Building a Rock Paper Scissors AI

How to ensemble six models to predict and avoid predictability

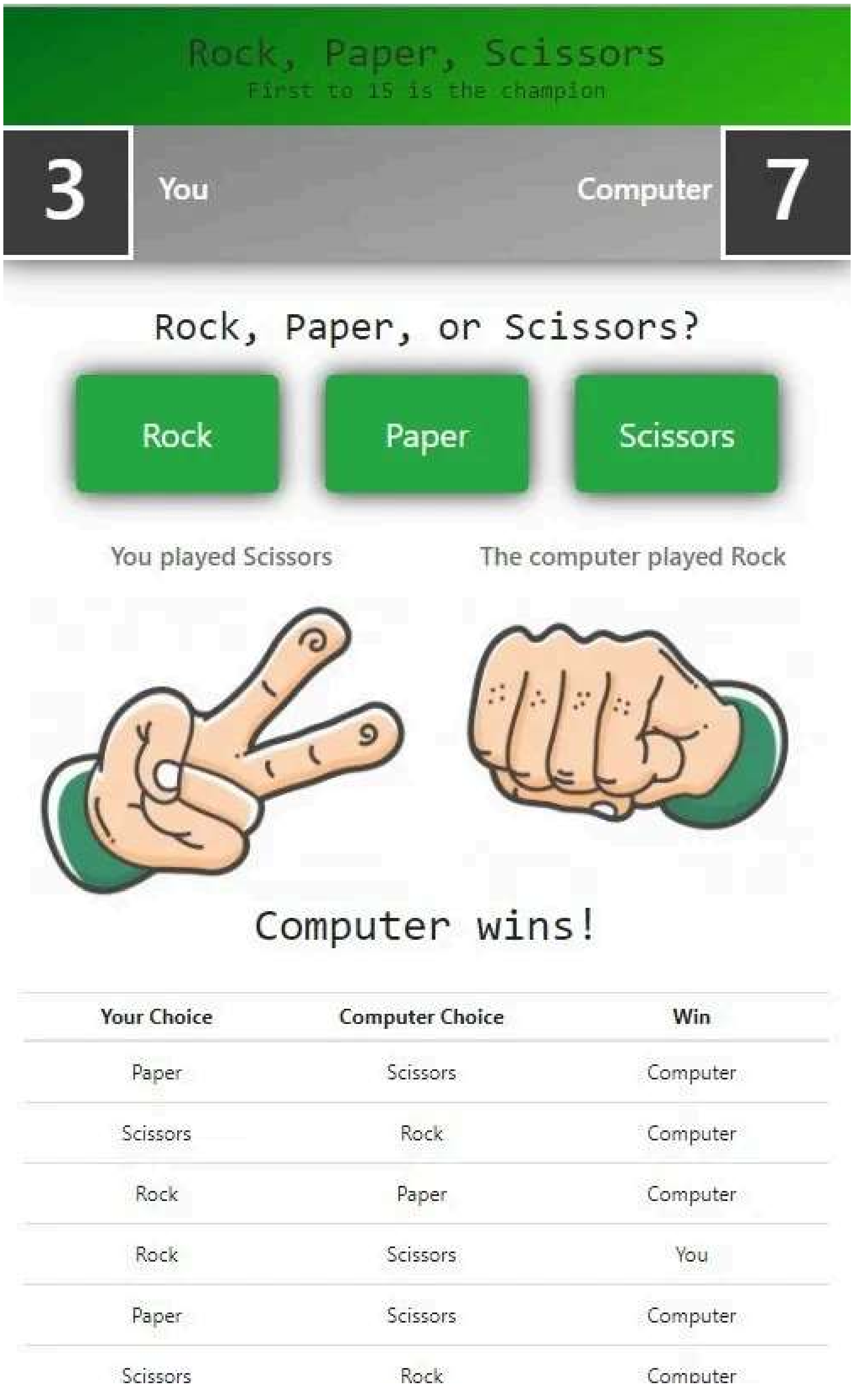
Austin Fischer · Follow

Published in Towards Data Science · 10 min read · Mar 24, 2021

159 1

In this article, I'll walk you through my process of building a full stack Python Flask artificial intelligence project capable of beating the human user over 60% of the time using a custom scoring system to ensemble six models (naïve logic-based, decision tree, neural network) trained on both game-level and stored historical data in AWS RDS Cloud SQL database.

Play the game here



Your Choice	Computer Choice	Win
Paper	Scissors	Computer
Scissors	Rock	Computer
Rock	Paper	Computer
Rock	Scissors	You
Paper	Scissors	Computer
Scissors	Rock	Computer

Watch my video presentation of the project here, or jump to the video by section below

Overview

Video Clip here

Rock Paper Scissors caught my attention for an AI project because, on the surface, it seems impossible to get an edge in the game. These days, it is easy to assume that a computer can beat you in chess, because it can harness all of its computing power to see all possible outcomes and choose the ones that benefit it. Rock Paper Scissors, on the other hand, is commonly used in place of a coin toss to solve disputes because the winner seems random. My theory though, was that **humans can't actually make random decisions**, and that if an AI could learn to understand the ways in which humans make their choices over the course of a series of matches, even if the human was trying to behave randomly, then the AI would be able to significantly exceed 33% accuracy in guessing the player's decisions.

Each "game" in the app is composed of a series of "rounds" where the player and the AI each make a choice between rock, paper, or scissors, and the winner is determined. The game ends when either the player or the computer win a specified number of rounds. My target was to see if I could make the AI aware enough of human behavior to win over 55% of games.

First Iteration

Video clip here

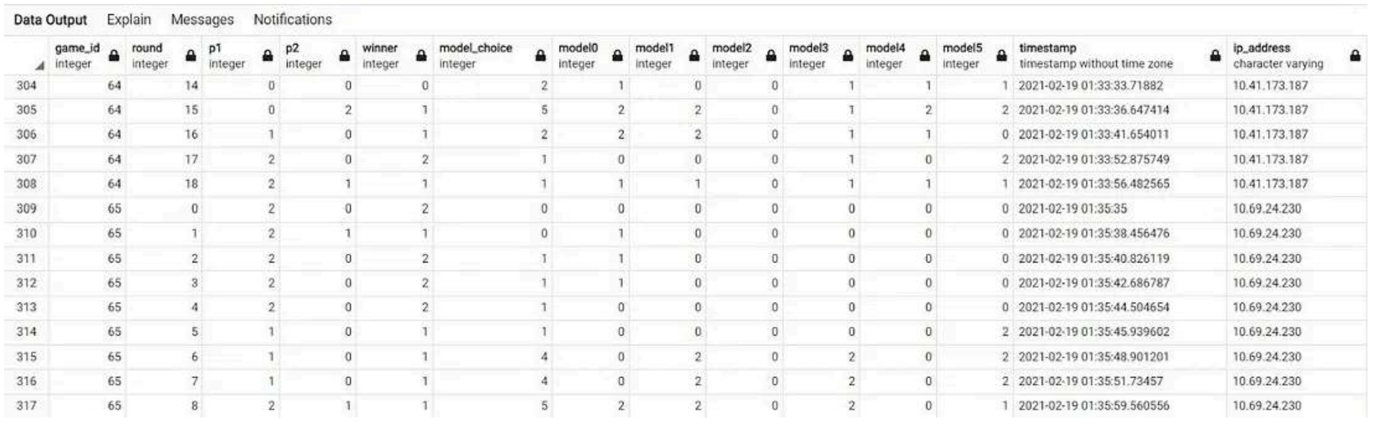
I started out by simply hard coding the different ways that I could think of that humans would make decisions: choosing the same thing over and over, choosing in a pattern, or trying to make the choice that they hadn't used in a while. I built models that would predict the player's next choice if they were using any of these methods, and then used logic-based criteria to try and decide which model fit the player's behavior based on a record of the previous rounds. This was the first stage of the project, ran in a Jupyter notebook, and initially played pretty well. It would fall into certain patterns easily, however, and could be reliably tricked by a savvy player.

At this point, I realized that there were a lot of improvements that I could make and got excited to flesh the project out more. I built a **Flask webapp** and hosted it on **Heroku** so that I could share it with friends. I then built a **cloud database on AWS** to capture the data from every time that it was played, knowing that this data could give me the power to build much more sophisticated models.

I began a process of analyzing the performance of my models and tweaking them. I also replaced the simple logic-based model selection process with a new ensembling system which I'll go into more detail about below. I created a **mobile-friendly app** using bootstrapped HTML, improved the design for a more engaging user experience, and then sent the link out on social media for my friends and family to play against and provide insights. With this data, I began to implement **machine learning models** alongside the **naïve logic-based** ones, which I will also go into more detail about below. At this point, progress slowed as the app's scope increased, and I had to work through package dependency, data quality, backend, and model exportation issues. However, I continued to iteratively update the app functionality in response to data analysis I was performing and new ideas I continued to have to improve it. These days, when I play against the AI, even knowing everything about how it works, I have a hard time beating it.

How the AI Works

Data



Data Output | Explain | Messages | Notifications

	game_id integer	round integer	p1 integer	p2 integer	winner integer	model_choice integer	model0 integer	model1 integer	model2 integer	model3 integer	model4 integer	model5 integer	timestamp timestamp without time zone	ip_address character varying
304	64	14	0	0	0	2	1	0	0	1	1	1	2021-02-19 01:33:33.71882	10.41.173.187
305	64	15	0	2	1	5	2	2	0	1	2	2	2021-02-19 01:33:36.647414	10.41.173.187
306	64	16	1	0	1	2	2	2	0	1	1	0	2021-02-19 01:33:41.654011	10.41.173.187
307	64	17	2	0	2	1	0	0	0	1	0	2	2021-02-19 01:33:52.875749	10.41.173.187
308	64	18	2	1	1	1	1	1	0	1	1	1	2021-02-19 01:33:56.482565	10.41.173.187
309	65	0	2	0	2	0	0	0	0	0	0	0	2021-02-19 01:35:35	10.69.24.230
310	65	1	2	1	1	0	1	0	0	0	0	0	2021-02-19 01:35:38.456476	10.69.24.230
311	65	2	2	0	2	1	1	0	0	0	0	0	2021-02-19 01:35:40.826119	10.69.24.230
312	65	3	2	0	2	1	1	0	0	0	0	0	2021-02-19 01:35:42.686787	10.69.24.230
313	65	4	2	0	2	1	0	0	0	0	0	0	2021-02-19 01:35:44.504654	10.69.24.230
314	65	5	1	0	1	1	0	0	0	0	0	2	2021-02-19 01:35:45.939602	10.69.24.230
315	65	6	1	0	1	4	0	2	0	2	0	2	2021-02-19 01:35:48.901201	10.69.24.230
316	65	7	1	0	1	4	0	2	0	2	0	2	2021-02-19 01:35:51.73457	10.69.24.230
317	65	8	2	1	1	5	2	2	0	2	0	1	2021-02-19 01:35:59.560556	10.69.24.230

Data is recorded during the course of the game and kept in a local SQLAlchemy table that is passed to the models every turn to aid them in their respective decision making processes. It includes all of the above columns except for game_id and ip_address. At the end of every game, data is sent to an AWS cloud hosted database with those two columns appended, which can be accessed to assess model performance and train ML models.

Ensembling

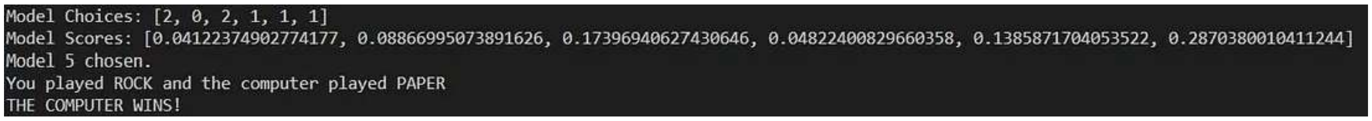
Video clip here

Every round, the computer_choice function chooses which model it will use as the AI's choice for the coming round. This is done by scoring the performance of each model given the current game record.

$$\text{score} = \frac{\sum_{j=0}^{n} a_j \cdot j^2}{\sum_{j=0}^{n} j^2} \quad \textit{where } a_j = 1 \textit{ if model won, } a_j = -1 \textit{ if model lost, and } a_j = 0 \textit{ if model tied}$$

This scoring system is quadratic in order to prioritize recent model performance, making it responsive to strategies that a player might be using.

Firstly, this allows the model to overcome simple patterns that a player may be using, such as playing the same choice constantly or switching choices in a repeated pattern. Secondly, it ensures that the same model is not consistently repeated, which would allow for the player to figure out how it worked and beat it. If the player did figure out a model, it would lose and quickly earn a negative score due to the priority that quadratic scoring places on the most recent rounds. And finally, it allows the models that may more accurately predict the player's thinking to be used more frequently. Here is an example of what the backend looks like over the course of a round:



```
Model Choices: [2, 0, 2, 1, 1, 1]
Model Scores: [0.04122374902774177, 0.08866995073891626, 0.17396940627430646, 0.04822400829660358, 0.1385871704053522, 0.2870380010411244]
Model 5 chosen.
You played ROCK and the computer played PAPER
THE COMPUTER WINS!
```

Naïve Models

Video clip here

Each of the models takes a different approach to understanding how a player will make their next choice. The first four models make very specific assumptions about the way a player is making choices, and thus are only applicable in the context of a player acting in that given way.

- **Model 0**: Chooses the choice that would lose to or beat the player's previous choice. Based on the assumption that a player will continue to either not repeat or to repeat previous choices.
- **Model 1**: Vector-based choice based on past three rounds. Based on the assumption that a player will be using a certain pattern (ex. Rock, Paper, Scissors, Rock, Paper, Scissors, etc.)
- **Model 2**: Chooses the choice that would beat the player's most frequent recent choice. Based on the assumption that a player has a certain choice

that they keep making.

- **Model 3**: Chooses the choice that would beat the player's least frequent recent choice. Based on the assumption that the player will try to play the choice they haven't played in a while.

With the ensembler built to favor models that are predicting accurately in recent rounds, these models will eventually rise in score to become the deciding model in the computer's choice if the human player repeats the behavior on which their assumption is built enough times.

Machine Learning Models

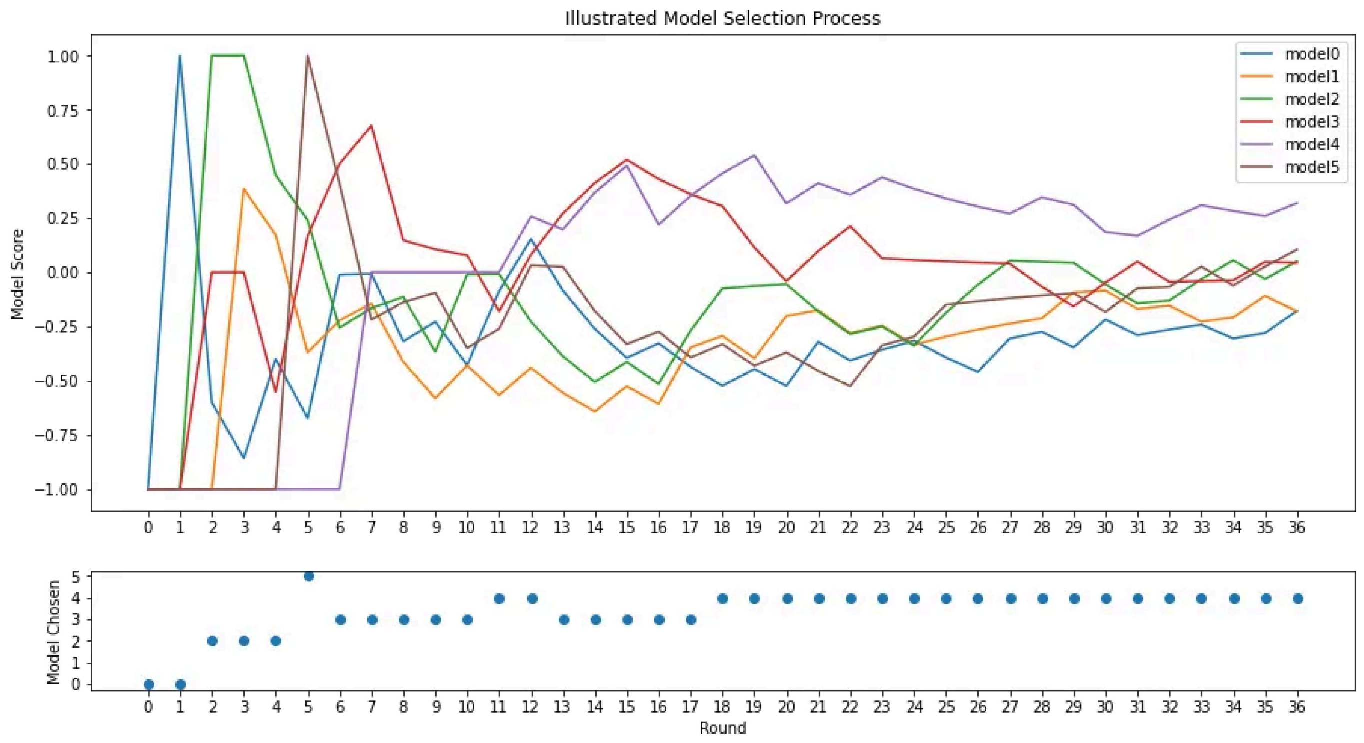
The last two models are more sophisticated, one being a decision tree and the other being a neural network, both trained on the historical dataset of games played against the AI. These models take longer to kick in to the AI's decision making process because they need to see at least 7 rounds and 5 rounds of player data respectively to feed into their ML models as input. These models are made aware not only of the player's choices from previous rounds, but also the computer's choices, who the winner was, which model the AI used to make its choices, and each individual model's choices for the previous rounds.

- **Model 4**: Uses a pickled scikit-learn neural network model trained on historical data to predict and beat the player's next choice based on data from the previous 7 rounds.
- **Model 5**: Uses a pickled scikit-learn decision tree model trained on historical data to predict and beat the player's next choice based on data from the previous 5 rounds.

Beginning the game using the naïve models helps to solve the cold start problem, where the AI knows nothing yet about the player who it is starting a new game against. (The only use of a random number generator is for the very first round, as it is necessary to make the AI not entirely predictable, otherwise random numbers were entirely avoided in the building of this app.) The initial AI is therefore doing the best it can by its logic-based models until it gets a better understanding of the player over a number of rounds, and can turn on its more intelligent models. See the figure below to understand how models were chosen by their scores over the course of one game on record.

Ensembling Model Selection Visualization

Video clip here

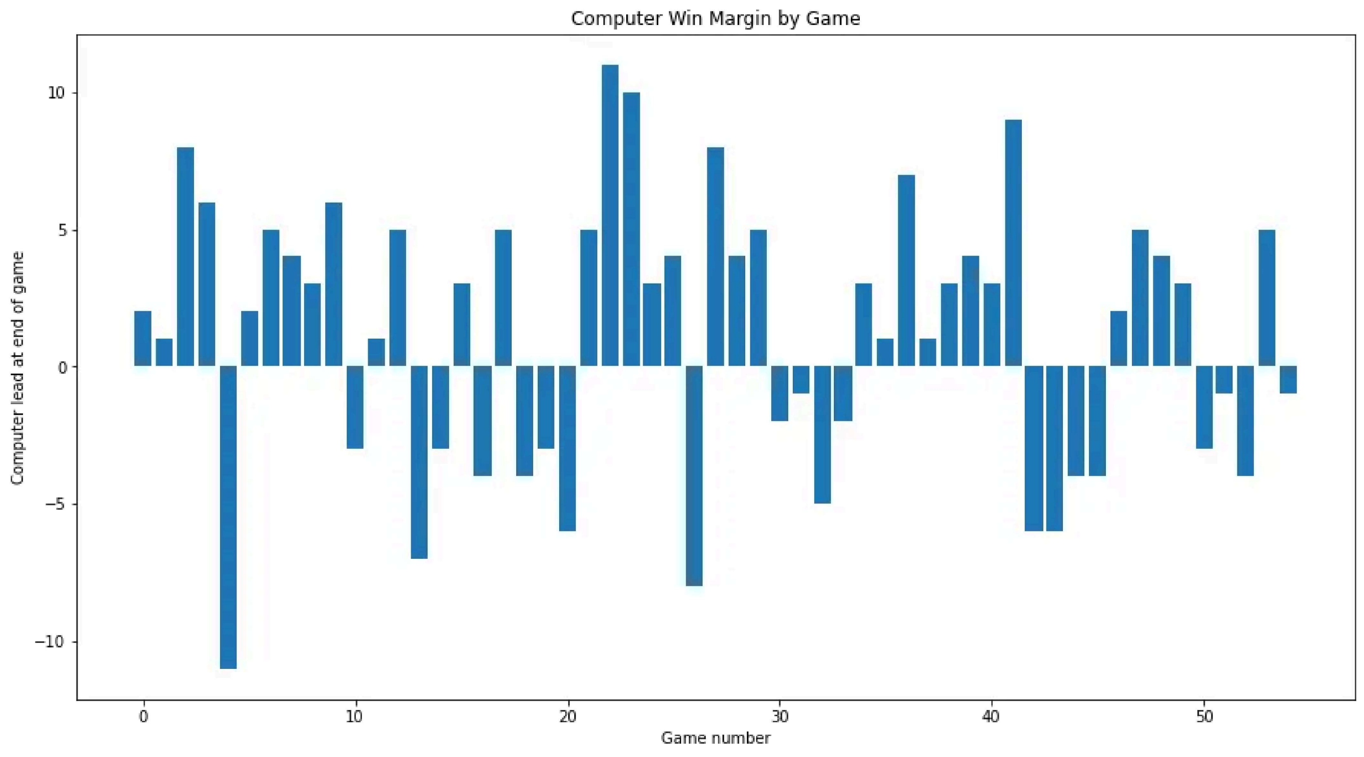


This image is meant to illustrate the model selection process on a sample game, and not to represent the general efficacy of the individual models.

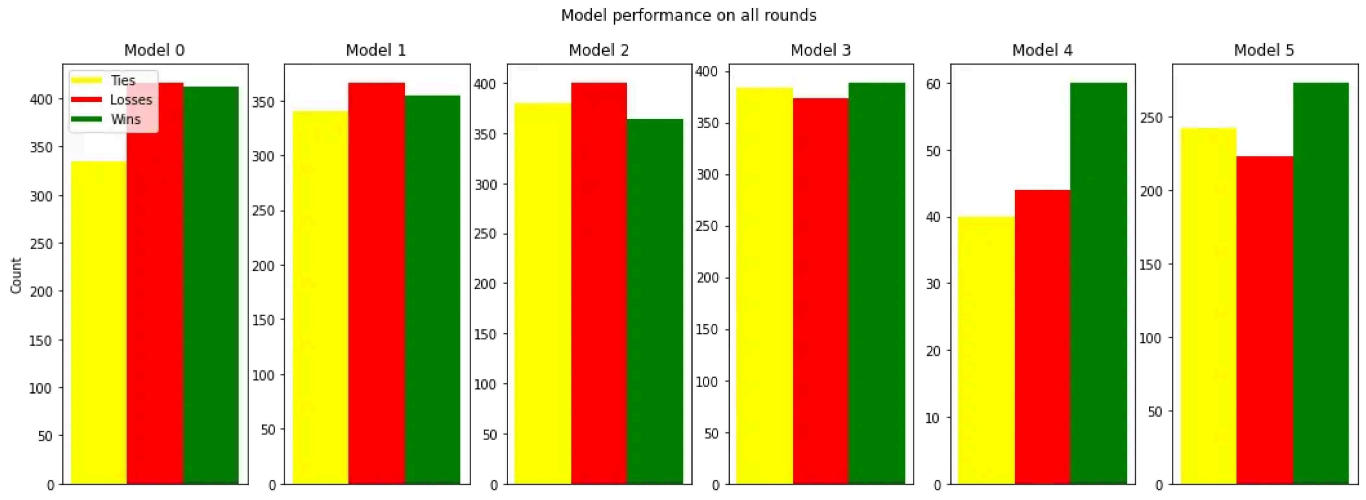
Statistics

Video clip here

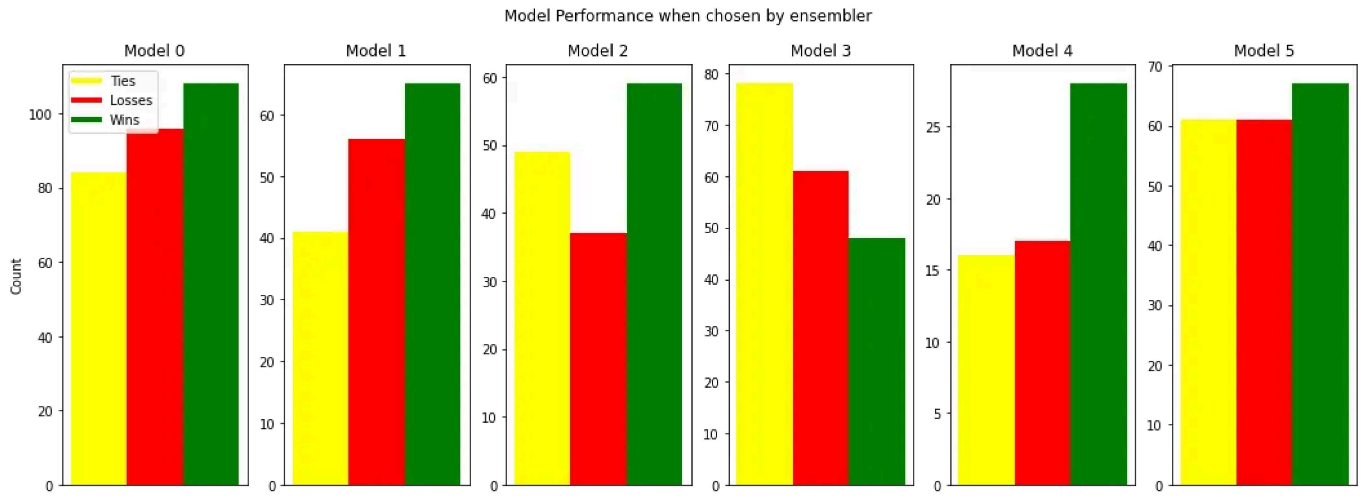
My goal was to have the AI win over 55% of the time. Currently, the AI's win percentage sits at 61.8%.



To analyze how the models perform, we will look at two datasets: one that shows the result of the model's choice in every round against what the player actually chose that round, and the next showing how the models actually performed against the player's choice on rounds that the ensembler chose that model. As can be seen below, the top performing model on all rounds (judging by win / loss ratio) is Model 4, which has made the winning choice 203 times and the losing choice 168 times. The second best is the decision tree model at Model 5, which has made the winning choice 273 times and the losing choice 223 times. Note: these model scores are only calculated for the most recent iteration of the model, thus the lower total counts for the neural network at model 4.



Looking only at the rounds in which models were chosen by the ensembler, the top performing model is also the neural network Model 4 with 28 wins and 16 losses, followed closely by Model 2 with 59 wins and 37 losses. The naive models 0–2 can be seen to have a much higher win/loss ratio when they are chosen than they do across all rounds, which means that the ensemble is choosing the correct times to play the models in general. Comparing the two figures, the ensembler seems to use Model 2 especially effectively, while Model 3 actually experiences a much lower win/loss ratio when it is chosen by the ensembler, meaning that it is not being selected at the correct times.



Takeaways

This AI has been able to outperform my expectations by having two important characteristics. Firstly, it is capable of beating a player with its naive models when they fall into a simple pattern, and by its complex models when they might be following a more complex, but still measurable, decision making process. And secondly, it is itself hard to predict. As the person who programmed it, I still cannot predict what its choices will be when I am playing against it; a fact that came to be true especially after introducing the machine learning models. Despite the fact that there is no randomness whatsoever in its decision making after the first round, it is nonetheless sufficiently beyond the player's ability to calculate what the AI will choose.

The ensemble of models that powers this AI is an example of a framework that can be used for AI decision making in real time where data bear weak correlation to each other and most recent trends need to be prioritized over averages of the whole dataset.

Future ideas:

Some ideas to improve the AI:

- Hosting competitions to build a larger dataset for training and validation of new ideas
- Test an ensembler using a weighted average of model choices given their scores
- Cluster player behavior into categories (ex. calculating, risky, random, etc.) and then train ML models to each cluster

- Build a meta layer above the ensembler that learns to enact multi-round strategies through reinforcement learning

Another possibility would be to turn the app into a platform where data scientists or students can compete in building Game AI and trying to beat their peers' AI and move up a leaderboard.

Hopefully the methods that I've used in building this project can provide some inspiration in your own ML projects, and if you'd like to view any of my source code, the check out the GitHub repo for the project.

Play the game here

All images used in this article are created and owned by the author

Machine Learning Artificial Intelligence Software Development Data Science
Projects