Bankrupting the casino

Patrick Guerin

6 avril 2018

Abstract

The blackjack is a very popular game in casino. The first goal of this project was to evaluate the performance limits of the Q-learning for this game. Secondly, we aimed to see if an agent could learn to count the cards to improve its performance, and ultimately to beat the house. In that respect we evaluated the impact of different rules on the performance of the agent to be as close as possible to the reality.

Context

The Blackjack game

We implemented a simplified version of the Blackjack game:

The objective of the player is to obtain a total of card higher than the dealer without exceeding 21.

The score of a hand is computed by the face value of each card in the hand, with the Jack,Queen and King having a value of 10.

At the beginning of each game the player and dealer are both dealt two cards,and at this moment the player can only see the first card of the dealer.

At each turn the player has two possibilities: draw an additional card or stop and stay with its cards.

If the player draw a card, it gives him the opportunity to get a higher score, at the risk of exceeding 21 and losing the game.

When the player stops to draw cards, the dealer shows its second card and draw additional cards until its own score reaches 17 or more.

Card counting

Card counting is a strategy that consist of maintaining a mental count of the cards already dealt - or of a score based on those cards - in order deduce information about the remaining card in the deck.

However keeping in mind all the cards already dealt can be impossible depending of the number of deck used in the game.

Hence, strategies have been developed to take this fact into account.

We chose to use one of those strategies to define the action-state space. Indeed ideally our agent should keep in memory all the previous dealt card, but the then the action-state space blows up:

In las vegas the casinos use between 1 to 8 decks to deal the cards, for 4 decks used the number of possible states for the player cards only is around 54^4.

Consequently, we had to summarize the action-states.

Hi-Lo Strategy

We used a modified version of the Hi-Lo strategy.

This strategy consists of keeping a mental counter depending of the cards dealt.

the cards [1,2,3,4] are assigned a value of +1

the cards [10,Jack,Queen,King] are assigned a value of -1

the cards [5,6,7,8,9] are assigned a value of 0.

This strategy allows the player to know how many cards the ratio of low cards and high cards that have been dealt: If the counter is high, a lot of low cards have been dealt and the probability of high cards is increased.

Number of decks

The number of decks is very important in card counting, the most deck are used by the dealer, the less informative it will be to count the cards. Casinos are well aware of that and generally use multiple decks, to stay as close as possible from the reality, we will examine the performance of the agent depending of the number of decks used.

Frequency of deck shuffling

Shuffling the deck regularly is another rule used by the casinos to severely damage the effectiveness of card counting. We will examine its impact depending of the frequency of deck shuffling.

The Q-learning algorithm

Choice of the method

The game of BlackJack can be easily modeled as a markov chain and the Q-learning algorithm seemed to us to be particularly adapted derive the optimal policy.

The Q-learning is part of the temporal-differences methods and has several advantages deriving from it:

1- Contrary to dynaming programming, it allows the agent to learn directly from raw experience without a model of the environment's dynamics.

2- Contrary to Monte-Carlo methods, it allow the agent to update its estimations, without waiting for the final outcome.This means that our agent will be able to learn during a Blackjack game and not only at the end of it.

Moreover, because the mistakes of the agent during the training phase can be done at no cost and that the fact the Q-learning tend to be faster, we prefered the Q-learning over the SARSA algorithm.

Description of the method

Firstly, we define the Q-table, which links each pair of action-state to its (approximated) expected value . On the basis of this the agent will choose the action with the highest expected value given the current state.

The Q-table is first initialized at zero for all action-states, and we use Temporal Difference error based updates to fill it.

The update is made according to the following formula:

$$Q(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Bigg(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \Big[\underbrace{Q(s_t,a_t)}_{\text{old value}} - \underbrace{\max_a Q(s_{t+1},a) \Big]}_{\text{estimate of optimal future value}} \Bigg)}^{\text{learned value}}$$

You can find below a description of the various parameters and of the choices we made about them.

Learning rate(α)

The learning rate or step size determines to what extent newly acquired information overrides old information

Various condition have been defined in the literature to ensure a convergence[-@convergence], in practice, a learning rate of 0.1 is often used, this is what we used after several trials.

Discount factor (γ)

It is the weight given to the future rewards,we set γ to 1 since there is no interest for the agent to not take into account all futures states in the case of the blackjack game. Hence we set $\gamma = 1$ (which gave us the best results).

Exploration factor (ϵ)

The Exploration factor is part of ϵ-greedy strategy generally used to garantee that the agent explore all the state space a convenient number of times.

To ensure that our agent become uncertain as its learning progress we set

$\epsilon = 1/t$ where t is the number of episodes.

Implementation

The blackjack game is an application particularly interesting since we can use our own experience to model the environment to be as close as possible of the experience of a real player.

We have implemented the environment separately from the algorithms to mimics the reality. The environment is implemented in **Environment.R** file and the algorithms in the **Policies.R** file.

We created 2 new scripts to implement the card counting environment and algorithms: **Environment__count.R** and **Policies__count.R**.

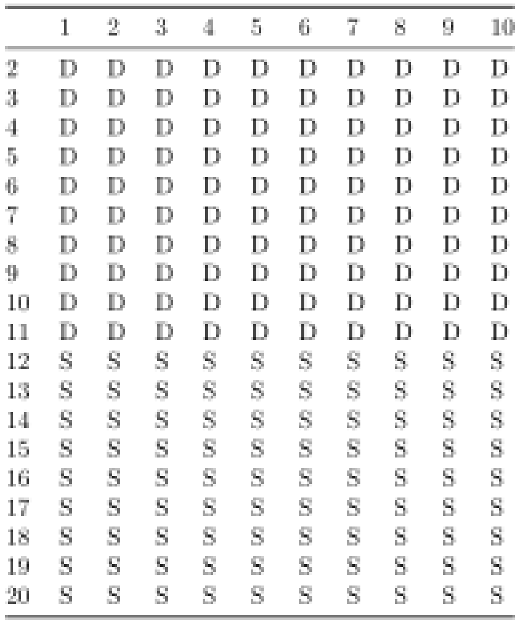
Benchmarks

In order to assess the performance of our agent, we have implemented a random strategy and a cautious strategy to serve as benchmarks to evaluate our performance.

In the random strategy, the agent take a random action (Draw or Stop) at each step.

In the cautious strategy, the agent choose to draw cards until he reaches 11 or more, then stop the drawing. With this strategy the player never have a hand exceeding 21. The strategy is illustrated below.

Figure 1: Cautious strategy



	1	2	3	4	5	6	7	8	9	10
2	D	D	D	D	D	D	D	D	D	D
3	D	D	D	D	D	D	D	D	D	D
4	D	D	D	D	D	D	D	D	D	D
5	D	D	D	D	D	D	D	D	D	D
6	D	D	D	D	D	D	D	D	D	D
7	D	D	D	D	D	D	D	D	D	D
8	D	D	D	D	D	D	D	D	D	D
9	D	D	D	D	D	D	D	D	D	D
10	D	D	D	D	D	D	D	D	D	D
11	D	D	D	D	D	D	D	D	D	D
12	S	S	S	S	S	S	S	S	S	S
13	S	S	S	S	S	S	S	S	S	S
14	S	S	S	S	S	S	S	S	S	S
15	S	S	S	S	S	S	S	S	S	S
16	S	S	S	S	S	S	S	S	S	S
17	S	S	S	S	S	S	S	S	S	S
18	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
20	S	S	S	S	S	S	S	S	S	S

Game Simulation

We used three mains functions to construct the environment:

1. The function **step** which handle one "turn" of the game, during one turn the agent choose one action between draw and stop. If the game end during a step, the reward is update: -1 if the agent lost, +1 if he lost and 0 for a draw. If we count the card it is also this function that increment the counter of the Hi-Lo strategy. The counter is incremented when the player become aware of the cards, as it would be in real life.
2. The function **party** which handle an entire party. Essentially it initialize of the players and loops the function **step**, calling the function **Qlearning** after each step.
3. The function **game** which handle an entire simulation and encompasses party. This function allows to pass various parameters:
    - the number of game desired in the simulation.

- the information displayed: if infos=TRUE one can see the parties as they are played.
- the strategy to apply: Random,Cautious or Q-learning.

If we count the cards it also allows to specify:

- The number of decks to use (the Q-table is automatically scaled depending of this parameter)
- The frequency of deck shuffling

We used three mains functions for the algorithm:

1. **rowQmatrix** which return the state in the Q-table given the cards of the player (and given the counter if we count the card).
2. **Qlearning** which update the Q-table using the Q-learning update.
3. **choose_action** which return the chosen action using an epsilon-greedy strategy.

States

In the simulation without card counting the features of the space are represented as follow:

Content	Possibles values
Player score	2-20
Dealer score	1-10

In the case with card counting we have:

Content	Possibles values
Player score	2-20
Dealer score	1-10
Count	Depend on deck size

Number of decks

As the number of decks grows, the number of values that the counter can take also grows and with it the dimension of the action-state space. To keep our Q-table as small as possible, we automatically scale the Q-table in function of the number of decks chosen (up to 10 decks).

we defined empirically the dimension of Count by observing the maximum count obtained after playing a consequent number of games, for a given number of decks. For example if we play with one deck, we will almost never have a count which is not between -10 and 10.

Results

Simple Q-learning BlackJack

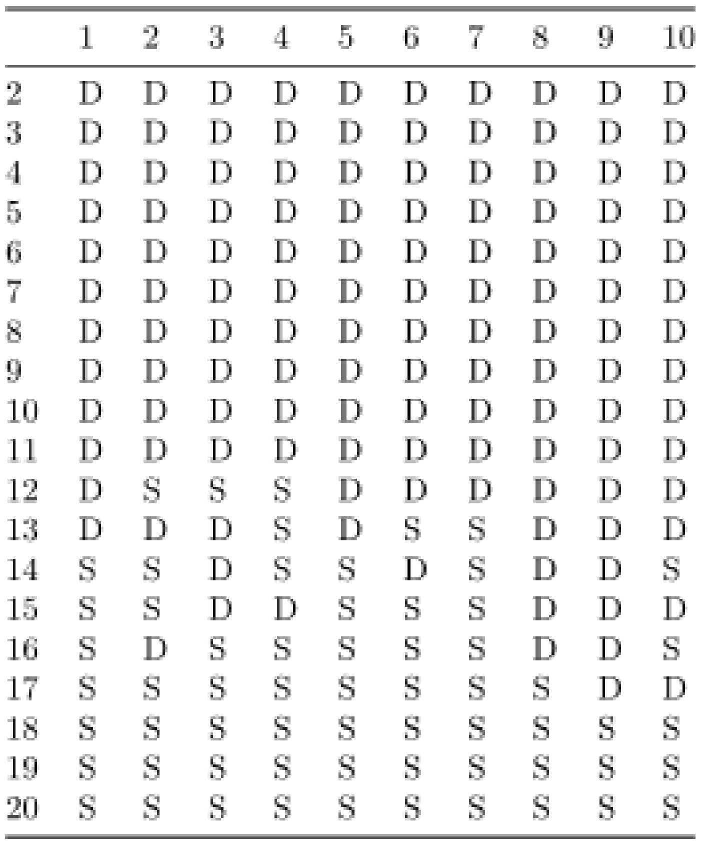
The policy obtained with Q-learning performed significatively better than a drunk player (random strategy) but couldn't equal the cautious strategy.

Table 3: winnings in percentage

random	Q-learning	Cautious
0.3	0.417	0.43

We can also vizualize the policy followed by our agent:

Figure 2: Learnt policy



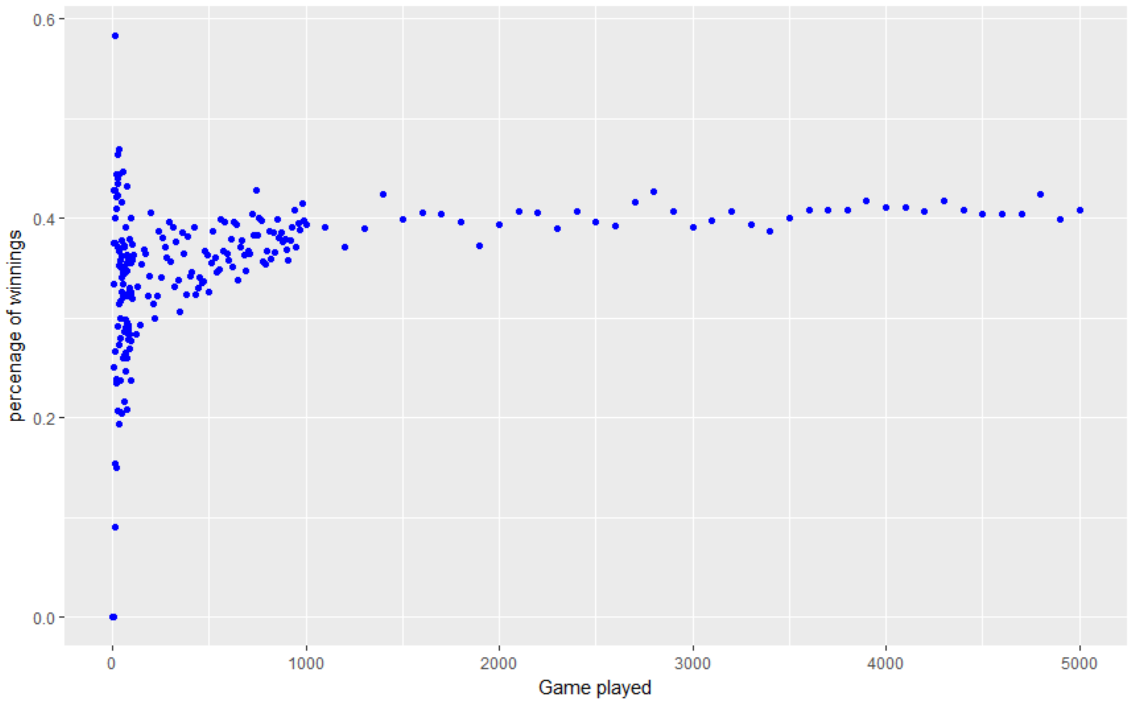
	1	2	3	4	5	6	7	8	9	10
2	D	D	D	D	D	D	D	D	D	D
3	D	D	D	D	D	D	D	D	D	D
4	D	D	D	D	D	D	D	D	D	D
5	D	D	D	D	D	D	D	D	D	D
6	D	D	D	D	D	D	D	D	D	D
7	D	D	D	D	D	D	D	D	D	D
8	D	D	D	D	D	D	D	D	D	D
9	D	D	D	D	D	D	D	D	D	D
10	D	D	D	D	D	D	D	D	D	D
11	D	D	D	D	D	D	D	D	D	D
12	D	S	S	S	D	D	D	D	D	D
13	D	D	D	S	D	S	S	D	D	D
14	S	S	D	S	S	D	S	D	D	S
15	S	S	D	D	S	S	S	D	D	D
16	S	D	S	S	S	S	S	D	D	S
17	S	S	S	S	S	S	S	S	D	D
18	S	S	S	S	S	S	S	S	S	S
19	S	S	S	S	S	S	S	S	S	S
20	S	S	S	S	S	S	S	S	S	S

We can observe that the agent never stop if its score is below 12 since it would be ineficient in 100% of the cases. In other words for this part of the state space, the agent has learnt the best possible strategy.

Apart that, our agent has learnt gain statistical intuition about the subtleties of the BlackJack; with score of 12 or more, It tends to Draw a lot more when the dealer has high cards (8,9,10), which is considered to be a good practice among BlackJack players.[@basic]

It can also be interesting to vizualize how long should we train the agent in order for it to converge:

Figure 3: Variance of the performance



In this case the size of the state-space is relatively low which allow allow a relatively fast convergence of the performance, the variance seems already pretty low for 3500 games played.

Unfortunately, we were not able to do better than the cautious strategy with Q-learning. Can we do better by counting the cards?

Counting the cards with Q-learning

We obtained fairly better results by adding a count of the card to the state space.

variance

Below,we constructed a learning curve for a game with 1,3 and 8 decks in order to have an idea of both the convergence rate and the influence of the number of decks on the performance of the agent.

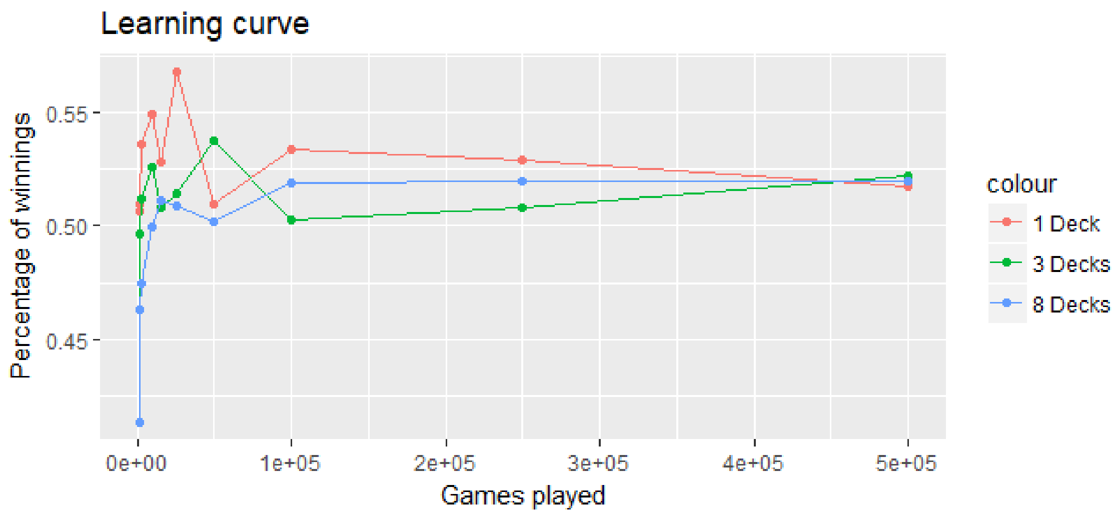
Surprisingly, the

shuffle?

Further works

This project confirms that the representation of the game via the proper states in crucial for performance in blackjack. The Hi-Lo strategy is fairly efficient, however it cannot be as efficient as considering most of the possible states. In that manner it would be interesting

Figure 4: Learning curve



to see in which measure a multi-layer perceptron (or other non-tabular methods) can help to represent the Q-table and help to consider even more states.

Indeed,the Universal Approximation Theorem shows that a multi-layer perceptron should be able to eventually represent any Q function if the network if sufficiently large.

Secondly, we did a bit of tuning of the learning rate parameter but it would be good to do more work and look into some more interesting annealing strategies.

Another avenue to explore is the betting aspect of blackjack, in reality the choice of the amount of the bet in each party is crucial for most of the counting strategies,and there is general recommendations about when to up the bet and when to lower it.

Finally a very useful path to investigate would be to parallelize the Q-learning algorithm to share the Q-table between several threads; It would lead to more accurate modeling of the states for the same amount of computation time.

References

@basic

nocite: | @Sutton @basic @strat