Team 2
Xiao Fei, Lihong Ji, Shanshan Zhang, Yanbin Zhu, Demokan Coban

Final Report

1. Final System Design

Our Multi-Agent-Decision-System (MADS) is capable of loading dataset, training and predicting the test data. By configuring the configure.xml file, users can set the number of classifier agents, number of train, validation and test data, random seed, etc. The system architecture is shown in Figurer 1.

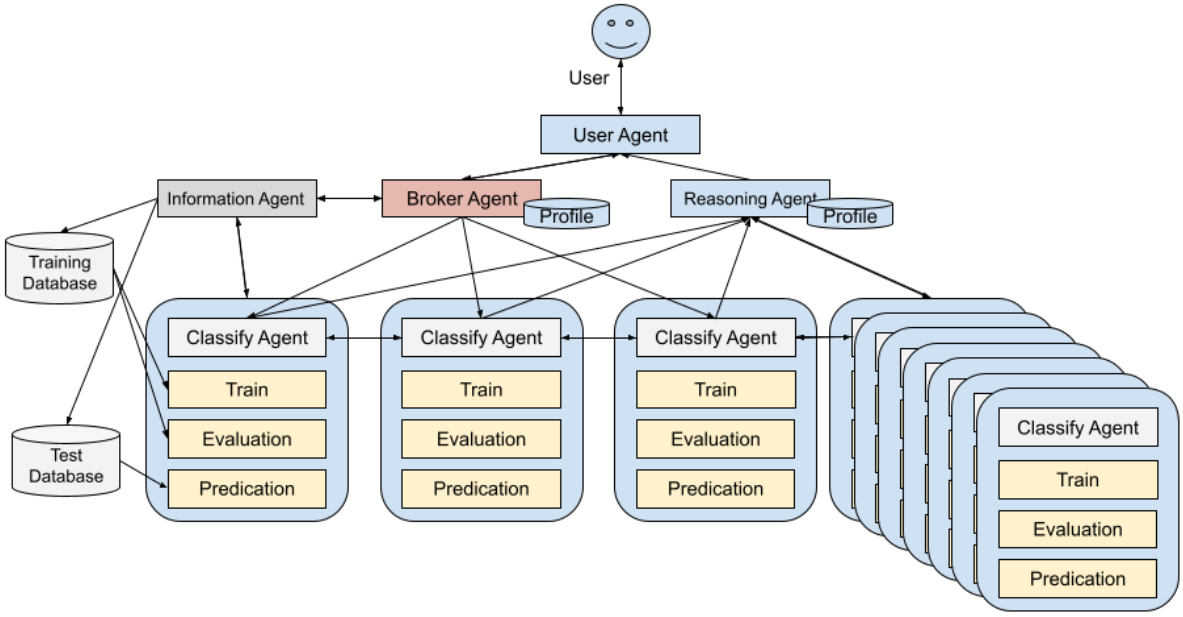

Figure 1. System architecture

The whole system can be divided into three phases. The first phase is preparing and training phases. When a user launches the system, the first phase starts and all the agents will be created and initialized. When all the agents get ready, it will automatically or manually enter the second phase, which is the training phase. In this phase, the information agent will load the data for training and validation. After the training phase, the evaluation of each classifier is stored into the reasoning agent. For the last phase, a set of test data will be predicted by qualified classifier agents and then using a voting mechanism to determine the final results.

2. Implementation

2.1 My Agent Base class

In our implementation, we created a base class called MyAgent which handles basic and common functions across all the inherited agents. The common functions are Setup, SearchAgent, SendMsgBehaviour, AutoReplyBehaviour, and ParseXML.

The base **Setup** function handles registration to the DFService. Therefore all the agents inherited to the base class will be registered to the DFService.

SearchAgent function is searching for specified type of agents from Direct Facilitator. For example, the Information Agent wants to send messages to all the classifier agents, so it will search “ClassifierAgent” at DFService. As a result, all the classifier agents are able to receive the message from the Information Agent.

SendMsgBehaviour is a helper function for an agent to send message or data to (a) receiver(s).

AutoReplyBehaviour is a helper function to automatically respond to an incoming message. Initially, we use this function to track communication between agents. Later, we muted the reply message function to keep the system simple.

ParseXML is a common function to parsing a xml file. All the inherited agents are able to read the configuration file and the option value. However, we only parsed the configuration in Broker Agent, Information Agent and Reasoning Agent.

2.2 User Agent

The User Agent handles user interaction by receiving messages from a user. When it receives a “Start” Request message, it then sends a “GetReady” request to the Broker agent. Similarly, when it receives a “Test” Request message, it forwards the “Test” request to the Broker agent.

As mentioned above in section 1, the user agent can also automatically start the system and if the autostart option is set to true. When the User Agent receives an “ImReady” message, the system is ready to train the classifier agents, so the User Agent will send a “Train” request to the Broker agent. The following flow chart demonstrates the User Agent.

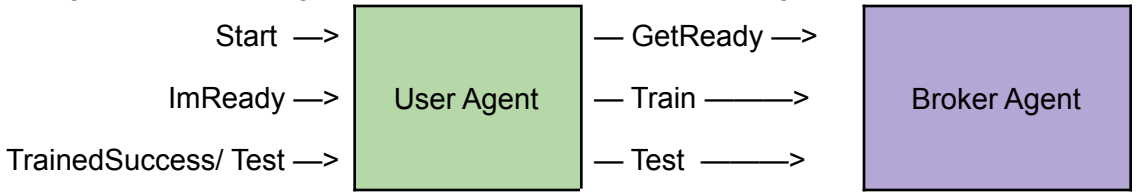

Figure 2. User Agent actions

2.3 Broker Agent

The broker agent handles the communication between the user agent and the information agent. When the system starts, it will request the information agent to prepare for loading the data set. If the information agent failed to prepare the data, it will repeatedly request the information to load the data until it's done. When the user agent requests to train or test the classifier, the broker agent will forward the request to the information agent.

The broker agent is also responsible for creating classifier agents when the system starts. The following graph shows the actions of the broker agents.

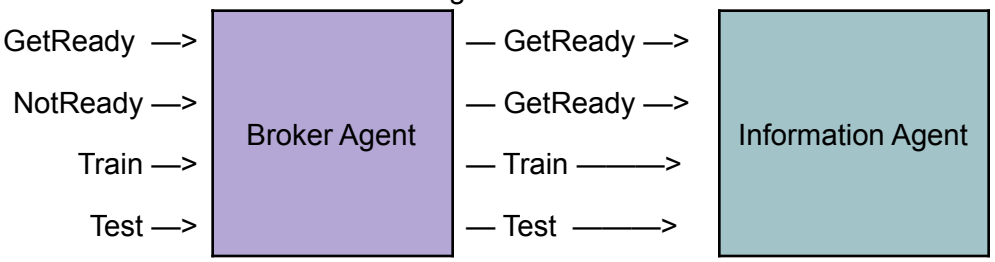

Figure 3. Broker Agent actions

2.4 Information Agent

The information agent handles the communication between the broker agent and the classifier agents. Its main responsibility is to prepare the data for training, validation, and testing. When it receives "**GetReady**" requests from the broker agent, it will load the data from the configured path. If loaded successfully, it will send a "ImReady" Request to the classifier agents. If not, it will reply to the broker agent with a "**NotReady**" request.

The information agent handles the "**Train**" request from the broker agent. It will select 300(or other number specified in configure.xml) training data and forward it to all the classifier agents. The "**Test**" request from the broker agent is handled by the information agent with CFP request to all the classifier agents. If a classifier agent is able to process the test data (all its 6 attributes are in the test data), it will reply to the information agent with a proposal. The information agent receives all the proposals and then sends the test data to each of the proposed data. The following graph shows the actions of the information agent.

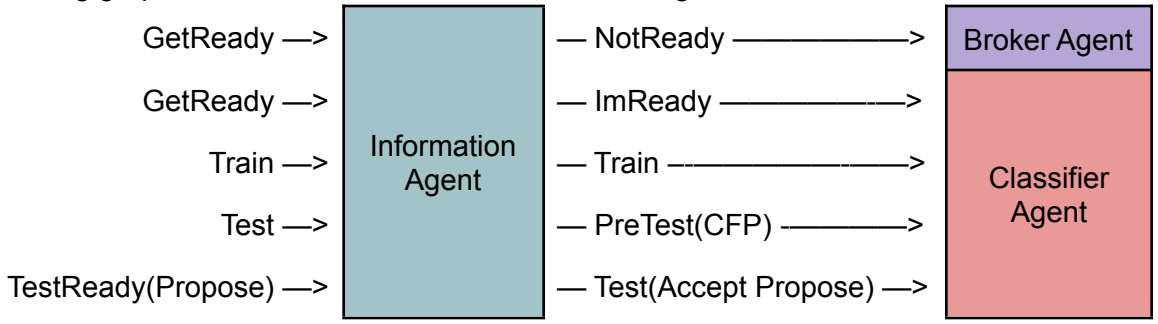

Figure 4. Information Agent actions

2.5 Classifier Agent

The classifier agent handles the communications between information agent and reasoning agent. When the system starts, the broker agent will create the classifier agent and randomly select 6(or other numbers specified in configure) attributes. When the agent receives an "**ImReady**" request from the information agent and the agent is also ready for training, the agent will forward "**ImReady**" request to the reasoning agent.

When the classifier agent receives a "**Train**" request from the information agent, it will train its classifier tree with the received training data. The training data contains 225 train data and 75 validation data. When the agent successfully trained and validated, the classifier agent will send a "**TrainedSuccess**" request and an evaluation results to the reasoning agent. The evaluation results contain the TruePositiveRate, Precision, Recall, FMeasure, etc.

When the classifier agent receives a "**PreTest**" CFP message, it will compare the test data attributes and its own preferred attributes. We store the attributes in a 32 bit integer so that the left 24 bit represents whether an attribute is used. To test if the classifier agent's attributes are satisfied with the test data's attributes, we simply do the following calculations.

```
public boolean Fit(int testAttrs){
   return ((testAttrs & m_attrInd) == m_attrInd);
}
```

If the classifier agent's attributes fit the test data's attributes, the agent will reply to the information agents with "**TestReady**" proposal.

After the information agent processes all the proposals, the classifier agent will receive the AcceptPropose message with the test data. The classifier agent then removes all unrelated attributes and evaluates the test data. After the test, the "**TestSuccess**" message with evaluation result is forwarded to the reasoning agent.

The following figure 5 shows the actions of the classifier agent.

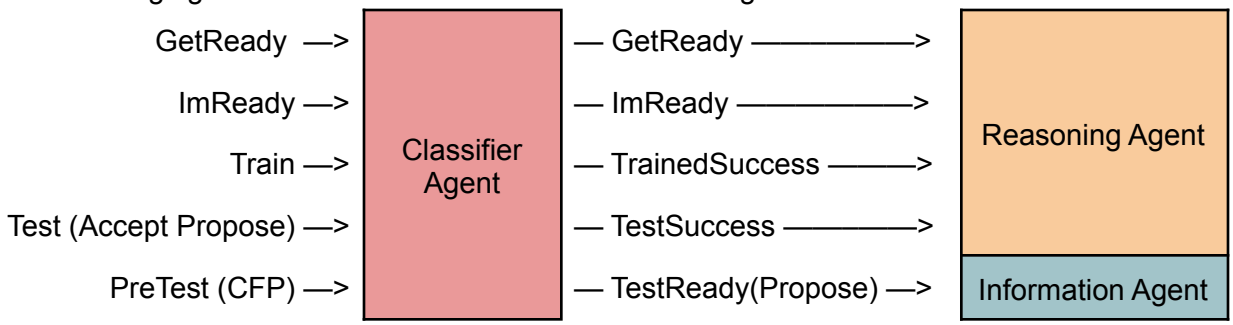

Figure 5. Classifier Agent actions

2.6 Reasoning Agent

The reasoning agent handles the communication between classifier agents and the user agent. When the reasoning agent receives an "**ImReady**" request from all the classifier agents, it will then forward the "**ImReady**" request to the user agent. When the reasoning agent receives a "**TrainedSuccess**" request from all the classifier agents, it will store all the evaluation metrics in a hashmap with corresponding classifier agents as the key. After the test, the "**TestSuccess**" request message is sent by classifier agents. The reasoning agent will process the request by storing all the test results in a hashmap with corresponding classifier agents as the key. The following figure shows the actions of the reasoning agent.

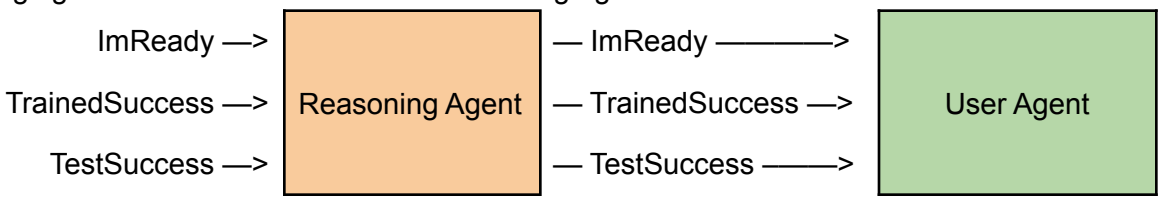

Figure 6. Reasoning Agent actions

2.7 Coordination Mechanism

After all the test results were received, we will use a voting mechanism to determine the final result. If the uniform (plurality) voting is used, each classifier agent will have one vote to its result, the final result is the highest number of votes. Since the result is a binary class, we can use the following formula to calculate the result where $R_i \in \{0, 1\}$; $W_i \in [0, 1]$; In the plurality voting, the $W_i = 1$; and the threshold is 0.5.

$$voting = \frac{\sum_{i=0} R_i W_i}{\sum_{i=0} W_i}$$

$$result = \begin{cases} 1 & \text{if, } voting \geqslant threshold \\ 0 & \text{if, } voting < threshold \end{cases}$$

If we take the W_i as the evaluation metrics (i.e. Precision, Recall, F-Measure) from the ith classifier agent's training evaluation, the final result is then based on the trained evaluation. We will further evaluate the results in section 3.

3. Result

For evaluation purposes, we will compare all the voting measurements in one test. We used 30 classifier agents for the test. As we can see in Table.1 there are 6 agents proposed for the test request. And the 15 test data was predicted with around 80% accuracy. All the different voting measurements output similar accuracy. And if we take a look closely, the first two agents with 100% accuracy in the training dataset, output very different results (difference marked in red). 8

of 15 data were classified differently by the classifier19 and classifier9. On the other hand, the classifier25 gets 100% accuracy for the test.

Agent	Result				TPR	FPR	Precision	FMeasure	Accuracy
classifier19	[1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0]				1	0	1	1	1
classifier9	[0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1]				1	0	1	1	1
classifier3	[1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1]				.960	.055	.960	.960	.960
classifier17	[0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0]				.987	.008	.987	.987	.987
classifier27	[1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0]				.920	.063	.932	.920	.920
classifier25	[1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1]				.973	.045	.974	.973	.973
GT	**[1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1]**								
Uniform	[1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1]							$\beta = 1$	
	8	4	3	0	1	3/7	8/11	32/11	.80
TPR	[1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1]								
	6	6	1	2	.75	1/7	6/7	24/9	.80
FPR	[1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1]								
	7	5	2	1	.875	2/7	7/9	28/10	.80
Precision	[1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1]								
	6	5	2	2	.75	2/7	.75	24/10	11/15
Recall	[1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1]								
	6	6	1	2	.75	1/7	6/7	24/9	.80
FMeasure	[1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1]								
	6	5	2	2	.75	2/7	.75	24/10	11/15
Accuracy	[1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1]								
	6	6	1	2	.75	1/7	6/7	24/9	.80
Metrics	**TP**	**TN**	**FP**	**FN**	**TPR**	**FPR**	**Precision**	**FMeasure**	**Accuracy**

Table.1 The result of all voting measurement

4. Conclusion

Revising the initial system design, we have implemented most parts of the system. And we have run our *MADS* 100 times and the average accuracy is around 82%. There is still a lot of room for improvement, such as careful selection of attributes. For our current system, the 6 attributes were randomly selected, which may result in imbalanced attributes. We can also assign weights for each attribute, the most frequent attributes of top K classifiers will be assigned a higher weight.

Another problem we may fix in future is to handle 0 or few classifier proposals. The fewer the classifier agent proposes the test, the worse the result may get. We can deal with this problem with dynamic strategy. If there is 0 proposal from classifier agents, we can send a request to the broker agent for adding new classifier agents. If the number of proposals is less than a minimum requirement, we can either add new classifier agents or use simple plurality voting for the final result.

For the communication and coordination between agents, we believe that our system should utilize more of the ContractNet protocols. The message between agents can be more structured and uniformed.

Lastly, the functions we plan to implement but not yet implemented is the saving trained classifier mode and the train and the test results.