Artificial Intelligence
Project

Jesper Holmblad, Fredrik Mårtensson

January 23, 2020

Abstract

Games without exact solutions are common ways to learn machine learning. For that purpose we'll be making a crude machine learning based agent for five card poker. We'll start by explaining how this implementation is done and then give it the PEAS description of which the feature vector is based upon. Prior to the intended tournament a problem with the server was observed. This problem lead to most groups, with us included, withdrawing from the tournament. The server issue shut down any hope of testing the model in the environment it was built for. Post tournament this was fixed and revealed that the model had only learned a bias.

Contents

1 Introduction

Games without exact solutions are common ways to learn machine learning. For that purpose we'll be making a crude machine learning based agent for five card poker. In short, five card poker is like Texas hold em but with five cards on hand and no cards on the board. All players get a hand, place some bets based on their perceptions, hands are shown and the winner takes the pot. For a more detailed description see the PEAS description section.

There's more than one way to approach this task of making a poker agent. A handful of solutions have been considered but the set can be narrowed down by considering the restrictions the game applies to our model. We have hidden information which excludes all informed search algorithms. There's also randomness involved which excludes all exact solutions. This leaves us with:

- Uninformed search like BFS but with probability
- Machine learning methods
- Bayesian Network
- Reflex agent that accounts for probabilities

Another method that works with all opponent based games is to combine an opponent model with recursion to enhance the performance. This opens up for search tree optimisations such as expectiminimax.

Methods and information used in this project comes from: Artificial Intelligence: A Modern Approach (AIMA), Google crash course, Keras documentation and Scikit documentation.

1.1 AIMA

The course book [1] for the course and the course material is based on.

1.2 Google crash course

Google offers a free course[1] focused on teaching the essentials of ML. They teach what type of ML should be used in different situations as well as how and why preproccesing is done. How feature selection, data mining, normalization, quantiles, embeddings is done is based on the information given in the crash course.

1.3 Keras documentation

Keras documentation[2] offers technical explanation of various algorithms and generalises the syntax for them. It's gives the mindset that any model reduced to their input tensor, output tensor and model specific parameters. This is intended for structuring large models made of several smaller ad hoc models but the structure helps on smaller projects as well.

1.4 Scikit documentation

Scikit[3] helps make ML a causal experience. The framework contains all the basic forms of ML as well as the tools needed for data processing. The documentation offers many examples on how to use their framework allowing you to stitch several examples together rather than getting in depth with the documentation for simple implementations.

[1] `https://developers.google.com/machine-learning/crash-course`
[2] `https://keras.io/models/about-keras-models/`
[3] `https://scikit-learn.org/stable/user_guide.html`

2 Theory

2.1 Normalisation

Normalisation is a method of transforming multiple features into the same scale. Without normalization, two features of different lengths with three points, such as points 1,2,3, may appear to have two (1 and 2) points which are reasonably similar. After normalization, the data can be represented differently where 2 and 3 now have greater similarities than 1 and 2. By transforming the data we can get a better picture of how the data is structured and both axes in this case are represented on the same scale.

2.2 One-hot encoding

In order for machine learning not to confuse two things like having a relationship where one is better than the other, we use one-hot encoding. Let's say that the agent's feature vector consists of the following actions [Call, Raise, Fold] and each action can change place with another. It is not possible to give each action its own number [1,2,3] and use it as feature vector for the machine learning as they have no direct relation to each other were 1 is considered better than 2 and 2 is better than 3. When using one-hot encoding, the model will look like the following [0,0,0,0,0,0,0,0,0]. These are three actions where each action gets a combination of numbers ex:

1. Call: [1,0,0]
2. Raise: [0,1,0]
3. Fold: [0,0,1]

This is merged into a vector as follows [1,0,0,0,1,0,0,0,1]. The method has high space complexity for many features but can quickly be built a feature vector for machine learning.

3 PEAS description

"For the acronymically minded, we call this the PEAS (Performance, Environment, Actuators, Sensors) description".[1] The following chapters will briefly explain where each part of PEAS is implemented in the project.

3.1 Performance

Since the project is based on classification it makes sense to use accuracy as performance measure.

3.2 Environment

Five card poker contains a set of objects, mechanics and a flow as described below.

From the environment it is possible to determine the agent's design and which areas it can be used for. A short summery of environmental variables can be observed below.

- *Partially observable:* Since the agent can't see the opponent's hand
- *Multiagent:* Five agents in the game.
- *Stochastic:* Each time the deck is involved randomness is included.
- *Sequential:* The agent uses previous turn in order to decide the current action.
- *Dynamic:* There is a time limit and if no response given to server the agent is kicked.
- *Discrete:* Turned based game.
- *Known:* The agent knows what the actuators do.

3.2.1 Objects

Serve as anchor points for actuators. Could also be seen as name spaces.

1. *Chips:* A players currency. when it runs out the player loses.
2. *Cards:* The building block of hands. Has rank and colour.
3. *Hands:* A combination of 5 cards. Given to the player by the game server.
4. *The pot:* The accumulated bets from all players.
5. *Player:* The player/agent playing the game. The player owns chips.

3.2.2 Mechanics

These are operations/actuators belonging to game objects.

Chips mechanics

- *Match:* Means matching the highest current bet
- *Raise:* Raising means increasing ones bet over the highest current bet which is turn forces all other players to respond. A raise must be equal to or greater than the previous raise, this is to prevent players from stalling the game.

Cards mechanics

- *Display:* Cards can be made visible
- *Compare:* Cards can be compared by both rank and colour

Hands mechanics

- *Discard:* The act of throwing one or more cards which then gets replaced. This action can be seen by all players

- *Display:* Hands can be made visible
- *Compare:* This determines which hand is the strongest by first looking at which category each hand belongs to. The hand with the rarest category wins. If both hands fall into the same category it compares the rank of the cards specific to the category. If category and the rank is the same the hands are viewed as equal.

3.2.3 Flow

The operations performed by the game itself. These can be called in any state machine fashion but in this case they are simply called in order.

1. Ante
2. The first betting round is made.
3. If any players remain in the game, then each player can choose to change cards.
4. The second betting round is made.
5. Showdown, where the player hands are compared and the winner(s) will receive the pot (or part of the pot, in case of multiple winners).

3.2.4 Actuators

Mechanics that belong to the player

- *Open:* Performs the raise actuator from chip without setting the minimum raise.
- *Call:* Trigger match actuator from chip
- *Check:* Also triggers chips match actuator
- *Raise:* Bound to chips raise actuator
- *Fold:* This actuator is unique for the player. It sets the agent state to folded meaning that the agent will not be affected by the game flow until the start of next round.
- *All-in:* Raise equal to the players chips.

3.2.5 Sensors

The sensors of the agents are it's inputs and the inputs are the features in the feature vector. This means the sensors are:

- The agent's hand (6255)
- Chips for all players
- Previous and current action/actuator for all players.

Example of sensor readings:
[6255, 250, 250, 250, 250, 250, 'Player_Call', 'Player_Check', 'Player_All-in', None, 'Player_Fold', None, 'Player_Call', 'Player_Check', 'Player_Call', 'Player_Check']

4 Method

As mentioned the chosen method is to make an agent based on machine learning so we'll start by explaining how this implementation is done. We'll also give our agent a PEAS description as this was requested. Finally a hypothesis about the expected result is made.

4.1 Machine learning

Machine learning is a method of generating a solution based on statistics. The framework Keras divides the main ML models into three attributes:

1. The model layers
2. The output tensor
3. The input tensor

Our model consist of a single layer SVM and output is defined as a trinary class representing whether or not the current state will lead to a loss, undisputed win or a normal win. This leaves only the input.

Since Sci-kits framework is intended for one dimensional input the shape of the input has to be [Batch, Feature] where batch is determined by Scikit. This means we only have to determine what the feature vector in order to finish the model. However the feature is the most important component so we'll be explaining it in depth. The feature vector creation can be split into three steps: feature selection, data mining and data processing.

4.2 Feature selection

The basic idea behind our feature vector is to provide a state vector that covers all the data at a point in time. The game state would simply be the current pot as well as each players hand and chips for a total of $1 + 2 * n$ where n is the number of players. However since we don't know the hand of each player these would have to be estimated by another upstream machine learning. But since this model is a single layer SVM that is not possible so the hand for each opponent has to be scraped leaving us with $2 + n$ features.

Since the pot is not something given from the server, this must be calculated from the client side. However, with several packet losses, a good method for extracting the information could not be produced. Thus, the pot was excluded as a possible feature for the model.

Because the output is to be a trinary class we also need to include the actuator we intend to use for each player so that we may iterate the actuators to figure which of them are viable. The actuator of our opponent is only known if they make their move before us. It could be approximated upstream but once again; it's a single layer model. So a naive fix to this is to give out ML features that we do know which would correlate to the opponent actuators. Taking a stab in the dark, one can guess that the actuators from the previous turn would provide some insight to this so those were added as well putting the new feature length at $1 + 3 * n$.

4.3 Data mining

Data mining involves searching and extracting information relevant to the selected features from large amounts of data. First step is to get a large amount of data. It was difficult to extract information from the client side because of missed packets and incorrect responses. To solve this the data is picked directly from the server. Secondly the data needs to be filtered down to the information relevant to our model. This is done by searching for keywords known to be near the information and then extracting it with string operations.

Below includes two feature vectors extracted by data mining where example 1 is an accepted feature and example 2 is not since it's missing a feature, which is represented by the -1 in the first feature. Which in turn means that the agent will never find out what his own hand is. Thus this feature vector will be scraped. The examples below is built upon the following vector were the target agent that is playing is in a fixed position.

[targetHandStrength, targetChips, targetWin, p1Chips, p2Chips, targetAction1,targetAction2, p1Action1, p1Action2, p2Action1, p2Action2, p3Action, p3Action2, p4Action1, p4Action2]

Vector Example 1:
[6261, 880, 905, 730, 855, 630, 'Player_Check', 'Player_Check', 'Player_Check', 'Player_Check', 'Player_Check', 'Player_Check', 'Player_Check', 'Player_Check', 'Player_Check', 'Player_Check']

Vector Example 2:
[-1, 250, 0, 250, 250, 250, 250, 'Player_Fold', None, 'Player_All-in', None, 'Player_Call', 'Player_Check', 'Player_Call', 'Player_Check', 'Player_Call', 'Player_Check']

4.4 Processing

Before sending the data into the model it needs some processing. This involves normalization, quantisation and embedding. Normalisation isn't necessary as most ML models can adapt their weights to do this for us but it's done anyways as this speeds up the learning process in most cases.

As we're working with first order ML it's important that the features are first order polynomials as well. In other words they need to be linearized. This is trivial if you know the maths behind the generated data. However if you are clueless then there's a crude yet effective method that always works: quantiles. Simply split the data into n equally sized sets, these are called quantiles. Each data point is relabelled with the number of the set it falls into.

Finally, to handle categorical features they are processed into embedded features. To embed a feature one first needs to create an embedding. These can be quite complex so to simplify we'll resort to the most primitive embedding there is. The embedding of choice is one-hot encoding.

The following were observed from the feature vector before and after utilising normalization, quantile and one-hot encoding:

Before preprocessing:
[2831, 805, 830, 855, 780, 730, 'Player_Check', 'None', 'Player_Check', 'None', 'Player_Check', 'None', 'Player_Check', 'None', 'Player_Check', 'None']

After preprocessing:
(0.08571428571428572, 0.6470588235294118, 0.5588235294117647, 0.8, 0.5428571428571428, 0.5882352941176471, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0)

The first feature represent hand strength. Note that 2831 correspond to three of a kind according to the hand evaluator API and after being processed it turns into 0.0857 or 8.57%. The probability of getting three of a kind or better is 2.87%. This shows the performance of the processing method chosen. As the data set is increased the processed feature should in theory approach the

mathematically derived value.

4.5 Expected behaviour

Using this method we expect to see no result. Why is no result expected? This is because at the time the model was created there was no way to interact with the server without getting disconnected.

However in the off chance that the server does function as intended the expected result would be that the agent makes decisions that lead to slightly higher win rate than random and some ability to generalise on hand strength and chips.

Also by running the model model in batches and train on the newly acquired data it should imitate genetic algorithms where the training data can be seen as the genetics. This should enable the model generate the data most needed for training.

5 Experiment and tournament result

Prior to the tournament a problem was observed. Data from the server did not arrive as expected. This meant that the agents were kicked from the server for incorrect response or lack of response. The messages could come up as empty arrays "[]" or with multiple messages bundled together into one which caused major problems.

This lead to most groups, with us included, withdrawing from the tournament. With only one group in our bracket there weren't enough players to host a tournament so we have nothing to show from that test.

The server issue didn't just stop the tournament but also shut down any hope of testing the model in the environment it was built for. Furthermore it made it near impossible to acquire viable training data. A valiant attempt was made but out of the 1500 data points extracted only 44 were considered usable.

Post tournament a fix for the server issue was acquired. This resulted to loads of bugs and errors being uncovered, both practical and theoretical. When trained data from random agents the model defaults to checking. When trained on data from random agents mixed with ML based agents it resorts to folding. Even if the other agents don't raise the model will still fold. This suggest that it's acting heavily based on a bias while disregarding most the input features. We can showcase a test accuracy of 80% however with this kind of error it's safe to assume it guessed on losing as four out of five player do lose.

6 Conclusion

The server issue teaches us (once again) that the testing environment has the final say on the result. Unfortunately the environment comes pre-compiled and undocumented making it a dead end for anyone trying to fix it. However a fix was found which allowed us to confirm that the issue was caused by something called Nagle's algorithm.

As for the model. We can conclude that the chosen metric can prove misleading. Switching metric to spearman correlation would eliminate any model that relies on bias alone. The model itself is no condition to be competing in tournament

What could be reworked then? Instead of using a one dimensional feature vector one would likely be better of using an input such as [batch, phases, players, features]. Say we store four features belonging to five players over the four phases of a round then we'd have a total of batch*4*5*4 = batch*80 features as input. This is a lot more features but because of the structure it would be much easier to generalise since we can now process the different dimensions with models specialised on those types of data.

Of course we wouldn't suggest changing the feature without a model in mind. Using a multi layered model we could combine the best of all ML models into one. Time series data is preferably processed by recurrent models such as RNN, LSTM, and GRU. The data from each player could be fed one at a time into a neural net that outputs an embedding for each. Since we know the final layer has to perform comparisons we could opt for a deep neural net or perhaps a convolutional model, that way the activator function can mimic the non-linear properties of the comparison. As a cherry on top an ad hoc model could be built to determine our opponents hand strength.

These changes to the model would be challenging to implement while being constrained by Sci-kits framework. Switching over to Keras would enable a lot of interesting builds to be created.

References

[1] Russel P S & Norvig. Artificial Intelligence: A Modern Approach. 3:rd edition. Prentice Hall; 2010.