Motivation. The Mind Reader Bot (http://www.mindreaderpro.appspot.com) tries to predict if you will click either the left arrow or the right arrow, given previous clicks. If it predicts correctly, the computer advances, and if not, you advance. Clearly, a random number generator would roughly tie the computer’s prediction. But humans can’t usually win, because we’re terrible random number generators! We instead want to develop a principled way to predict these bit sequences, perhaps even by combining a number of algorithms to create a meta-algorithm to adapt to anyone’s mind.

1 Bit Prediction

Given a sequence \(y_1, y_2, \ldots\) of bits, we’d like to predict the sequence. If the sequence is i.i.d. Bernoulli, then it’s best to just predict the majority of the prior inputs. The asymptotic proportion of correct predictions is

\[
\max(p, 1 - p)
\]

We know this because by the Central Limit Theorem, the average \(\bar{y}_t\) over \(y_i\) approaches \(p\). But in fact, we can make a similar claim for any sequence:

**Claim 1.1.** For any arbitrary sequence, there is a method that ensures we perform as well as the imbalance of 1’s and 0’s in the sequence.

Figure 1: An image of the mind reader bot, where the bot is far ahead of the user.
So as long as the sequence is unbalanced, we can predict it with better than 50% probability. Thus, to win against the computer it’s advantageous to use balanced sequences because otherwise the computer can exploit the imbalance.

First, we can show that a simple majority algorithm will not work. One easy counterexample to consider is an alternating sequence of 0’s and 1’s.

Instead, we present David Blackwell’s solution [Bla95]. Let \( L_t = (\bar{y}_t, \bar{c}_t) \) be the proportion of 1’s and the proportion of correct predictions. The quadrant of \( \bar{y}_t, \bar{c}_t \) determines the strategy. If you are in the top quadrant, your \( \bar{c}_t \) is greater than the maximum of \( \bar{y}_t \) and \( 1 - \bar{y}_t \) and so you are set. If you are in the left quadrant, you predict 0, and if you are in the right quadrant, then you predict 1. If you are in the bottom quadrant, you draw a line from \((0.5, 0.5)\) to \( L_t \), and output 1 with probability equal to the intersection of the line and \( y = 0 \), see Figure 2 for more details. Note that this is a mixed strategy – no deterministic strategy can work!

![Figure 2: An image of David Blackwell's quadrants.](image)

The Bayes predictor [Bla95] works well for almost all sequences, and it’s much easier to prove good convergence, but it may be thwarted by certain sequences (e.g. the alternating sequence). In the online learning framework, we want results to hold for arbitrary sequences. If the data are in fact i.i.d (amenable to Bayes prediction, and not really “learnable”), learning-based approaches usually still give good results.

### 2 Potential-based Methods

How do we deal with prior assumptions? In the static prediction paradigm, you predict probabilistic models based on the data, estimate the model parameters, then make predictions. In the online prediction paradigm, you often go directly to data prediction, because in some sense there’s no model to estimate.

**Example.** Suppose you have a binary matrix that determines when a certain user like a certain movie. We make sequential predictions whether a user likes a movie, and our mistakes are corrected in real time. Is there a principled method to make few mistakes? Because of similarities between
users, the final matrix will likely be near to a low-rank matrix – but how do we use this “prior” information?

In online supervised learning, we aim to predict \( y_1, \ldots, y_n \), where at each time \( t \) we observe some side information \( x_t \), predict an outcome \( \hat{y}_t \), and observe the real outcome \( y_t \). If the sequence has “structure”, \( \sum_{t=1}^{n} |\hat{y}_t - y_t| \) should be small. Using the prior knowledge of the problem, we want to design a function \( \phi \) of the sequence such that the error is bounded by \( \phi(x_1, y_1, \ldots, x_n, y_n) \). To do better on some sequences, you must do worse on some other conditions, by basic probability. In fact, this is a sufficient condition to choose the function \( \phi \).

For our low-rank movie matrix example, we have an updated formulation for this \( \phi \):

\[
\sum_{t=1}^{n} |\hat{y}_t - y_t| \leq \min_{w \in \mathcal{F}} \sum_{t=1}^{n} |\langle w, x_t \rangle - y_t| + C_n(\mathcal{F}; x_1, \ldots, x_n).
\]

This essentially states that you want to do better than if you knew the data before and picked the best low-rank matrix \( w \) for the data, plus a complexity term \( C_n \). Accordingly, the first term captures the \( x/y \) relationship, and the second accounts for the complexity of the \( x \)'s.

**Potential functions.** It was shown recently that the space of achievable \( \phi \)s can be characterized by martingale inequalities [Rak19]. Optimal predictions are of the form

\[
\hat{y}_t = \frac{1}{2}((U(x_1, y_1, \ldots, x_t, +1) - U(x_1, y_1, \ldots, x_t, -1))
\]

for some potential function \( U \) that can be derived from \( \phi \). This is as easy as it gets: we compute two values and subtract them! There’s no optimization required, as compared to SVMs or other learning methods. In addition, this works for arbitrary lengths of data, and there’s no need to rely on an i.i.d. nature of data.

**Back to Matrix Completion.** In the movie matrix completion problem, there’s an efficient prediction strategy such that for any sequence of user-move pairs and ratings,

\[
\# \text{ errors} \leq \# \text{ errors of best rank-} r \text{ explanation} + O(\sqrt{rd_1d_2(N_c \wedge N_r)})
\]

where \( d_1 \) is the number of users, \( d_2 \) is the number of movies, and \( N_c, N_r \) are the most frequently presented movies and users, respectively. And if the sampling is roughly uniform, we get back statistical bounds under the i.i.d assumption.

### 3 Predictable Sequences

In online linear optimization, you do not observe any side information \( x_t \), but predict a vector \( \hat{y}_t \in \mathcal{K} \subset \mathbb{R}^d \), and observe the correct vector afterwards. Similarly to the above, the best we can hope for is:

\[
\sum_{t=1}^{n} \langle \hat{y}_t, y_t \rangle \leq \min_{w \in \mathcal{K}} \sum_{t=1}^{n} \langle w, y_t \rangle + C_n(\mathcal{K}; y_1, \ldots, y_n).
\]
Example. Prediction with expert advice or weighted majority. Here \( \hat{y}_t \) is a probability distribution of \( d \) experts, \( K \) is a simplex, and \( q_t \) are losses of experts. For example, on each round you might have \( d \) stock experts, and each gives you a a prediction. You only care about their loss function, so you can “massage” the situation back to the above setup.

4 No-Regret Methods

We can modify the above function \( C \) to make use of a machine learning method. We’d like to come up with an algorithm that would do as well as any algorithm would expect to do, in hindsight. Adding a machine learning model gives you a way to hedge.

4.1 No-regret dynamics for zero-sum two-player games

In zero-sum two-player games, let \( M_{i,j} \) and \( -M_{i,j} \) be a \( d_1 \times d_2 \) matrix indicating the cost for player 1 (resp. player 2) when the two players choose their moves as \( i \) and \( j \). At each step, player 1 picks \( q_t \in \Delta_{d_1} \), and Player 2 picks \( p_t \in \Delta_{d_2} \) (one a row, one a column). The first player receives \( M p_t \), while the second receives \( q_t M \).

The minimax value is:

\[
\min_{q \in \Delta_{d_1}} \max_{j \in [d_2]} q^T M e_j = \max_{p \in \Delta_{d_2}} \min_{i \in [d_1]} q^T M p = \min_{q \in \Delta_{d_1}} \max_{p \in \Delta_{d_2}} q^T M p.
\]

If both players employ a no-regret strategy, then

\[
\frac{1}{n} \sum_{t=1}^{n} q_t^T M p_t - \min_{q \in \Delta_{d_1}} \frac{1}{n} \sum_{t=1}^{n} q^T M p_t \leq \frac{1}{n} \text{Reg}_n^{(1)}
\]

and

\[
\frac{1}{n} \sum_{t=1}^{n} (-q_t^T M) p_t - \min_{p \in \Delta_{d_2}} \frac{1}{n} \sum_{t=1}^{n} (-q_t^T M) p \leq \frac{1}{n} \text{Reg}_n^{(1)}.
\]

Adding the two, we see that

\[
\max_{p \in \Delta_{d_2}} q_t^T M p - \min_{q \in \Delta_{d_1}} q^T M \bar{p} \leq \frac{1}{n} (\text{Reg}_n^{(1)} + \text{Reg}_n^{(2)}).
\]

However, we know that

\[
\min_{q \in \Delta_{d_1}} q^T M \bar{p} \leq \max_{p \in \Delta_{d_2}} \min_{q \in \Delta_{d_1}} q^T M p = \min_{p \in \Delta_{d_1}} \max_{p \in \Delta_{d_2}} q^T M p \leq \max_{p \in \Delta_{d_2}} q_t^T M p.
\]

Hence, the pair \((\bar{q}, \bar{p})\) is within \( \frac{1}{n} (\text{Reg}_n^{(1)} + \text{Reg}_n^{(2)}) \) from the minimax. So \((\bar{q}, \bar{p})\) is very close to an actual Nash equilibrium.

Intuitively, you start with a uniform distribution over who’s better, and update your perception as the rounds progress. On each round, \( M p_t \) is conveyed, and you see the expected cost of every action that could be chosen. Eventually, \((\bar{q}, \bar{p})\) is very close to an actual Nash equilibrium.
\( C_n \) would typically grow as \( \frac{1}{\sqrt{n}} \). But in fact, according to [Das15], you can obtain much faster convergence, at a rate of \( \frac{\log(n)}{n} \). It required a very complicated construction where the two players agree on some schedule, i.e. the dynamics are no longer uncoupled. The paper left an open question whether there’s a simpler method that does the same.

What kind of extra information can one get? Typically you can scales as something like \( \sqrt{n} \), but if you work a bit harder you can get something like \( \sqrt{\sum_{t=1}^{n} \|y_t\|^2} \). With some extra information about \( y_t \), the next element in the sequence, you can improve to

\[
\sqrt{\sum_{t=1}^{n} \|y_t - \mathcal{M}_t\|^2}.
\]

In particular, for two player games, this extra information is that the previous move of a player is a good proxy for the next move, because the player is doing some kind of slow optimization. So the previous move is a good proxy for the next move.

### 4.2 Optimistic Mirror Descent

The method to update is called Optimistic Mirror Descent, and it works like this:

\[
g_{t+1} = \arg\min_{v \in \mathcal{K}} \eta \langle v, y_t \rangle + D_{\phi}(v, g_t)
g_{t+1} = \arg\min_{v \in \mathcal{K}} \eta \langle v, M_{t+1} \rangle + D_{\phi}(v, g_{t+1}).
\]

If \( \mathcal{M}_t \) is informative, you actually do better. You can incorporate any knowledge. There are all sorts of papers playing with different choices of \( \mathcal{M}_t \).

The first line alone would give the usual mirror descent, which encompasses usual gradient descent (if \( d \) is a distance or a norm), exponential weights (if \( d \) is a KL-divergence), and so forth. The modification of the second line allows one to assume that \( \mathcal{M}_{t+1} \) is correct. It’s a simple modification, and the results can be related to Mirror Prox [He15], have extensions to partial information and smooth convex programming, and have been implicated recently in GAN training.

### References


