COMP 3211
Fundamentals of Artificial Intelligence
Final Project Report

Topic:
In-depth Analysis of Felix: the Cat in the Sack

Supervisor:
SONG, Yangqiu

Authors:
LIANG, Zibo (20256837)
LIAO, Kunjian (20256368)
LIU, Qinhan (20328953)
ZHANG, Ziyao (20256928)
ZHANG, Zizheng (20256796)

Date:
23 November 2017
# Table of Content

1 Abstract ................................................................................................................................. 1

2 Introduction ........................................................................................................................... 1
   2.1 Problem Definition ........................................................................................................... 1
   2.2 Game Rules ...................................................................................................................... 1
   2.3 System Inputs and Outputs ............................................................................................. 1
       2.3.1 Inputs ....................................................................................................................... 1
       2.3.2 Outputs ................................................................................................................... 1
   2.4 Evaluation ......................................................................................................................... 1

3 Literature Review .................................................................................................................. 1

4 Methodology ........................................................................................................................ 2
   4.1 Approach .......................................................................................................................... 2
       4.1.1 Supervising Learning ............................................................................................... 2
       4.1.2 Deep Q Learning (DQN) ........................................................................................ 3
   4.2 Applied Libraries ............................................................................................................. 4

5 Experiment ........................................................................................................................... 4

6 Result ...................................................................................................................................... 4
   6.1 Experiment Results .......................................................................................................... 4
       6.1.1 MI Approach ............................................................................................................. 4
       6.1.2 DQN Approach ....................................................................................................... 4
   6.2 Error Analysis .................................................................................................................. 5

7 Conclusion ............................................................................................................................. 5

8 Appendix ................................................................................................................................ 5

9 Reference ................................................................................................................................ 5
1 Abstract

The success of Alpha Go inspires us that artificial intelligence could solve problems in certain gaming areas and give out the local optimal solutions. In this report, we will focus on the game called “Felix the Cat” which is not a perfect information game, and show how to apply the technical of machine learning into this kind of game. The result of experiments proves that traditional learning methods are also effective for the non-perfect information games, and we can finally get a relatively intelligent agent compared with the randomly moved agents and the agents with some naïve strategies.

2 Introduction

2.1 Problem Definition

With the huge success and the inspiration of the Alpha Go, we would like to apply the same strategy to other games and see whether this method has the universality to all the games. The project aims to utilize the technical in artificial intelligence to create an advanced gaming agent typically for the game Felix the Cat.

2.2 Game Rules

*Felix: The Cat in the Sack* is a 4-player auction game. Each player has 15 tokens and an identical set of 10 cards at the beginning of the game. The game consists of 10 rounds. Each round has two stages, the Selling Stage and the Bidding Stage.

At the Selling Stage, each player plays one card from the hand and orderly faces it down into Central Area. At the Bidding Stage, the first card in the Central Area is revealed, then players take turns to bid for the cards using their tokens. The player either increases the bid or quits the bid in his turn. If a player chooses to quit, he reveals a card in the Central Area, takes back his/her betting tokens and receives a certain amount of reward. At the end of the game, player with the highest score is the winner.

*(The details of rules will be shown in the appendix of the paper.)*

2.3 System Inputs and Outputs

2.3.1 Inputs

1. Game rule information: total number of rounds, configuration of default deck, max bid to exceed each time, skip rewards;
2. Player index of current agent, deck of current agent;
3. Current round, the first mover of this round, current stage;
4. Cards in the central area, current highest bid, next skip rewards;
5. Players’ information: inferred deck, score, token, bid, skipped or not.

2.3.2 Outputs

1. Which card to sell;
2. How many bids to exceed the last player.

2.4 Evaluation

The game core written by Python sends the current information on game and players to the agent and requests an output from the agent. One player can be controlled by human, a built-in agent or an imported agent.

Two built-in agents, the random agent and the naïve agent, are provided in the game core, as training partners of the agent at early stage. Namely, the random agent makes decisions based on random selection. The naïve agent, equipped with a certain amount of hard-coded logic, makes minimally reasonable decisions.

A dashboard is available, where the user can assign agents to each player, set the number of game plays and select game display mode.

3 Literature Review
According to the research of S.B. Kotsiantis [1], the strategies for our Felix agent could be generated by analysing a dataset. This approach is called supervised learning. Comparing to the reinforcement learning taught in lectures, supervised learning utilizes datasets with known inputs and outputs and represents them using the same set of features. We found the feasible implementation method of supervised learning from Fabian Pedregosa’s research [2]. The Python programming language could provide maturing ecosystem of scientific computing libraries we need for exploratory data analysis. Additionally, a Python module, Scikit-learn has integrated sufficient machine learning algorithms for our supervised learning approach. With Scikit-learn, it is possible for us to maintain an easy-to-use interface with the Python language.

We chose the detailed machine learning methods (or classifiers) based on Chih-Chin Lai’s paper [3]. There are three classifiers discussed in this paper, Naïve Bayes, K-nearest neighbour, support vector machine. For our instance, the basic concept of Naïve Bayes is to find whether a strategy could give positive outcome by looking at whether this strategy appears in the strategy sets leading to a winning result. The K-nearest neighbour classifier would calculate the distance between a new strategy set and all strategy sets in our training history. Then, it would assign a classification to the new strategy set using the k samples in training history that have nearest distance. Lastly, the support vector machine utilizes the concepts of statistical learning theory and structural minimization principle.

From the above research about tools we may exploit, our team has formed a basic blueprint of implementing this Felix AI agent. We shall develop this project based on Scikit-learn module and Python programming language, and make use of the built-in classifiers including Naïve Bayes, K-nearest neighbour, support vector machine etc.

4 Methodology

4.1 Approach

4.1.1 Supervising Learning

The basic approach we designed is to gather datasets from rule-based agents, then determine the strategies of our supervised learning agent based on those datasets.

As mentioned in the literature review, in machine learning, the datasets have to be represented by the same set of features. Since the game has two stages, bidding and selling, two sets of features need to be adopted for this agent. The agent records fifty-one features in the bidding stage such as bidding price history, number of players choosing to skip. Thus, strategies at every step can be generated by classifiers once datasets are available.

The initial generation of this agent is based on the training data sets of random agents who make random choices at every step. The decisions made by each random agent were recorded in a txt file. Various classifiers will analyse the decisions made by the winning random agent, and provide strategies for the supervised learning agent based on those decisions. From our observation, the agents generated from Naïve Bayes and support vector machine generally have better performance. However, this agent was not intelligent enough after the first round of training and regression. Its winning rate versus random agents is even lower than 25%. Afterwards, the data sets were replaced by the playing history of the trained agents. Thus, the second generation of agent could be trained based on this new data sets. After several
rounds of iterations, the winning rate of our supervised learning agent approached 45%.

Aiming to further increase the winning rate, other approaches have been tried by the team. A rule-based agent that makes decisions based on our own understanding of the game was programmed to facilitate the training of the supervised learning agent. Additionally, another agent that acts as the rule-based agent or as the random agent with equal probabilities was also developed. However, the training and regression results were not as expected. It seemed that after the supervised learning agent learned from the rule-based agent, it would be dominated by that agent during the subsequent games.

The optimal result the team got using supervised learning approach was forty-five percent winning rate versus random agents.

4.1.2 Deep Q Learning (DQN)

Deep Q learning (DQN) is a kind of reinforcement learning that combines Q learning and Neural network. The traditional Q learning has a bottleneck. When the problem we want to solve becomes complex, the number of state becomes considerable, which makes it very difficult to store all the Q value of state into a table. What's more, get Q value from table is also time consuming. In our game, the number of state is big which implies that it is impossible for us to use the traditional Q learning to solve this problem, so we choose to use deep Q learning instead.

In DQN, we will build a neural network (NN) which will give out the Q value of each action for a state. We also know the real Q value. We can calculate the error of the estimation and send it back to the NN and improve the estimation of it. In addition to this structure, DQN has two major factors to make it powerful. These two factors are experience replay and Fixed Q-targets.

DQN has a memory bank used to learn previous experience. DQN is a kind of off-policy learning method. Experience replay means that we can let the model to learn the experience it undergoes now, the experience in the past as well as the experience of others. This kind of policy will destroy the dependency of each experience and make the learning process more efficient.

Fixed Q-target is also a method to destroy the dependence of experience. We will build two neural networks with same structure but different parameter. One of the networks is used to get the estimated Q value and it updates the parameter frequently. Another network is used to get the real Q value and the parameter is updated less frequently. This will make the learning of model more efficient.

Implementation:

This is the whole algorithm of the DQN. It contains for part.

a) Traditional Q learning;

b) Memory for store experience;
c) Calculate the Q value using neural network; d) Fixed Q-target.

This is the neural network to get the estimated Q value. This neural network is divided into two different layers. We divide the network into two layers and compute the weight and bias of them. The structure of the other network is the same.

4.2 Applied Libraries

Scikit-Learn, Tensorflow, Python.

5 Experiment

In the experiment, we first gather all the selling and bidding decisions from 4 random agents playing against each other and use the data of only the winners to train the 1st generation machine learning agents. Depending on the machine model used, each agent is named differently, e.g. the agent based on “Support Vector Machine” is called “svm” agent. There are in total 4 agents: “svm” (Support Vector Machine) agent, “nn” (Neural Network) agent, “nb” (Naive Bayes’) agent and “lr” (Linear Regression) agent.

6 Result

6.1 Experiment Results

6.1.1 MI Approach

Relationship between generation of agents and the winning rates is showed as follows.

From 1st generation agents to 4th generation agents, the winning rate of the agents against random agent is clearly increasing gradually. For the 4th generation agents, it achieves around 50% winning rate against other 3 random agents. Notice that 25% winning rate for each player is a fair game, 50% winning rate is already a breakthrough we made.

The reason why keeps evolving the agents does not result in a better agent may due to the limited number of features we provide. Since there are 4 players selling 10 cards each one in a game and the number of bidding decisions of a game is limited by the token numbers but not explicit, the total number of cases one may encounter exceeds $(10!)^4 = 10^{26}$. Comparing to the total cases, we have only 51 or 65 features to feed the machine learning model. As the situation gets more and more complex while evolving the agents, the model may be in under fit, then the actually winning rate drops as we do see in the graph.

6.1.2 DQN Approach

Relationships between gaming times and the winning rates:

From the figure we see that there is really not much difference as for the winning rate of our DQN agent after 20000 games learning both positive and negative cases. The problem that we are not achieving any improvements may due to the huge number of states of the whole game. Foreseeably, we only tried out less than $1/10^{22}$ of the total states, which is far from equipping the agent with enough intelligence to outperform random agents.
6.2 Error Analysis

For 6.1.1, one better solution is to build a model for each round. On round 1, only 51 or 65 features are involved. While on every round after round 1, we keep track of all former decisions made and concatenate them together to feed the model for this round. Then for round n, we have exactly 51+n or 65+n features. Then the problem of under fit may be solved and the overall winning rate should improve.

In 6.1.2, the only resolution is to keep the agent trying for enough cases due to the methodology of reinforcement learning is exactly trying out to learn, so if we find out a way for the agent to play the game quicker, it may finally get competent. But still, the large amount of trials is the burden we have to overcome.

7 Conclusion

From the results of the experiments we can clearly tell that our training methods indeed increase the winning rate of the targeting agent. The increasing scale of the winning rate is about 30% on average and therefore can be considered as a sound prove that our training strategy is effective in this game. However, there are still some improvements we would like to add into the system, which will be discussed in the future paper.

8 Appendix

Game rules in details

a) Felix: The Cat in the Sack is a 4-player auction game. Each player has 15 tokens and an identical set of 10 cards at the beginning of the game.

b) The game consists of 10 rounds. Each round has two stages, the Selling Stage and the Bidding Stage.

c) The card set includes “+15 Cat”, “+11 Cat”, “+8 Cat”, “+5 Cat”, “+3 Cat”, “0 Cat”, “-5 Cat”, “-8 Cat”, “Big Dog”, and “Small Dog”.

d) A “Cat” card is worth the same score as specified in its name. For instance, a “+15 Cat” is worth 15 points.

e) “Dog” cards have special effects when computing the score of the purchased cards at the end of the Bidding Stage. The “Big Dog” removes the “Cat” card with the highest value and the “Small Dog” removes the “Cat” card with the lowest value.

f) At the Selling Stage, each player plays one card from the hand and orderly faces it down into Central Area.

g) At the Bidding Stage, the first card in the Central Area is revealed, then players take turns to bid for the cards using their tokens. The player either increases his/her bid or quits the bid in his/her turn. If a player chooses to quit, he/she reveals a card in the Central Area, takes back his/her betting tokens and receives a certain amount of reward which depends on how many players have already quit the bid at that time. The last player remained in the Bidding Stage wins the bid and purchases the cards, after which the game proceeds to the next round.

h) At the end of the game, player with the highest score is the winner.

9 Reference