A thesis submitted in partial satisfaction of the
requirements for the degree of
Master of Computer Science and Engineering
in the Graduate School of the
University of Aizu

Towards Building a Machine Learning-based AI System for the Game of Soccer

by
m5171132
Seishu Itabashi
B.S. (the University of Aizu) 2013

Reviewed by
Professor Maxim Mozgovoy, Main Referee
Professor Igor Lubashevsky,
Professor Vitaly Klyuev
March 2015

The thesis titled

"Towards Building a Machine Learning-based AI System for the Game of Soccer"

by

m5171132
Seishu Itabashi

is approved

_____

Main referee Date

_____

Date

_____

Date

University of Aizu

Spring 2015

Index

List of Figures, List of Tables

List of Figures

List of Tables

Acknowledgments

I would like to thank Professor Mozgovoy who helped my writing the master thesis. He gave me a lot of idea to think up my scheme and got data set of real soccer used in this thesis. If he did not help me, I could not write this work.

Abstract

The development of procedure to build the soccer AI on the basis of observations of real soccer games and machine learning is and important topic for people who study team sport games, such as soccer. A skillful soccer AI system can further reveal successful game tactics and help develop better soccer video games. However, it is difficult to implement such AI with hand-coded rules. The purpose of this study is to contribute to the process of building data-driven AI for soccer with the help of real soccer recordings and machine learning techniques.

In the present work we will consider an important subproblem of *pass analysis*. We will analyze real soccer recordings to learn which game situations are suitable for passes, and which passes have sufficient probability to reach their receivers. The scheme relies on the learning model built with LIBSVM, a popular library of Support Vector Machines procedure. Furthermore, we will compare the proposed method of pass success/failure prediction with the method implemented in the open source soccer agent team program called *agent2d*. Finally, we will evaluate both original and modified agents within the framework of a well-known 2D soccer engine *Robocup 2D soccer simulator*. As a result, while the pass success rate of the original agent2d was not surpassed, I made a number of important observations and have suggestions for further improvements of the proposed AI algorithm.

At the introductory part, the background and motivation of this work are explained. Next, the fundamental technologies and the scheme to build AI are explained. Finally, the environment and result of experiment by using AI that is built by proposed scheme, consideration for result, and conclusion are provided.

1 Introduction

Experimenting with human behavior via human-computer interaction is a challenging and interesting topic with many open problems. A challenge to give the artificial intelligence the ability for judgment by means of a variety of methods was addressed in numerous projects, but nothing comes to get equal with high-level circumstantial judgment ability of humans until now[1]. The development of high-quality artificial intelligence supports the possibility of high-quality simulation and analysis, and can contribute to the development of nearly every research field[2]. However, the human circumstantial judgment is based on analysis and interpretation of large amounts of data[3], so it is very difficult to implement the ability for equal-level circumstantial judgment of AI with hand-coded rules[4].

In this work I analyze the process of building an AI system by learning from real soccer recording as one of the way to solve the problem. Soccer is the sport where the situation on the field changes rapidly, and the cooperation of many people is required. In addition, it is very popular worldwide, and is commonly used for general studies of team tactics with the help of virtual soccer simulators. The progress of the artificial intelligence development techniques in soccer helps researchers of sport tactics, soccer coaches, and people who develop soccer video games. Furthermore, it has the possibility to be applied in many other fields where the artificial intelligence is used.

For our experiments, I use Roboup 2D soccer simulator. Robocup is a domain where AI research and robotics do meet[5]. There are various leagues in Robocup and numerous research and development activities are conducted in each league. In the RoboCup soccer, it is regarded as important to win soccer games in competition. Furthermore, it is required that the ball is controlled securely. Among Robocup leagues, Robocup 2D soccer simulator represents the simplest environment. Each agent and ball is shown just as a circle on a two-dimensional field. An actual screenshot of simulator is shown in Figure 1. In this environment, the developers can concentrate on pure AI development. Due to its popularity, Robocup 2D soccer simulator is the most mainstream simulator to research tactics of the soccer and the circumstantial judgment of the AI agent.

As environment for research, we used Ruby 1.9.3 for programming the scripts that analyze the recording of real soccer game and construct learning models from analyzed data set. Furthermore, we use JavaScript and HTML5 for the visualization of the recordings of real soccer game recordings. In addition, I adopted the open source AI agent team program agent2d[6] to use in the simulator, and to implement the proposed scheme. The agent2d system can work as a sample team in Robocup 2D soccer simulator alone and has high extensibility. When agents in the program agent2d do the action in simulation, they generate numbers of actions that they do in the current state, and evaluate these generated actions based on information that agents has. The information includes seeing, hearing, and their own condition. Each agent independently gathers and processes incoming information. My idea was to

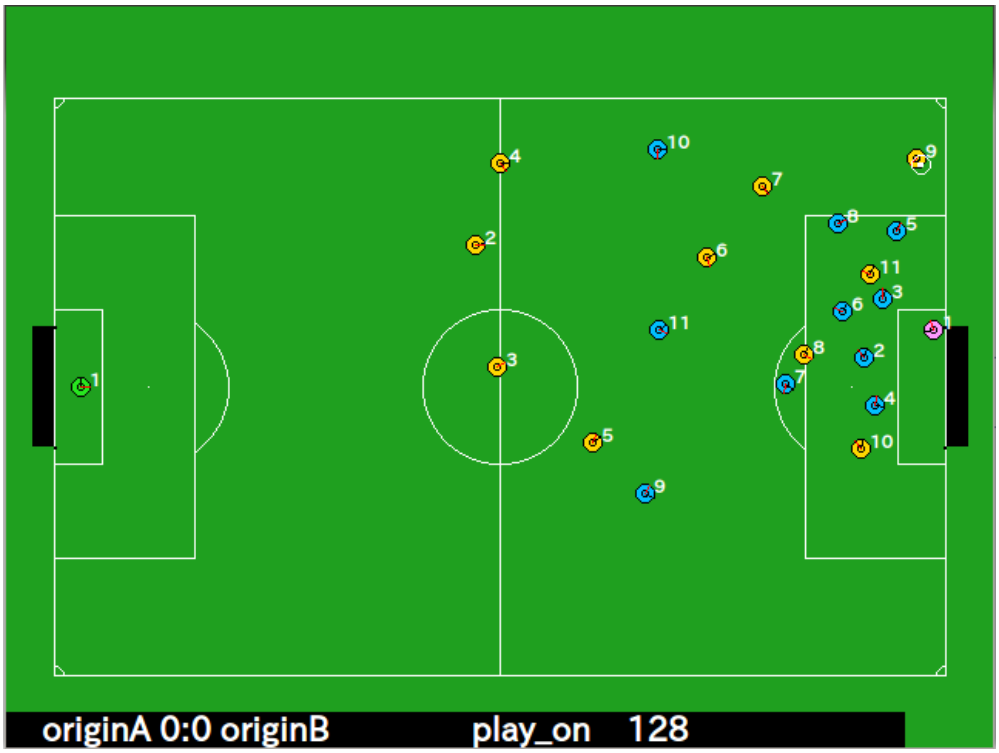

Figure 1. Screenshot of Robocup 2D soccer simulator

improve the evaluation method of the agent2d. Similarly, I used Support Vector Machine as an algorithm of machine learning and decision making.

In the next section we will discuss the fundamental technologies used. Next, we will consider the proposed scheme in detail. Finally, the results obtained with the proposed scheme and the ideas for future improvements, and the discussions are provided.

2 Fundamental Technologies

2.1 Support Vector Machine

Support Vector Machine is one of the algorithms for classification and discrimination. The original algorithm of Support Vector Machine was invented by Vladimir N. Vapnik and the current standard incarnation that is called *soft margin* was proposed by Vapnik and Corinna Cortes in 1993, and published in 1995[7]. Support Vector Machine is one of the best method to solve the classification and discrimination problem. Recently, it is expected to be improved by applying the method of deep learning[8]. Support Vector Machine possesses following three features[9]:

1. Discrimination surface is constructed by margin maximization policy. For this reason, we can expect high generalization ability.

In the linear discrimination problem on two-dimensional space like Figure 2, one of the way to solve problem is making the border to classify two patterns that belong to different class. In this thesis, I call the border that is decided to classify patterns *surface for discrimination*. In this example, surface for discrimination is just one line because of this problem is on two-dimensional space. Large numbers of lines that can classify two patterns completely can be drawn like Figure 3.

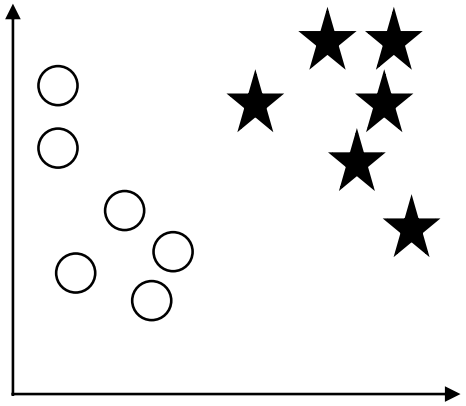

Figure 2. Space that performed by two vectors

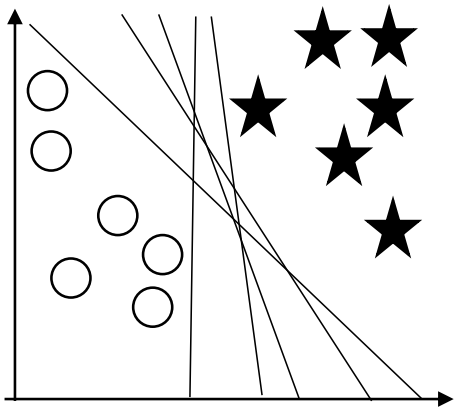

Figure 3. Large numbers of lines that can classify two patterns completely can be drawn

Generally, pattern recognition algorithm needs to identify unknown pattern not only known pattern. Considering from this viewpoint, the best surface for discrimination goes along just center of two already known patterns. This means surface for discrimination goes along the point as distance between that and nearest already known pattern is maximum like Figure 4. The distance is called *margin*. Statistically, when this way to decide surface for discrimination is adopted, probability of the pattern identified correctly becomes highest. It is expected that

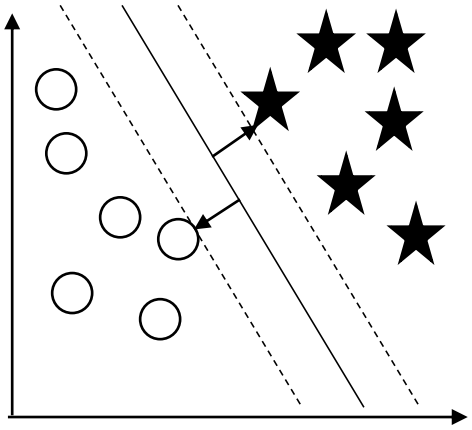

Figure 4. Surface for discrimination on maximum margin

Support Vector Machine has high generalization ability because Support Vector Machine adopts this way to decide the surface for discrimination.

2. Local optimal solution problem is never occurred.

Generally, learning machine decides the surface for discrimination by using already known pattern and updating the parameters of a machine. This process is called *training*. In other algorithms that use backpropagation, like neural networks, a local optimal solution problem can be obtained if training stops in the local area. In Support Vector Machine, the process of training relies on the square optimization problem using the method of Lagrange multipliers. In the square optimization problem, the local optimal solution must be same the solution in wider area. Because of this reason, the local optimal solution problem is never obtained in Support Vector Machine.

3. Support Vector Machine adopts *kernel function*.

Some pattern recognition problem cannot be identified linearly. For example, in the problem shown in Figure 5, it cannot be defined border line that can classify those two patterns correctly. However, it can be defined border line that can classify the patterns correctly by translation the space by the function that changes value x_2 to $(x_1 + 1)$ times of x_2 (see Figure 6).

The space that is translated by function like this case is called *feature space*. High

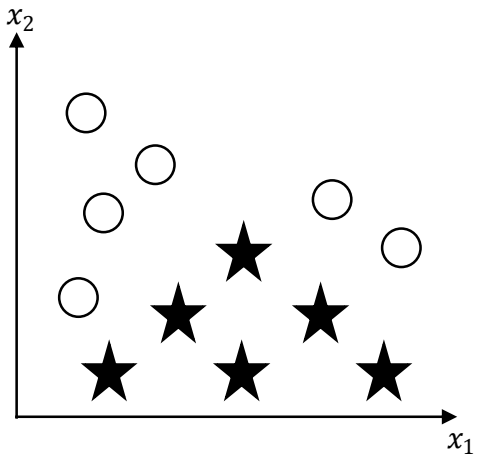

Figure 5. The pattern that cannot identify by line

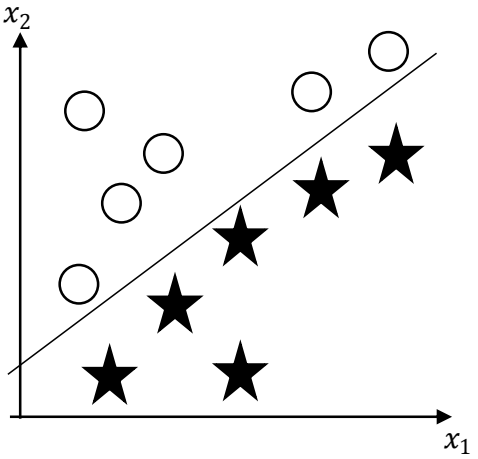

Figure 6. Translated patterns in space

generalization is expected by reflecting prior knowledge about the pattern to the function to translate, however, it is needed that computing about the positions of each patterns in space expressly to define the function to translate. For this reason, the function that is called *kernel function* is defined in substitution for the function to translate in Support Vector Machine. The kernel function is performed like $K(x, x')$ and gives the way to compute inner product of two patterns in feature space directly

without computing the positions of each patterns in space expressly. This feature often reduces computational complexity. Generally, the approach that reduces computational complexity and extends the way to analyze to high-dimensional feature space based on inner product by using kernel function is called *kernel trick*. The kernel function has various type. In this thesis, I adopt the Gaussian kernel. The Gaussian kernel is one of the most often used kernel and can be used general-purpose. The Gaussian kernel is expressed with the following formula[10];

$$K(x, x') = exp(-\gamma \|x - x'\|^2)$$

In this formula, γ is parameter that is adjusted by user.

Furthermore, Support Vector Machine is implemented in many libraries and can be used in various environments. In this work I relied on LIBSVM — a library of Support Vector Machine developed by Chih-Chung Chang and Chih-Jen Lin. This library has extensive API, and much used in academic field[11].

2.2 The AI Agent Team Program: agent2d

The open source AI team program *agent2d* is one of the team programs for Robocup 2D soccer simulator. The program agent2d is created in 2006 and kept developing by Hidehisa Akiyama[12]. The program agent2d can exhibit the minimum function to work as a sample AI agent team on a simulator natural. The agents gather information by themselves and store that for decision-making. The information includes seeing, hearing, and their own condition. The condition includes stamina, and speed. The seeing and hearing of agents are implemented as following process. Each agent gets message that includes the information that each agent saw (for example, position of other players and ball, line of field, goal) and heard (for example, what referee say, what other players say) in field from server every cycle. When agents need the information to make decision, agents extract the information they need from message. These information values are not necessary always correct because they are not always up-to-date. Because the agents gather information by using their own sight and ears, there is a limit of range for gathering information. The agents evaluate reliability of their data from elapsed time from getting information. They generate numbers of actions (for example, tackle or kick the ball) that they will do in the next cycle and evaluate each action based on information and reliability each cycle. When they kick the ball, they generate numbers of routes for kick the ball and evaluate each routes based on information and reliability. When they do each action, they compute the information required to evaluate. For example, they compute the

distance from position of receiving point to position of nearest opponent and position of receiver based on the information they have when they do the pass action. Numbers of conditional expressions are implemented in the file sample_field_evaluator.cpp of agent2d, and the evaluation score increases or decreases on the basis on conditional expressions and computed information. Finally, the agents try to perform the action that has the highest evaluation score.

2.3 TRACAB

TRACAB is one of the tracking system for real game of soccer invented by Chyron Hego in Sweden[13]. Development of TRACAB started early in 2003. This project aimed of creating a system capable of delivering real-time positioning of objects in sports. TRACAB enables real-time tracking of players, ball, and referee, and exporting the data set by using exclusive sixteen cameras and software (see Figure 7). The original technology was intended to support military projects of SAAB, one of

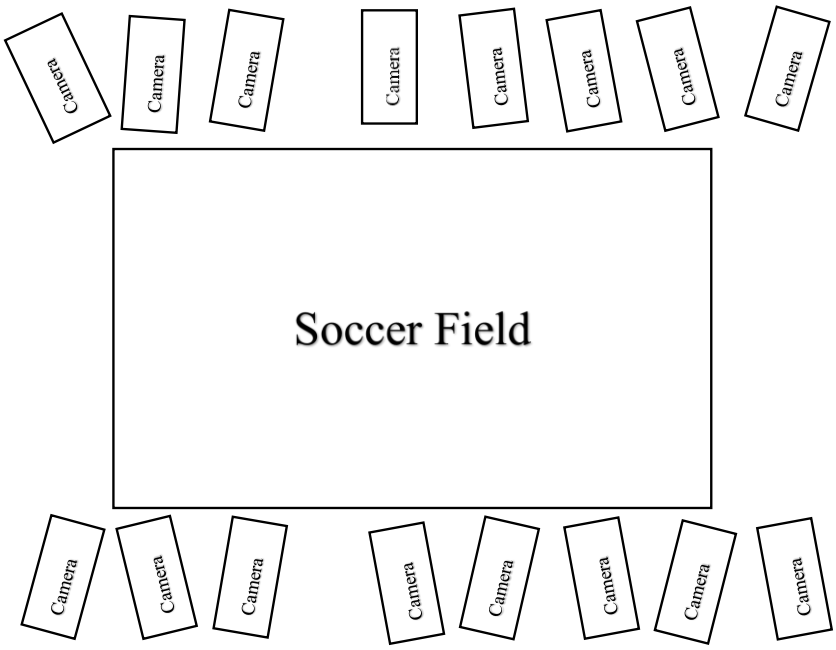

Figure 7. The field and cameras of TRACAB

the military companies in Sweden[14], and was used for data processing. Later this technology was adopted in FIFA World Cup 2010 as an official data tracking system. Furthermore, this technology was also adopted in Premier League, Bundesliga, and La Liga. The data set in this thesis is created with this system.

3 Proposed Scheme

3.1 Overview

This scheme enables to build AI agent that has the method to evaluate pass route based on machine learning and real soccer recording. This scheme adopts both LIBSVM and original agent2d method to evaluate pass route. Because I estimate that adopting both method give more performance to the scheme because LIBSVM learns just the distance between the pass sender and the pass receiver, the distance between the pass sender and the nearest opponent, and the distance between the pass receiver and the nearest opponent in this scheme, and original agent2d method can use other information (for example, distance from goal line, distance from touch line) to judge. It is expected that improvement of performance by evaluation by each method independently and adding up the scores. The flow is description below.

1. The timing of player who perform the pass action is extracted from real soccer recording, and written to the annotated text file (it describes the timing of player pass action) by the script learner.rb.
2. LIBSVM reads the text file that is written in step 1, learns the situations when the player passed/did not pass the ball, and outputs the learning model for detecting the pass timing.
3. The pass timing is detected by LIBSVM from real soccer recording.
4. The timing of the pass success/failure is judged from extracted pass timings, and the annotated text file (it describes whether pass is successful or not), is created by the script predict.rb.
5. LIBSVM reads the text file that is written in step 4, learns the situation when the pass succeeds/fails, outputs the learning model for prediction that the pass will succeed or fail judging using the current situation.
6. The team program agent2d_svm reads the learning model produced during the step 5.
7. When the agents in team program agent2d does the pass action in the simulator, they generate numbers of pass routes, evaluate each routes, and give score based on evaluation method of the team program agent2d.
8. The agents in the program agent2d call the LIBSVM and give the information that they gathered in current state and where they will pass the ball in next state to LIBSVM. LIBSVM predicts the success probability of the pass based on given information and learning model produced in the step 5. When the prediction of LIBSVM is success, the evaluation score is increased. When it is not, the score is

decreased.

9. The agents make decision based on the evaluation score.

3.2 Data Format of Real Soccer Recordings

The data set of real soccer recording used in this thesis consists of colon-separated main chunks. They can be Integer or String chunks. Strings can be just strings or they can be arrays of objects represented as strings. In the latter case the positions in the array are separated with semicolons. The individual properties of each object are separated with commas.

The data set of real soccer recordings consists of three main chunks. The first is an integer chunk containing the frame count of the current frame. The frame count is unique for each frame, and is generated by the tracking system. The second chunk is an array of 29 player and referee candidate target objects. The last chunk is an array of one object: the ball.

The chunk 1 is just integer data. The data type of chunk 2 is string-represented array of up to 29 objects. Each object contains the properties shown in Table 1. The data type of chunk 3 is a string-represented array of one object. This object contains the properties shown in Table 3[15].

Table 1. Description for the chunk 2 in real soccer data set

Properties	**Valid values**	**Remarks**
Target's assigned team	0 or 1 or 3, integer	1=Hometeam, 0=Awayteam, 3=Referee. Other values are used for internal purposes.
System target ID	1 to 29, integer	
Assigned jersey number	-1, or 1 to 99, integer	Jersey numbers are 1-99. Jersey -1 is "unassigned" except for team 3 (the referee).
Pitch position x	-5250 to 5250, float	
Pitch position y	-3400 to 3400, float	

Table 2. Description for the chunk 3 in real soccer data set

Properties	**Valid values**	**Remarks**
Pitch position x	-5250 to 5250, float	
Pitch position y	-3400 to 3400, float	
Pitch position z	0 to infinity, float	Note that the position of the ball center is displayed. Normal z position for a ball on ground is thereby 10 cm.
Ball owning team	"H" or "A", string	"H"=Hometeam, "A"=Awayteam
Ball status	"Alive" or "Dead", string	"Alive"=In play, "Dead"=Not in play

3.3 Model of Proposed System

The whole model of system that is proposed in this thesis is shown in Figure 8. The script rs_analyzer.rb is the script that executes all other scripts and LIBSVM in this system. The user of this system can call all function of system by using this script. The script learner.rb is the script to make text file for training to detect the timing that the player did the kick action. This script reads the real soccer recordings data set and outputs text file for the training phase. The script predict.rb is the script to make text file for training to classify the pass action. This script reads the real soccer recordings data set and identifies the timing that player did the pass action by using learning model that is made by the script learner.rb and LIBSVM. Furthermore, the script that classifies the kick action is success or not and outputs the text file for training. The script rs_svm.rb is the script to use LIBSVM from Ruby. This script is called by the script predict.rb and provides the way to use LIBSVM. The team program agent2d_svm is the team program for Robocup 2D simulator based on the program agent2d. The program sample_field_evaluator.cpp in this team program reads the learning model file made by the script predict.rb, and evaluates the pass route based on prediction of LIBSVM. This system enables AI agent for Robocop 2D simulator to be able to evaluate the pass route based on learning from the real soccer recording data set by this using these scripts. The script field_visualize.rb and visualize.html are the scripts that visualize the real soccer recordings data set. The details of the scripts in this system are described following section.

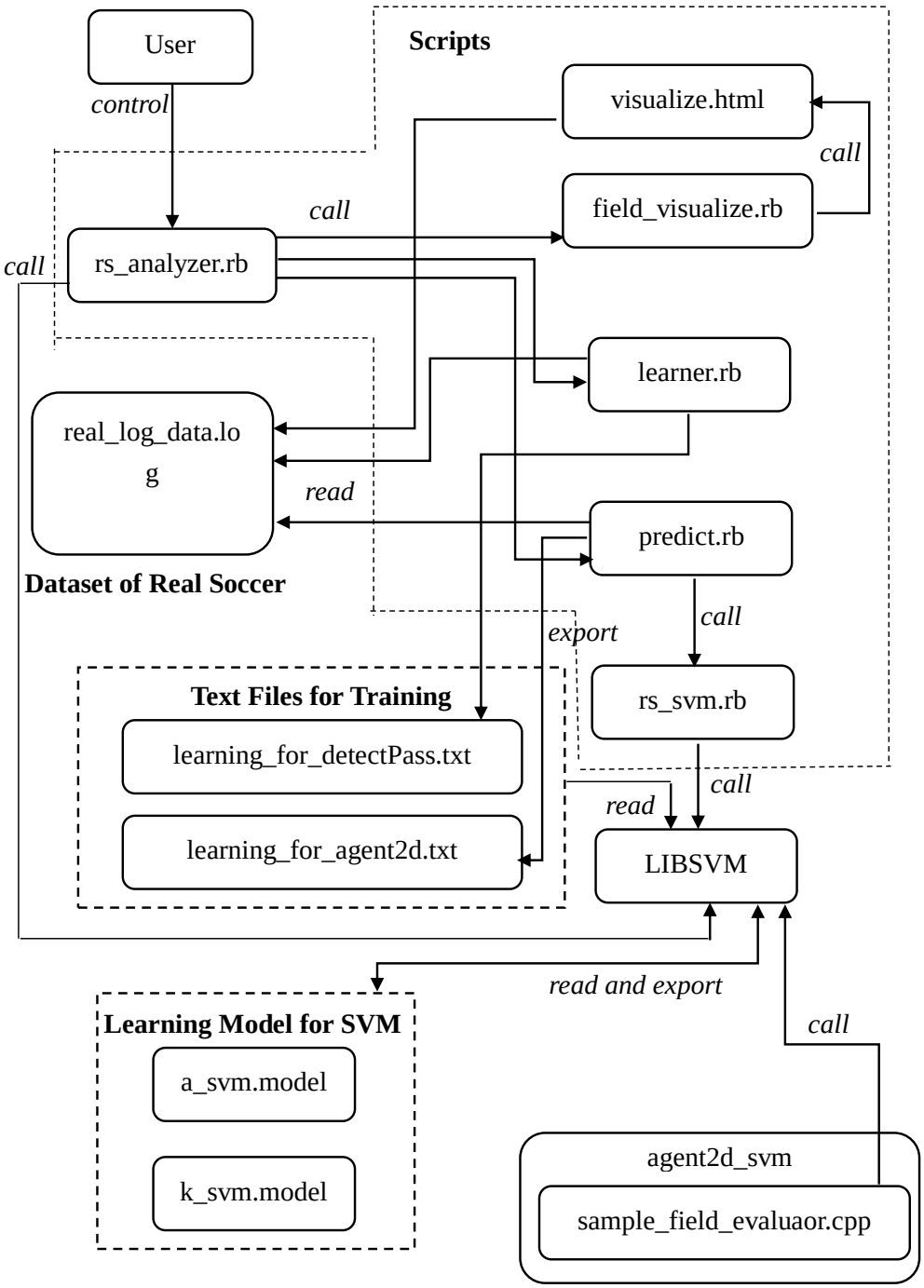

Figure 8. The whole model of proposed system

3.4 System Controller: rs_analyzer.rb

The script rs_analyzer.rb is the controller of the whole system. This script calls the functions of other scripts and LIBSVM upon user request.

3.5 Pass Timing Classifier: learner.rb

The script learner.rb can read the real soccer recording data set and output the text file for training. When this script is called, it starts reading real soccer recordings data set and writes to the text file for training in the file format of LIBSVM library. In this thesis, I defined the timing that the player did the pass action on the following conditions:

- Ball moved distance in one time frame exceeds 100 cm.
- The margin between ball moved distance in previous time frame and current time frame exceeds 100 cm.

If data in the time frame fulfills these two conditions, the script learner.rb classifies the time of the frame as “pass action”. Otherwise, the script classifies the time frame as “no pass action”.

The text file that is written by this script includes the parameters shown in Table 3. The sample string of this text file is description is shown below:

1 1:4.242640687119285 2:3030.694309890062 3:3026.451669202943

Table 3. Data format of text file for training with detecting kick timing

Name	Index number	Valid value	Remarks
Class	nothing	0 or 1, integer	1=pass frame 0=no pass frame
Ball move distance in the previous time frame	1	0 to infinity, float	Computed from positions.
Ball move distance in the current time frame	2	0 to infinity, float	Computed from positions.
The difference between the previous and the current ball move distance	3	0 to infinity, float	Computed from positions.

3.6 Success of Pass Classifier: predict.rb

The script predict.rb can read the real soccer recording data set and call the LIBSVM to predict the time when player did the kick action. Furthermore, this script outputs the text file for training to evaluate the pass route from predicted time frame. In this thesis, I defined the success of pass action with following conditions:

- When ball-possessing player changed after pass action, the ball-possessing player is not same as the player that kicked the ball.
- When ball-possessing player changed after ball was kicked by the original player, the ball-possessing team is the same as the team that kicked the ball.

If data in the time frame fulfills these two conditions, the script predict.rb classifies the pass action as “success”. If the-possessing player is changed after pass action and the ball-possessing team is changed, the script predict.rb classifies the pass action as “failure”.

The text file produced by this script includes the parameters shown in Table 4. Whole parameters are divided by 10 000 to adjust to Robocup 2D simulator. A sample string of this text file is shown below:

1 1:0.19254674757055754 2:0.06627367501504651 3:0.07380027100221244

Table 4. Data format of text file for training with pass route evaluation

Name	**Index number**	**Valid value**	**Remarks**
Class	nothing	0 or 1, integer	1=success 0=failure
The distance between the pass sender and the pass receiver	1	0 to infinity, float	Computed from positions.
The distance between the pass sender and the nearest opponent	2	0 to infinity, float	Computed from positions.
The distance between the pass receiver and the nearest opponent	3	0 to infinity, float	Computed from positions.

3.7 Improved Team Program: agent2d_svm

The program agent2d_svm is the team program for Robocup 2D simulator based on agent2d-3.1.1. The base program agent2d evaluates the probability of successful pass as explained in the previous section. The program agent2d_svm implements an additional evaluation method. The program agent2d_svm uses additional evaluation method based on prediction by Support Vector Machine. When agent2d_svm passes the ball, agent2d_svm generates the pass routes and evaluates each pass routes similarly to agent2d. In addition, agent2d_svm calls LIBSVM with the information of the distance between the pass receiver and the nearest opponent, the distance between the pass sender and the nearest opponent, and the distance between the pass sender and the pass receiver to LIBSVM. If the prediction of LIBSVM is “success”, the evaluation score is increased. If the prediction is “failure”, the evaluation score is decreased. Finally, the program agent2d_svm selects the pass route that has the highest evaluation score. When the evaluation score is lower than some other action

(for example, tackle), the agents do not initiate pass, and do another action that has the highest evaluation score.

3.8 The Visualizer of Real Soccer Recording: field_visualize.rb and visualize.html

These scripts implement the capability to visualize the real soccer recording data set. The script field_visualize.rb shows the size of the soccer field and opens the visualize.html. The html document visualize.html reads the real soccer recordings data set and draws the whole field of data set by using canvas method on HTML5 and Javascript (see Figure 9). This function is useful to check the data set and debugging.

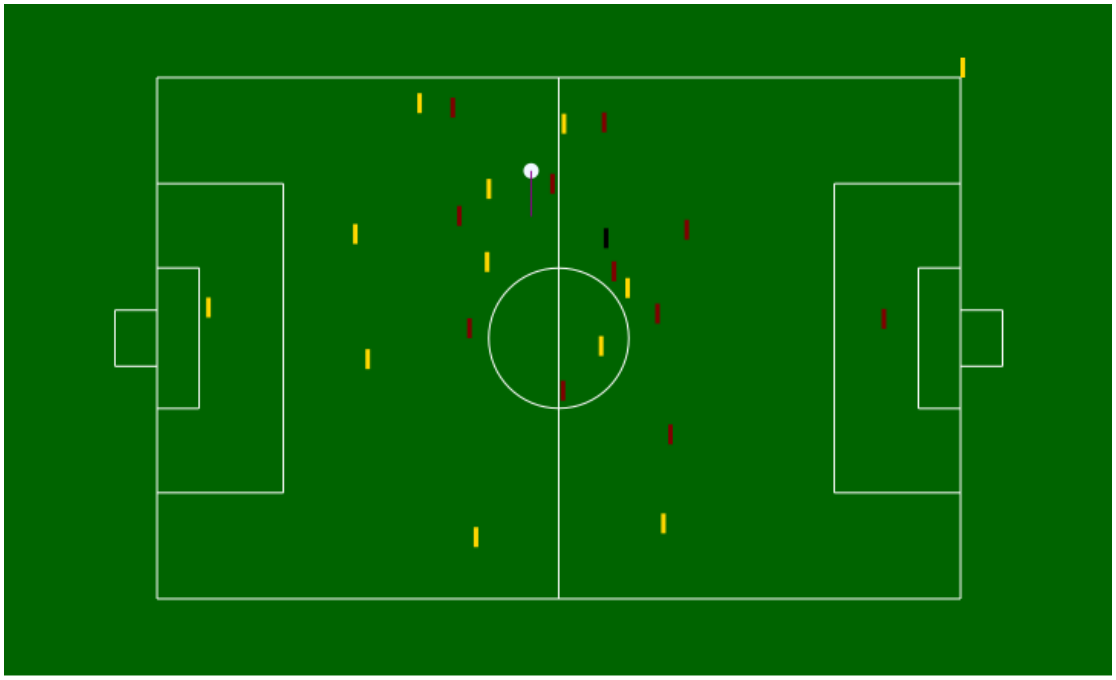

Figure 9. Visualization of real soccer recordings

4 Experiment and Result

4.1 Experiment

For testing the performance of AI that is improved by proposed scheme, I made the match that two AI agent teams compete each other on Robocup 2D soccer simulator. One of the team is the original agent2d, the other is agent2d_svm. The match is carried out ten times. Finally, I got the log data of these ten matches. I analyzed the data and computed the pass success rate. In this thesis, the pass success is defined that

when the agent which kicked the ball is not same as the agent that kicked the ball previous time, these two agents belong to same team. Furthermore, pass success rate is also defined as following formula:

$$Pass\ Success\ Rate = Pass\ Success\ Times/(Pass\ Success\ Times + Pass\ Failure\ Times)$$

4.2 Environment

In this work, we used the Robocup 2D simulator. Robocup 2D simulator is initially created by Itsuki Noda in 1993 and developed by a number of contributors. The last stable version of the simulator is dated 2005[16]. This means the simulator attained full maturity, so the developers of teams do not need to implement new rules and functions anymore. The simulator adopts the Client-Server model to implement the dispersed multi-agent system. The model of the simulator is shown in Figure 10: rcssserver is the server program of the simulator, and rcssmonitor is the program for visualization of the simulation. Several agents communicate to the server independently. This means that several agents are also controlled independently. Several agents can communicate each other via rcssserver. In this work we will use rcssserver-15.2.2 and rcssmonitor-15.1.0. The environment of the experiment Ubuntu (one of the Linux distributions).

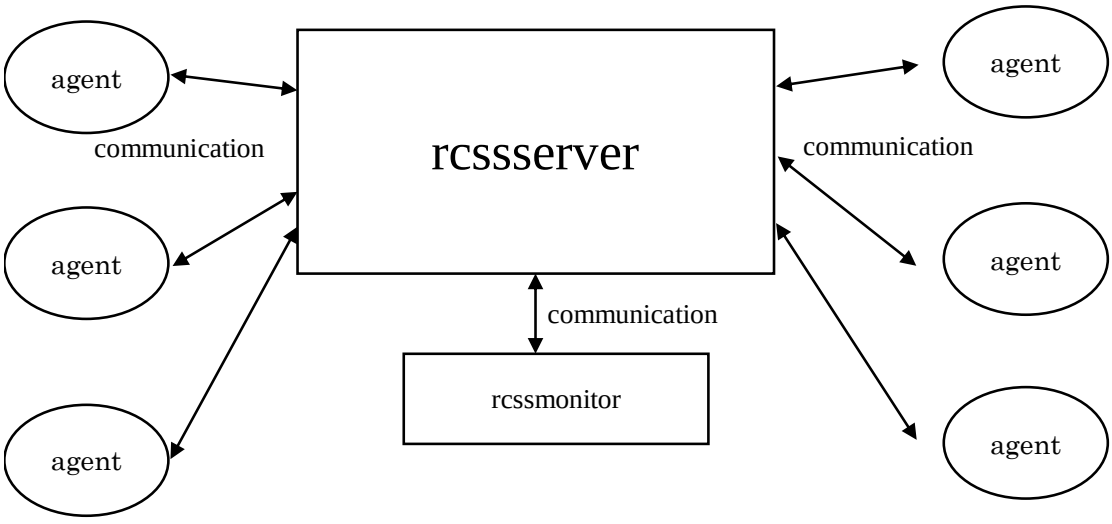

Figure 10. Model of Robocup 2D simulator

4.3 Result

The pass success rate of the original agent2d is shown in Table 5. The pass success rate of agent2d_svm is shown in Table 6.

Table 5. The pass success rate of natural agent2d

Match	**Pass times**	**Success**	**Failure**	**Rate[%]**
1	110	82	28	74
2	123	88	35	71
3	99	67	32	67
4	101	71	30	70
5	107	76	31	71
6	98	74	24	75
7	90	60	30	66
8	104	75	29	72
9	105	82	23	78
10	107	78	29	72
Total	1044	753	291	72

Table 6. The pass success rate of agent2d_svm

Match	Pass times	Success	Failure	Rate[%]
1	98	71	27	72
2	116	82	34	70
3	97	65	32	67
4	105	75	30	71
5	102	70	32	68
6	91	67	24	73
7	92	63	29	68
8	98	69	29	70
9	106	82	24	77
10	104	76	28	73
Total	1009	720	289	71

The total rate of original agent2d is 72% and one of agent2d_svm is 71%. Comparing two results, pass success rate of agent2d_svm is lower 1% than pass success rate of the original agent2d, so the performance results of these two systems are comparable.

5 Discussion

As it can be seen, the modified agent2d_svm shows comparable performance with the original agent2d. I can suggest the following reasons for not surpassing the original algorithm in terms of pass success rate.

First, the learning model produced by learner.rb can be improved. The script learner.rb exports the text file to train for detect the timing that the player did the pass action. This part is very important because it relate to the whole procedure after this step. In the script learner.rb, I defined the condition of detect the pass timing. The condition in this thesis adopts just ball moved distance (see Section 3.5). This condition can be changed to a more advanced formula to increase accuracy. Similarly,

the attributes that are stored in the text file for training can be modified to advance accuracy. For example, the speed of the ball.

Second, it is possible to improve the learning model of predict.rb. The script predict.rb exports the text file to train for additional evaluation method based on Support Vector Machine for the pass action. However, the conditions that I defined to classify the pass as success or failure are derived from observation, and are correct to a certain degree. The attributes in text file for training provided by predict.rb, however, can modified to a certain degree. The distance to the nearest opponent and the distance between the passer and the receiver are used now. The implementation the function for computing the rate of threat from situation and some formula might help to advance the performance of agent2d_svm.

Third, it is possible to change the algorithm. In this thesis, I tried to improve agent2d with Support Vector Machine. However, the algorithm can be improved and we can use it to improve the scheme. I can suggest to try to use the Support Vector Machine with deep learning that I introduced in this thesis.

Finally, the real soccer recordings show the behavior of real, not ideal, teams. They also can perform unsuccessful pass actions, and thus our agent2d_svm could learn these erroneous patters. However, I do not see it as a problem, since the real task of learning from real data is to create a human-like AI, not the most skillful AI.

6 Conclusions

In this thesis, I tried to build a new scheme to develop an AI agent for soccer based on real soccer recording and machine learning. I achieved that and tested the performance of AI agent made according to the proposed scheme. As a result, the hints to improve AI agent for soccer by machine learning and real soccer recording are given. Proposed scheme enables to analyze and visualize real soccer recording. Furthermore, the scheme enables to make learning model for Support Vector Machine by learning from annotated data of real soccer recording. Finally, the scheme enables AI agent for soccer to get additional evaluation method that is to evaluate the pass route based on prediction by Support Vector Machine.

It is possible to improve the behavior performance by changing or adding attributes into the scripts learner.rb and predict.rb. There are many options to change in the attributes in text file for making learning model. Furthermore, the conditions for definition of pass timing have room for improvement, too. There are many options to

improve the proposed scheme, since the attributes and the conditions for learning have many combinations.

Finally, I could build the scheme to build AI agent team based on real soccer recording and machine learning. Furthermore, it became clear the scheme has the room for improvement and the hints to improve scheme were given through the experiment.

Reference

[1] Makoto Nagao, 1992, “Jinkou Chinou To Ningen (Artificial Intelligence and Human)”, Iwanami Shinsho, 248

[2] Jason Brownlee, “Practical Machine Learning Problems”, http://machinelearningmastery.com/practical-machine-learning-problems/ (December 13 2014)

[3] Yanai Kohsuke, Mita Hideyuki, Iba Hitoshi, “Robot learning of cooperative behavior using Genetic Programming”, http://ci.nii.ac.jp/naid/110003187220 (December 13 2014)

[4] Junichiro Hirayama, Thawonmas Ruck, “Applying of Human Decision-Making Behaviors to RoboCup Software Agnets”, http://ci.nii.ac.jp/naid/110003187272/ (December 13 2014)

[5] Katsuhiro Yamashita, Tomoharu Nakashima, Hidehisa Akiyama, “Constructing Prediction Models of Opponent Positions in RoboCup Soccer”, http://winnie.kuis.kyoto-u.ac.jp/sig-challenge/SIG-Challenge-B301/SIG-Challenge-B301-03.pdf (December 6 2014)

[6] agent2d, http://rctools.sourceforge.jp/pukiwiki/index.php?agent2d (December 6 2014)

[7] Takashi Onoda, “Support Vector Mahine no Gaiyo (The outline of Support Vector Machine)”, http://www.orsj.or.jp/~archive/pdf/bul/Vol.46_05_225.pdf (December 6 2014)

[8] Yichuan Tang, “Deep Learning using Linear Support Vector Machines”, http://deeplearning.net/wp-content/uploads/2013/03/dlsvm.pdf (December 6 2014)

[9] Computational Intelligence Lab (Ritsumeikan University), “Support Vector Machine”, http://www.sys.ci.ritsumei.ac.jp/project/theory/svm/svm.html (December 6 2014)

[10] Asa Ben-Hur, Jason Weston, “A User's Guide to Support Vector Machines”, http://pyml.sourceforge.net/doc/howto.pdf (December 6 2014)

[11] Chih-Chung Chang, Chih-Jen Lin, “LIBSVM － A Library for Support Vector Machines”, http://www.csie.ntu.edu.tw/~cjlin/libsvm/ (December 6 2014)

[12] Robocup tools agent2d releases, http://sourceforge.jp/projects/rctools/releases/55186 (December 17 2014)

[13] TRACAB, http://tracab.hegogroup.com/ (December 6 2014)

[14] SAAB, http://www.saabgroup.com/ (December 17 2014)

[15] Data Stadium, www.datastadium.co.jp/index.html (December 6 2014)

[16] Hidehisa Akiyama, “Robocup 2D Guide Book - 1.0”, http://iij.dl.sourceforge.jp/rctools/46021/RoboCup2DGuideBook-1.0.pdf (December 6 2014)

Appendix

rs_analyzer.rb----------

```
require "./rs_svm.rb"
require "./predict.rb"
require "./learner.rb"
require "./field_visualize.rb"

puts "Exit:exit lerningModel for detect pass:0 learningModel for agent2d:1 Visualize:2"
while str = STDIN.gets
	if str.chomp == "exit" then
		break
	elsif str.chomp == "0"
			LearningF = LearningFileEx.new()
		LearningF.leaningFileExport("./real_log_data.log")

		/input parameter to C and gamma/
		system("svm-train -c 8192.0 -g 0.000030517578125 learning_for_detectPass.txt
k_svm.model")
	elsif str.chomp == "1"
		Predict = PredictKick.new()
		Predict.predictKick("./real_log_data.log")

		/input parameter to C and gamma/
		system("svm-train -c 8.0 -g 8.0 learning_for_agent2d.txt a_svm.model")
	elsif str.chomp == "2"
		Field = Field_visualize.new()
		Field.fieldsize_comp("./real_log_data.log")
		Field.visualization("./real_log_data.log")
	end
puts "Exit:exit lerningModel for detect pass:0 learningModel for agent2d:1 Visualize:2"
end
```

learner.rb----------

```
/
This script makes text file for training to detect pass timing
/

class LearningFileEx

    def leaningFileExport(input_file)

        /data set/
        f = File.open(input_file,'r')

        /text file for learning/
        fw1 = File.open("./learning_for_detectPass.txt",'w')



        /The array for contain the divided data set/
        chunk = Array.new()
        players_and_referee = Array.new()
        mans_data = Array.new(6)
        ball_data = Array.new(7)

        /The array to store previous state/
        mans_data_pre = Array.new(6)
        ball_data_pre = Array.new(7)

        /flag to recognize the top of file/
        fir_flag = 0

        /ball owning player number/
        ball_own_pl = 0

        /ball moved distane in previous state/
        ball_moved_pre = 0

        /ball owning team/
        ball_own_team = 0
```

```
count = 0
sum = 0

/ball moved distance in current state/
ball_moved = 0

a=0
b=0

f.each do |line|
        count = count + 1
        dist_bp_min = 1000000.0

        chunk = line.split(":")
        players_and_referee = chunk[1].split(";")
        ball_data = chunk[2].split(",")

        if fir_flag == 1 then
                ball_moved_pre = ball_moved
                ball_moved = Math.sqrt((ball_data[0].to_f-ball_data_pre[0].to_f)*(ball_data[0].to_f-
ball_data_pre[0].to_f)+(ball_data[1].to_f-ball_data_pre[1].to_f)*(ball_data[1].to_f-ball_data_pre[1].to_f))
                ball_data_pre = ball_data
        else
                fir_flag = 1
        end

        sum = sum + ball_moved

        /chunk[0]:Count
         chunk[1]:ball
         chunk[2]:players_and_referee

         ball_data[0]:X
         ball_data[1]:Y
         ball_data[2]:Z
         ball_data[3]:Speed
         ball_data[4]:owning Team
```

```
                ball_data[5]:"Alive" or "Dead"
                /

                players_and_referee.each do |p_and_r|
                    mans_data = p_and_r.split(",")

                    /detecting the ball owning player/
                    if mans_data[0] != 3 && dist_bp_min > Math.sqrt((ball_data[0].to_f-
mans_data[3].to_f)*(ball_data[0].to_f-mans_data[3].to_f)+(ball_data[1].to_f-
mans_data[4].to_f)*(ball_data[1].to_f-mans_data[4].to_f)) then
                        dist_bp_min = Math.sqrt((ball_data[0].to_f-
mans_data[3].to_f)*(ball_data[0].to_f-mans_data[3].to_f)+(ball_data[1].to_f-
mans_data[4].to_f)*(ball_data[1].to_f-mans_data[4].to_f))
                        ball_own_pl = mans_data[1]
                        ball_own_team = mans_data[0]
                    end

                    /
                    mans_data[0]:Team(0:Away 1:Home 3:Referee)
                    mans_data[1]:Number
                    mans_data[3]:X
                    mans_data[4]:Y
                    mans_data[5]:Speed
                    /

                end

                if fir_flag == 1 then
                    /Writing to text file as LIBSVM learning file format/
                    /ball moved distance, ball moved distance in previous state, the margin between
2 parameters are contained/
                    if ball_moved > 100 && ball_moved - ball_moved_pre > 100 && a < 3000 then
                        fw1.puts "1 1:" + ball_moved_pre.to_s + " 2:" + ball_moved.to_s + " 3:" +
(ball_moved - ball_moved_pre).to_s
                        a = a + 1
                    elsif b < 3000
                        fw1.puts "0 1:" + ball_moved_pre.to_s + " 2:" + ball_moved.to_s + " 3:" +
(ball_moved - ball_moved_pre).to_s
```

```
                    b = b + 1
                end
            end


        end

        average = sum/count

        f.close
        fw1.close

    end

end
```

predict.rb----------

```
/
This script makes text file for training to evaluate the pass route
/

class PredictKick

    Svm = RsSvm.new()

    def predictKick(input_file)

        /data set/
        f = File.open(input_file,'r')

        /text file for training/
        fw = File.open("./learning_for_agent2d.txt",'w')

        /The array for contain the divided data set/
        chunk = Array.new(3)
        players_and_referee = Array.new(29)
        mans_data = Array.new(6)
        ball_data = Array.new(7)

        /The array to store previous state/
        ball_data_pre = Array.new(7)

        /ball owning player/
        ball_own_man = Array.new(6)

        /passer data/
        pass_starter = Array.new(6)

        pass_start_flag = 0
        kicker_team = 0
        cycle_count = 0

        /distance between passer and enemy/
        dist_enemy_player = 10000.0
```

```
fir_flag = 0
i = 0

count_flag = 0

ball_own_pl = 0
ball_own_pl_pre = 0
ball_own_team = 0

ball_moved_pre = 0
ball_moved = 0

f.each do |line|
    dist_bp_min = 1000000.0

    chunk = line.split(":")
    players_and_referee = chunk[1].split(";")
    ball_data = chunk[2].split(",")

    if fir_flag == 1 then
        ball_moved_pre = ball_moved
        ball_moved = Math.sqrt((ball_data[0].to_f-ball_data_pre[0].to_f)*(ball_data[0].to_f-
ball_data_pre[0].to_f)+(ball_data[1].to_f-ball_data_pre[1].to_f)*(ball_data[1].to_f-ball_data_pre[1].to_f))
        ball_data_pre = ball_data
    else
        fir_flag = 1
    end

    /chunk[0]:Count
     chunk[1]:ball
     chunk[2]:players_and_referee

     ball_data[0]:X
     ball_data[1]:Y
     ball_data[2]:Z
     ball_data[3]:Speed
     ball_data[4]:owning Team
```

```
ball_data[5]:"Alive" or "Dead"
/

players_and_referee.each do |p_and_r|
    mans_data = p_and_r.split(",")

    if ball_data[4] == "H" then
        if mans_data[0].to_s == "1" && dist_bp_min > Math.sqrt((ball_data[0].to_f-
mans_data[3].to_f)*(ball_data[0].to_f-mans_data[3].to_f)+(ball_data[1].to_f-
mans_data[4].to_f)*(ball_data[1].to_f-mans_data[4].to_f)) then
            dist_bp_min = Math.sqrt((ball_data[0].to_f-
mans_data[3].to_f)*(ball_data[0].to_f-mans_data[3].to_f)+(ball_data[1].to_f-
mans_data[4].to_f)*(ball_data[1].to_f-mans_data[4].to_f))

            ball_own_man = mans_data
            ball_own_team = ball_data[4]

        end
    elsif ball_data[4] == "A" then
        if mans_data[0].to_s == "0" && dist_bp_min > Math.sqrt((ball_data[0].to_f-
mans_data[3].to_f)*(ball_data[0].to_f-mans_data[3].to_f)+(ball_data[1].to_f-
mans_data[4].to_f)*(ball_data[1].to_f-mans_data[4].to_f)) then
            dist_bp_min = Math.sqrt((ball_data[0].to_f-
mans_data[3].to_f)*(ball_data[0].to_f-mans_data[3].to_f)+(ball_data[1].to_f-
mans_data[4].to_f)*(ball_data[1].to_f-mans_data[4].to_f))

            ball_own_man = mans_data
            ball_own_team = ball_data[4]
        end

    end


    /
    mans_data[0]:Team(0:Away 1:Home 3:Referee)
    mans_data[1]:Number
    mans_data[3]:X
    mans_data[4]:Y
```

```
                mans_data[5]:Speed
                /

            end

            if Svm.predict_pass(ball_moved_pre,ball_moved,ball_moved - ball_moved_pre) == 1.0 &&
pass_start_flag == 0 then
                pass_start_flag = 1
                pass_starter = ball_own_man
                kicker_team = ball_own_team
                dist_enemy_player = 10000.0


                players_and_referee.each do |p_and_r|
                    mans_data = p_and_r.split(",")

                    if pass_starter[0] != mans_data[0] && mans_data[0] != "3" &&
dist_enemy_player > Math.sqrt((pass_starter[3].to_f - mans_data[3].to_f)*(pass_starter[3].to_f -
mans_data[3].to_f)+(pass_starter[4].to_f - mans_data[4].to_f)*(pass_starter[4].to_f -
mans_data[4].to_f)) then
                        dist_enemy_player = Math.sqrt((pass_starter[3].to_f -
mans_data[3].to_f)*(pass_starter[3].to_f - mans_data[3].to_f)+(pass_starter[4].to_f -
mans_data[4].to_f)*(pass_starter[4].to_f - mans_data[4].to_f))
                    end

                end

                count_flag = 1

            end

            if count_flag == 1 then
                cycle_count = cycle_count + 1
            elsif count_flag == 0 then
                cycle_count = 0
            end
```

```
                if pass_start_flag == 1 && pass_starter[1] != ball_own_man[1] && kicker_team ==
ball_own_team then
                    dist_enemy = 10000.0


                    players_and_referee.each do |p_and_r|
                        mans_data = p_and_r.split(",")

                        if ball_own_man[0] != mans_data[0] && mans_data[0] != "3" && dist_enemy
> Math.sqrt((ball_data[0].to_f - mans_data[3].to_f)*(ball_data[0].to_f -
mans_data[3].to_f)+(ball_data[1].to_f - mans_data[4].to_f)*(ball_data[1].to_f - mans_data[4].to_f)) then
                            dist_enemy = Math.sqrt((ball_data[0].to_f -
mans_data[3].to_f)*(ball_data[0].to_f - mans_data[3].to_f)+(ball_data[1].to_f -
mans_data[4].to_f)*(ball_data[1].to_f - mans_data[4].to_f))
                        end

                    end


                    distance_pr = Math.sqrt((ball_own_man[3].to_f -
pass_starter[3].to_f)*(ball_own_man[3].to_f - pass_starter[3].to_f)+(ball_own_man[4].to_f -
pass_starter[4].to_f)*(ball_own_man[4].to_f - pass_starter[4].to_f))

                    fw.puts "1 1:" + (distance_pr/10000).to_s + " 2:" +
(dist_enemy_player/10000).to_s + " 3:" + (dist_enemy/10000).to_s

                    pass_start_flag = 0
                    passer_risk = 0
                    recieve_risk = 0
                    count_flag = 1

                elsif pass_start_flag == 1 && pass_starter[1] != ball_own_man[1] && kicker_team !=
ball_own_team then
                    dist_enemy = 10000.0


                    players_and_referee.each do |p_and_r|
                        mans_data = p_and_r.split(",")
```

```
                        if ball_own_man[0] != mans_data[0] && mans_data[0] != "3" && dist_enemy
> Math.sqrt((ball_data[0].to_f - mans_data[3].to_f)*(ball_data[0].to_f -
mans_data[3].to_f)+(ball_data[1].to_f - mans_data[4].to_f)*(ball_data[1].to_f - mans_data[4].to_f)) then
                            dist_enemy = Math.sqrt((ball_data[0].to_f -
mans_data[3].to_f)*(ball_data[0].to_f - mans_data[3].to_f)+(ball_data[1].to_f -
mans_data[4].to_f)*(ball_data[1].to_f - mans_data[4].to_f))
                        end

                    end


                    distance_pr = Math.sqrt((ball_own_man[3].to_f -
pass_starter[3].to_f)*(ball_own_man[3].to_f - pass_starter[3].to_f)+(ball_own_man[4].to_f -
pass_starter[4].to_f)*(ball_own_man[4].to_f - pass_starter[4].to_f))



                    /writing to text file for training/
                    /file contains distance between passer and reciever, passer and nearest enemy,
reciever and nearest enemy/
                    fw.puts "0 1:" + (distance_pr/10000).to_s + " 2:" +
(dist_enemy_player/10000).to_s + " 3:" + (dist_enemy/10000).to_s

                    pass_start_flag = 0
                    passer_risk = 0
                    recieve_risk = 0
                    count_flag = 0


                end

            end

            f.close
            fw.close
        end

end
```

rs_svm.rb----------

```
require 'svm'

class RsSvm

      M = Model.new("k_svm.model")

   def predict_pass(dist_pre,dist,distances)
      res = M.predict([dist_pre,dist,distances])
      return res
   end

end
```

field_visualize.rb----------

```
class Field_visualize

    @@max_width = 0.0
    @@max_height = 0.0

    /computing the field size/
    def fieldsize_comp(input_file)

        f = File.open(input_file,'r')

        chunk = Array.new(3)
        ball_data = Array.new(7)

        f.each do |line|

            chunk = line.split(":")
            ball_data = chunk[2].split(",")

            if @@max_width < ball_data[0].to_f && ball_data[5] == "Alive" then

                @@max_width = ball_data[0].to_f

            end

            if @@max_height < ball_data[1].to_f && ball_data[5] == "Alive" then

                @@max_height = ball_data[1].to_f

            end

            /chunk[0]:Count
             chunk[1]:ball
             chunk[2]:players_and_referee

             ball_data[0]:X
             ball_data[1]:Y
             ball_data[2]:Z
```

```
                ball_data[3]:Speed
                ball_data[4]:owning Team
                ball_data[5]:"Alive" or "Dead"
                /

            end

            puts "Field Width = " + (@@max_width*2).to_s + ", Field Height = " + (@@max_height*2).to_s

            f.close

        end

        /exec the visualizer/
        def visualization(input_file)

            system("firefox visualize.html&")

        end



end
```

visualization.html----------

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html>

 <head>
    <title>visualize</title>
    <script type="text/javascript" src="rsvisualize.js"></script>
 </head>

 <body>
    <form name="log">
    <input type="file" id="selfile"><br>
    </form>
    <canvas id="field" width="725" height="440"></canvas>
    <script>
    var obj1 = document.getElementById("selfile");
    var j = 0;

    //When the file selected, do
    obj1.addEventListener("change",function(evt){

    var file = evt.target.files;

    //Making FileReader
    var reader = new FileReader();
    //Reading as a text
    reader.readAsText(file[0]);

    //Processing after reading
    reader.onload = function(ev){
        var readline = new Array();
        readline = reader.result.split("¥n");


        var i = 0;
        while(readline[i] != null){
```

```
            i++;
        }


        var id = 0;
        id = setInterval(function(){animat(readline,i,id);},40);
}},false);

//for animation, using anvas method on HTML5
function animat(lines,i,id){
        console.log(lines[j]);
        var canvas = document.getElementById("field");
        var ctx = canvas.getContext("2d");
    var canvas_h = canvas.height;
    var canvas_w = canvas.width;

        var chunk = new Array();
        var player_and_referee = new Array();
        var ball_data = new Array();

    var n = 2;

    ctx.fillStyle = "#006400"
        ctx.fillRect(0,0,canvas_w,canvas_h);

    ctx.strokeStyle = "#ffffff"
    ctx.beginPath();
    ctx.moveTo(100,50);
    ctx.lineTo(canvas_w - 100,50);
    ctx.moveTo(canvas_w - 100,50);
    ctx.lineTo(canvas_w - 100,canvas_h - 50);
    ctx.moveTo(canvas_w - 100,canvas_h - 50);
    ctx.lineTo(100,canvas_h - 50);
    ctx.moveTo(100,canvas_h - 50);
    ctx.lineTo(100,50);
    ctx.moveTo(canvas_w/n,50);
    ctx.lineTo(canvas_w/n,canvas_h - 50);
```

```
ctx.moveTo(100-55/n,canvas_h/n - 36.6/n);
ctx.lineTo(100,canvas_h/n - 36.6/n);
ctx.moveTo(100-55/n,canvas_h/n - 36.6/n);
ctx.lineTo(100-55/n,canvas_h/n + 36.6/n);
ctx.moveTo(100,canvas_h/n + 36.6/n);
ctx.lineTo(100-55/n,canvas_h/n + 36.6/n);

ctx.moveTo(canvas_w-100+55/n,canvas_h/n - 36.6/n);
ctx.lineTo(canvas_w-100,canvas_h/n - 36.6/n);
ctx.moveTo(canvas_w-100+55/n,canvas_h/n - 36.6/n);
ctx.lineTo(canvas_w-100+55/n,canvas_h/n + 36.6/n);
ctx.moveTo(canvas_w-100,canvas_h/n + 36.6/n);
ctx.lineTo(canvas_w-100+55/n,canvas_h/n + 36.6/n);

ctx.moveTo(100,canvas_h/n - (36.6+55)/n);
ctx.lineTo(100+55/n,canvas_h/n - (36.6+55)/n);
ctx.moveTo(100+55/n,canvas_h/n - (36.6+55)/n);
ctx.lineTo(100+55/n,canvas_h/n + (36.6+55)/n);
ctx.moveTo(100+55/n,canvas_h/n + (36.6+55)/n);
ctx.lineTo(100,canvas_h/n + (36.6+55)/n);

ctx.moveTo(canvas_w-100,canvas_h/n - (36.6+55)/n);
ctx.lineTo(canvas_w-100-55/n,canvas_h/n - (36.6+55)/n);
ctx.moveTo(canvas_w-100-55/n,canvas_h/n - (36.6+55)/n);
ctx.lineTo(canvas_w-100-55/n,canvas_h/n + (36.6+55)/n);
ctx.moveTo(canvas_w-100-55/n,canvas_h/n + (36.6+55)/n);
ctx.lineTo(canvas_w-100,canvas_h/n + (36.6+55)/n);

ctx.moveTo(100,canvas_h/n - (36.6+55+110)/n);
ctx.lineTo(100+(55+110)/n,canvas_h/n - (36.6+55+110)/n);
ctx.moveTo(100+(55+110)/n,canvas_h/n - (36.6+55+110)/n);
ctx.lineTo(100+(55+110)/n,canvas_h/n + (36.6+55+110)/n);
ctx.moveTo(100+(55+110)/n,canvas_h/n + (36.6+55+110)/n);
ctx.lineTo(100,canvas_h/n + (36.6+55+110)/n);

ctx.moveTo(canvas_w-100,canvas_h/n - (36.6+55+110)/n);
ctx.lineTo(canvas_w-100-(55+110)/n,canvas_h/n - (36.6+55+110)/n);
ctx.moveTo(canvas_w-100-(55+110)/n,canvas_h/n - (36.6+55+110)/n);
```

```
        ctx.lineTo(canvas_w-100-(55+110)/n,canvas_h/n + (36.6+55+110)/n);
        ctx.moveTo(canvas_w-100-(55+110)/n,canvas_h/n + (36.6+55+110)/n);
        ctx.lineTo(canvas_w-100,canvas_h/n + (36.6+55+110)/n);

        ctx.stroke();

        ctx.beginPath();
        ctx.arc(canvas_w/n,canvas_h/n,91.5/n,0,Math.PI*2,false);
        ctx.stroke();



        //recognizing the file and drawing the ball
            chunk = lines[j].split(":");
            players_and_referee = chunk[1].split(";");
            ball_data = chunk[2].split(",");

            if(ball_data[5].indexOf("Alive") != -1) ctx.fillStyle = "#f0f8ff";
            else ctx.fillStyle = "#dc143c";

        ctx.beginPath();
        ctx.arc((parseFloat(ball_data[0])+(canvas_w-200)*10)/10/n+100,(parseFloat(ball_data[1])*(-
1)+(canvas_h-100)*10-parseFloat(ball_data[2]))/10/n+50, 5, 0, Math.PI*2, false);
        ctx.fill();

        ctx.strokeStyle = "#8b008b"
        ctx.beginPath();
        ctx.moveTo((parseFloat(ball_data[0])+(canvas_w-
200)*10)/10/n+100,(parseFloat(ball_data[1])*(-1)+(canvas_h-100)*10-
parseFloat(ball_data[2]))/10/n+50);
        ctx.lineTo((parseFloat(ball_data[0])+(canvas_w-
200)*10)/10/n+100,(parseFloat(ball_data[1])*(-1)+(canvas_h-100)*10)/10/n+50)
        ctx.stroke();

            players_and_referee.forEach(p_a_r);

            if(lines[j] == null) clearInterval(id);
            else j++;
```

```
    }

      //recognizeng file and drawing the player and referee
    function p_a_r(line,index,ar){

        var canvas = document.getElementById("field");
        var ctx = canvas.getContext("2d");
       var canvas_h = canvas.height;
       var canvas_w = canvas.width;
       var n = 2;

  mans_data = line.split(",");

  if(mans_data[0] == 0) ctx.fillStyle = "#FFD700";
  else if(mans_data[0] == 1) ctx.fillStyle = "#800000";
  else if(mans_data[0] == 3) ctx.fillStyle = "#000000";
  ctx.fillRect((parseFloat(mans_data[3])+(canvas_w-
200)*10)/10/n+100,(parseFloat(mans_data[4])*(-1)+(canvas_h-100)*10)/10/n+50,3,-13);

    }

  </script>
 </body>

</html>
```

sample_field_evaluator.cpp----------

```
// -*-c++-*-

/*
 *Copyright:

 Copyright (C) Hiroki SHIMORA

 This code is free software; you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation; either version 3, or (at your option)
 any later version.

 This code is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 GNU General Public License for more details.

 You should have received a copy of the GNU General Public License
 along with this code; see the file COPYING.  If not, write to
 the Free Software Foundation, 675 Mass Ave, Cambridge, MA 02139, USA.

 *EndCopyright:
 */

//This code is added the fucntion of prediction by SVM to original sample_field_evaluator.cpp (agent2d-3.1.1/src/).
//If you want to use this evaluator in your team, please replace your sample_field_evaluator.cpp to this file and ./configure and make again.
//And also, you need install LIBSVM.

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include "sample_field_evaluator.h"

#include "field_analyzer.h"
```

```
#include "simple_pass_checker.h"

 //libsvm
#include "svm.h"

#include <rcsc/player/player_evaluator.h>
#include <rcsc/common/server_param.h>
#include <rcsc/common/logger.h>
#include <rcsc/math_util.h>

#include <iostream>
#include <algorithm>
#include <cmath>
#include <cfloat>

// #define DEBUG_PRINT

using namespace rcsc;

static const int VALID_PLAYER_THRESHOLD = 8;
svm_node elem[4];

const char *model_filename = "a_svm.model";
svm_model *model = svm_load_model(model_filename);
double score = 0.0;

/*-------------------------------------------------------------------*/
/*!

 */
static double evaluate_state( const PredictState & state );



/*-------------------------------------------------------------------*/
/*!

 */
SampleFieldEvaluator::SampleFieldEvaluator()
```

```
{

}

/*-------------------------------------------------------------------*/
/*!

 */
SampleFieldEvaluator::~SampleFieldEvaluator()
{

}

/*-------------------------------------------------------------------*/
/*!

 */
double
SampleFieldEvaluator::operator()( const PredictState & state,
                                  const std::vector< ActionStatePair > & /*path*/ ) const
{
    const double final_state_evaluation = evaluate_state( state );

    //
    // ???
    //

    double result = final_state_evaluation;

    return result;
}


/*-------------------------------------------------------------------*/
/*!

 */
static
```

```
double
evaluate_state( const PredictState & state )
{

    score = 0.0;

    //getting information for prediction by SVM
    //if predicted good, add same evaluation point as basic score
    double    dist_from_ball = 100.0;
    double    dist_from_self = 100.0;

    elem[0].index = 1;
    elem[0].value = state.self().distFromBall()/100;

    elem[1].index = 2;
    state.getOpponentNearestTo(state.self().pos(),1,&dist_from_self);
    elem[1].value = dist_from_self/100;

    elem[2].index = 3;
    state.getOpponentNearestTo(state.ball().pos(),1,&dist_from_ball);
    elem[2].value = dist_from_ball/100;
    elem[3].index = -1;



    const ServerParam & SP = ServerParam::i();

    const AbstractPlayerObject * holder = state.ballHolder();

#ifdef DEBUG_PRINT
    dlog.addText( Logger::ACTION_CHAIN,
                  "========= (evaluate_state) ==========" );
#endif

    //
    // if holder is invalid, return bad evaluation
    //
    if ( ! holder )
    {
```

```
#ifdef DEBUG_PRINT
            dlog.addText( Logger::ACTION_CHAIN,
                          "(eval) XXX null holder" );
#endif
            return - DBL_MAX / 2.0;
        }

        const int holder_unum = holder->unum();



        //
        // ball is in opponent goal
        //
        if ( state.ball().pos().x > + ( SP.pitchHalfLength() - 0.1 )
             && state.ball().pos().absY() < SP.goalHalfWidth() + 2.0 )
        {
#ifdef DEBUG_PRINT
            dlog.addText( Logger::ACTION_CHAIN,
                          "(eval) *** in opponent goal" );
#endif

            score = 1.0e+7;



          if ( svm_predict(model,elem) == 1.0)

            {
#ifdef DEBUG_PRINT
            dlog.addText( Logger::ACTION_CHAIN,
                          "(eval) XXX is predected good" );
#endif
               score = score + 1.0e+7;
            }

            else score = score - 1.0e+7;

            return score;
        }
```

```
    //
    // ball is in our goal
    //
    if ( state.ball().pos().x < - ( SP.pitchHalfLength() - 0.1 )
         && state.ball().pos().absY() < SP.goalHalfWidth() )
    {
#ifdef DEBUG_PRINT
        dlog.addText( Logger::ACTION_CHAIN,
                      "(eval) XXX in our goal" );
#endif


    score = -1.0e+7;



      if ( svm_predict(model,elem) == 1.0)

        {
#ifdef DEBUG_PRINT
        dlog.addText( Logger::ACTION_CHAIN,
                      "(eval) XXX is predected good" );
#endif
          score = score + 1.0e+7;
        }

        else score = score - 1.0e+7;

        return score;
    }


    //
    // out of pitch
    //
    if ( state.ball().pos().absX() > SP.pitchHalfLength()
```

```
            || state.ball().pos().absY() > SP.pitchHalfWidth() )
    {
#ifdef DEBUG_PRINT
        dlog.addText( Logger::ACTION_CHAIN,
                      "(eval) XXX out of pitch" );
#endif

        score = - DBL_MAX / 2.0;

        return score;

    }


    //
    // set basic evaluation
    //
    double point = state.ball().pos().x;

    point += std::max( 0.0,
                       40.0 - ServerParam::i().theirTeamGoalPos().dist( state.ball().pos() ) );

#ifdef DEBUG_PRINT
    dlog.addText( Logger::ACTION_CHAIN,
                  "(eval) ball pos (%f, %f)",
                  state.ball().pos().x, state.ball().pos().y );

    dlog.addText( Logger::ACTION_CHAIN,
                  "(eval) initial value (%f)", point );
#endif

    //
    // add bonus for goal, free situation near offside line
    //
    if ( FieldAnalyzer::can_shoot_from
         ( holder->unum() == state.self().unum(),
           holder->pos(),
           state.getPlayerCont( new OpponentOrUnknownPlayerPredicate( state.ourSide() ) ),
```

```
                VALID_PLAYER_THRESHOLD ) )
        {
            point += 1.0e+6;
#ifdef DEBUG_PRINT
            dlog.addText( Logger::ACTION_CHAIN,
                          "(eval) bonus for goal %f (%f)", 1.0e+6, point );
#endif

            if ( holder_unum == state.self().unum() )
            {
                point += 5.0e+5;
#ifdef DEBUG_PRINT
                dlog.addText( Logger::ACTION_CHAIN,
                              "(eval) bonus for goal self %f (%f)", 5.0e+5, point );
#endif
            }
        }

          if ( svm_predict(model,elem) == 1.0)

            {
        #ifdef DEBUG_PRINT
            dlog.addText( Logger::ACTION_CHAIN,
                          "(eval) XXX is predected good" );
        #endif

              point = point + 1.0e+7;
            }

            else point = point - 1.0e+7;

        return point;
}
```