostly remain blocked, waiting for another agent that needs to ask for help by sending some task that needs to be done.
The main goal of each agent is performing certain tasks, that once aggregated, they are able to fulfill the goal of the system. As we have explained before, the goal of the system is detecting if the information about a firm seems or not altered, meaning that the firm could be fraudulent. This goal cannot be achieved without the actions of the agents. During the training phase, it is true that the Final Classifier Agent is almost idle, and does not participate. Nevertheless, in the testing phase, everyone participates, performs tasks and contributes to serving the test queries to the human user.

Therefore, we are considering the situation where we have to solve a more or less complex problem, and we divide it into a set of tasks that can be performed by different agents. In our system, we mostly have one agent of each type, except for the Classifier Agents, that we have at least 10 of. However, given that we are assuming a controlled environment, we do not have to worry about one stopping working. If this situation happens, the system will simply stop its execution.

The task decomposition and allocation is mainly done static, given that we have decided which tasks each agent will perform. However, during the testing phase, it is true that we can consider it dynamic, given that only if the test instances contain the needed attributes for a particular classifier, then that Classifier Agent will predict them.

The synthesis of the results is mostly done by the Final Classifier Agent during the testing phase, given that it is the one that will produce the final predictions that will be later displayed to the user. During the training phase, we are not producing any kind of result, but we are simply training the models.

The type of cooperation mechanism our agents follow, does not exactly fall to any of the learned methodologies. Given that we have deliberative agents that do not negotiate, we can consider coalitions or Generalized Partial Global Planning (GPGP). The most similar case would be coalitions, but the agents do not obtain any profit, nor is there some kind of mechanism that generates the coalitions taking into consideration the capabilities of each agent and the requirements of the task. Given that we mostly have one agent of each type, they have been designed to be able to perform specific tasks. Also, there is always only one coalition at a specific time that can be formed.

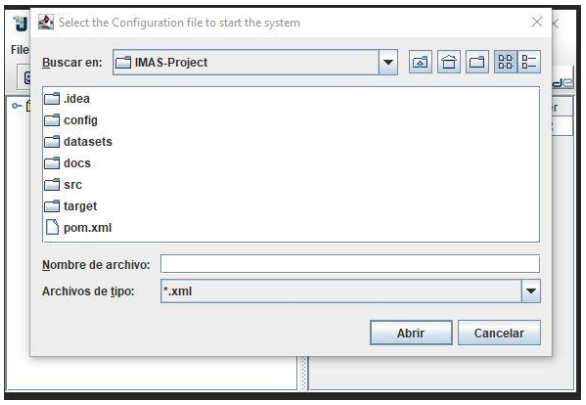
For this reason, we are not going to consider our system to follow one of the studied mechanisms, but rather they simply fully cooperate during the training phase and during the testing phase. During the training phase, the User Agent, Data Manager Agent, and the Classifier Agents cooperate to train the models. During each testing phase, all the Classifier Agents that can classify those test instances, and the other 3 types of agents cooperate to predict them.

The way our agents coordinate their actions is by directly sending FIPA messages between them. The User Agent is the one that initiates each of the 2 mentioned phases, given that is the one that receives the information needed to start them. The other agents are simply blocked, waiting for a message to come. When one of the agents receives a message, it performs its tasks, sends its results back, and is blocked again waiting for a new message to come.
However, we have to also take into account the finite state machine each agent has (i.e. those explained states in the implementation details), that will guide each agent to know at which phase they are. For example, when the system is ready to perform test queries, all the other agents will know the system is in that state, and all of them will be ready to wait for a message related to that phase. This is achieved by changing the state of each agent when this agent performs some task, and thanks to all the agents being designed by us. That is, given that we have developed all of them, we are assuming that all the agents are in line with each other, so we will never have agents in states corresponding to a different moment of the execution.

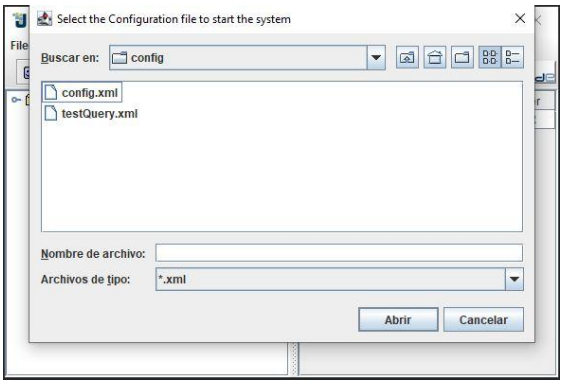
4. Results

In order to obtain the results, the user first needs to set up the system & run the A-DSS. The instructions for the same could be found in our project's README: https://github.com/QuimMarset/IMAS-Project/blob/main/README.md

After the system starts to run, the system asks the user to select the configuration file as shown in Img. 2. Img. 1 window will appear to help the user to navigate through the files:



Img. 1 Starting the System



Img. 2 Selecting the System Configuration

After loading the configuration correctly, the system trains the classifiers and through a log, it announces the results of the training model as shown in Img. 3. Here we can see that the first shown classifier obtained a 100% accuracy rate and that it belongs to the tenth created Classifier Agent. The second classifier had an accuracy of 88% of its training and it belongs to the fourth Classifier Agent.

Then the system asks the user for the Test queries and gives two different options as shown in Img. 4. Either select a pre-existing query or allow the system to select randomly. In case that the first option is selected, a window will pop out as seen in Fig. 5, where the user can select the testQuery.xml file.

```
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     0
                 1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     1
Weighted Avg.    1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000

=== Confusion Matrix ===

  a  b   <-- classified as
 52  0 |  a = 0
  0 23 |  b = 1

Accuracy: 1.0
ene 09, 2022 4:17:08 PM Agents.ClassifierAgent trainModel
INFORMACIÓN: Classifier classifierAgent_10 statistics:
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,980    0,333    0,862      0,980    0,917      0,721    0,837     0,867     0
                 0,667    0,020    0,941      0,667    0,780      0,721    0,837     0,790     1
Weighted Avg.    0,880    0,233    0,887      0,880    0,874      0,721    0,837     0,842

=== Confusion Matrix ===

  a  b   <-- classified as
 50  1 |  a = 0
  8 16 |  b = 1

Accuracy: 0.88
ene 09, 2022 4:17:08 PM Agents.ClassifierAgent trainModel
INFORMACIÓN: Classifier classifierAgent_4 statistics:
```

Img. 3 Training Model Results from two Classifier Agents

```
The classifiers have finished training. Test queries can be done now
Select querying mode:
1. Load an xml containing the query
2. Let the system randomly select 15 instances with 20 attributes each
Choice:
```

Img. 4 Choosing between Random Queries and XML file

After selecting the query, the system will start to predict the outcome of the instances. In Img. 5 we can see the results of the prediction versus the actual classification. We can see that all predictions were 100% accurate and that the system classification was correct.

```
Test Query Results:
Test instance 0: Predicted = Altered | Actual = Altered
Test instance 5: Predicted = Normal | Actual = Normal
Test instance 10: Predicted = Altered | Actual = Altered
Test instance 12: Predicted = Altered | Actual = Altered
Test instance 14: Predicted = Altered | Actual = Altered
Test instance 15: Predicted = Altered | Actual = Altered

Test Instances have been classified
Introduce again another test query
```

Img. 5 Results of the Predictions for the XML file

If the second option was selected, where the system selects random instances to be predicted, it will show the user which instances were selected and which attributes were selected to form the query, as shown in Img. 6.

```
Random generated query:
Instance indices (starting from 0): [0, 34, 2, 35, 6, 38, 42, 44, 12, 13, 16, 23, 24, 30, 31]
Attribute names :[Sector_score, LOCATION_ID, PARA_A, Score_A, Risk_A, PARA_B, Score_B, Risk_B,
Now wait for the results to come
```

Img. 6 Selecting Random Queries

The resulting predictions from this random query, Img. 7, resulted to be 100% accurate as well as the first one shown in Img. 5.

```
Test Query Results:
Test instance 0: Predicted = Altered | Actual = Altered
Test instance 34: Predicted = Normal | Actual = Normal
Test instance 2: Predicted = Altered | Actual = Altered
Test instance 35: Predicted = Normal | Actual = Normal
Test instance 6: Predicted = Altered | Actual = Altered
Test instance 38: Predicted = Normal | Actual = Normal
Test instance 42: Predicted = Altered | Actual = Altered
Test instance 44: Predicted = Normal | Actual = Normal
Test instance 12: Predicted = Altered | Actual = Altered
Test instance 13: Predicted = Altered | Actual = Altered
Test instance 16: Predicted = Altered | Actual = Altered
Test instance 23: Predicted = Normal | Actual = Normal
Test instance 24: Predicted = Normal | Actual = Normal
Test instance 30: Predicted = Altered | Actual = Altered
Test instance 31: Predicted = Altered | Actual = Altered

Test Instances have been classified
Introduce again another test query
```

Img. 7 Results of the Prediction for the Random Queries

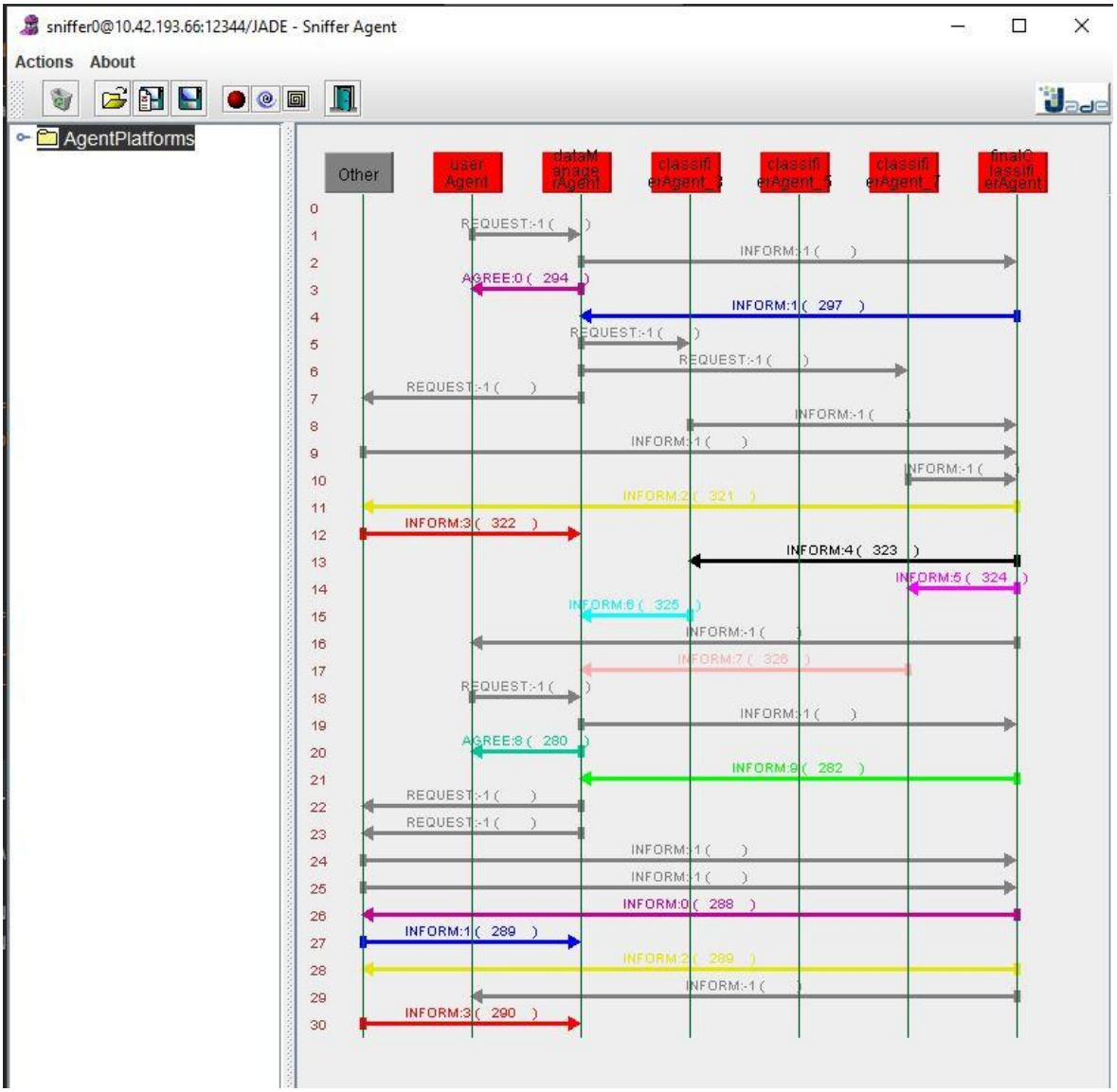
After performing several runs of the complete system, we can conclude that we are obtaining a validation accuracy between 80 and 100% per classifier. These are not bad results considering the random splitting of the instances and the attributes. In the shown test query, we can observe how we are able to correctly predict all the instances. This will not always be the case. For example, in the image below, we observe how we are performing some errors.
However, even though we are making some errors, we are still achieving very good results. This is because the ensemble of the different classifiers, although each one trained with few random attributes, is able to achieve better results than each individual classifier.

```
Test Query Results:
Test instance 32: Predicted = Normal | Actual = Altered
Test instance 40: Predicted = Normal | Actual = Normal
Test instance 41: Predicted = Normal | Actual = Normal
Test instance 42: Predicted = Altered | Actual = Altered
Test instance 43: Predicted = Normal | Actual = Normal
Test instance 12: Predicted = Altered | Actual = Altered
Test instance 45: Predicted = Normal | Actual = Normal
Test instance 14: Predicted = Altered | Actual = Altered
Test instance 46: Predicted = Normal | Actual = Normal
Test instance 48: Predicted = Normal | Actual = Normal
Test instance 19: Predicted = Altered | Actual = Altered
Test instance 20: Predicted = Altered | Actual = Altered
Test instance 23: Predicted = Normal | Actual = Normal
Test instance 28: Predicted = Normal | Actual = Altered
Test instance 29: Predicted = Normal | Actual = Normal
```

Img. 8 Another random test query

To see if the agents were communicating correctly when predicting the testing queries, a sniffer procedure was started from the Agent Platform. Img. 9 shows the Agents selected to be monitored, which were the User Agent, Data Manager Agent, three Classifier Agents and the Final Classifier Agent. Img. 10 shows the communication established between the agents after two runs of classification prediction.

Img. 9 Starting a Sniffer to see communication



Img. 10 Sniffer Result after two runs of Testing Queries

5. Conclusions

In this project, we have been able to develop an end to end Multi-Agent System that is able to perform a rather simple task in an enclosed environment. It is true that the task we are trying to achieve was quite simple, and sometimes the agent communication has complicated things a bit. However, we have been able to put into practice some concepts learned during the subject about the way the agents communicate and cooperate, and the necessity of cooperation.

Another conclusion we can extract is that even though JADE was created more than 20 years ago, it is still being developed because it grants the tools to develop MAS that are FIPA compliant. Moreover, it permits using the FIPA agent communication language. Therefore, even though it may not be the most cutting-edge framework, it is still very useful to develop this kind of simple yet powerful Multi-Agent System.

After having developed the system, and taking into account our design, we could consider the Final Classifier Agent to be more a problem than a help. In the end, this agent is simply computing a weighted average of the test predictions, but to do so, we need to perform some communications with the Data Manager Agent to receive some information like the number of existing classifiers. Therefore, we could have done this simple task in the Data Manager Agent itself, because it will have all the needed information and no extra message passing would be needed. The agent can be considered to be the one managing all the dataset instances.

6. References

- Ministry of Electronics and Information Technology (MEITY) (2018). https://archive.ics.uci.edu/ml/datasets/Audit+Data
- Frank, Eibe (NA). Weka J48 Documentation. WEKA Sourceforge. https://weka.sourceforge.io/doc.dev/weka/classifiers/trees/J48.html
- Martín, Mario (2018). Data Mining 3: Decision Trees. https://www.cs.upc.edu/~mmartin/DM4%20-%20Decision%20trees.pdf
- TutorialPoint (NA). What is WEKA?. https://www.tutorialspoint.com/weka/what_is_weka.htm
- Wooldridge, MIchael (2002, April). Intelligent Agents: The Key Concepts. Springer, Berlin, Heidelberg. https://campusvirtual.urv.cat/pluginfile.php/3953480/mod_resource/content/1/wooldridge95intelligent_weiss.pdf
- Batet et al (2012). Turist@: Agent-based personalised recommendation of tourist activities. https://campusvirtual.urv.cat/pluginfile.php/3953499/mod_resource/content/2/Turist%40%20ESWA%202012.pdf
- JADE FIPA, https://jade.tilab.com/papers/JADETutorialIEEE/JADETutorial_FIPA.pdf
- JADE API Documentation, https://jade.tilab.com/doc/api/index.html
- Weka Documentation, https://waikato.github.io/weka-wiki/documentation/
- Weka API Documentation, https://weka.sourceforge.io/doc.stable/

X.1 E-Portfolio

The e-portfolio of the project could be found in our GitHub repository at https://github.com/QuimMarset/IMAS-Project/blob/main/docs/ePortfolio.docx. Listed in the portfolio are the division of activities, task assignments, and the schedule & deadlines to be followed. On a high level, the division of work from a project perspective was as below:

1. User Agent - Shivani & Joaquim
2. Data Manager Agent - Joaquim
3. Classifier Agent - Rubén & Anne
4. Final Classifier Agent - Genevieve & Anne
5. System config changes - Shivani
6. Utils + refactoring - Joaquim, Shivani & Anne
7. Final report writing - Rubén, Joaquim, Shivani & Anne
8. Oral exposition slides - Everyone

This information can not be traced back via GitHub commits. It is crucial to take into consideration that at times when the task division was amongst subgroups, the code might be worked on by the subgroup as a whole but committed by **one member (not always being the case)**.

Owing to the differing course loads and timetables amongst the team members, there was a need to revise the initially proposed e-portfolio (submitted as part of the first delivery), but the team managed to largely stick to the schedules.

X.2 Meeting minutes

A record of all the meetings (scheduled or ad hoc), the attendees and the content discussed was kept on Moodle. Summarizing the same in the report for ease of access:

Date	**Attendees**	**Agenda**
8th October, 2021	(discussed online, no meeting)	• Google Drive workspace created and access shared • GitHub project created and access shared
11th October, 2021	Anne, Joaquim, Shivani, Genevieve, Ruben	First introductory meeting.
13th October, 2021	Anne, Joaquim, Shivani, Genevieve, Ruben	In-lab meeting to clear doubts & system issues on JADE.
14th October, 2021	Anne, Joaquim, Genevieve	Design discussion #1
15th October, 2021	Ruben, Joaquim, Anne	Design discussion #2
17th October, 2021	Anne, Joaquim, Shivani, Genevieve, Ruben	• Agent architecture of Data Divider Agent - debating between reactive and hybrid. • Reached a consensus that there is

		no "intelligent" or learning task associated with it, so it would be reactive. ● Implementation-level details for the Final Classifier Agent - Shivani to ask if it could be a Neural Network. ● Divided the parts for the presentation.
17th October, 2021	Anne, Joaquim	● Discuss how the Final Classifier Agent (i.e. the coordination mechanism) should work. ● Discuss the Reactivity property of some agents.
9th November, 2021	Anne, Joaquim, Shivani, Genevieve, Ruben	● Putting together the initial project layout + backbone code to read and process the dataset. ● Discussing deadlines & task assignments for the first draft of e-Portfolio.
7th December, 2021	Anne, Joaquim, Shivani, Ruben	● Discuss the state of the project at the current date. ● Reschedule and set new deadlines. ● Decide how the Final Classifier Agent will combine the predictions. ● Decide to start writing a draft of the final document.