Mind Reader

Final Project Report

6.883 Online Methods in Machine Learning

Guy Satat

1. Problem Statement

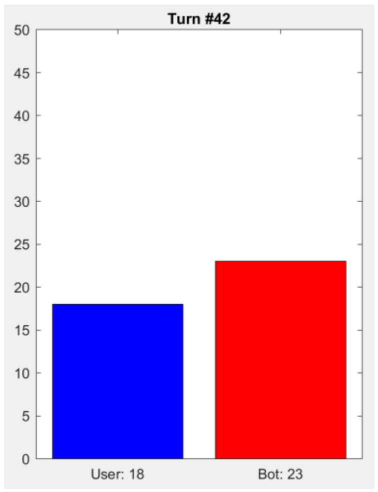
The mind reader machine developed here is motivated by Shannon's "A Mind-reading(?) Machine"(1953) and Hagelbarger's "SEER, A SEquence Extrapolating Robot"(1956). In this game the user is playing against the machine, the user selects a bit (in this implementation it's the right and left keys) and the machine is trying to predict which key the user will click. If the machine guesses correctly it gets a point, and otherwise the user gets a point.

If both players make their selection from a "real" i.i.d. process, they have equal chance of winning. However, for humans it's extremely hard to produce i.i.d. sequences, this is exactly what the machine is aiming to exploit.

2. Game Tutorial

The game is developed in Matlab. To play the game run the *Script.m* file. In that file you can choose the game target (default value is 50), and to ask the computer to generate i.i.d sequence for the user (this will be discussed later).

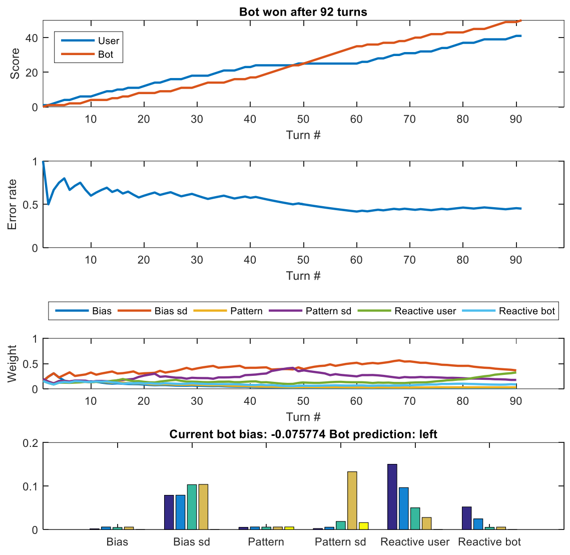
Once the game is lunched the user should click either the left or right keys to make his selection, the computer is making its selection in the background. The current score of the game is presented in a Matlab figure, as seen below.



The figure shows the current scores towards the game's target.

During the game the user can exit any time by hitting 'q'. The user can also choose the cheat by hitting 'c', this will open a second figure with current algorithm status and algorithm decisions.

The game ends when the first player reaches the game goal, then the game summary figure is shown (the same figure is shown during “cheating” mode).



Game summary figure. From top to bottom: The time evolving score. The algorithm’s error rate. The exponential weights evolution for aggregated predictors based on type. The current exponential weights for all predictors, the title provides the decision bias and the actual prediction.

3. Algorithm Description

The algorithm which defines the computer’s move is based on the Experts Setting as we learned in the class. The experts are a set of predictors (described later) which attempt to predict the next user move based on previous moves. The next subsection describes the various predictors, and will follow by a description of the exponential weights algorithm to aggregate these predictors.

First, we define some notations. Right key is assigned $+1$, left key is assigned -1. The user strokes is denoted by $u(1..n)$, and the algorithms predictions (strokes) is defined by $b(1..n)$, n is the current game turn. T is the game target.

The predictors can operate on the immediate keystrokes sequence $u(1..n)$, or on the flipping sequence defined by:

$$f(n) = \begin{cases} 1 & u(n) == u(n-1) \\ -1 & else \end{cases}$$

this sequence indicates flipping of bits, rather than the bits themselves.

3.1. Predictors

3.1.1. Bias Predictor

The bias predictor tracks the user strokes and searches for specific bias in the sequence (for example tends to hit more right key). This predictor can be tuned to search for bias in a given history length m, this allows to create multiple predictors each searching for bias with different memory. The expert prediction is defined by:

$$P_{B,u}(n+1) = \frac{1}{m}\sum_{i=1}^{m} u(n-i)$$

In my implementation I used four Bias Predictors with memories: $m = 5, 10, 15, 20$.

3.1.2. Flipping Bias Predictor

This essentially the same implementation of the Bias Predictor, but this predictor searches for bias in flipping sequence (for example hitting the right key k times followed by k times left key has zero mean, but the flipping sequence shows significant bias to hit the same key multiple times). This predictor can also operate on a given history m, such that:

$$P_{B,f}(n+1) = u(n)\frac{1}{m}\sum_{i=1}^{m} f(n-i)$$

In my implementation I used four Flipping Bias Predictors with memories: $m = 5, 10, 15, 20$.

3.1.3. Pattern Predictor

This predictor detects keystroke patterns (for example $[R, L, L, R, L, L, \dots]$, where R, L stands for right and left keys respectively). It searches for pattern length m .This predictor operates by the following procedure:

```
1. $pattern = u(n - m \,..\, n)$
2. $i = n,\ \ score_u = 0$
3. $while \quad pattern == u(i - m..i)$
4.         $score_u = score_u + 1$
5.         $pattern = shift(pattern)$
6.         $i = i - 1$
7. end
```

where $shift(\cdot)$ produces a cyclic shift of the input.

The prediction is defined by:

$$P_{P,u} = u(n-m)\frac{\min\{score_u, 2m\}}{2m}$$

where the score defines the confidence of the prediction, here I defined an ideal case as a situation where the pattern repeats itself two times.

In my implementation I choose to search for patterns of lengths: $m = 2,3,4,5,6$.

3.1.4. Flipping Pattern Detector

This predictor is implemented exactly as the previous predictor, but it operates on the flipping sequence instead (it searches for sequences such as: $[S, S, F, S, S, F, \dots]$, where S, F stands for same key and flipping respectively). The scoring algorithm operates exactly the same as the regular pattern predictor, with replacing the sequence $u(1..n)$ by $f(1..n)$, and produces $score_f$. The prediction is then:

$$P_{P,f} = u(n-m)\frac{\min\{score_f, 2m\}}{2m}$$

Similar to the previous predictor, the patterns lengths searched for are: $m = 2,3,4,5,6$.

3.1.5. Shannon Inspired Predictor (user reaction predictor)

In his paper, Shannon proposed the user reacts to winning and losing. He proposed the user reactions can be characterized in 8 ways:

1. User wins, played the same, wins again, he may play the same of differently.
2. User wins, played the same, losses, he may play the same of differently.
3. User wins, played differently, wins again, he may play the same of differently.
4. User wins, played differently, losses, he may play the same of differently.
5. User losses, played the same, wins again, he may play the same of differently.
6. User losses, played the same, losses, he may play the same of differently.
7. User losses, played differently, wins again, he may play the same of differently.
8. User losses, played differently, losses, he may play the same of differently.

While previous predictors ignored the winning / losing aspect of the game, this detector introduces this emotional aspect. In the context of previous predictors, this predictor operates on the flipping sequence $f(1..n)$ and on the user win/loss sequence defined by:

$$w_u(n) = \begin{cases} 1 & u(n) \neq b(n) \\ -1 & else \end{cases}$$

In my implementation I extended the predictor to operate on user with variable memory (Shannon's proposal assumed the user has a memory length of $m = 1$), for example the user might have a longer memory. This predictor is implemented by keeping a state machine to track how the user played the last time he was in a similar situation, the total number of states is just 2^{2m+1}. Here, m counts the memory of results to previous player attempts, and the extra 1 counts first win/loss.

Each time the user plays, the predictor updates the state machine S_m by the following procedure:

```
1. Based on $f(n-m-1 \dots n-1)$, and $w_u(n-m-2 \dots n-1)$ map to the appropriate state
   index $i$.
2. If $S_m(i) == 0$ then:              %i.e. no meaningful history
       $S_m(i) = 0.3\, f(n)$                     %update the state with low score
3. Else if $S_m(i)\, f(n) == 0.3$       %i.e. this is the second time the user is doing the same thing
       $S_m(i) = 0.8\, f(n)$                     %update the state with higher score
4. Else if $S_m(i)\, f(n) == 0.8$       %i.e. this is the third time the user is doing the same thing
       $S_m(i) = f(n)$                           %update the state with highest score
5. Else                                 %i.e. the user played different then the prediction
       $S_m(i) = 0$                              %update the state with zero confidence
```

The motivation behind these updates is to increase confidence of the decision if the user reacts the same way as he did previously when encountered a similar situation.

Finally, the prediction of the next step is defined by: $f(n-m..n)$, and $w_u(n-m-1..n)$ map to the appropriate state index i, and predicting $P_R(n+1) = S_m(i)$.

In my implementation I choose these memory lengths: $m = [0,1,2,3]$. The $m = 0$ case corresponds to a user that purely reacts to winning or losing, without memory of what brought the win/loss (for example whenever losing flip the decision). While the longer memory predictors include the shorter ones, due to the non-linear update when the user reacted different than expected, the short memory predictors would react faster to change in behavior.

3.1.6. Hagelbarger's Inspired Predictor ("emotional" bot algorithm)

Hagelbarger proposed a very similar algorithm to Shannon's, basically, he proposed the machine should react to winning or losing and play the next step based on the last it encountered such a case. Essentially this is exactly the same implementation as the previous approach, but operating on the machine's strokes and win/loss sequence:

$$b_f(n) = \begin{cases} 1 & b(n) == b(n-1) \\ -1 & else \end{cases}$$

$$w_b(n) = \begin{cases} 1 & u(n) == b(n) \\ -1 & else \end{cases}$$

For this implementation I used the same memory lengths as the previous example.

3.2. Predictors aggregation, and algorithm decision

There are a total of 26 predictors in my implementation (the number can easily be changed by adding or removing memory lengths, this can be done in the *game_parameters.m* file). The predictors results are aggregated with the exponential weights algorithm. We define an index $j = 1..N$ to run over all $N = 26$ predictors, such that $P_j(n)$ is the $j - th$ expert prediction for time n. First we use "soft-max" to weigh all the predictors based on their performance:

$$W_j(n+1) = \frac{\exp\{-\eta \sum_{s=1}^{n} |u(s) - P_j(s)|\}}{\sum_{j=1}^{N} \exp\{-\eta \sum_{s=1}^{n} |u(s) - P_j(s)|\}}$$

with: $\eta = \sqrt{\frac{\log N}{2T-1}}$.

This allows to choose a bias for the current decision:

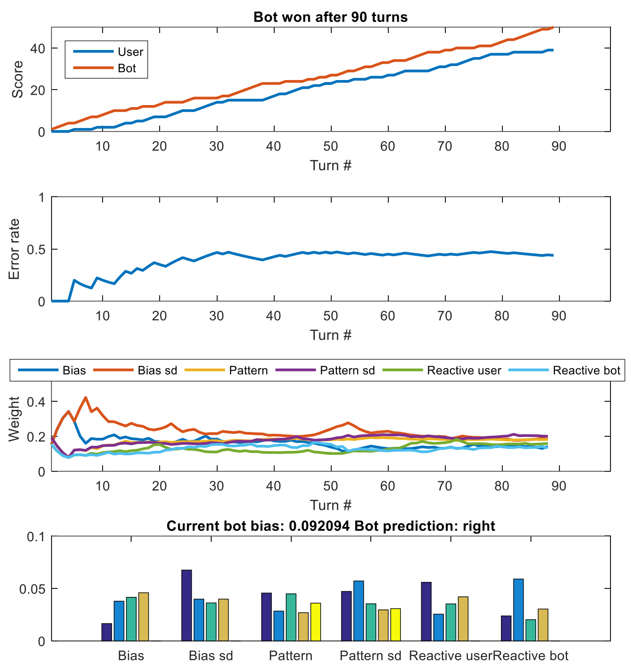
$$q_t(n+1) = \sum_{j=1}^{N} W_j(n+1)\, P_j(n+1)$$

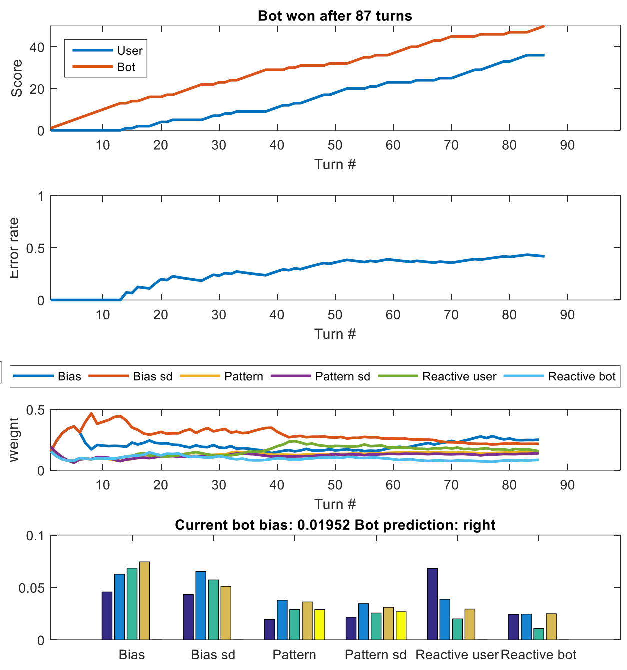
Finally, the algorithm takes a random decision with bias q_t such that:

$$b(n+1) = Rademacher\ with\ mean\ q_t$$

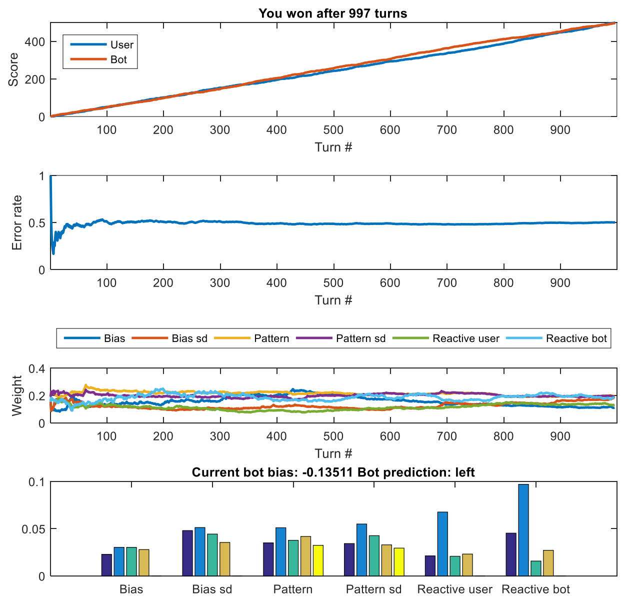
4. Results

The game was played by several users, here are a couple examples of the final results:





Here are results for a computational random user playing against the algorithm (game target is 500):



We note that the game lasted for almost the maximum duration (997 turns out possible 999), that the error rate is 50%, and that there is no preferred expert.

5. Brief Code Overview

Here is a short description of the code structure:

1. *script.m*
   Starts the game, it allows to choose the game length and whether the user should play or the computer should use an i.i.d user to compete with the algorithm.

2. *game_parameters.m*
   Contains a list of predictors' memories, use this file to add and remove experts.

3. *game.m*
   Manages the game, including main game loop, asking the algorithm and user to play, display results etc.

4. *getkey.m*
   Handles keyboard interface (written by Jos van der Geest, downloaded from Matlab Central).

5. *bot.m*
   Manages the algorithm, calls all experts to ask for predictions, runs the exponential weights algorithm.

6. *bias_detector.m*
   A class to implement a bias detector, has a memory parameter, and a flag to denote if it operates on the $u(n)$ or $f(n)$ series.

7. *pattern_detector.m*
   A class to implement the pattern detector, similarly has a pattern length parameter, and a flag for the $u(n)$ or $f(n)$ series.

8. *reactive_detector.m*
   A class to implement the reactive user, has a memory length parameter and keeps the corresponding state machine. Receives as input either the user $f(1..n)$, $w_u(1..n)$ sequence (Shannon like) or the algorithm strokes $f_b(1..n), w_b(1..n)$ sequences (Hagelbarger like).

6. Future Work

There are several things to explore in the context of this work:

- Predictions:
    - The Shannon and Hagelbarger detectors can be implemented on the direct key strokes sequence.
    - All detectors can be explored with other parameters (memory length, ways to score the predictions).
    - It would be interesting to explore other reactive based predictors (how people respond to winning or losing).
    - Reducing the number of experts to expedite learning (decreasing $\log N / 2T$).

- User behaviors:
    - It seems that users behave different the first time they play the game and from a (not statistically meaningful) observation it appears they do better the first time they play. I suspect this is because they are less engaged the first time, and so can produce more random sequences.
    - It would be interesting to explore and understand how users react to this game, for example do people have some common behavior model (similar to what Shannon suggested). Putting this game online can help to evaluate this.
- Other ways to perform the task:
    - Is it possible to solve this problem in the context of Cover's algorithm? What are the desired sequences? Can we learn them from many players?
    - If the user produces non-random sequence, it means it's compressible, that can allow to predict the next bit.