Simulating Prediction Markets That Include Human and Automated Agents

by

Wendy Chang

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2009

Author .... [illegible]
Department of Electrical Engineering and Computer Science
May 22, 2009

Certified by .... [illegible]
Thomas W. Malone
Director, MIT Center for Collective Intelligence
Thesis Supervisor

Accepted by .... [illegible]
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Simulating Prediction Markets That Include Human and Automated Agents

by

Wendy Chang

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2009, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this work I study the interaction of sophisticated trading agents with simpler agents in a prediction market. The goal is to simulate markets with both human and computer agents, and investigate ways to maximize the performance of these markets. I start with the neural net-based agent that is currently used in CCI's collective prediction experiments on football plays. By tuning their training and risk affinity, I configure a "smart" agent to represent the sophisticated computer traders. I implement three types of simple agents to approximate human traders – two are rule based, and one uses aggregate human data from lab experiments. By exploring different combinations of smart versus simple agents, I showed that it is possible for mixes of agents to outperform either types alone. This result is consistent with the larger goal of the collective prediction project, which is to show that humans and computer agents combined in a prediction market can do better than either alone.

Thesis Supervisor: Thomas W. Malone
Title: Director, MIT Center for Collective Intelligence

Acknowledgments

I would like to thank Professor Tom Malone for his constant support and warm encouragement. This research effort has been both productive and fun under his guidance. I would also like to thank Chris Hibbert and Yiftach Nagar, both of whom were always available for questions, ideas and feedback. Finally, special thanks to two people who watched football with me – Rob Laubacher for providing formation data on the game plays, and Alvin Liang for checking the outcomes of training plays.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

The purpose of this study is to investigate the use of computer agents to enhance the performance of prediction markets. Specifically, the agents are designed to aid in the Center of Collective Intelligence's (CCI) Collective Prediction experiments.

In this study I examine the performance of our neural net-based agents using different training data. I also examine the performance of the agents with different risk affinity. With these parameters fine-tuned, I then use the agents to trade in the same prediction markets used in the Collective Prediction experiments. I use three types of simplistic agents to simulate human traders in a market: two rule-based, and one using aggregate human data from lab experiments. I examine the effect different mixes of neural net and simple agents have on market performance.

1.2 Background

A prediction market is a betting exchange set up for the purpose of predicting the outcome of some future event. Prices are tied to a trader's belief of the possibility of a certain outcome. The market setup incentivizes individuals to gather information and make sophisticated predictions about the outcome. Like in any other market, a trader may also operate by spotting market inefficiencies and exploiting them for profit.

Prediction markets have been shown to be useful tools in estimating the probability of future events. Gjerstad [1] shows the market equilibrium price is close to the mean belief of the participating traders. The information-aggregating nature of a market and its ability to allow traders to update their beliefs are key factors of success. This model has a lot of potential applications and documented uses. Oft-cited examples include the outcomes of presidential races and sporting events, and sales forecasting.

Established prediction markets include Foresight Exchange (FX), an open online market where players use play money to predict future events in topics ranging from politics to entertainment. Another well known example is Hollywood Stock Exchange (HSX). Players trade shares of actors, films and related options. In 2007, this exchange correctly predicted 32 of 39 Oscar nominees in selected categories, and also 7 of 8 winners in the top categories. [2]

Automated agents using any trading strategy can be introduced to prediction markets, in the hopes that they will augment market accuracy. Automated agents are capable of processing large amounts of information and producing decisions based on predetermined rules systematically. This quality may allow them to complement human traders in the economy. Many staple AI tools are good candidates for facilitating the decision-making of an agent, such as support vector machines and neural nets.

CCI runs a ongoing study on prediction markets called Collective Prediction. We wish to demonstrate that the right mix of computer agents and human participants will maximize the accuracy of a prediction market. We recruit subjects to trade in a prediction market on the outcome of football plays. Automated agents trade alongside the human traders in a portion of the markets, and we compare effect they have on market accuracy.

Chapter 2

System

2.1 System Overview

The experiment is run on two separate software packages, Zocalo and Collective Prediction. Zocalo [3] is an open source prediction market toolkit actively maintained by Chris Hibbert. It enables prediction market trading of multiple setups: binary vs multi-outcome markets, book orders vs market make, etc. Zocalo provides an user interface where a human subject could trade in open prediction markets. Our computer agents, as well as our system, communicates with Zocalo via a custom RPC protocol.

Collective Prediction was developed by Jason Carver, designed specifically to interact with Zocalo to run CCI's collective prediction experiments. [4] This system has two major components: the news server, which controls the flow of a football experiment, and the agent manager, which spawns and manages the computer agents. The two components are designed to run independently of each other, to preserve modularity and ensure graceful failure. CP is currently geared towards running experiments on football plays, but could easily be modified to run other experiments.

To run a football experiment, the news server first resets the balance of all participants' accounts and loads up a set of football plays. For each play, the news server plays a video clip of the action leading up to the play, stopping right when the teams are in formation. Then it opens up a prediction market on Zocalo for traders, both human and computer, to trade in. Human traders use Zocalo's graphical user interface to trade, while agents enter trades via RPC calls. The duration of the market can be automatically configured, or hand-controlled by an experiment administrator. Once the time is up the news server closes the market. At the end of all plays, the news server records results and terminates.

Computer agents may enter at any point during the experiment. The agent manager initializes the agents, then launches them in a staggered fashion. Once launched, the agent will find the news server to enter the current market, and attempt trading at regular intervals. The agent manager provides a centralized place for keeping track of the agents present during an experiment, and for recording data about agent actions.

2.2 Computer Agents

A computer agent produces a single prediction for the probability of PASS for every market that it enters. That prediction then becomes its target price for trading if it decides to trade. Though an agent may see a market multiple times, the prediction does not change.

We currently have four types of agents implemented – a neural-net based agent, two simple rule-based agents, and a deterministic agent. The underlying neural nets for the neural net agents are implemented using Joone [5], an open source neural net toolkit. We use the JooneTools API to create a standard 3-layer neural net. We choose to use a sigmoid transform function for our output layer, which limits the output to within the range of 0 to 1. This is appropriate for our binary classification problem. Using football play data as input, the neural net outputs a probability for the play being a pass.

A parameter we call *biasForAction* is a key configuration for a neural net agent. Once the underlying neural net produces a prediction, we use this parameter to evaluate whether it is "worth it" to trade. An risk-averse agent would only enter a trade if it's confident about the outcome – in other words, if its prediction is close to either extreme. *biasForAction* specifies a range from either probability extreme in which the agent will trade. For example, an agent with a bias of 10 will trade only if its prediction falls in the range [90, 100], or [0, 10].

The rule-based agent asks a simple question – how many yards are left to first down? If this parameter *yardsToDown* is larger than 5, the agent will predict a higher likelihood of pass. Else if *yardsToDown* is < 5, the agent will predict a lower likelihood of pass (higher likelihood of run). The agent uses a base prediction for the overall likelihood of pass, and adds a deviation depending on the value of *yardsToDown*. The pass baseline and the size of the deviation can be adjusted. Also, we have the option of adding a small random perturbation to the deviation and to the final prediction.

Another rule-based agent uses visual information from the football videos to make their decision. This is an effort to capture a piece of information available only to our human traders in lab. We manually coded up the formation of the teams (e.g. shotgun formation with one running back) for the agent to consider. The agent uses simple rules based on this information to trade. For clarity, this paper refers to these agents as "visual" traders, and the rule-based agents using *yardsToDown* as "dumb" traders.

Finally, the deterministic agent is the simplest in concept of the agents. The agent takes a predefined set of numbers as input, and uses them as target prices in its trades. This construct is useful for simulating past behavior of a group in an experiment on the same football plays.

Chapter 3

Experiment Design

3.1 Controlled Parameters

The experiment can be run with the presence of only human or computer traders, or both. Full-scale experiments involving human subjects require extra procedures, such as a live demo of the software and training rounds. In this study I focus on experiments with computer traders ("agents") only. For consistency, I keep the experiment parameters in accordance with those we use in the lab.

Each experiment consists of 20 markets, each corresponding to a football play. The plays come from the 2008 Fiesta Bowl, between West Virginia University and Oklahoma University (WVUvOU). Excluding plays that are neither pass nor run, and plays used for training rounds in human experiments, we use plays number 3 through 30 in the 1st quarter. In an experiment with human traders, we show a video of the action leading up to a play before opening the market. In agents-only experiments this step is omitted.

Another modification that allows agents-only experiments to run much faster is the reduction of market duration. In the mixed experiment we allow $3\frac{1}{2}$ minutes per market, or 210000 ms. The agents are set to a polling period of 20 seconds or 20000 ms. In the agents-only version we reduce both times by a factor of 10, for a market length of 21000 ms and polling rate of 20 ms. An agent can make a trade each time it polls an open market. Each agent can buy or sell a maximum of 1000 shares per trade. This number is consistent with the setting we place on the agents in lab experiments.

Each market is opened on Zocalo with the presence of a market maker funded with \$10000. The use of a market maker promotes more trading activity, versus using book orders. I use a pool of 16 agents, which is close to the average number of participants in a humans-only run. In mixed experiments, both types of traders get the same starting balance of \$10000 to be used across all 20 plays. However, for this study I set a starting balance of \$50000 to prevent agents from zeroing out during the early rounds. This is merely a convenience measure. In practice, only one pool of agents (the one trained with the game KUvVT) needed the extra balance to stay active.

The neural net agent takes 3 inputs for each football play: the down number (1-4), number of yards to first down (within (0, 10] in most cases, can be larger with penalty), and whether the previous play in the game was a pass (1, 0). The default training data used in the lab are plays from the first quarter of a 2007 NFL game between Washington Redskins and New York Giants (WASvNYG). For this study we add data from the first quarters of two more games to the default training set. For the parameter *biasForAction*, I set a mid-range value of 30. This filters out some of the trading activity when the agents are less confident, while still maintaining participation in a lot of plays.

We define two types of dumb agents for this study, "dumb" and "dumbest". Both use the same baseline pass prediction of 54%. This number comes from Carver's thesis and is derived from the joint statistics of WAS and NYG in 2007, the two teams that play our default training game. The "dumbest" agent simply outputs the pass baseline as its prediction for every play. The "dumb" agent uses a deviation of 10 – when *yardsToDown* is $>=5$, it predicts $54 + 10 = 64\%$ pass likelihood; otherwise, it predicts 44%. Neither agent type uses randomness in producing predictions.

We also use the visual agent for this study. The visual agent uses the same baseline pass prediction as the dumb agents, but only trades when it receives useful information about a play's formation. A "shotgun" formation with one running back standing next to the quarterback receives a deviation of 20 towards pass. A shotgun formation with no running backs next to the quarterback is considered a strong indication of a pass and receives a deviation of 30. Thus, the visual trader predicts $54 + 20 = 74\%$ pass for a "shotgun-plus-one", $54 + 30 = 84\%$ pass for a "bare shotgun", and enters no trades for all other formations.

The deterministic agent is initialized using aggregate data from lab experiments run on the same 20 football plays. For its target prices, I look at the closing market prices from 20 lab experiments. Excluding plays where computer agents participated, I collected 10 closing prices for each play. The deterministic agent's target price of a play is set to the average of the corresponding prices.

Collective Prediction is written with many configurable parameters to handle different testing situations. For a complete list, see Appendix A.

3.2 Experimental Setup

The experiments are grouped into five sets, each exploring the effect of an independent variable on the agents' performance. The first two sets focus on fine tuning the neural net agent's performance, and in the third set I explore the interaction of these "smart" agents pitched against the simpler rule-based ones. The fourth set compares mixes of smart and visual agents, and the fifth compares mixes of smart and deterministic agents.

3.2.1 Neural Net Training Data

I experiment with different training data sets for this part. The data comes from plays from the first quarter of three separate games. The first one is our lab training game, WAS versus NYG. This is the training set we use for the agents in our lab studies. The other two sets come from college games. They are the 2009 Cotton Bowl between Mississippi and Texas Tech (MISSvTTU), and the 2008 Orange Bowl between Kansas and Virginia Tech (KUvVT).

I compare the performance of four pools of agents: one for each of the training sets, and one trained with all three sets.

3.2.2 Neural Net Agent Bias for Action

The parameter *biasForAction* can take on any value between 0 and 50. An agent with a bias of 50 corresponds to the most risk-seeking trader – it will trade in every single market it enters.

I compare the performance of five pools of agents, each with *biasForAction* values of 10, 20, 30, 40, and 50.

3.2.3 Mixes of Neural Net Agents and Rule-Based Agents

In this part I pitch the more sophisticated neural net agents against our more simplistic rule-based agents. Starting with a pool of neural net agents only, I gradually add more "dumb" agents, keeping the total number of agents at 16. I test the performance of "smart" versus "dumb" agents, as well as "smart" versus "dumbest" agents. For each pairing, I tested 5 pools – 16 neural net agents, 14 smart plus 2 dumb, 12-plus-4, 8-plus-8, and finally 16 "dummies". Mixes with a large number of dumb agents versus a few smart agents are not included because the agents are too dominant even when there were only a few of them.

3.2.4 Mixes of Neural Net Agents and Visual Agents

In this part I substitute the dumb agents with our visual agents. Using homogenous pools of both agent types as benchmarks, I wish to create a mix that outperforms both. I start out with two mixes, 4-12 and 12-4. Using the same number of agents, I then adjust the *maxShares* parameter, which determines the size of the trade placed by an agent each time it sees a market. In scaling the trade sizes I hope to get the best of both worlds – maintaining the extreme predictions when smart agents are in the market, but still allowing small traders to determine the price when the smart agents are absent.

I compare 4 mixes: 12-4, 4-12, and the same two mixes at scaled bet sizes (5000 for smart agents, 500 for visual agents).

3.2.5 Mixes of Neural Net Agents and Deterministic Agents

In this part I study mixes of smart agents and deterministic agents initialized with human data. Similar to the previous part, I study 4 mixes: 12 smart plus 4 deterministic, 4 smart plus 12 deterministic, 12-4 with scaled price (5000-500), and 4-12 with scaled price.

Chapter 4

Results

In this study as well as in full-scale CP experiments, we use root mean squared error (rmse) as an important measure of performance. We define the error of a market as the difference between the closing price and the price corresponding to the outcome of the event. In particular, a pass play is represented by a price of 100 cents ($1) and a run by 0 cents. For example, a market that closes at 70 cents has error of 30 if the outcome is pass and 70 if it's run. We take the square of each of the errors, and get the square root value of their average.

Due to the *biasForAction* parameter, a market with neural net agents can sometimes close with no trading activity. This give an error of 50 regardless of the outcome. For each experiment, I keep track of the number of markets in which the agent pool made trades. I also keep track of the number of markets in which the market price was in the same direction as the actual outcome. While these are less meaningful as measures of an agent pool's overall performance, they provide us with a way to "characterize" a pool – risk-averse but accurate versus risk-seeking but inaccurate, for example.

4.1 Neural Net Training Data

	KUvVT	MISSvTTU	WASvNYG	ALL
# played	19	18	20	8
# correct	11	13	15	7
mean rmse	54.59	43.48	43.15	42.92

Table 4.1: Trials on Different Training Sets

Table 4.1 shows the run results of agent pools trained with the different training sets. The "# played" row shows the number of plays in which trading activity was recorded, out of 20 plays total. The "# correct" shows the number of plays in which the market prediction was in the correct direction (> 50 for pass, < 50 for run). I ran multiple trials with each agent pool to obtain the average rmse. While small deviations exist between the ending prices of each experiment, agents trained with the same data entered the same markets each time and traded in the same direction.

By the mean rmse, I find that KUvVT is the worst (SD=0.27) training set. With an average rmse over 50, it actually fares worse as a predictor than an agent that performs no trades at all. A one-way ANOVA test on the data indicates a significant difference between the sets (one-way ANOVA, $F_{3,8}$=188.68, $p < 0.0001$). However, post-hoc Tukey tests show no significant difference between the rmse's of ALL (SD=0.6), WASvNYG (SD=0.3) and MISSvTTU (SD=1.25).

While statistical tests produced no clear winner between the sets, other statistics tell us that these sets of agents behaved very differently in experiment. The pool trained with MISSvTTU played in 18 of the 20 plays, correctly predicting 13 of them. WASvNYG traded in every single play, correctly predicting 15 of them. It does not push prices to extremes as often, only going within 15 of an extreme in 4 plays.

ALL only participated in 8 plays, getting 7 of them correct. Of these 7 correct plays, the pool posts ending prices that are close to the correct price extreme in 6. The low rmse's of these plays compensates for the plays not participated in. The worst performer KUvVT traded in almost every play, but only predicted about half of them correctly. Worst yet, it posted many extreme prices – 5 of its wrong predictions have market prices within 15 of the wrong price extreme.

4.2 Neural Net Agent Bias for Action

	BIAS-50	BIAS-40	BIAS-30	BIAS-20	BIAS-10
# played	20	17	8	7	5
# correct	14	12	7	6	5
mean rmse	42.56	43.27	42.92	43.86	44.54

Table 4.2: Trials on Different Values of *biasForAction*

Table 4.2 shows the results on the runs with 5 agent pools having respective *biasForAction* values of 50, 40, 30, 20, and 10. The agent pools use the ALL training set. The data set for BIAS-30 is identical to the one for ALL in the previous section. As one would expect, as we move to smaller bias for action the accuracy of the pool increases. At bias = 10, the pool of agents predict 5 out of 5 plays correctly. All sets predicted these 5 plays correctly with strong closing prices. All sets with bias >= 20 also shared 2 more predictions with strong closing prices, with 1 hit and 1 miss. These correspond to the high confidence plays.

The rmse seems to have a slightly upward trending effect as the bias decreased. But at BIAS-40 the rmse rises up slightly higher than both of its neighbors BIAS-30 and BIAS-50. A closer look at the closing market prices reveals the intricate effect *biasForAction* values can have on market performance. Of the markets that are only played in by BIAS-40 and BIAS-50, two ended up predicting opposite outcomes. The ending prices in both directions are weak, grouping around the middle value of 50.

The individual neural nets have a small degree of variation in their output, even though they are trained on the same data set. In the BIAS-50 case, the ending price showed the majority opinion from all the predictions. However, in BIAS-40 agents whose prediction falls within [40, 60] do not trade. So a few number of traders whose prediction falls outside of this range can single-handedly decide the outcome, which may be opposite the majority's opinion.

There was a significant effect of the value of *biasForAction* on mean rmse for the 5 conditions (one-way ANOVA, $F_{4,10}$=7.38, p=0.004912). Post hoc comparisons using the Tukey HSD test indicated only significant differences between BIAS-50 (SD=0.29) and BIAS-10 (SD=0.4), and BIAS-30 (SD=0.6) and BIAS-10. BIAS-20 (SD= 0.42) and BIAS-40 (SD=0.66) are not significantly different from each other, or from any other sets.

We have seen that in a pool of homogenous agents, *biasForAction* value alone does not improve overall performance. However, it is a useful parameter that allows us to represent agents with identical underlying neural nets as traders of very different behavior. In the next section, we simulate a "smart" agent by setting the *biasForAction* at 30. In sections 4 and 5, we also experiment with increasing their trading power to be larger than that of the less informed agents.

4.3 Mixes of Neural Net Agents and Rule-Based Agents

	16-0	14-2	12-4	8-8	0-16
# played	8	20	20	20	20
# correct	7	13	13	13	13
mean rmse	42.96	43.46	45.01	47.12	48.95

Table 4.3: "Smart" versus "Dumbest" Agents

	16-0	14-2	12-4	8-8	0-16
# played	8	20	20	20	20
# correct	7	14	14	13	12
mean rmse	42.96	43.35	45.21	47.01	48.17

Table 4.4: "Smart" versus "Dumb" Agents

Table 4.3 and Table 4.4 show the results from runs mixing different numbers of neural net ("Smart") agents versus rule-based ("Dumb"/"Dumbest") agents. The groups are labeled by the numbers of each type of agent, in the pattern "Smart-Dumb". The earlier results from a homogenous pool of 16 smart agents are included in the first column for comparison. The last column of each table shows the performance of the dumb agents on their own. Since the rules used by these agents are deterministic, and their trade prices sufficiently large, they always push the market price to their target. Therefore these results are deterministic. Figure 4-1 shows the play-by-play error measurements of the three homogenous mixes. Notice that smart agents finish many markets with squared error = 2500, which is equal to a ending price of 50. These are the markets where they abstained from trading. A squared error measure of > 2500 indicates a prediction in the wrong direction.

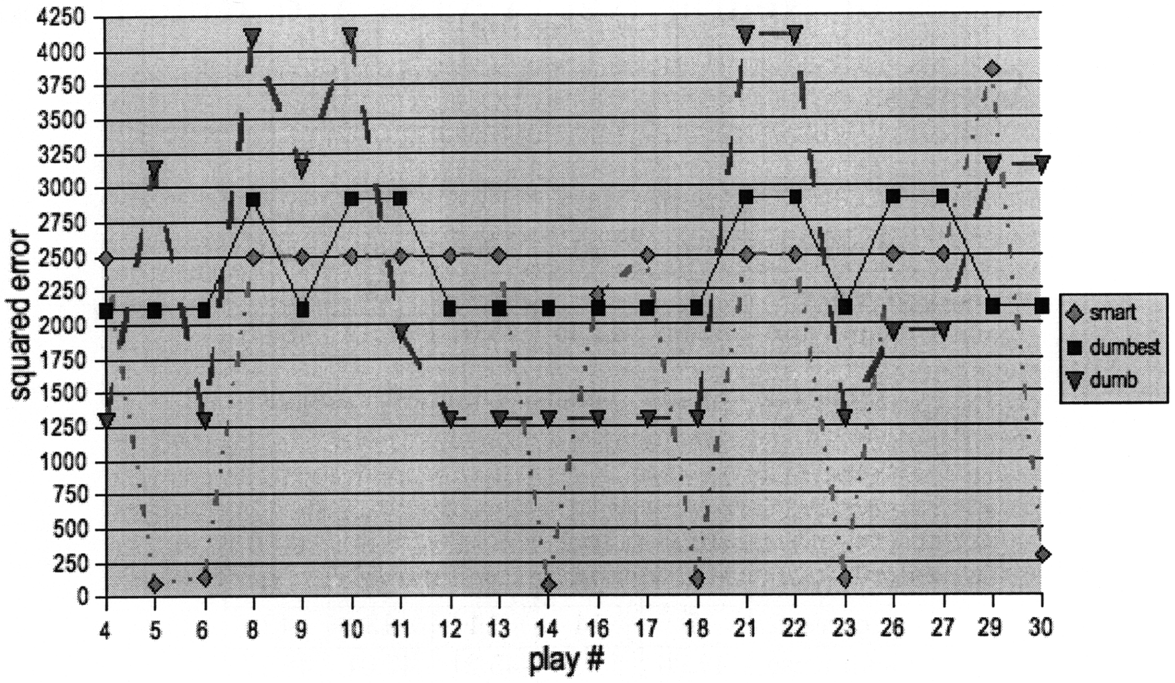


Figure 4-1: Play-by-Play Performance by Homogenous Agent Pools

The "Dumbest" agents predicted all 20 plays at 54%, and was correct in 13 of them. These include all 7 of the correct predictions by the smart agents. When these dumb agents are present, the error measurements of these 7 markets suffered since their predictions brought down the market closing prices from the correct extreme. The dumb agents contributed 6 correct predictions on their own, but with weak prices. It also added to the error with 7 wrong calls. Overall, we see a clear trend of their negative effect: the control group with no dumb agents performed the best. The rmse grew with the number of dumb agents in the market.

The mixes with "Dumb" agents fared only a little bit better. On their own, the dumb agents predict 12 correct plays with a rmse of 48.17. Like the homogenous dumbest-agent mix, they push the market price to their target in every market. They predict all pass calls at 64% and all run calls at 44%. In this set, the smart agents differ with the dumb ones in prediction direction in two markets. With 12 and 14 smart agents in the mix they were able to "correct" the direction of the prediction, but not with only 8 smart agents. With both rule-based agents, we are only able to achieve a mix whose performance approaches that of a homogenous pool of smart agents. There was no significant difference between the means for the 16-0 mix (M=42.96, SD=0.6), the 14-2 mix with dumb agents (M=43.35, SD=0.5), and the 14-2 mix with dumbest agents (M=43.46, SD=0.5) at the $p<0.5$ level (one-way ANOVA, $F_{2,6}=0.85$, p=0.473131).

4.4 Mixes of Neural Net Agent and Visual Agents

agent mix	16-0	12-4	4-12	0-16
# played	8	12	12	10
# correct	7	10	10	8
mean rmse	42.96	41.49	42.49	45.24

Table 4.5: Smart-Visual Agent Mixes

trade sizes	1000	5000-500		1000
agent mix	16-0	12-4	4-12	0-16
# played	8	12	12	10
# correct	7	9	10	8
mean rmse	42.96	44.87	41.66	45.24

Table 4.6: Smart-Visual Agent Mixes, Scaled Trade Sizes

In this section I compare mixes of smart versus visual agents. Besides simply adjusting the ratio of smart to visual agents, I also try using different *maxShares* values for each trader to control their representation in the pool. Table 4.5 shows the results of the unscaled mixes (*maxShares* = 1000 for both agent types). Table 4.6 shows the result of the mixes where smart agents have *maxShares* = 5000, and visual agents *maxShares* = 500. Values from unscaled homogenous pools are included in both tables for comparison.

From the statistics about 0-16, we can see that the visual agents are a very different pool than our dumb agents from previous sections. Instead of trading in a most plays with a low hit rate, this pool trades only in 10 of the 20 plays with 8 correct calls. It has a rmse of 45.24, compared at 48.17 for the dumb agents. Like the dumb agents, however, this pool is also deterministic since they produce the same target prices every time.

One-way ANOVA shows significant difference between the homogenous smart agents (SD=0.6) and the four mixes ($F_{4,10}$=58.54, p<0.0001). In particular, two mixes were better than 16-0 : 12-4 (SD=0.15) and 4-12-scaled (SD=0.11). The difference between these two mixes is nonsignificant. The mix 12-4-scaled (SD=0.24) is significantly worse than 16-0. Finally, the mix 4-12 (SD-0.13) shows no significant difference from 16-0. However, this mix is in turn not found to be significantly different from 4-12-scaled.

Results from comparisons against the homogenous visual pool are less convoluted. There is a significant difference between 0-16 and the mixes ($F_{4,10}$=436.4, p<0.0001). Three mixes are better than 0-16: 12-4, 4-12-scaled, and 4-12. There is still no significant difference between the two best mixes 12-4 and 4-12-scaled. Our worst mix 12-4-scaled shows nonsignificant difference from 0-16.

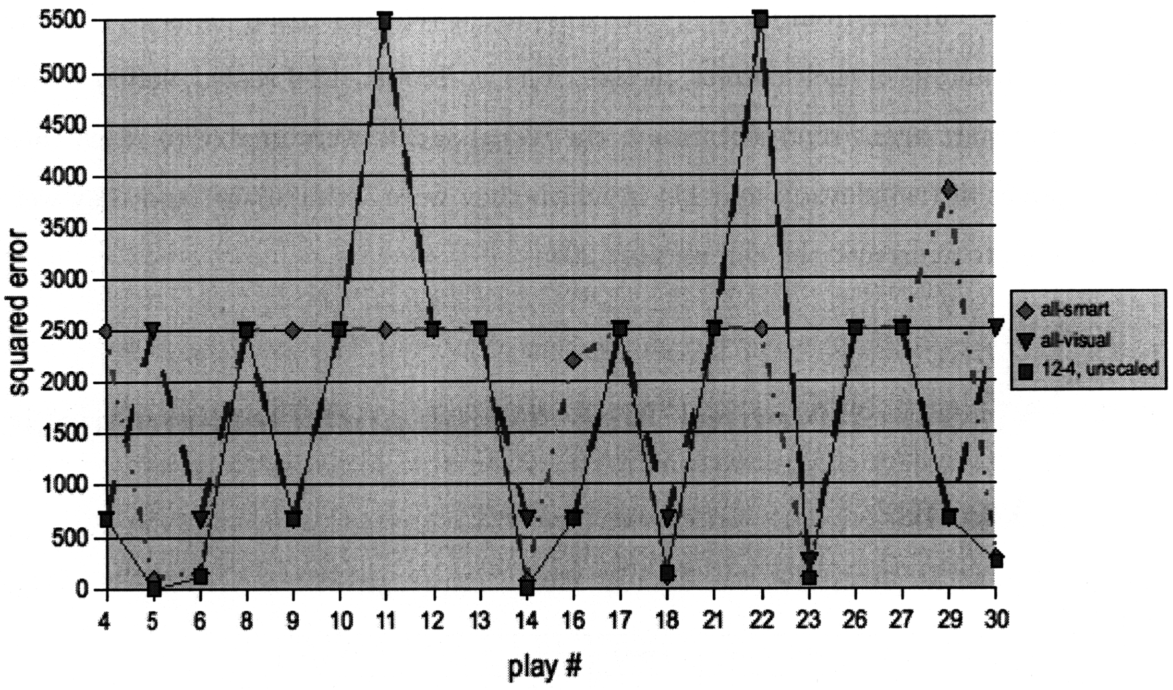


Figure 4-2: Play-by-Play Performance, Smart-Visual Mix

Figure 4-2 shows the play-by-play performance of our best mix, 12-4, against both homogenous pools. We can see that the two homogenous pools trade in a somewhat similar manner. They both withhold judgement on low confidence plays, resulting in a lot of markets closing at price = 50 (squared error = 2500). But once they decide to enter the market they often post aggressive prices, indicated by the large peaks and troughs in this graph. The smart agents suffer one bad call at play 29. The visual traders have two at plays 11 and 22, with much larger penalty since their prices were more extreme in the wrong direction.

The 12-4 mix was successful in following the better of the two homogenous pools on almost every market. There was enough representation from both trader types to push the prices to their targets in the absence of the other type. The mix suffers from the large penalty of the wrong guesses on plays 11 and 22, where the smart agents did not trade. However, it was able to choose the correct call on play 29, where the visual agents indicated a stronger confidence than the smart agents.

The other successful mix, 4-12-scaled, has much of the same play-by-play behavior as 12-4. 4-12 suffered from not enough smart agent representation, causing it to not take full advantage of their correct guesses. The worst mix, 12-4-scaled, suffered from too much smart agent representation – the visual agents were unable to reach their target prices in two plays (9 and 11) in which they were acting alone, and they were also unable to "correct" the call on play 29.

4.5 Mixes of Neural Net Agent and Deterministic Agents

agent mix	16-0	12-4	4-12	0-16
# played	8	20	20	20
# correct	7	18	18	17
mean rmse	42.96	37.17	41.56	41.71

Table 4.7: Smart-Deterministic Agent Mixes

trade sizes	1000	5000-500		1000
agent mix	16-0	12-4	4-12	0-16
# played	8	20	20	20
# correct	7	17	18	17
mean rmse	42.96	39.08	37.53	41.71

Table 4.8: Smart-Deterministic Agent Mixes, Scaled Trade Sizes

Tables 4.7 and 4.8 show the results of Smart-Deterministic mixes. Table 4.7 shows the mixes where agents have even trading power, and Table 4.8 where the trade sizes are scaled (5000 to 500).

The first interesting thing to note is that the deterministic agents do remarkably well. A statistical test indicates significant difference between homogenous pools of smart and deterministic agents at the $p<0.05$ level (two-tailed $t=3.6084$, $df=4$, $p=0.0226$). In light of this result, it is less justifiable for us to continue calling the neural net agents the "smart" ones; but I will preserve the term for consistency. Like the rule-based agents, the homogenous pool of deterministic agents have deterministic end prices since they always push the price to target. This pool trades in all 20 plays with 17 correct predictions. Although it is very adept at picking the right trade direction, this pool is also very conservative in its betting. Only 9 of its target prices fall outside of the range [40, 60], and only 3 outside of [30, 70]. This behavior directly influences the choice of mixes I examine in this section – the deterministic agents don't need a lot of representation in a mixed pool to reach their low targets. On the other hand, the neural net agents need to have a lot of influence to be able to reach their extreme target prices.

One-way ANOVA test indicates a significant difference between the mixes and the homogenous smart agent pool ($F_{4,10}$=22.87, p<0.0001). In particular, three mixes outperform 16-0 (SD=0.6): 12-4 (SD=0.11), 12-4-scaled (SD=1.91), and 4-12-scaled (SD=0.33). The other mix, 4-12 (SD=0.17), shows no significant difference from the homogenous smart agent pool. Another ANOVA indicates the same results against the homogenous deterministic pool ($F_{4,10}$=18.39, p=0.000133): significantly better performance from 12-4, 12-4-scaled, 4-12-scaled, and no significant difference from 4-12. Both tests indicate no significant pairwise difference between the three best mixes.

Looking at the plays in more detail helps us understand the difference between the mixes' performances. The two homogenous pools differ in the direction of their predictions in two plays, out of 8 total that smart agents trade in. In play 5, the smart agents have the right prediction; in play 29, the deterministic agents get it right. This is the only play that the smart agents miss. In play 29 and two more plays, the smart agents' end price is not more extreme than the deterministic agents' price. In the 5 other plays the smart agents post much more extreme prices than the deterministic ones, achieving smaller rmse's.

The mix that did not outperform either homogenous pools, 4-12, suffered the same problem as the unscaled mixed pools with dumb agents. The aggressive prices from the smart agents were reduced down to more conservative ones, which negatively affected the rmse's on smart agents' correct calls. The mix followed the deterministic agents on both critical plays, getting play 5 wrong and play 29 correct.

The other three mixes overcame 4-12's deficiency by giving a bigger share of the market to the smart agents. It turns out simply adjusting the numbers of agents to 12-4 was enough for the smart agents to push the prices to extremes, and correct the outcome of play 5. As it happens, the smart agents don't post a very aggressive price in play 29, so the deterministic agents are still able to override them to get the correct prediction. Scaling the trade sizes to 5000 and 500 on 4-12 achieves similar results. In 12-4-scaled, though, we start the see the negative effect of giving the smart agents overwhelming influence. They are now so powerful that they override the deterministic agents on play 29, resulting in one less correct prediction than 12-4 and 4-12-scaled.

Figure 4-3 shows the performance of 4-12-scaled against both homogenous pools. This pool nicely captures the "best of both worlds", almost always following the better of the two pools and avoiding both of the high-penalty spikes. It only underperformed the better of the two homogenous pools in one play (play 11), where the smart agents did better by not trading.

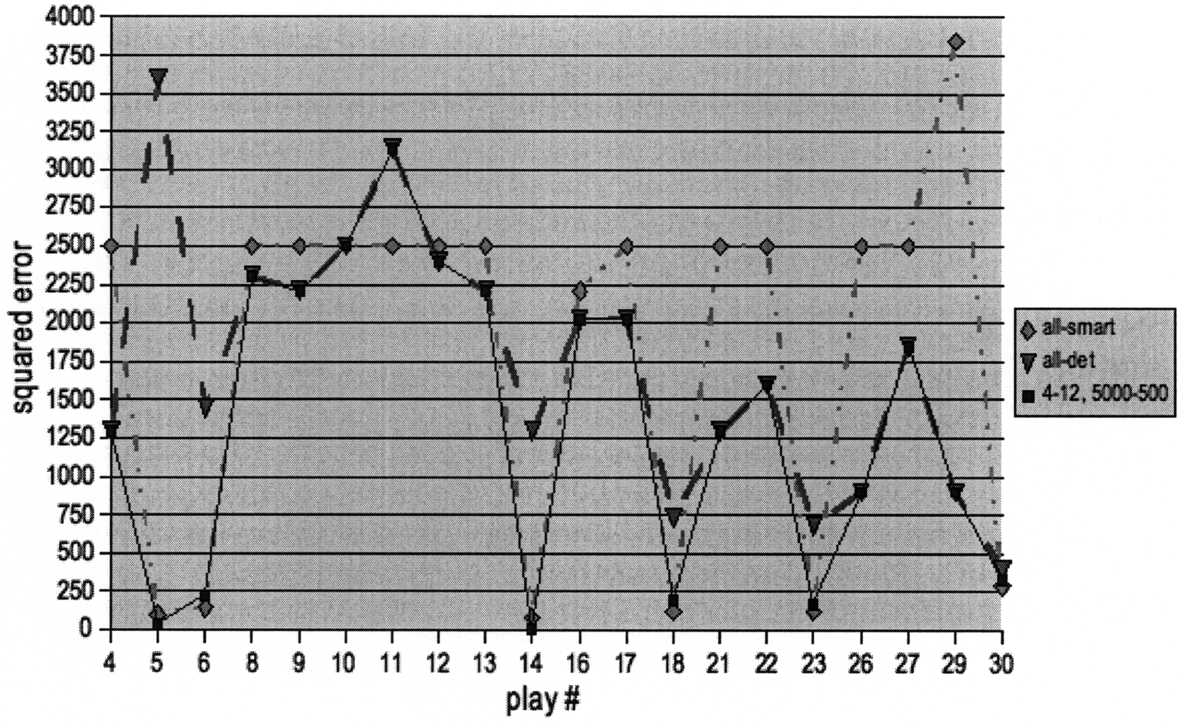


Figure 4-3: Play-by-Play Performance, Smart-Deterministic Mix

Chapter 5

Discussion

In the first two sets of experiments, I tested the effect of training sets and *biasForAction* values on neural net agent performance. I find that the neural net became more discriminating after being trained with three sets of data instead of just one. Keeping the value *biasForAction* small also keeps the agents accurate, with the tradeoff that it will enter fewer markets. Rather than having a "best" value of *biasForAction*, it is more useful to think of the parameter as a way to control agent behavior. Agents with small *biasForAction* values could do well in increasing the overall accuracy of a market with many other traders. Agents with large *biasForAction* values can be useful in providing liquidity to the market.

From the results of the first two sections, I configured a "smart" neural net agent to use in trading alongside our dumb agents. My homogenous pool of smart agents entered few markets, but tended to push the price close to either extreme in those markets. It was highly accurate in predicting these plays but abstained from other plays. Even though agent mixes showed the ability to improve market price in individual cases, overall their success was limited. In most cases, they underperformed the homogenous smart agent pool. This is largely due to the limitations of the dumb traders. They were simply so inaccurate that it wasn't worth introducing them.

The mixes with visual agents fared much better. The homogenous pool of visual traders is a better predictor than the dumb agents, but worse than the smart agents.

However, with proper representation of both agent types in the mix, mixed pools outdid either types acting alone. The two agent types have similar trading styles, abstaining from a lot of plays but trading aggressively once they enter a market. Together they traded in more markets and with more successes than either one alone.

Mixes with deterministic agents were also successful. On their own, they are slightly more accurate than the smart agents. I was able to find mixed pools of smart and deterministic agents that outperform both homogenous pools. The two agent types were good complements, the deterministic one trading often and modestly and the neural net agents trading infrequently but aggressively. Successful mixes combined the correct predictions made by both agent types, and preserved the extreme prices posted by the smart agents.

In examining the closing prices of our mixes, it would appear that the market simply gave us a weighted average of the traders' predictions. This seems to be less than what we expect by using a market setup. What makes a market more than an averaging mechanism?

I used simplistic rule-based agents to simulate the presence of less knowledgeable individuals in a prediction market. While the rules used by these agents may reasonably be used by a human trader, our model lacks a significant component of a real market – that of market interaction. Both the sophisticated and simplistic agents act oblivious to outside information. This includes trading activity in the market, results of past markets, and knowledge about other traders in the market. But considering these factors is essential in having the market price reach an equilibrium that represents the collective belief of its participants. Without market-aware traders, the market will remain inefficient. Thus, the existing model has yet to fully realize the potential of the market mechanism to generate good predictions.

Chapter 6

Conclusion and Future Work

In the first part of this study I explored the effects of different training sets and risk affinity on the neural net agents. With the results I configured a "smart" agent, which I then used to trade alongside three different simple agents. The simple agents are meant to approximate the human agents in a lab study. The "dumb" agent uses a simple rule based on the number of yards left to first down in a play. The visual agent uses formation information. The deterministic agent trades in accordance with the aggregate trading history of human subjects in our lab studies.

By exploring different combinations of smart versus simple agents, I showed that it is possible for mixes of agents to outperform either types alone. I found pools of both smart-visual and smart-deterministic agents that achieved better results than the corresponding homogenous pools. This result is consistent with the larger goal of the collective prediction project, which is to show that humans and computer agents combined in a prediction market can do better than either alone.

In this study, I relied on trial and error to find good combinations of smart and simple agents. The best mixes differed with the simple agents types, which have different trading behaviors. The obvious next question, then, is how do we achieve the ideal mix in every prediction market?

One approach to achieving the ideal mix is to have a "market referee" who assigns weights to the different agents' predictions. I effectively used this approach in this study, dictating the influence each agent type should have by adjusting their ratios and trade sizes. As we have seen, however, crafting the ideal mix requires in-depth knowledge of the strengths and weakness of each agent type, plus careful fine tuning to get the best results. This may be infeasible for a real market setting, with many more trader types and complex interactions that are absent from our simple model.

We have reasons to hope that market mechanisms will automatically keep the price efficient. A simple agent that trades the same trade repeatedly may quickly lose all its balance to smarter traders in the market and die out. Traders that successfully combine their prediction and trading strategy will gain more influence over time and bring the market to its efficient price. To capture this effect in my simulation framework, we would need to improve our agents to be more market-aware.

It is useful to think of every trader as having two components – a "predictor", which generates the trader's own belief about the outcome of the play, and a "controller", which decides how the trader should use his/her prediction. Our neural net agent has a form of controller in the *biasForAction* parameter, which prevents the agent from trading in plays in which it has low confidence. On the other hand, our simpler agents lack this component. Once they produce a prediction, they will repeatedly enter the trade regardless of what may be going on in the market. The controller can base its rules on many factors, such as the current market price, risk affinity, budget constraints, and success in past trades. Current market price and success in past trades both affect a trader's self-confidence, which is often a key parameter in shaping its trading strategy. An agent's self-confidence can also be affected by its perception of other traders in the market.

Every trader should have a clearly defined goal of maximizing its own profit. To that end, a trader may decide to act based solely on its belief of the event probability. However, there should be other traders present who take other factors into consideration. For example, an agent may speculate about the price movement of the market rather than the outcome of the event, and attempt to capitalize on that. Another agent may adjust its trade size according to the current market price. The presence of a variety of traders will promote richer interaction in the market.

Another key advantage of using a market is that it allows traders to continually update their beliefs. In lab experiments, we observe big learning effects from our human traders. Some become better at predicting the outcome of the plays as they build a history of plays that happened; others develop more effective strategies to buy a prediction low and sell high to make a profit. Including learning effects in our simple agents, which are meant to approximate human subjects, has potential for substantially improving their performance. This can also apply to our smart agents. A typical feature of a neural net is the ability to evolve over time. Our neural net agent learns during the training period, but does not retain information learned during the actual experiment. Enabling learning could add to the neural net's accuracy over the course of an experiment as it adapts to the specific game.

The experiments in this study provide a first step in observing the effects of mixing agents in a prediction market. To obtain a more effective prediction market, we need to promote more interaction amongst the traders, and enhance our trader model to include more behaviors that occur in real markets.

Appendix A

Collective Prediction System Parameters

This section lists the configurable parameters of the Collective Prediction software. Default values reflect the settings used in the experiments of this study.

A.1 System Parameters

Name	Function	Default Value
RUN_LOCAL	Run Zocalo from a local installation, or from a remote server.	true
RUN_ WINDOWS	Whether the software is running on Windows operating system. This affects the application used to play the videos.	false
AGENTS_ ONLY	Run experiment with only computer agents.	true
MANUAL_ ADVANCE	Whether an experiment administrator needs to manually control when to end a market and advance to the next. If false, the rounds advance automatically.	false
autoRunPause	The amount of trade time (in milliseconds) per round.	21000
skipVideo	Skip showing videos. Used for agent-only sessions.	true
defaultGame	The default experiment game.	WVUvOK
MM_FUNDS	The amount of money available to the market maker per market.	10000
AGENT_ START_ BALANCE	The amount of money available to each trader (human or computer) at the start of the experiment.	50000
firstRealRound	First real round of data collection, after training plays for human subjects. The system automatically resets all account balances. Does not apply when AGENTS_ONLY is true.	4
agentsFirstHalf	In a lab study, agents only participate in either the first or second 10 plays. This controls when the agents start/stop. Does not apply when AGENTS_ONLY is true.	true

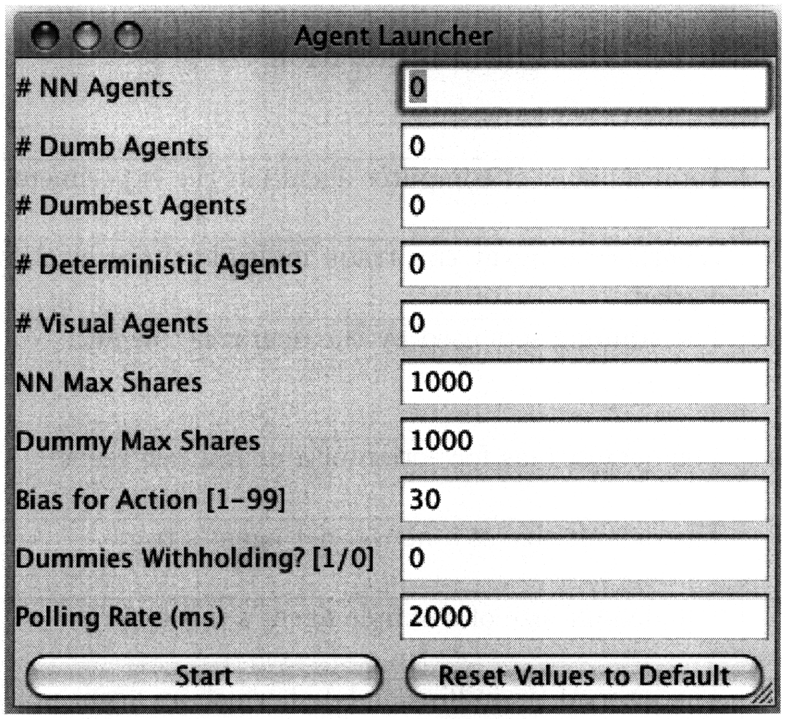
Table A.1: System Parameters

A.2 Agent parameters

Name	Function	Default Value
NUM_ AGENTS	Total number of computer agents in the experiment.	16
REPEAT_ TRADING	Whether an agent can trade multiple times in a single market.	true
trainingGames	The training sets used by the neural net agent.	{WASvNYG, MISSvTTU, KUvVT}
global_ bias-ForAction	The default bias for action of a neural net agent.	30
global_ maxShares	The default size of a neural net agent's trade.	1000
dummy_max_ shares	The default size of a simple agent's trade.	1000
pollingRate	The interval (in milliseconds) at which an agent polls for the newest market.	2000 (agents-only) / 20000 (humans)

Table A.2: Agent Parameters

A.3 Experiment Parameters



# NN Agents	0
# Dumb Agents	0
# Dumbest Agents	0
# Deterministic Agents	0
# Visual Agents	0
NN Max Shares	1000
Dummy Max Shares	1000
Bias for Action [1-99]	30
Dummies Withholding? [1/0]	0
Polling Rate (ms)	2000

Figure A-1: Agent Launcher Window

For the agent-only experiments in this study, I control most of the independent variables through the small graphical interface shown in Figure A-1. The agent launcher allows an experimenter to quickly set up mixes of different agent types. I also include the parameters *biasForAction*, *maxShares* and *dummy _max _shares* for easy access. The values default to the settings in the table above. There is an additional parameter, "Dummies Withholding", which I did not use in this experiment. When this value is set to true, the simple agents will withhold their trades if the market price is in the same direction as its prediction but closer to the extreme. We achieve a similar effect in this paper by outnumbering the simple agents heavily.

Appendix B

Code Excerpts: Computer Agents

Much of the core code for Collective Prediction is excerpted in Carver's thesis. Here I excerpt the code for our three simple agents, which were developed for this study. The rule-based agent using *yardsToGoal* is based on an earlier version, and the other two are new.

B.1 Rule-Based Agent Using *yardsToGoal*

```
package edu.mit.cci.predecon.agents;

/**
* Dummy Agent
* @author Jason
*
* Also referred to as Rule-based agent, it makes a very simple guess about football
* plays.
*/
public class DummyAgent extends TradingAgents {

    /*
    * We implement three types of "dumbness" here. (1, 2 deterministic, 3 has
    * randomness)
    * 1. "Dumbest": agent bets the pass baseline (54%) everytime.
    * 2. "Dumb" : agent looks at yards to first down: if < 5 bet 44%, else bet 64%
    * 3."Dumb-Random": similar to "Dumb" , but uses randomness in the passBaseline
    * and the deviation.
```

```
*
* market_aware only implemented for the market maker case.
*/

private static final double passBaseline = 54.0;
private static final double base_deviation = 10;

private final boolean USE_RANDOMNESS;
private final boolean USE_DEVIATION;
private final boolean MARKET_AWARE;
private double expectPass;

public DummyAgent(String username, boolean useRandomness, boolean useDevia-
tion, boolean marketAware, int maxShares) {
    super(username, maxShares);
    USE_RANDOMNESS = useRandomness;
    USE_DEVIATION = useDeviation;
    expectPass = USE_RANDOMNESS ?
    passBaseline + (Math.random()-0.5)*20 : passBaseline;
    MARKET_AWARE = marketAware;
}


protected void gotNewMarket(String newMarket, FootballPlay play) {
    double myEstimate = expectPass;
    double deviation = USE_DEVIATION ? (USE_RANDOMNESS ?
            Math.random()*10 : base_deviation) : 0;
    int ytf = play.getYards();
    if(ytf == -1){
        Util.out("yards to 1st was unusable, agent aborting trade");
        return ;
    }
    else if(ytf <= 5)
        myEstimate -= deviation;
    else
        myEstimate += deviation;

    //get price
    double price = getMarketPrice(newMarket, PASS);

    if(price == -1) {    //book orders
        //intended for markets with no Market Maker
        int spreadAroundEstimate = 10 - (int)deviation;
        placeMarketTrade(newMarket, BUY, PASS,
            (int)Math.round(myEstimate - spreadAroundEstimate),maxShares);
        placeMarketTrade(newMarket, SELL, PASS,
            (int)Math.round(myEstimate + spreadAroundEstimate),maxShares);
        }

        else {    //market maker
            mmTrade(newMarket, (int)Math.round(price), myEstimate);
        }
    }


    /*
    * trade with the market maker.
    */
    protected void mmTrade(String newMarket, int price, double myEstimate) {
        int myPrice = (int)Math.round(myEstimate);
        boolean sameDirection = (price > 50) ? myPrice > 50 : myPrice < 50;
        boolean withhold = MARKET_AWARE && price != 50 &&
        sameDirection && ((price > 50) ? myPrice < price : myPrice > price);
        if (!withhold) {
            //buy up to / sell down to desired price
            TradeType tradeType = price < myPrice ? BUY : SELL;
            placeMarketTrade(newMarket, tradeType, PASS, myPrice, maxShares);
        }
    }
}
```

B.2 Rule-Based Agents Using Visual Data

```
package edu.mit.cci.predecon.agents;

/**
* Visual Agent trades according to formation information.
* If sees shotgun formation with no running backs near the quarter back,
* considers a strong visual cue for pass.
* SG formation with 1 running back - weak cue for pass.
* Other formations - do nothing.
*
* @author wendyc
*
*/
public class VisualAgent extends DummyAgent {

    private static final double visual_deviation_weak = 20;
    private static final double visual_deviation_strong = 30;
    public VisualAgent(String username, boolean marketAware, int maxShares) {
        super(username, false, false, marketAware, maxShares);
    }

    @Override
    protected void gotNewMarket(String newMarket, FootballPlay play) {
        double myEstimate = getPassBaseline();
        boolean willTrade = false;
        switch (play.getFormation()) {
        case shotgun_bare :
            willTrade = true;
            myEstimate += visual_deviation_strong;
            break;
        case shotgun_one_rb :
            willTrade = true;
            myEstimate += visual_deviation_weak;
            break;
        default :
            break;
        }
        if (willTrade) {
            //get price
            double price = getMarketPrice(newMarket, PASS);

            if (price == -1) {
                //no market maker; not implemented
                return;
            }
            mmTrade(newMarket, (int)Math.round(price), myEstimate);
        }
    }
}
```

B.3 Deterministic Agents Using Predefined Prices

```
package edu.mit.cci.predecon.agents;

/**
* @author wendyc
*
* A deterministic agent that trades based on the input prices we feed it.
* Can withhold trades using the same rules as parent.
*/
```

```
public class DeterminAgent extends DummyAgent {

    public DeterminAgent(String username, boolean marketAware, int maxShares) {
        super(username, false, false, marketAware, maxShares);
    }

    @Override
    protected void gotNewMarket(String newMarket, FootballPlay play) {
        int myPrice = agentPrices.get(play.getId()-1);
        double price = getMarketPrice(newMarket, PASS);
        if (price == -1) {
            //no market maker; not implemented
            return;
        }
        mmTrade(newMarket, (int)Math.round(price), myPrice);
    }
}
```

Appendix C

Experiment Software Setup Instructions

1. Setup Collective Prediction

   - Import Eclipse project Prediction Economy Agents
     svn : //portend.org/trunk/Prediction_Economy_Agents
   - Settings are in ExpConstants.java. To run off of local build, set RUN_LOCAL = true. See list of params for other settings.
   - Server addresses are in RPCClient.java. If running locally, make sure the address Zocalo_Local is is set to the proper IP.
     (e.g. "*http* : //18.95.6.69 : 8180/*RPC*2")
   - Get football video and put in folder "football" in project folder.

2. Install Zocalo

   - Download the latest copy of Zocalo. Currently at
     http : //portend.org/zocalo – PM – Caching – bin.tar
   - Unarchive and move resulting folder "zocalo" to convenient location.
   - Setup is done through editing the file */etc/zocalo.conf*. Make the following changes:
   - Add "RPC.server.enable = true" to enable RPC. This lets our system talk to zocalo.
   - Change "initial.user.funds" to 10000.
   - Mail info. Fill in all relevant fields including host, user, port, secure, and password.
   - Important: If the line "useCostLimit = true" is not in the file, add it. This configures the Zocalo UI to look the way we want it to for the experiment.
   - Important: After a new install, make sure the UI looks right. A normal user should NOT be able to see trading history and closed markets. If this is not the case, contact Chris Hibbert for the right install package.

- Consult *INSTALL* and */etc/CONFIGURE* for additional info.

3. Setup Play Accounts

   - Go to *localhost* : 30000/*Welcome.jsp* to create the following accounts: *NewsServer*, *zoc*1 - *zoc*20 (humans), *aa*0 - *aa*9 (agents).
   - Fund the market maker with extra cash. From RPCClient.java, uncomment line in main class for granting cash and run once.

4. Create Eclipse Run Targets

   - Create Java Application targets for two classes: AgentLauncher and NewsServer.

5. Start Zocalo

   - From Zocalo install folder, start database by running ./bin/startDB
   - Then start Zocalo by running ./bin/zocalo.sh
   - To stop Zocalo properly, run ./bin/shutdown.sh

6. Start Collective Prediction

   - Run target *NewsServer* to start experiment. Run *AgentLauncher* to launch agents when desired. Be sure to give neural net agents enough time to initialize.

Bibliography

[1] Steven Gjerstad, 2004. *Risk Aversion, Beliefs, and Prediction Market Equilibrium*, Microeconomics 0411002, EconWPA.

[2] Hollywood Stock Exchange, LLC (January 23, 2008). *Hollywood Stock Exchange Traders Hit 80% of Oscar Nominations for the 80th Annual Academy Awards.* Press release. Retrieved on 2008-08-08.

[3] Hibbert, C. *Zocalo: An Open-Source Platform for Deploying Prediction Markets.* CommerceNet Labs, CN-TR-05-02, February 2005.

[4] Carver, J. *Architecture of a Prediction Economy.* May 2008.

[5] Marrone, P. *Joone - Java Object Oriented Neural Engine.* August 1 2008. `http://www.jooneworld.com/`.