applied sciences

MDPI

Article

Woc-Bots: An Agent-Based Approach to Decision-Making

Sean Grimes and David E. Breen *

Department of Computer Science, Drexel University, Philadelphia, PA 19104, USA; spg63@drexel.edu
* Correspondence: david@cs.drexel.edu

Received: 30 September 2019; Accepted: 29 October 2019; Published: date

Abstract: We present a flexible, robust approach to predictive decision-making using simple, modular agents (WoC-Bots) that interact with each other socially and share information about the features they are trained on. Our agents form a knowledge-diverse crowd, allowing us to use Wisdom of the Crowd (WoC) theories to aggregate their opinions and come to a collective conclusion. Compared to traditional multi-layer perceptron (MLP) networks, WoC-Bots can be trained more quickly, more easily incorporate new features, and make it easier to determine why the network gives the prediction that it does. We compare our predictive accuracy with MLP networks to show that WoC-Bots can attain similar results when predicting the box office success of Hollywood movies, while requiring significantly less training time.

Keywords: classification; prediction; multi-agent; wisdom-of-crowds; Hollywood; feature-extension; collective-intelligence; swarm

1. Introduction

We, humans, want to predict the future; disease outbreak and risk factors, business success, economics, and many more applications can benefit from better forecasting. Researchers have developed many tools to help us make predictions, with artificial neural networks (ANNs) being a current popular choice. ANNs can be used for classification, allowing us to take, for example, a series of features about an upcoming movie and determine, with fairly high accuracy, if the movie will be successful. ANNs, however, typically require a large amount of training data and compute time, and they do not generalize well to other topics. We cannot use an ANN trained on Hollywood movies to help us determine if some sports team will win an upcoming game or where the next 'hot spot' in an epidemic will be; they are inherently inflexible. Recent efforts are improving their flexibility by adding to their basic design, as seen in transfer learning [1], however this increases complexity and compute/data requirements while further obfuscating the internal workings of an ANN, making it *even more* difficult to answer the "why" about some outputted classification [2].

Prediction markets (PM) are designed to determine the probability of a future event taking place. Well-designed PMs encourage agents, human or computer-based, to contribute information to the market through trading shares and incentivizing correct, truthful information sharing, and then aggregate the information from individual agents into a collective knowledge [3]. PMs work because the aggregate knowledge of the group will generally be more precise and complete than the knowledge that any individual within the group holds. However, participants are expected to be well-informed in the topic being predicted, which as of current technology, requires human participants [4]. Additionally, computer agent-based PMs are difficult and programmer-intensive to create. Othman said on computer-based agents, "agent-based modeling of the real world is necessarily dubious. Attempting to model the rich tapestry of human behavior within economic structures—both the outstandingly bad and the terrifically complex—is a futile task."[5] Even if it were possible to model the complexity of

human knowledge and decision-making within some narrow topic it would be extremely difficult to generalize across topics. We can consider simpler agents that don't attempt to model human interaction. These agents have been studied in simple, academic, scenarios with some success when their behavior is limited in possible actions and their opponents are not adversarial [6,7]. However, work done by Othman and Sandholm [8] has shown that simply changing the order in which the agents participate in the market can drastically impact the outcome of the market, indicating that "markets may fail to do any meaningful belief aggregation."

An alternative to PMs is Wisdom of the Crowd (WoC). WoC takes the approach that the opinion of a large, diverse group will be more accurate than any individual opinion within the group given a sufficiently competent aggregation mechanism [9]. The classic example that demonstrates this is guessing how many jelly beans are in a jar at a county fair. Typically no individual is consistently able to get close to the correct amount, but the aggregate opinion of the group is generally very close to the correct number of jelly beans. WoC doesn't expect or require expert knowledge, Scott Page said "the squared error of the collective prediction equals the average squared error minus the predictive diversity" [10]. This means the more diverse the crowd, the smaller the predictive error.

In this paper we present a robust, computer-agent-based approach to making predictions about the success of Hollywood movies that can be easily distributed across multiple computational nodes [11]. We take a WoC approach, using simple agents (WoC-Bots) without expert knowledge that are trained with different, small, subsets of features that describe the movies. This initially gives us a group of agents with a diverse and independent set of knowledge. The agents interact with one another socially, sharing some knowledge, determining the trust they have in other agents and the confidence they have in their own opinion, and changing their opinion given enough evidence. Following this interaction an overall conclusion is drawn from the crowd using a trust and performance-based aggregation mechanism. Our system was compared with traditional multilayer perceptron (MLP) networks trained with the full set of features available to the agents, as well as a subset of the most highly correlated features. We show that WoC-Bots are able to achieve more accurate classification results, with reduced training time and resistance to feature drop-out.

2. Methods & Design

The test scenario for this research involves predicting if a movie will be a success. Success was defined as the reported revenue being greater than 2× the reported budget for the movie. It is difficult to determine exactly what revenue is considered a success, and it differs on a movie-by-movie basis. However, advertising and promotion budgets are generally less than the production budget, which indicates studios should start to see some positive cash flow if a movie makes 2× the production budget [12]. Additionally, defining success as we did split our data, discussed more in Section 2.1, roughly equally between success (47.5%) and failure (52.5%).

2.1. Datasets & Libraries

We primarily used two datasets for this work:

1. The Movie Database (TMDb)(https://www.kaggle.com/tmdb/tmdb-movie-metadata/)
2. MovieLens (ML) https://www.kaggle.com/grouplens/movielens-20m-dataset/) [13]

The MovieLens dataset provides information for more than 27,000 movies, while the TMDb dataset includes 5000 movies. Only movies found in both datasets, with complete information for all features used for classification, were considered. The features used for classification are listed in Table 1. A note about the genre feature; only the first two listed genres were considered for each movie in classifiers that used the genre feature(s). Each genre was assigned a unique numeric ID. We were left with 4722 possible movies for testing and training, however 1023 movies appeared to contain incorrect information, e.g., negative values for movie budget or movie revenue; these movies were removed from our testing and training subsets. We used both datasets to help reduce sparse areas in the data

for less popular and older movies. The data was split into two subsets, testing and training, with 2959 randomly selected examples used for training and 740 examples used for testing. The training subset was randomly selected from the full dataset at the start of each simulation, with the remaining examples being used for testing.

Table 1. Features available for classification.

Features	Description
`budget`	given to all agents, reported budget for movie
`tmdb_popularity`	dynamic variable from TMDb API attempting to represent interest in movie
`revenue`	used for sanity checks, reported revenue
`runtime`	unreliable metric for success without including genre information
`tmdb_vote_average`	average score from TMDb, can be combined with ML average
`tmdb_vote_count`	total votes for a movie from TMDb, can be combined with ML count
`ml_vote_average`	average score from ML, can be combined with TMDb average
`ml_vote_count`	total votes for a movie from ML, can be combined with TMDb count
`ml_tmdb_genres`	combined genre information from TMDb & ML; first 2 listed genres used
`vote_average`	combined `tmdb_vote_average` and `ml_vote_average`
`vote_count`	combined `tmdb_vote_count` and `ml_vote_count`

The data was transformed in the following ways:

- Movies were matched between the two datasets based on ID, using "movieId" and "tmdbId" values provided in the ML dataset.
- The ML dataset used a 0–5 rating system while the TMDb dataset used a 0–10 rating system, the ML ratings were multiplied by 2.
- Only overlapping genres from each dataset were considered; e.g., if, for Toy Story, the ML dataset lists it as "action, animation, family" and the TMDb dataset lists Toy Story as "family, adventure, animation", the movie was considered to fall into only the "animation" and "family" genres.

Sample training and testing CSV files encompassing the transformed data can be found at the following url https://data.mendeley.com/datasets/gj66mt4s4j/2, while the code required to reproduce the results presented in this article can be found at https://github.com/spg63/MDPIApplSciCodeRepo. The code will be made available upon request to Sean Grimes or David E. Breen. Eclipse Deeplearning4j (DL4J) (https://deeplearning4j.org/) [14] is "an open-source, distributed deep-learning project in Java and Scala spearheaded by the people at Skymind (https://skymind.ai/), a San Francisco-based business intelligence and enterprise software firm." DL4J (versions 0.9.1 and 1.0.0-beta3) was used as a neural network library, providing the core multilayer perceptron classifier used by each agent (discussed in more detail in Section 2.2.1). Additionally, DL4J was used to build and test the larger MLP classifiers that were compared with our agent-based approach. All non-DL4J library code was written in Kotlin (https://kotlinlang.org/) (versions 1.3.20 - 1.3.41), running on the Java Virtual Machine (JVM) (https://www.java.com/en/download/) (versions 1.8.0_151 - 1.8.0_211). All feature, agent history data, and trained agents were stored in various databases, using SQLite3 (https://www.sqlite.org/index.html) as the database engine.

2.2. Agent Design

Agents are designed to be modular, presenting an interface that includes different algorithms for all aspects of their behavior. Agents are responsible for coordinating and managing the following functions:

- Classifier: MLP classifier with a single hidden layer
- Classifier configuration: shape, depth, activation and optimization algorithms
- Initialization algorithm: How the agents are initialized within the interaction space
- Movement algorithm: How the agents move within the interaction space
- Interaction algorithm: How (if) an agent interacts with other agents
- Scoring algorithm: How an agent reaches the conclusion it does; a combination of the internal classifier and information learned while interacting with other agents

2.2.1. Classifier

Each agent contains a very small, very simple MLP classifier. All agents were configured with similar classifiers, each containing 2–4 input nodes (one for each feature), a single hidden layer containing `numInputNodes * 2` number of nodes, and an output layer with two output nodes, one for each of the two output classes, "success" and "failure". All classifiers used the DL4J implementation of the Adam updater [15] (learning rate), softmax activation function [16], and traditional stochastic gradient descent for optimization.

Each agent's classifier was given a single feature in common with all other agents, the movie `budget`. Other features were spread across multiple agents, occasionally in pairs (e.g., `budget` & `vote_count` & `vote_average`), but more frequently a single feature in addition to `budget`. Classification performance was the determining characteristic in assigning features to agents, with very low (<50% accuracy) performing combinations dropped during early testing in favor of decreased computational complexity.

2.2.2. Agent Initialization & Movement

All agents currently participate and interact within a centralized 'interaction arena' (`Arena`) managed by a central controller responsible for registering agents, confirming that their initial location is valid, and validating each movement. All agents must be initialized within the bounds of the arena and into an empty space. All movements must be within the bounds of the arena and there can be at most two agents occupying any space within the arena. Interactions between three or more agents at the same location are not currently supported. Centralized control is currently being used to ease implementation, however it is not a requirement. Agents are capable of validating location and movement on their own, or if required, having some number n other agents confirm all positioning as valid in a decentralized manner. The `Arena` can take any 3D shape comprised of rectangles allowing for arrangements from a simple 4×4 square to something more complex, with multiple rooms, floors, and restricted movement between each, e.g., a simulated building.

Agents are currently responsible for maintaining a history of their movements within the arena, a history of interactions, and a history of how each interaction affected their internal belief. Historical information for each agent is stored in memory during each iteration and dumped to individual SQLite tables for long-term storage.

Agents are initialized with an `InitializationAlgorithm` and a `MovementAlgorithm` which implement simple interfaces, `init()` and `move()`, respectively. `init()` has a single goal, initialize the agent in the arena in an empty space. Initialization can be random, or account for complexities like location of other agents, placing agents with similar features close together, localizing similar information, or spreading them out to facilitate transmission of information between dissimilar agents. Localizing similar information may allow a group of similar agents to come to an optimal conclusion, whereas spreading similar agents out may allow the best performing agents to convince others of their

opinions [17]. The `InitializationAlgorithm` used in this work randomly initialized agents within a rectangular arena.

`move()` also has a simple goal, move the agent within the arena. `move()` can be as simple or complex as necessary; randomly selecting a space within the arena and 'teleport' the agent to the new space, or it can require the agent to move towards some target location or another agent. Agents interact when two agents move onto the same space. The `MovementAlgorithm` used for this work randomly moved agents in a "Manhattan-like" fashion, allowing each agent to move one step north, south, east, or west, within the bounds of the arena.

2.2.3. Interaction & Scoring

The `InteractionAlgorithm` and `ScoringAlgorithm` are both designed to be modular, with the `InteractionAlgorithm` being responsible for deciding with whom an agent should interact, truthfulness, and trust updating. The `InteractionAlgorithm` is required to implement three functions, `shouldInteract()` which determines if the agent is interested in interacting with another agent, `truth()` which determines if the agent should be truthful with another agent, and `updateTrust()` which tries to update the other agent's trust score. `updateTrust()` is allowed to be a NO-OP function when it is not desirable to update other agents' trust scores. This work assumes all interactions are acceptable, doesn't limit repeat interactions, and requires all interactions to be truthful.

The `ScoringAlgorithm` determines how interactions update an agent's internal belief state. Agents are initialized with specific internal values, referenced in Table 2, that are (in part) updated during each interaction. Many of these values are made available to other agents during interaction, allowing each agent to determine how certain another agent is, what that agent's initial classification values were, and how much influence it will allow the agent to have over its current belief.

Table 2. Internal scoring variables.

Variable	Description
`current_prediction`	true if prediction for movie is success
`trust_score`	initialized to classifier precision, updated by other agents
`features`	a list of features used by the agent's classifier
`prior_performance`	long-term history of agent performance, varied between 0.7 and 1.3 where 1.0 is average performance
`certainty`	an average of classifier accuracy and precision, multiplied by `prior_performance`, bounded by 0.5 and 1.5
`eval_accuracy`	initial classification accuracy
`eval_precision`	initial classification precision
`eval_recall`	initial classification recall
`confidence`	biased value based on an average of accuracy, precision, and recall favoring whichever is deemed most important

Similar to the previous algorithm interfaces, the `ScoringAlgorithm` used by each agent allows the scoring to be implemented in a way most appropriate to the given problem. The algorithm is required to implement a single function, `updatePrediction` which updates the current binary prediction based on information from the most recent interaction. The `ScoringAlgorithm` used in this study works as follows: initially the agent (agent a) determines how willing it is to accept information from another agent (agent b), this is a function of a's current `certainty`, where $a_{certainty}$ represents a's current `certainty` and $a_{acceptance}$ represents a's willingness to accept information from b

$$a_{acceptance} = 1.0 - a_{certainty} \quad (1)$$

Agent a then determines how much influence b should have ($b_{influence}$).

$$b_{influence} = b_{confidence} * a_{acceptance} * b_{trustCertainty}, \tag{2}$$

where $b_{confidence}$ represents b's `confidence` and $b_{trustCertainty}$ represents b's `trust_score` $*$ `certainty`. b's influence is modified based on its prior performance where $b_{priorPerf}$ represents b's prior performance, a value between 0.7 and 1.3, as noted in Table 2, and $b_{correctedInfluence}$ represents this modified value,

$$b_{correctedInfluence} = b_{priorPerf} * b_{influence}. \tag{3}$$

$b_{correctedInfluence}$ is multiplied by -1 if b's opinion (success or failure) differs from a's opinion. a's new certainty, $a_{certainty}$ is now calculated by Equation (4), where a's certainty is increased if both a and b have the same belief and is diminished if they disagree,

$$a_{certainty} = a_{certainty} + b_{correctedInfluence}. \tag{4}$$

a now checks if it should flip its opinion, which it does if $a_{certainty}$ is less than 0.50. Finally, a updates its certainty if its opinion changed,

$$a_{certainty} = 1.0 - a_{certainty}. \tag{5}$$

2.3. Opinion Aggregation

Effective opinion aggregation is an open question with many different possible approaches [18]. This research hopes to contribute more to this area in the future. We implement a voting system, where each agent receives a maximum of 100 possible votes for their preferred outcome, success or failure. We considered three methods of vote aggregation. The first and simplest method gives equal weight to each agent regardless of performance, the Unweighted Mean Model (UWM) [19]. The second method gives each agent votes based on prior accuracy, where an 80% accuracy rate would result in 80 votes, similar to the Weighted Voter Model presented in [20]. Agents are initially allowed 50 votes each until an accuracy for prior performance can be determined. The third method we used is similar to the second, however it also takes into account the trust score that other agents are allowed to modify, giving more granular control over how much influence an agent has on the aggregate opinion. Total votes for agent a is represented by $a_{totalVotes}$, where $a_{priorAccuracy}$ represents a's prior accuracy and a_{trust} represents a's trust score,

$$a_{totalVotes} = \left(\left(a_{priorAccuracy} + a_{trust}\right)/2\right) * 100. \tag{6}$$

During an interaction the agent, a, is allowed to modify another agent's, b, trust score (b_{trust}) based on how agent b has performed in the past, if agent a and b are in agreement ($doAgree$), and if prior information that a has received from b was correct. Agent a will check its interaction history, look for any interactions with b to determine what percent of interactions gave advice that was correct ($b_{percCorrect}$). If there are no prior interactions the trust score will not be modified. The trust score can be modified a maximum of 5% during each interaction.

$$b_{trust} = b_{trust} + 0.05\left(b_{percCorrect} * b_{priorPerf}\right) * doAgree \tag{7}$$

A high-level overview of the training, interaction, and voting process can be seen in Figure 1. The internal MLP for each agent is trained using available training data, the agents are presented with a binary question, they are initialized in an arena where they move and interact for some number of steps (based on time or total interactions). Agents are then assigned some number of votes based on the system described in Section 2.3 and they vote at the end of the interaction period.

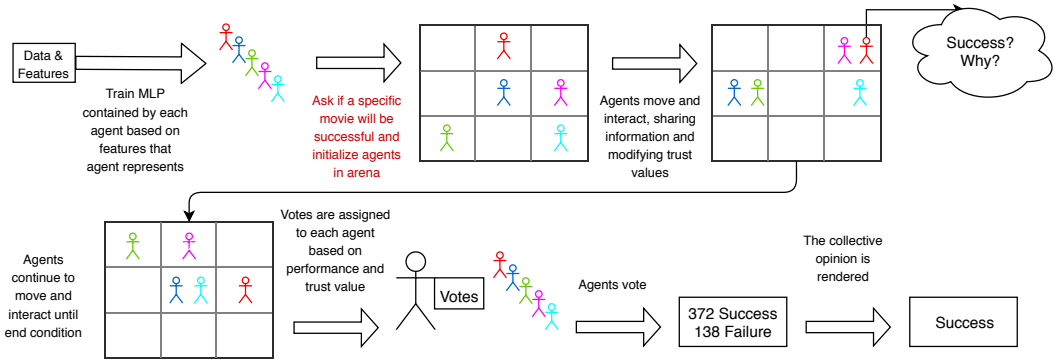


Figure 1. Agent training, initialization, movement, interaction, and voting.

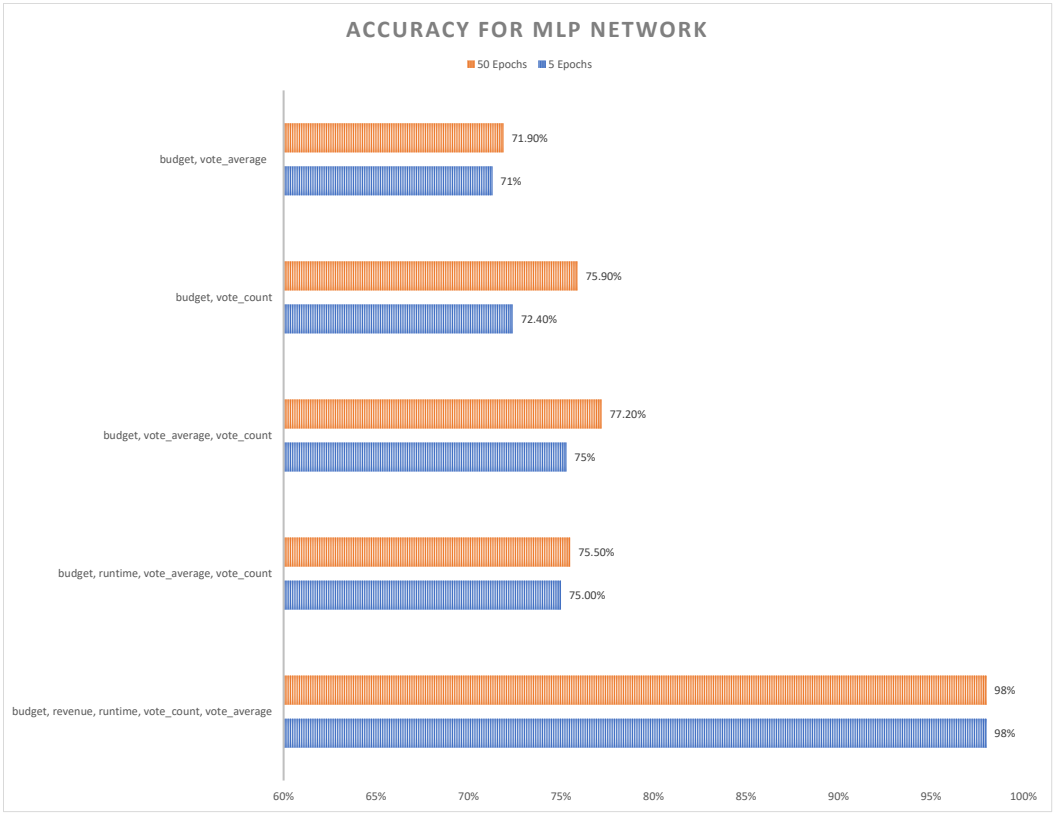
3. Results

We compared the results from our social, agent-based approach to the results produced by multiple configurations of a traditional, monolithic MLP developed in DL4J. All agent classifiers were trained on 4× Nvidia GTX 1070 GPUs using Cuda (versions 10.0–10.1 update 2) through the DL4J library. Table 3 shows the configuration and accuracy under different training conditions for each of the 10 agents. All agent classifiers were trained in parallel, taking an average of 2.8 s for 5 epochs and 22 s for 50 epochs; there was no accuracy improvement beyond 50 epochs. Once trained, the agents can participate in any decision-making configuration without re-training their classifiers. The best performing agent classifier was the `budget`, `vote_count`, `vote_average` agent, with the most important feature being `budget`, followed by `vote_count`. The worst performing agent was the `budget`, `runtime` agent.

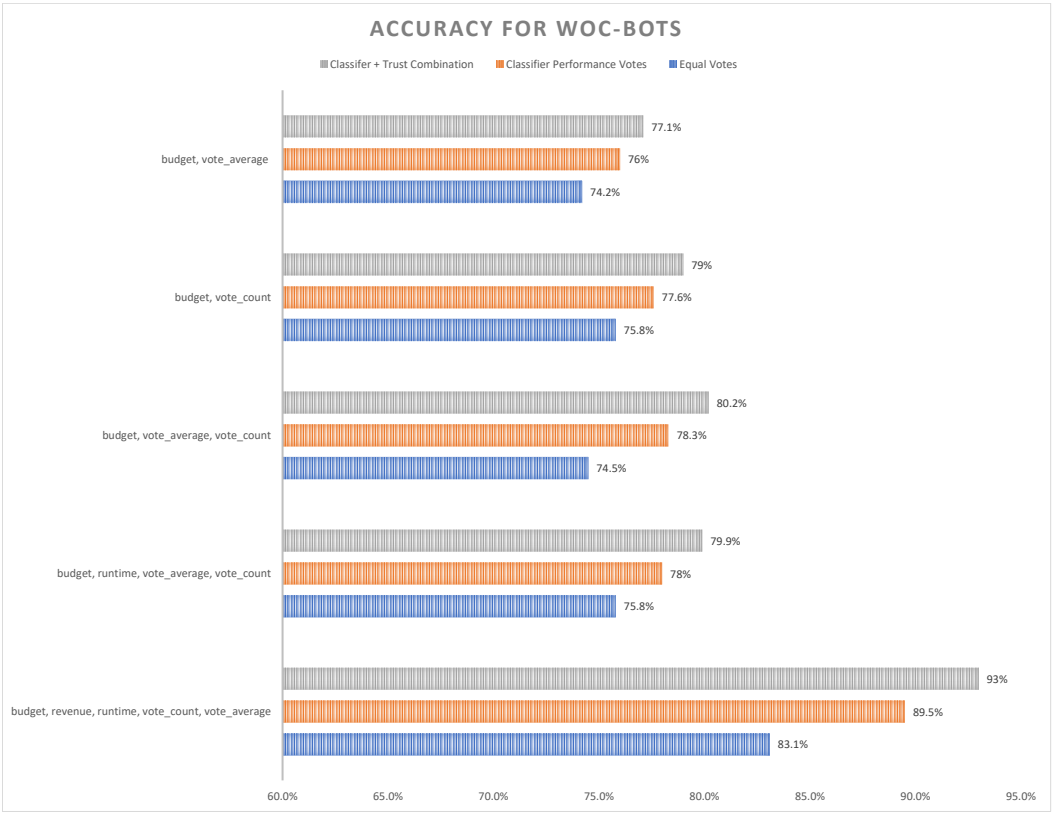
Table 3. Agent Classifier Accuracy for 5 and 50 epochs.

Features	5 Epochs	50 Epochs
`budget`, `revenue`	98%	100%
`budget`, `vote_average`, `vote_count`	77.2%	77.6%
`budget`, `tmdb_popularity`, `vote_average`, `vote_count`	75.4%	75.7%
`budget`, `vote_count`	75.7%	75.5%
`budget`, `tmdb_popularity`, `tmdb_vote_average`, `tmdb_vote_count`	72.8%	74.9%
`budget`, `tmdb_vote_count`, `ml_vote_count`	73%	73.4%
`budget`, `ml_vote_average`, `ml_vote_count`	62.2%	64.1%
`budget`, `ml_vote_count`	60.3%	61.9%
`budget`, `tmdb_vote_average`	60.9%	61.4%
`budget`, `runtime`	53.9%	56.4%
Average (`budget`, `revenue` agent removed)	67.93%	68.99%

We tested 10 MLP networks, with a variety of feature sets, and with one containing the final `revenue`. Figure 2a shows the accuracy for five classifier configurations when trained for 5 and 50 epochs. The figure shows the change in classifier accuracy when various features have been removed as inputs into the network.



(**a**) Accuracy for single hidden layer MLP classifier. The y-axis shows which feature were included.



(**b**) Accuracy for agent interaction. The y-axis shows features included.

Figure 2. Comparison of MLP and Woc-Bots performance.

All MLP classifiers were trained individually with an average training time of 2.6 s for 5 epochs and 21.2 s for 50 epochs. Comparing training times with the social agents, training 10 MLP classifiers in parallel took 1 min and 1 s (3 min and 32 s if computed sequentially) vs. 22 s to train 10 agents for 50 epochs. It should be noted that inference is slower using WoC-Bots; it takes an average of 260 milliseconds to test 740 examples using an MLP classifier incorporating all the features listed in Table 1, while it takes the WoC-Bots, encompassing the same feature set, an average of 13.4 s to test the same 740 examples. But, once trained, the agents can be reconfigured to compute new prediction results for different feature sets, without requiring retraining, unlike a monolithic MLP.

Accuracy results from five configurations of our social, agent-based prediction system can be found in Figure 2b. We show results for three aggregation mechanisms after 50 epochs of training: (1) unweighted equal voting, (2) votes assigned based on initial classifier performance, and (3) votes assigned based on classifier performance and agent trust (described in Section 2.3), with method (3) consistently out-performing methods (1) and (2). Method (1) is represented by the blue bar, method (2) by the orange bar, and method (3) the grey bar. We tested similar configurations across our agents and MLP networks. Data from Movielens and TMDb was combined, as described in Section 2.1, with no agent receiving information from only one source.

Similar to Figure 2a, Figure 2b's labels show which features were included in the interaction. Feature distribution across agents was optimized for accuracy, within the limits of available features. Five agents participated in each interaction, with the `budget`, `vote_average` and `budget`, `vote_count` interactions being comprised of five copies of the same agent.

WoC-Bots out-performed the MLP classifier in all cases except where final revenue was included as a feature, indicating that our aggregation method does not give enough weight to an agent with exceptionally good performance. We tested removing a highly correlated (http://ibomalkoc.com/movies-dataset/) feature, `vote_count`, which caused a performance decline in both the MLP and social agents, with the MLP network accuracy declining 4% compared to a decline of 1.9% in WoC-Bots, indicating our agents are more resistant to feature drop-out. We also tested removing an unimportant feature, `runtime` which showed a 1.7% performance increase in the MLP network and only a 0.3% increase for WoC-Bots, indicating poorly performing agents have little impact on other agents during the interaction period and receive few votes during opinion aggregation. Statistical analysis confirms that `runtime` is not highly correlated in both the TMDb dataset and an ensemble dataset combining the Movielens and TMDb data, as used in this article [21,22].

Figure 3 shows the performance of MLP networks and WoC-Bots as features are systematically added. The results presented in this figure are produced via the classifier performance & trust aggregation mechanism. The agents are configured to allow for maximum agent participation without duplicating agents in any simulation testing more than two features. Four copies of an agent, representing `budget` and `vote_count`, participated in the first simulation. Four agents participated in the `budget`, `vote_count`, `popularity` simulation, eleven agents participated in each of the following four-feature simulation. Twenty-six agents participated in the `budget`, `vote_count`, `vote_average`, `runtime`, `popularity` simulation with one agent receiving five features, five agents receiving four features, 10 agents receiving three features, and 10 agents receiving two features.

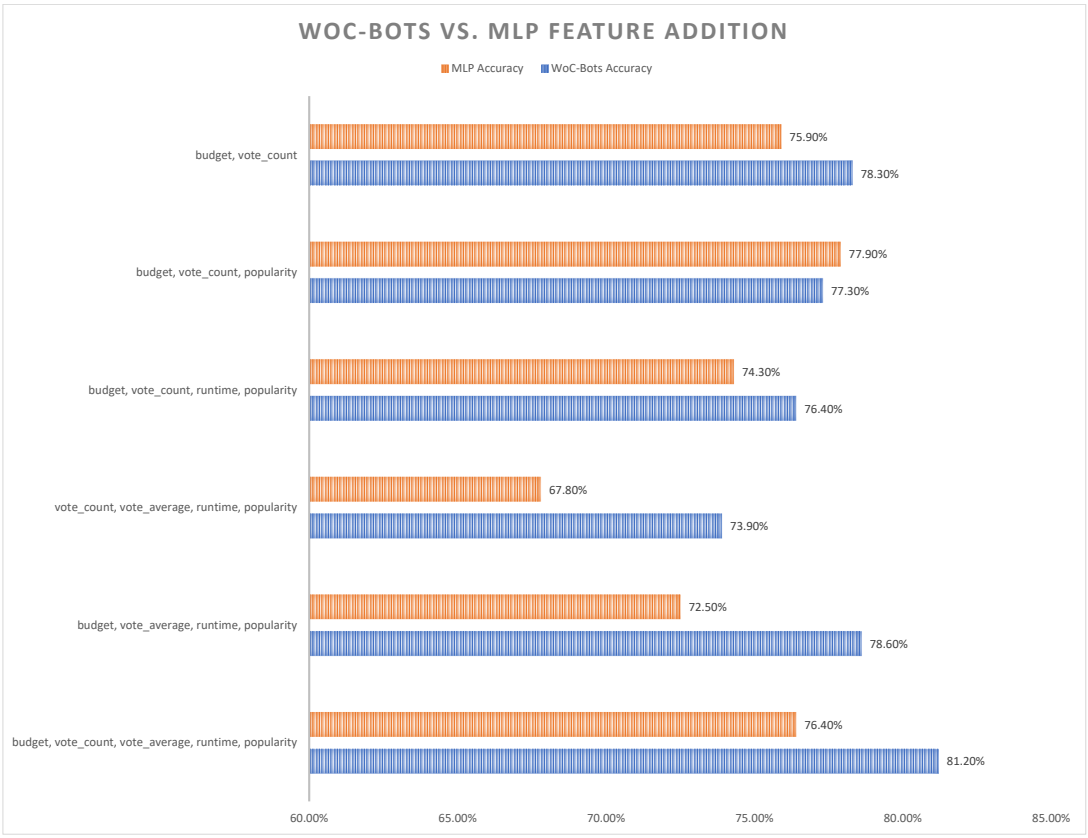


Figure 3. Accuracy of MLP vs WoC-Bots w/ Max agent configuration while adding features.

In five out of six simulations WoC-Bots out-performed the MLP network, and significantly out-performed the MLP network when the most important feature, `budget`, was removed. WoC-Bots performed best when all features were available, and when the maximum number of unique agents were participating in the simulation. The MLP network performed best when the three most highly correlated features were the only features being considered. This performance difference indicates WoC-Bots are able to gather additional information from features that are less correlated with `revenue` without a net negative impact to their accuracy from the additional feature noise.

Figure 4 shows the accuracy for training epochs 1–50 for an MLP network and WoC-Bots. The network was configured with five features, `budget`, `vote_count`, `vote_average`, `runtime`, and `popularity`. Five agents participated in the simulation with four agents receiving two features and one agent receiving five features. Each two-featured agent received `budget` as a feature and one other feature from the list of features. No agent was duplicated. We choose this agent and feature configuration to make as fair and direct comparison with an MLP network as possible despite WoC-Bots performing better when more agents participate in the simulation, as seen in the `budget`, `vote_count`, `vote_average`, `runtime`, `popularity` simulation in Figure 3 where 26 agents were allowed to participate. WoC-Bots are out-performed in this configuration, slightly, by the MLP network when trained for more than 40 epochs, however they are able to more quickly integrate information compared to the MLP network, reaching an optimum (76.3%) at 20 epochs vs. the MLP optimum (76.8%) at 40 epochs.

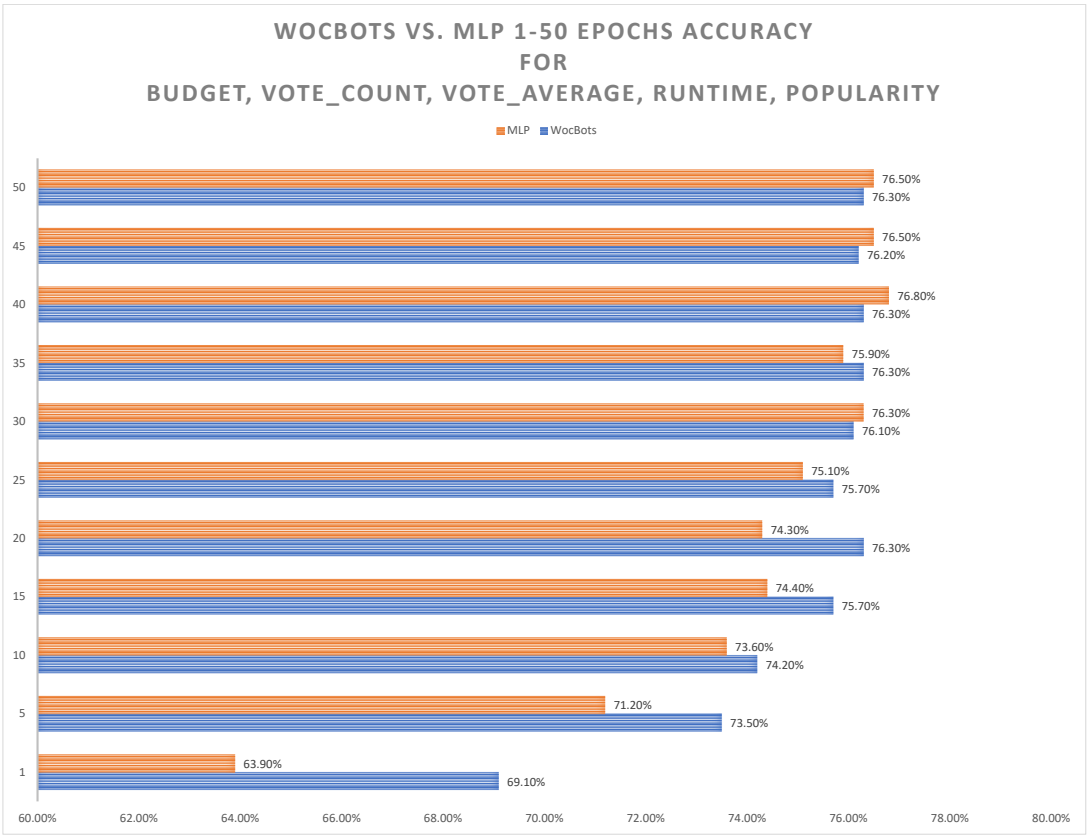


Figure 4. Accuracy of MLP and WoC-Bots for five features over 1–50 epochs.

4. Discussion

Our system can easily include new features by creating a new agent to represent those features, allowing the agent to be added to the next "interaction" without the need to re-train a network or employ complex, dynamically expandable networks found in [2] when incorporating new data. Additionally, this design allows us to quickly test the impact of removing features or testing various combinations of features without the time-consuming re-training step required when changing features in a full MLP network. This allows us to easily removing features, like `runtime`, to test how they impact the final prediction.

WoC performance, or a computer-agent-based version of it, depends on two attributes, a diverse and independent crowd and an aggregation mechanism that assigns appropriate weights to individuals within the crowd to reach the correct collective decision [23]. We also know from Othman [5] that it is not reasonable to develop a computer agent that accurately represents the intricate and diverse knowledge that humans have. Therefore, we need to find the right balance between independently thinking agents within a crowd and information sharing to better represent the diverse knowledge of human agents.

5. Conclusions & Future Work

We have demonstrated a robust, flexible alternative to traditional ANN methods for making predictions about specific future events. Our implementation takes ideas from prediction markets, wisdom of crowds, and multi-agent systems to use simple, modular agents in a social setting to answer binary questions. Our results show that we can attain similar results to that of a multilayer perceptron when classifying Hollywood movies, while requiring less training time and offering more flexibility and prediction options. Further, our system is robust, demonstrating only a 1.9% loss in accuracy

when losing the `vote_count` feature versus a 4% loss in accuracy when the same feature was removed from the MLP network.

We have three main areas to focus on for improvement in the future. Recent work on deep neural networks is starting to explain "why" we get certain output. However, there is still a long way to go before we have the ability to easily answer this question [24]. Our system offers a framework, using a multi-agent approach, that should allow us to answer "why" more easily; at the core of each agent is a *very* simple, single hidden layer MLP. Agents track all interactions, how those interactions affect their internal belief, and how they change the trust value of other agents. Given the state of the system during a simulation, the internal belief and trust scores of each agent, as well as each agent's interaction history, we can follow the history of each agent, starting with its initial belief post-classification, through each interaction, allowing us to see when and why an agent's belief changed (or stayed the same).

The two other areas we will address in future work are (1) the interaction, movement, and initialization algorithms, allowing us to change and optimize the distribution and flow of information and (2) the aggregation mechanism. We will use theories from swarm intelligence [25] to better aggregate the information that each agent possesses in a manner that better extracts information from the **correct** agents while limiting the impact that **incorrect** agents have on the collective opinion. Unanimous A.I. (https://unanimous.ai/) has a unique, swarm-based aggregation method that is capable of arriving at a collective answer. Unanimous A.I. maintains a "human-in-the-loop" approach [26], where their 'swarm' is comprised of humans, answering binary and non-binary questions by working together to move a virtual puck to the collective answer [27]. We prefer a computer-agent-based approach that allows for new agents to be created as needed to answer questions as they come up. Our future work will focus on implementing a swarm-based algorithm to produce an "emergent prediction" from a group of relatively simple, modular agents.

Author Contributions: Conceptualization, S.G.; methodology, S.G. and D.B.; software, S.G.; investigation, S.G. and D.B.; resources, S.G. and D.B.; data curation, S.G.; writing–original draft preparation, S.G.; writing–review and editing, S.G. and D.B.; supervision, D.B.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

MDPI	Multidisciplinary Digital Publishing Institute
DOAJ	Directory of open access journals
WoC	Wisdom of the Crowd
MLP	Multi-layer Perceptron
ANN	Artificial Neural Network
PM	Prediction Market
TMDb	The Movie Database
API	Application Programming Interface
ML	MovieLens
DL4J	Deeplearning4j
JVM	Java Virtual Machine
UWM	Unweighted Mean Model

References

1. Olivas, E.S. *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques: Algorithms, Methods, and Techniques*; IGI Global, Hershey, PA, USA, 2009.
2. Yoon, J.; Yang, E.; Lee, J.; Hwang, S.J. Lifelong learning with dynamically expandable networks. *arXiv* **2017**, arXiv:1708.01547.

3. Parkes, D.C.; Seuken, S. Prediction Markets. In *Introduction to Economics and Computation: A Design Approach.*; Cambridge University Press: Cambridge, UK, 2016; Chapter 18.
4. Yi, S.K.; Steyvers, M.; Lee, M.D.; Dry, M.J. The wisdom of the crowd in combinatorial problems. *Cogn. Sci.* **2012**, *36*, 452–470.
5. Othman, A. Zero-intelligence agents in prediction markets. In Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, Honolulu, HI, USA, May 2007; Volume 2, pp. 879–886.
6. Chen, Y.; Reeves, D.M.; Pennock, D.M.; Hanson, R.D.; Fortnow, L.; Gonen, R. Bluffing and strategic reticence in prediction markets. In Proceedings of the International Workshop on Web and Internet Economics, San Diego, CA, USA, December 2007; pp. 70–81.
7. Dimitrov, S.; Sami, R. Non-myopic strategies in prediction markets. In Proceedings of the 9th ACM Conference on Electronic Commerce, Chicago, IL, USA, June 2008; pp. 200–209.
8. Othman, A.; Sandholm, T. When do markets with simple agents fail? In Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, Toronto, Canada, May 2010; Volume 1, pp. 865–872.
9. Ertekin, Ş.; Rudin, C.; Hirsh, H. Approximating the crowd. *Data Min. Knowl. Discov.* **2014**, *28*, 1189–1221.
10. Page, S.E. *The Difference: How the Power of Diversity Creates Better Groups, Firms, Schools, and Societies-New Edition*; Princeton University Press: Princeton, NJ, USA, 2008.
11. Helsinger, A.; Thome, M.; Wright, T. Cougaar: A scalable, distributed multi-agent architecture. In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, The Hague, Netherlands, October 2004; Volume 2, pp. 1910–1917.
12. De Vany, A. *Hollywood Economics: How Extreme Uncertainty Shapes the Film Industry*; Routledge: Abingdon, UK, 2003.
13. Harper, F.M.; Konstan, J.A. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.* **2016**, *5*, 19.
14. Team, D.; others. Deeplearning4j: Open-source distributed deep learning for the JVM. *Apache Softw. Found. Licens.* **2018**, 2.
15. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
16. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
17. Garnett, P.; Bissell, J. Modelling Social Networks Reveals How Information Spreads. 2013. Available online: http://theconversation.com/modelling-social-networks-reveals-how-information-spreads-18776 (accessed on 10 June 2019).
18. Du, Q.; Hong, H.; Wang, G.A.; Wang, P.; Fan, W. CrowdIQ: A New Opinion Aggregation Model. In Proceedings of the 50th Hawaii International Conference on System Sciences, Hilton Waikoloa Village, HI, USA, January 2017.
19. Hastie, R.; Kameda, T. The robust beauty of majority rules in group decisions. *Psychol. Rev.* **2005**, *112*, 494.
20. Valentini, G.; Hamann, H.; Dorigo, M. Self-organized collective decision making: The weighted voter model. In Proceedings of the International Conference on Autonomous Agents and Multi-agent Systems, Paris, France, May 2014; pp. 45–52.
21. Awasthi, D. Exploratory Analysis of Movies on TMDB. Available online: https://rstudio-pubs-static.s3.amazonaws.com/369891_b123051c3cb64da5a6d22a8d0b6e0d84.html (accessed on 19 October 2019).
22. Malkoc, I. The Movies Dataset. Available online: http://ibomalkoc.com/movies-dataset/ (accessed on 23 August 2019).
23. Surowiecki, J. *The Wisdom of the Crowds*; Anchor Books, a division of Random House: New York City, NY, USA, 2005.
24. Shwartz-Ziv, R.; Tishby, N. Opening the black box of deep neural networks via information. *arXiv* **2017**, arXiv:1703.00810.
25. Zhu, Y.F.; Tang, X.M. Overview of swarm intelligence. In Proceedings of the IEEE International Conference on Computer Application and System Modeling, Taiyuan, China, October 2010; Volume 9, pp. V9–400.
26. Rosenberg, L. Artificial Swarm Intelligence, a Human-in-the-loop approach to AI. In Proceedings of the 30th AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, February 2016.

27. Rosenberg, L.; Pescetelli, N.; Willcox, G. Artificial Swarm Intelligence amplifies accuracy when predicting financial markets. In Proceedings of the IEEE 8th Annual Conference on Ubiquitous Computing, Electronics and Mobile Communication, New York City, NY, USA, October 2017; pp. 58–62.