CS557 Project 2

**What was the experimental setup?**

I initially started with the suggested setup of copying the RBotRandom over to my java file bestPlayer and then proceeded to try different approaches. The first few ideas I had were to first get the number of possible guesses for each player then find the ones with only 1 possible guess and then remove that possibility from any other guesses with the same guest. This was a baby step to then lead into the other probability calculations. I got better results than the random guess and while using my model I lost a major percentage of accuracy when decoding (I think) resulting in better results than my ML model. The next setup was to get a way to run and test all the results against each other while trying to improve the probability guess results. I made a method in my bestPlayer class that outputted the players KB and the random results and the probability results. I then proceeded to make a python file that called the game suspicion and parsed the output and then used the ML model to generate the models guesses. After having all the guesses generated, I used NumPy mean calculation to compare all the results means.

**What method was used to calculate the probabilities?**

The method used to calculate the probabilities changed drastically while making, improving, and fixing the probability calculations. I initially was doing the calculations all with counting the guesses in loops and using up to 9 variables stored in an array list to store each value within a single players guest guess. This seemed to work, but I am positive on the math and accuracy that was implemented. While looking around on forums for help, I found a large amount of usages with javas built in functions to determine the permutations, probabilities and even predations.

After I found usages of the built-in functions for calculating permutations and probabilities within Java, I went through a large amount of trial and error to use these in my own predictions. In this process I found most usages and implementations were very compact and hard to follow. I was unfamiliar with the usage of lambda, Stream, and Collections in Java which almost all the implementations I found using Javas functions to calculate permutation and probabilities used most of these. I proceeded to familiarize myself with these features in Java in order to try and apply the Java functions to my guess method.

Once I applied the permutation to my method, I was getting similar results to when I calculated them by looping through all possible orders of the guesses. On the other implementation of using the built-in functions of Stream with the permutations, I was able to almost double my initial calculations accuracy. Then I realized that I was also calculating permutations of zero with my results and found dropping them increased the performance.

The idea was to weed out the guesses with a known guess (only one guest) from the other players guesses and then find the lowest permutations within the possible guesses.

The last step was only keeping the highest permutation/probabilities for my results and inserting them in a formatted list which I could report back to the game functions.

**Problems?**

My initial problems arose from my lack of understanding with other methods in the calculations for probabilities. I would have had less problems but also less accurate results (unless I found the reasoning why manually calculating the permutations were worse) without using Javas functions. I solved most of these by researching online with the collection of class notes on calculation dependent variables.

The other major problems I faced was using my ML model which was written using Python to calculate the guesses using the same results as the random guess and probability guess. I wanted to test the results using the same game outcomes to have a fair representation. I tried initially to call my python from java but was unsuccessful and ended up making a method to print out the players KBs, random results, and probability results to the system which I then called the Java file from Python and parsed the output. I was able to then run my model on the same game KBs as the other random and probability guesses. I finally formatted all the results to the same format which allowed me to calculate the means on each result to test and compare results.

The last problem was one of the longest but also unrelated to P2. It was the same issue I had in P1 with transforming the encoded data from my ML model back into the format to get the results. I am not sure if I ended up getting this correctly as I can view all the probabilities and see that it is vary close to being right with a mean\_squared\_error of 0.02 and then when I decode the data it is only about 50% accurate. This does not seem right, and my need revisited.

Also, good to note that my ML model is having an error of losing the “known” players guess. If there is only one guess on a player guest, the other models choose that because it has a 100% probability and the random choice only chooses from one choice. I believe this is in the process of decoding, but it may be an error on my models’ design all together.

**How were the results?**

I have run maybe a hundred games and seen results varying for each method (random, probability, and ML model) but with each test the random and ML model are similar with the random being more accurate most the time. As for the probabilities results, I saw it was more accurate that both the random and ML model every time. The probabilities guesses vary from around 60% accuracy to 95% accuracy with an average around 30% vs the other two average around 50%. I tried this on 5 players and 6 players which my model seemed to do better than random but still not better than the probability guesses. I have converted the probabilities guess into the same encoding as my model which when comparing the mean\_squared\_error appeared to be very close to the same as the models. I still do not know why when using the oneHotEncoder invers\_transformation (decoding) causes the model to lose a large scale of accuracy as I did not test this part in P1.

**Outside credit?**

Oracle.com for a the usages of predicate.java, Function.java, BiFunction.java, Consumer.java, and a few other built in functions for java that I used through my method mapGuessProb() in bestPlayer.java.

Geeksforgeeks.org and stackoverflow.com for implementing certain java functions and using lambda, Maps, Stream, and Collections in java.

Baeldung.com for guides on Stream.

**Run the same test I did**

You should be able to run the file temp.py after compiling the game Suspicion.java and see the three results.