Building a BlackJack AI

Basic Concept: an Ai classifies a blackjack game situation as stand (‘should-not-hit’) or hit (‘should-hit’) using 4 pieces of information: [value of players hand (int), player has ace (0/1), value of dealers first card (int), dealer has ace (0/1)]

# Choosing an algorithm

I chose to use the k nearest neighbors because I expected the data space to be very non-linear. with pockets of ‘should-hit’ and pockets of ‘should-stand’ overlapping each other. The way I labeled training data would have a lot of similar points in the same place, because sometimes ‘hitting’ is the right thing to do, but you bust, and my basic evaluating functions evaluateHand and evaluateGame in Blackjack.py I used for labeling would label that as the wrong move. I wanted to average out the nearest neighbors so that I could capture the right move for positions were the “right” move only works some of the time.

# Training the algorithm

## Random Strategy used to build some training data

First I had the AI randomly decide to hit or stand. Literally “hit if randf(0,1) < .5 else stand”. This resulted in a low win rate, but still surprisingly ok. I think this was because statistically the dealer busts 30% of the time. The variance of the random strategy win percentage was very high, indicating to me that the times it won a lot were flukes. Because sometimes it won a lot, sometimes it lost a lot.

I ran 500 games of that, which was ~2000 rounds of play (because if the ai hits, the game continues, it only ends when it busts or stands)

I saved these 2000 lines to a file, evaluating and labeling them as I went. (BlackJack.py line 76 and 91)

## First attempt at using the model

Then, I built a knn model using that data, and reran the above process. But this time, I didn’t use the random strategy. I queried the model using the current game state, and did what it predicted. (BlackJackAi.py line 37)

I ran this for 500 games, and then built the model from scratch again. At this point, I was doing better than the random strategy by 10%, but I still wasn’t beating an AI that used the dealer strategy: “hit if handValue < 16 else stand”. That did better than my model by about 6%.

## Modifying the model using what I learned from the first attempt

I realized I might not be using enough training data to be able to average out “hit was right but accidentally busted cases”. So I increased k a little bit, and ran games until I had about 500,000 rounds of training data.

## Result:

When I used this model to play, It did GREAT! I consistently beat the random strategy by 20%, and the dealer by 5%, with very low variance. It was very cool

If you want to run it yourself,

1. Run BlackJack.py. Set the variables at the top of that script to useModel = False and overWriteTrainingData = True. This will generate the first set of training data.
2. Run BJAICreate.py. it makes the model.
3. Now you can run BlackJack.py again, but set useModel to True. If you want more training data, change overWriteTrainingData to False, so every time you run BlackjackAi.py the results are appended to the last run. About 5-10 times should generate enough data to reproduce what I did with appending on.
4. Run BJAICreate.py again to recreate the model (this step takes a few minutes)
5. Run BlackJack.py, and you should get about a ~70% win percentage over the dealer! Pretty neat!