# Hybrid Machine Learning Algorithms


Umar Syed

Princeton University

Includes joint work with:

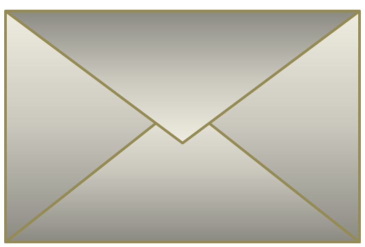
Rob Schapire (Princeton)

Nina Mishra, Alex Slivkins (Microsoft)

# Common Approaches to Machine Learning

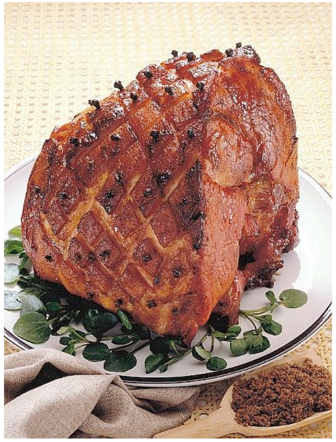
- Supervised Learning: Learn a concept from examples.

- Example: Spam Filtering



=


or

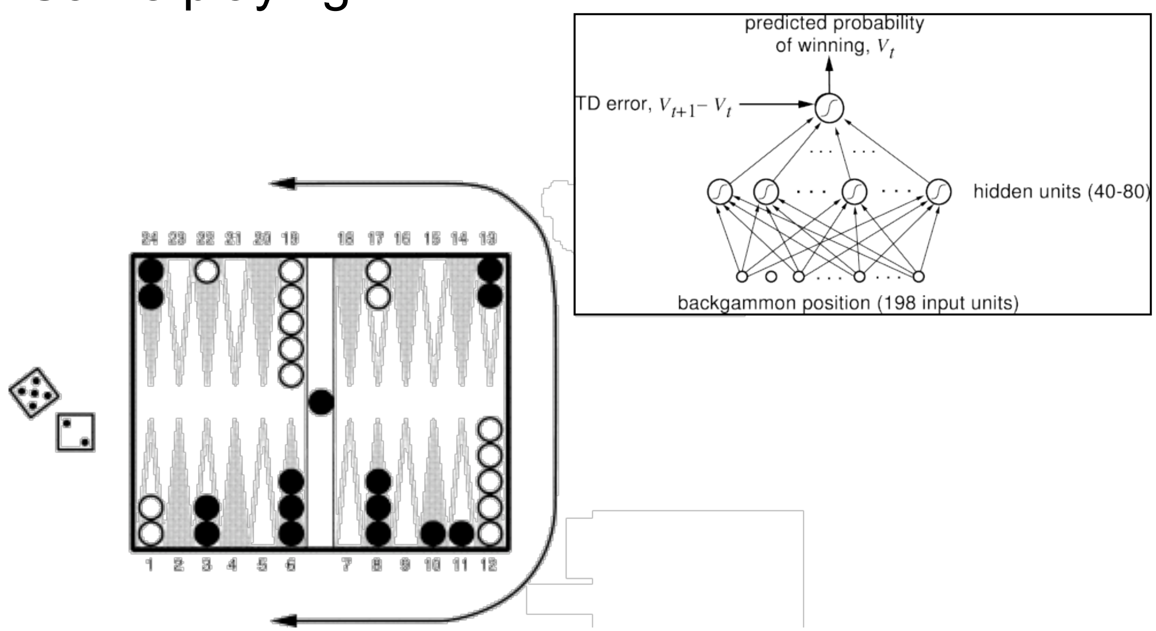


?

# Common Approaches to Machine Learning

- Reinforcement Learning: Learn optimal behavior from interactive experience.

- Example: Game playing




[Tesauro 1995]

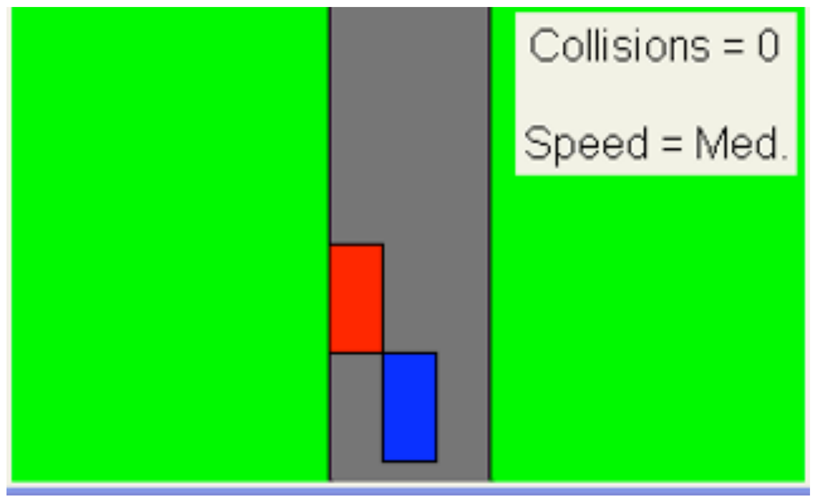
# This Talk

- In this talk, I will describe approaches to machine learning that combine features of supervised and reinforcement learning:

  1. Apprenticeship Learning
  2. Bandit Problems with Events

# Apprenticeship Learning

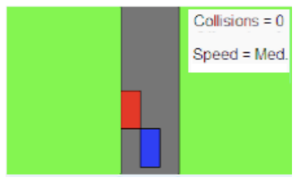
# Application: Car Driving



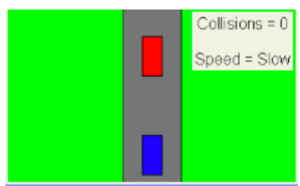

- Problem: Learn a good driving policy for this environment.

# What is a Policy?

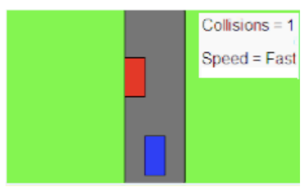
- A *policy* assigns a driving action to each possible environment state:




→ Move Right




→ Accelerate




→ Brake

# Reinforcement Learning Procedure

1. Assign a *reward* to each environment state.
   - e.g. Reward = (1 $\times$ high-speed?) + (-10 $\times$ collide?)

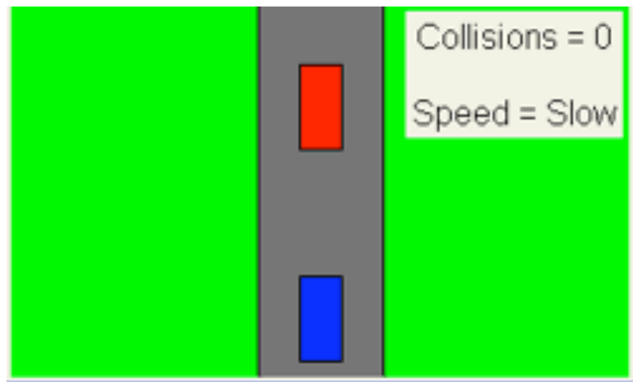
3. Let $V(\pi)$ be the average total reward for following driving policy $\pi$.
   - $V(\pi)$ is called the *value function*.

4. By repeated interaction with the driving environment, find a driving policy $\pi^*$ such that

$$\pi^* = \arg\max_{\pi} V(\pi)$$

# Value of a Policy – Example

- When a car follows this driving policy…



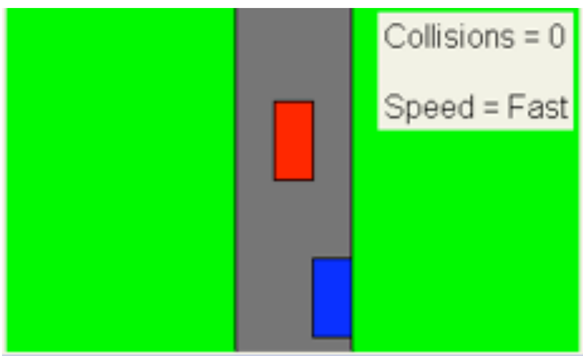

…then at the end of an <u>average</u> driving episode:

1. The car was at high-speed in 3.5 time steps.
2. The car collided with another car in 57 time steps.

- $\therefore$ Value of policy = $(1 \times 3.5) + (-10 \times 57) = -566.5$

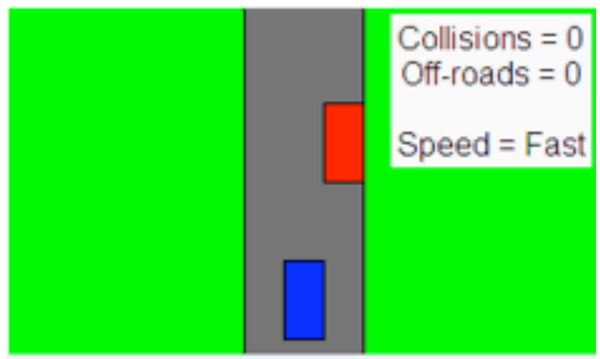
# Drawbacks of Reinforcement Learning

- Usually need to tune rewards manually. This can be tricky.




1st attempt

Forgot penalty for off-road!



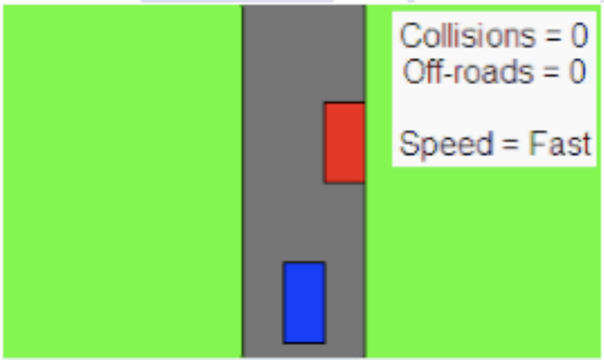

2nd attempt

Collision penalty too low!


Nth attempt

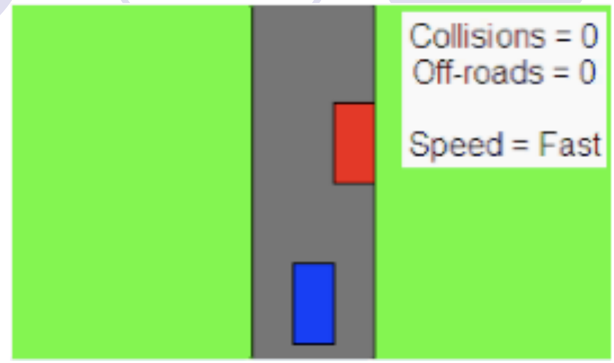
Good!

# Apprenticeship Learning (Abbeel & Ng, 2004)




Expert policy $\pi^E$

Learning algorithm




Apprentice policy $\pi^A$

- Given: Demonstrations from an *expert policy* $\pi^E$.
- Objective: Learn an *apprentice policy* $\pi^A$ such that

$$V(\pi^A) \geq V(\pi^E)$$

where the value function $V(\pi)$ is unknown.

# Apprenticeship Learning

- Our contribution: New algorithm for apprenticeship learning that:

  1. Is simpler,
  2. Is more efficient, and
  3. Outputs a better apprentice policy

  than existing algorithms.

# Assumptions

- Definition: Let $\mu(\pi)$ be the *feature vector* for policy $\pi$. For example:
  - $\mu_1(\pi)$ = Average total collisions for $\pi$ (negated).
  - $\mu_2(\pi)$ = Average total off-roads for $\pi$ (negated).
  - $\mu_3(\pi)$ = Average total high-speed time steps for $\pi$.

- Main Assumption: There exists $w^*$ such that

$$V(\pi) = w^* \cdot \mu(\pi)$$

where $w^*$ is an <u>unknown</u> convex combination.

# A Basic Question

- Let's first ask what can we learn <u>without</u> the expert demonstrations.

- Can we always learn a policy $\pi$ such that $V(\pi)$ is large?

- In general no, because we have no way of computing $V(\pi)$.

- But we can learn a policy that is good in a <u>conservative</u> sense.

# A "Max-Min" Policy

- Objective: Find policy $\pi^*$ satisfying

$$\pi^* = \arg\max_\pi \min_w w \cdot \mu(\pi)$$

- In other words, choose $\pi^*$ so that $V(\pi^*)$ is as large as possible for adversarially chosen weights.

- Since true weights $w^*$ are unknown, this is a suitably conservative approach.

# A "Max-Min" Policy

- Objective: Find policy $\pi^*$ satisfying

$$\pi^* = \arg\max_{\pi} \min_{w} w \cdot \mu(\pi)$$

- Objective is essentially a *two-player zero-sum game*.
  - Player 1 (algorithm) wants to maximize $V(\pi)$.
  - Player 2 (nature) wants to minimize $V(\pi)$.

- Policy $\pi^*$ is an *optimal strategy* for Player 1.

# Computing Optimal Strategy

- An optimal strategy for two-player zero-sum games can be computed with a linear program.

- Size of linear program is proportional to size of the *payoff matrix*, which defines possible game outcomes.
  - e.g. Payoff matrix for the game “Rock, Paper, Scissors”:

| | Rock | Paper | Scissors |
|---|---|---|---|
| Rock | 0 | +1 | -1 |
| Paper | -1 | 0 | +1 |
| Scissors | +1 | -1 | 0 |

# Computing Optimal Strategy

- Payoff matrix for the game $\max_{\pi} \min_{w} w \cdot \mu(\pi)$:

Policies

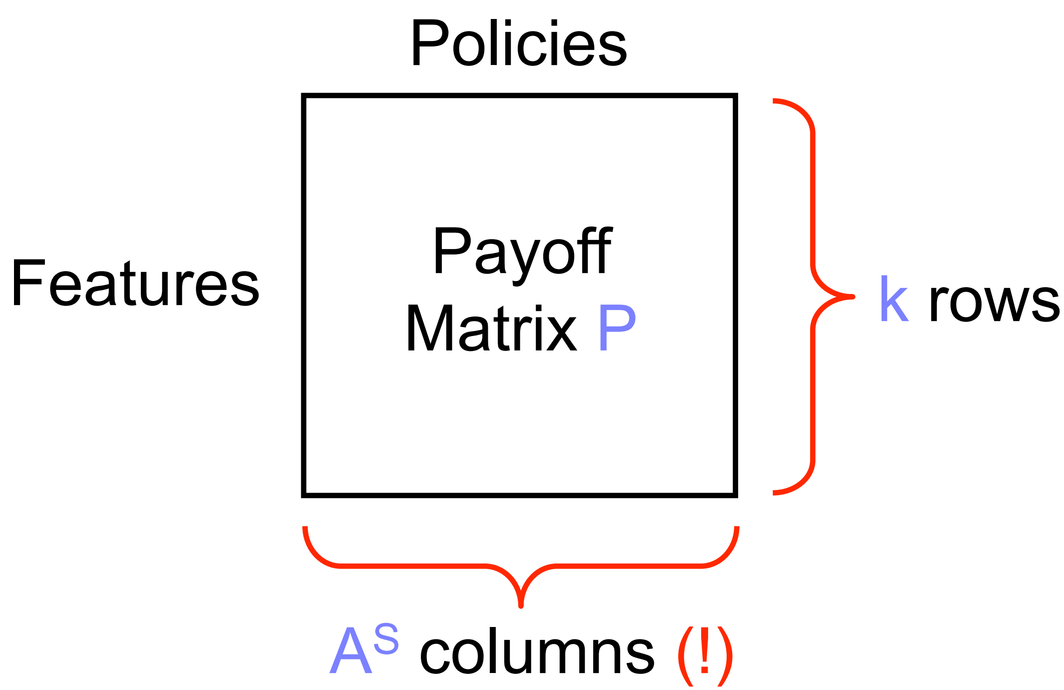
Features

Payoff Matrix $P$

where $P(i, j) = \mu_i(\pi^j) = i^{th}$ feature value for $j^{th}$ policy.

# Computing Optimal Strategy

- Problem: Payoff matrix is too big!




- k = # of features
- S = # of environment states
- A = # of driving actions

# Computing Optimal Strategy

- Solution: Use a multiplicative weights algorithm (Freund & Schapire, 1996) instead.
- MW algorithm can compute optimal strategies for large — even infinite — payoff matrices.
- MW algorithm is closely related to boosting, online learning.

# MW Algorithm for Apprenticeship Learning

1. Maintain a weight vector $w_t$ over several rounds.

2. In round $t$, compute best policy $\pi_t$ for reward function

   $$R(s) = w_t \cdot \mu(s)$$

   using a standard reinforcement learning algorithm.

3. Update $w_t \rightarrow w_{t+1}$ by shifting weight to features where $\pi_t$ does badly.
   - The update is essentially the boosting update.

4. After $T$ rounds, output $\pi^*$ chosen uniformly at random from $\{\pi_1, \ldots, \pi_T\}$.

# Analysis

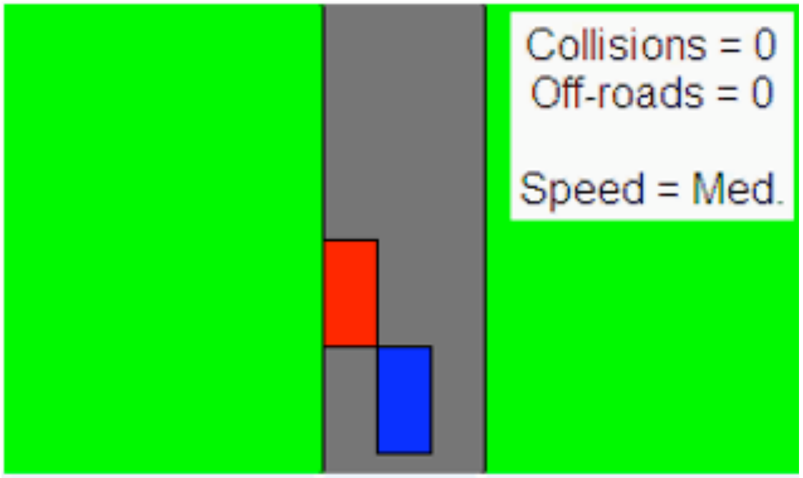
- Theorem: After $O(\log k)$ iterations of MW algorithm, it outputs a policy $\pi^*$ such that

  $$E[V(\pi^*)]$$

  is as large as possible for adversarially chosen $w^*$.

  (Expectation is over randomness in algorithm; $k$ = # of features)

# Demo




Conservative driving policy:
Drives as fast as possible without hitting other cars or going off-road.

# Back to Apprenticeship Learning

- Given demonstrations from an expert policy $\pi^E$, follow this procedure:

1. Use demonstrations to estimate $\mu(\pi^E)$.
   - i.e. Estimate expert's average total crashes, off-roads, etc.

2. Use MW algorithm to solve this objective

New!

$$\pi^A = \arg\max_\pi \min_w [w \cdot \mu(\pi) - w \cdot \mu(\pi^E)]$$

   - i.e. MW will choose $\pi^A$ so that $V(\pi^A) - V(\pi^E)$ is as large as possible for adversarially chosen weights.

# Analysis

- Theorem: After $O(\log k)$ iterations of MW algorithm, it outputs a policy $\pi^A$ such that

  $$E[V(\pi^A)] - V(\pi^E)$$

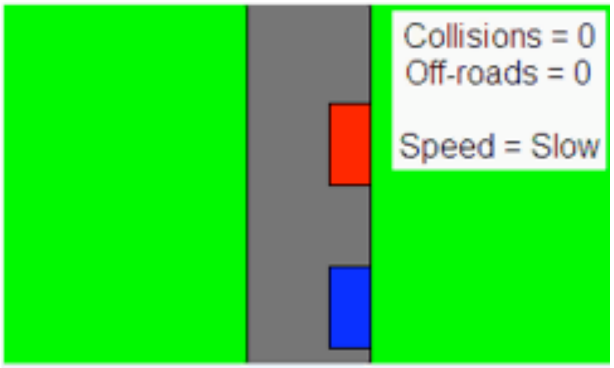
  is as large as possible for adversarially chosen $w^*$.

  (Expectation is over randomness in algorithm; $k$ = # of features)

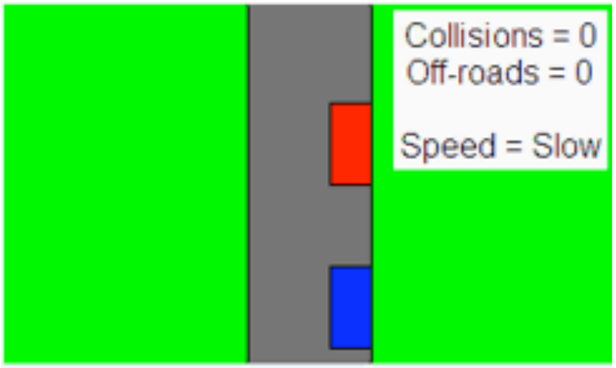
- Corollary: $E[V(\pi^A)] \geq V(\pi^E)$

  Proof idea: Algorithm can always choose $\pi^A = \pi^E$, so we have $E[V(\pi^A)] - V(\pi^E) \geq 0$.

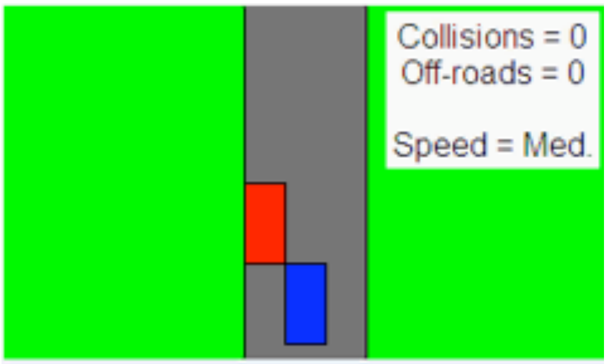
# Demo




Expert



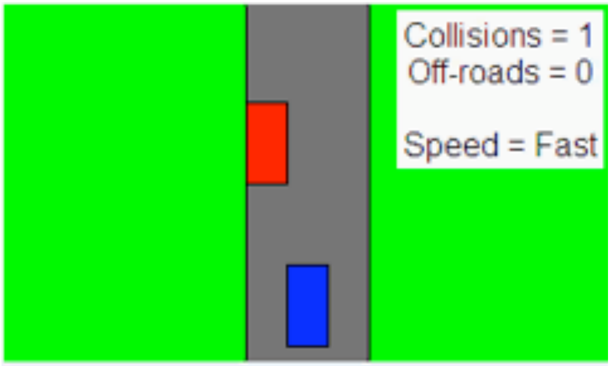

Projection algorithm
(Abbeel & Ng, 2004)



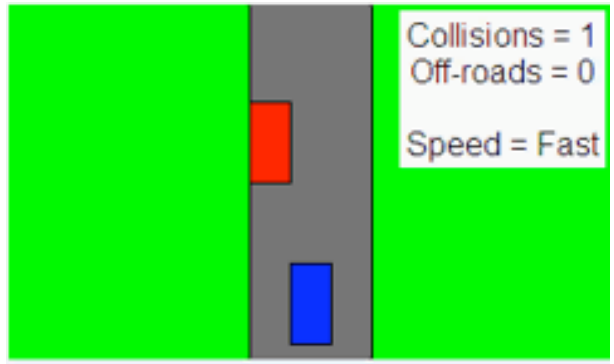

MW algorithm

- Projection algorithm mimics expert.
- MW algorithm learns <u>better</u> policy than expert!
  - This can happen when expert policy is *dominated*.

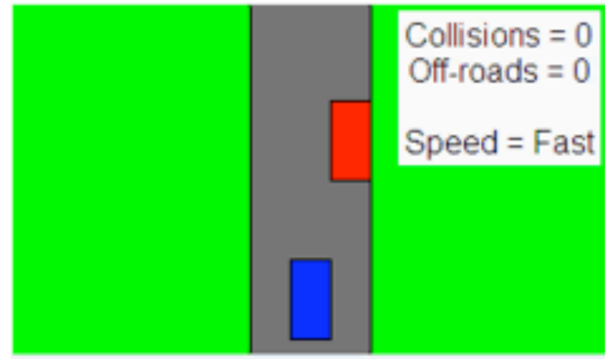
# Demo




Expert




Projection algorithm
(Abbeel & Ng, 2004)




MW algorithm

- An example where expert policy can't be ignored.

# Comparison of Algorithms

| | Projection algorithm | MW algorithm |
|---|---|---|
| No. of iterations | $O((k / \epsilon^2) \log (k / \epsilon))$ | $O((1 / \epsilon^2) \log k)$ |
| Post-processing | Requires QP solver | None |
| Guarantee | $\lvert V(\pi^A) - V(\pi^E) \rvert \leq \epsilon$ | $V(\pi^A) \geq V(\pi^E) - \epsilon$ and possibly $V(\pi^A) \gg V(\pi^E)$ |
| Applicable without expert? | No | Yes |

$k$ = # of features

# Refining the Analysis

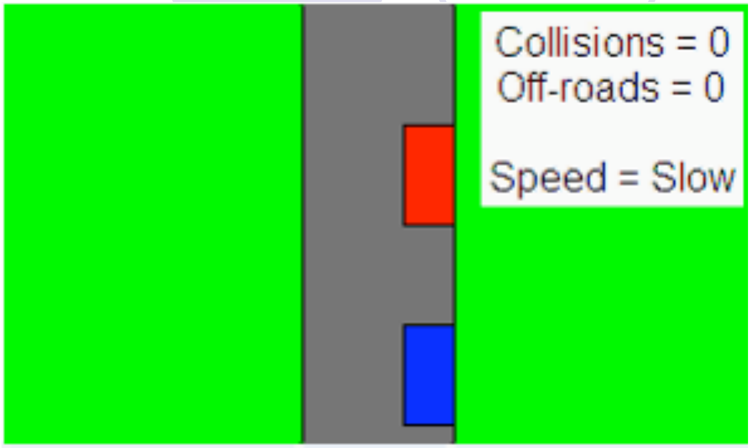
- Recall our earlier objective:

$$\pi^* = \text{arg max}_\pi \ \text{min}_w \ w \cdot \mu(\pi)$$

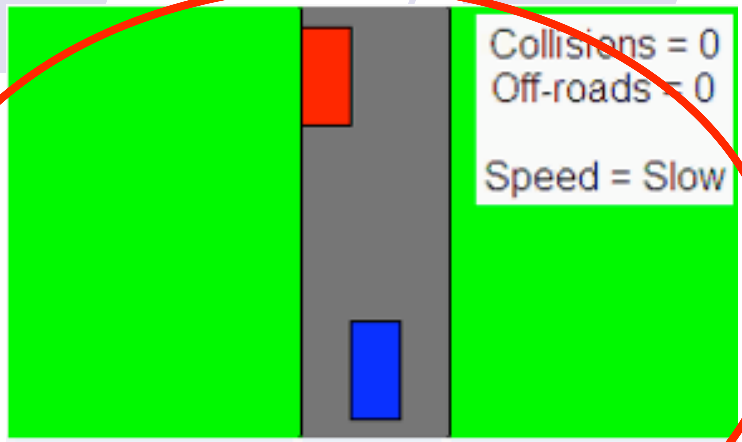
- Put differently: Choose $\pi^*$ so that smallest element of vector $\mu(\pi^*)$ is maximized.

- What can we say about the <u>other</u> elements of $\mu(\pi^*)$?

# Refining the Analysis




$\pi_1$: “Bad”




$\pi_2$: “Bad, but on-road”

- Smallest elements in $\mu(\pi_1)$ and $\mu(\pi_2)$ are equal.
  - Both have the same average number of collisions.
- But $\pi_2$ is clearly a better policy.
- So which policy does MW algorithm converge to?

# Refining the Analysis

- Theorem: Under certain conditions, MW outputs a policy $\pi^*$ such that

  Smallest element of $E[\mu(\pi^*)]$ is maximized,

  and further, among all such policies, MW outputs a policy $\pi^*$ such that

  2nd smallest element of $E[\mu(\pi^*)]$ is maximized,

  and so on for 3rd smallest, 4th smallest, ...

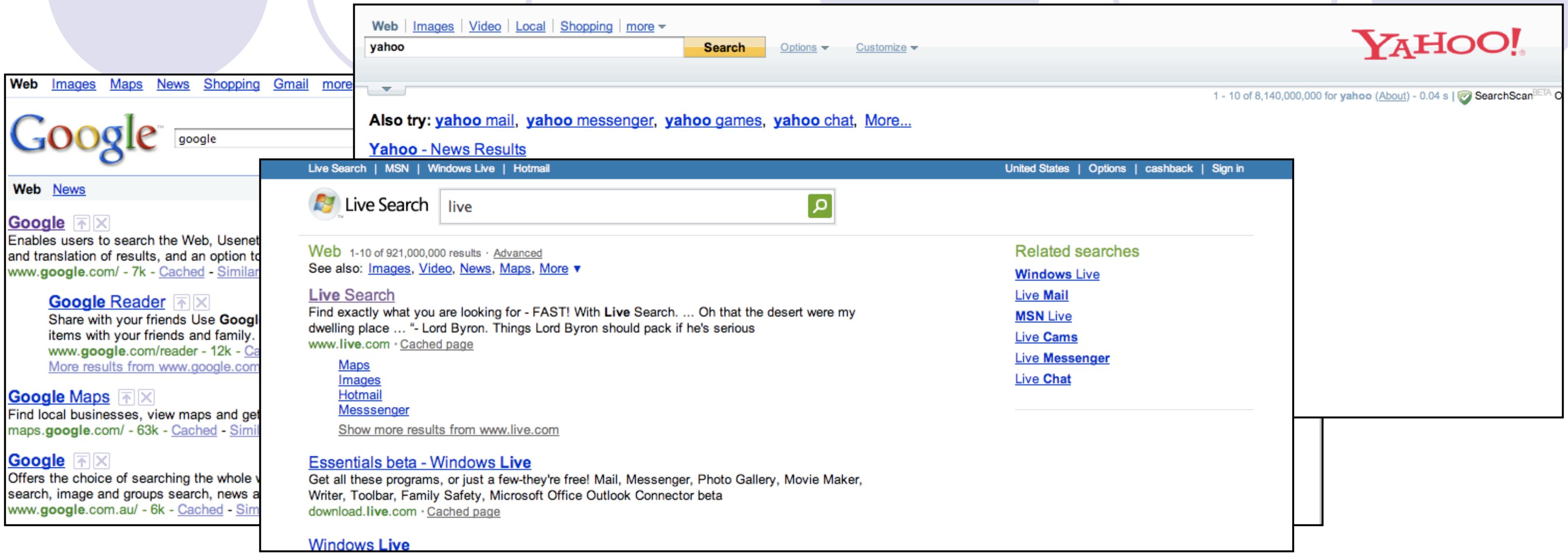
# Refining the Analysis

- Proof Idea:
  - Recall that the MW algorithm maintains a weight vector $w_t$.
  - MW constantly adjusts $w_t$ so as to place the most weight on the "hardest" feature.
  - We show that MW also places the second-most weight on the second-hardest feature, and so on.

- Note: Result applies to the generic MW algorithm. Has implications beyond apprenticeship learning (e.g boosting).

# Bandit Problems with Events

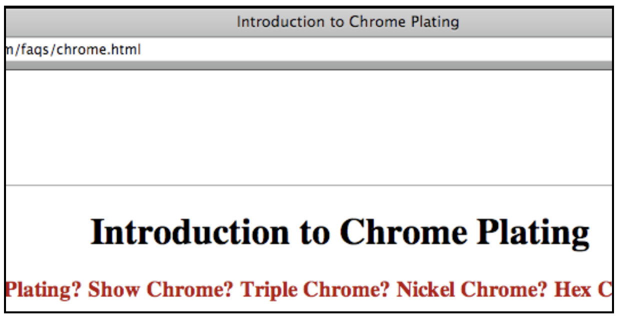
# Application: Web Search

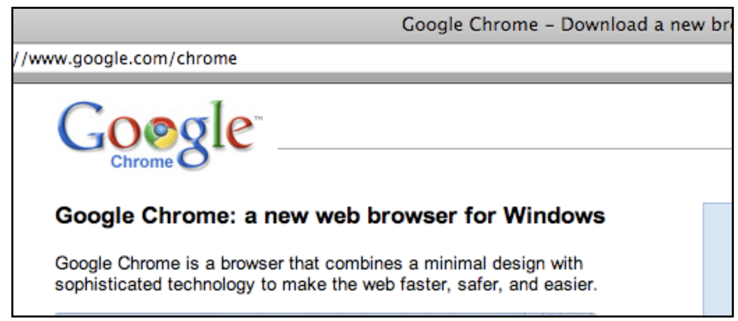



- Users issue *queries* to search engine, which tries to return the most relevant documents.
- Sometimes, a query will acquire a different meaning in response to an external *event*.

# Examples of Event-Sensitive Queries

Query = “chrome”



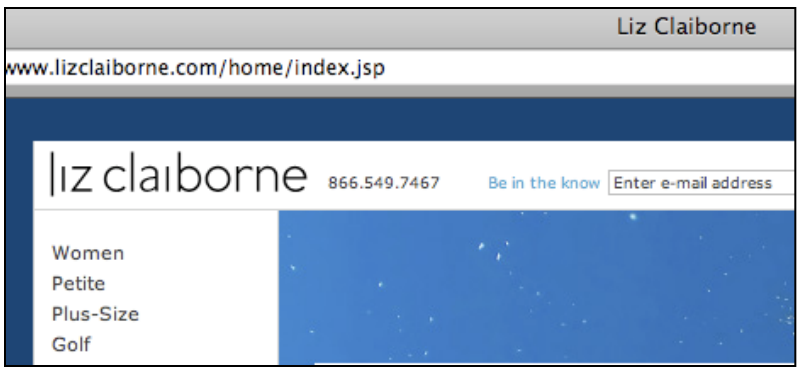




Event

Google releases “Chrome”
web browser

# Examples of Event-Sensitive Queries

Query = “liz claiborne”





Event

Liz Claiborne dies

- Problem: Provide better search results for event-sensitive queries.

# The “Bandit” Approach to Search

- Methods like PageRank uses the link structure of the web to determine the most relevant document for a query.
- These methods are slow to adapt to abrupt changes in query meaning caused by external events.
- Our approach is to use <u>feedback from user clicks</u> to determine the most relevant document.
  - Document gets more clicks $\Rightarrow$ Document is more relevant.
- We call this a *bandit problem with events*.

# Bandit Problem

- Fix a query $q$, and let $D$ be a set of candidate documents for $q$.

- For time $t = 1 \ldots T$:
  1. User searches for $q$.
  2. Search engine returns one document $d_t \in D$ to user.
     (For simplicity, we focus on returning the single best document.)
  3. User clicks on $d_t$ with <u>unknown</u> probability $p_{d_t,t}$.

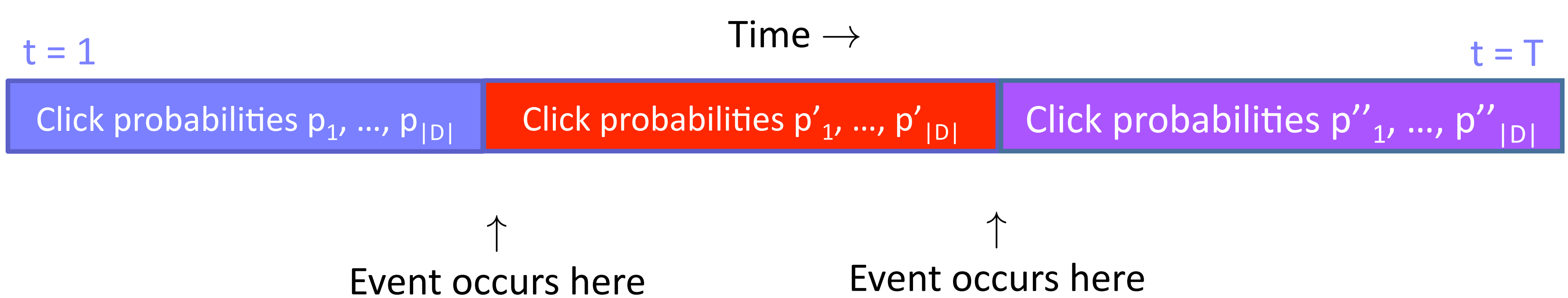
- Goal: Choose $d_1, \ldots d_T$ to maximize expected total number of clicks.

# Bandit Problem with Events

- Assume at most $k$ events can occur during $T$ searches.
  - Times of events are <u>unknown and arbitrary</u>.

- $p_{d,t} \neq p_{d,t+1} \Leftrightarrow$ An event occurs at time $t$
  - New click probabilities are <u>unknown and arbitrary</u>.

- So values of click probabilities can be divided into phases:



# Bandit Problem with Events

- Our contribution: A new algorithm that:

1. Is O(log T) suboptimal for this problem, while existing algorithms are $\Omega(\sqrt{T})$ suboptimal.
2. Performs better than existing algorithms in experiments on web search data.

# UCB Algorithm (Auer et al, 2002)

- Suitable for the special case of $k = 0$ (no events).
  - i.e. $p_{d,t} = p_d$ for all documents $d$.

Regret

- Theorem [Auer et al 2002]:

$$E_{UCB}[\text{\# of clicks}] \geq E_{Opt}[\text{\# of clicks}] - O(\log T)$$

  where Opt is the algorithm that always chooses the document with highest click probability.

- Details interesting, but not relevant: our algorithm uses UCB as a black-box subroutine.

# Handling Events

- UCB algorithm assumes that click probabilities are fixed, i.e. no events.
- So what happens if there are events?

# Lower Bound

- Theorem: If the click probabilities can change even *once*, then for *any* algorithm:

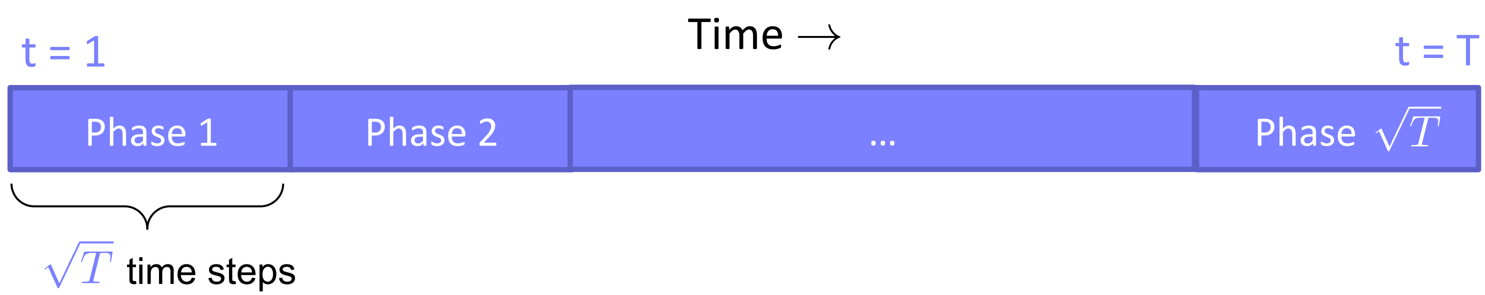
$$E_{\text{Algorithm}}[\text{\# of clicks}] \geq E_{\text{Opt}}[\text{\# of clicks}] - \Omega(\sqrt{T})$$

- The number of impressions $T$ can be very large (millions/day).
- So even one event can result in large regret.

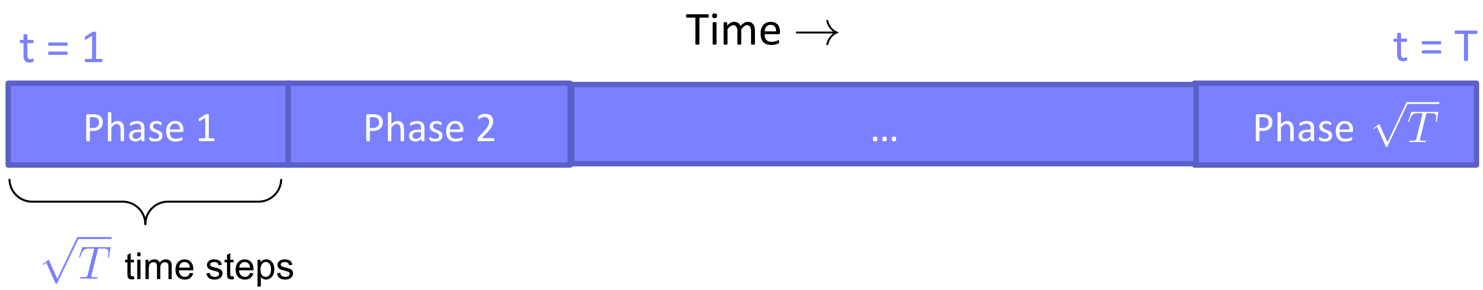
# Lower Bound

- Proof sketch: Let there be two documents, x and y, with click probabilities $p_x > p_y$.

- Divide the time period into $\sqrt{T}$ phases, each of length $\sqrt{T}$:



# Lower Bound




- For any algorithm A, there are two cases:
  1. A selects both documents at least once per phase.
     $\Rightarrow$ $\Omega(\sqrt{T})$ regret.
  2. A selects only document x in some phase. Then in that phase, increase $p_y$ so that $p_y > p_x$.
     $\Rightarrow$ $\Omega(\sqrt{T})$ regret.

- Note: This lower bound is not based on $|p_x - p_y|$ being small (unlike existing lower bounds).

# Lower Bound – Caveat

- Proof of lower bound assumes that algorithm has no knowledge about when events occur.
- But this is not really the case!

# Signals Correlated with Events

- An event related to query q may have occurred if:
  - Increased volume for q.
  - Increased mentions of q in news stories/blogs.
  - Increased reformulations of q by users.
  - Change in geographical distribution of q
    - e.g. “independence day” in US vs. India

# Event Oracle

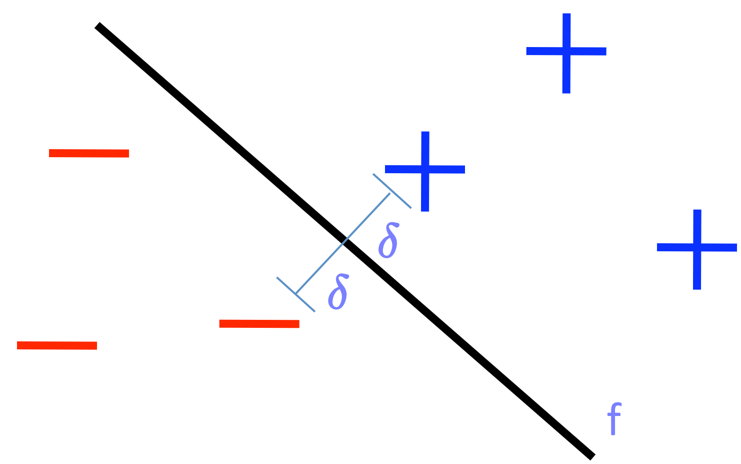
- Suppose that on every time step $t$ we receive a feature vector $x_t$.

- Assume there exists an *event oracle* function $f$ such that:

$$f(x_t) = +1$$
$$\Updownarrow$$
Event occurs at time $t$.

# Event Oracle

- Assume that f is a linear separator in feature space:
  $f(x) = \mathrm{sign}(w \cdot x)$ for some unknown weight vector w.




- Also assume a margin $\delta$ for all examples from the separator.

# UCB Algorithm with Event Oracle

- For each time $t = 1 \ldots T$:
  1. User searches for $q$.
  2. Choose document $d_t$ according to UCB algorithm.
  3. Receive feature vector $x_t$.
  4. If $f(x_t) = +1$ then <u>restart</u> UCB algorithm.

$t = 1$ Time → $t = T$

| Click probabilities $p_1, \ldots, p_{|D|}$ | Click probabilities $p'_1, \ldots, p'_{|D|}$ | Click probabilities $p''_1, \ldots, p''_{|D|}$ |
|---|---|---|

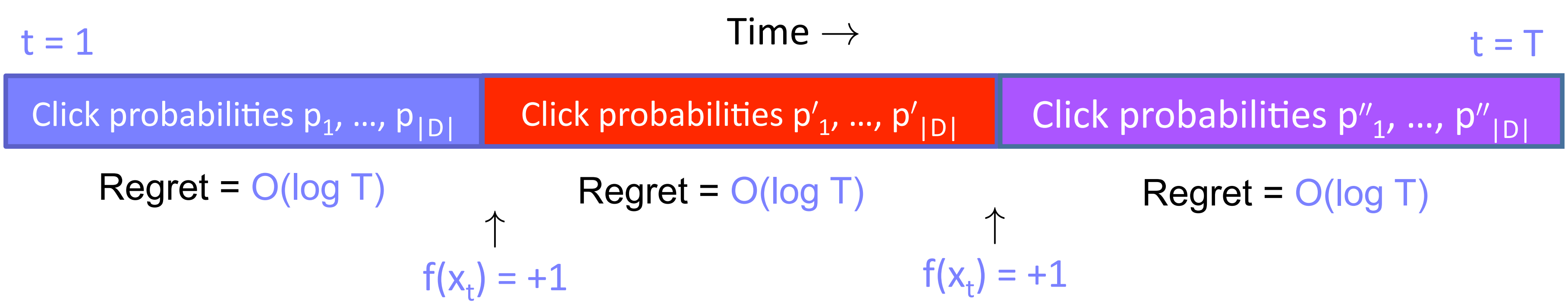
↑ $f(x_t) = +1$ ↑ $f(x_t) = +1$

# UCB Algorithm with Event Oracle

- Theorem: $E_{UCB+Oracle}$[# of clicks] $\geq$ $E_{Opt}$[# of clicks] – O(k log T),

  where k = number of events.

- Proof: UCB algorithm suffers O(log T) regret in each phase, and there are O(k) phases.

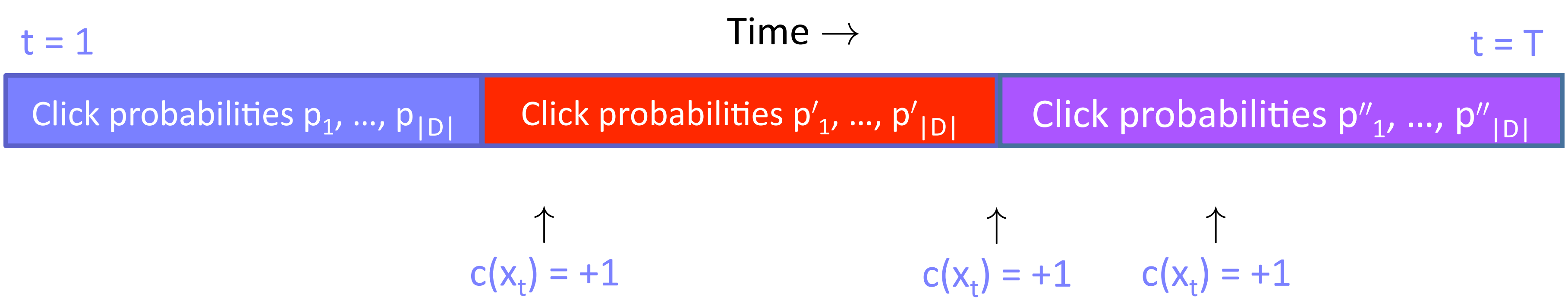


# Event Classifier

- Of course, the event oracle function $f$ will not be known in advance.
- So we want to <u>learn</u> a function $c \approx f$ as we go, from examples of events and non-events.
- This is a classification problem, and we call $c$ an *event classifier*.

# UCB Algorithm with Event Classifier

- For each time $t = 1 \dots T$:
  1. User searches for $q$.
  2. Choose document $d_t$ according to UCB algorithm.
  3. Receive feature vector $x_t$.
  4. If $c(x_t) = +1$, then <u>restart</u> UCB algorithm.
  5. If $c(x_t) = +1$ prediction was wrong, then <u>improve</u> $c$.



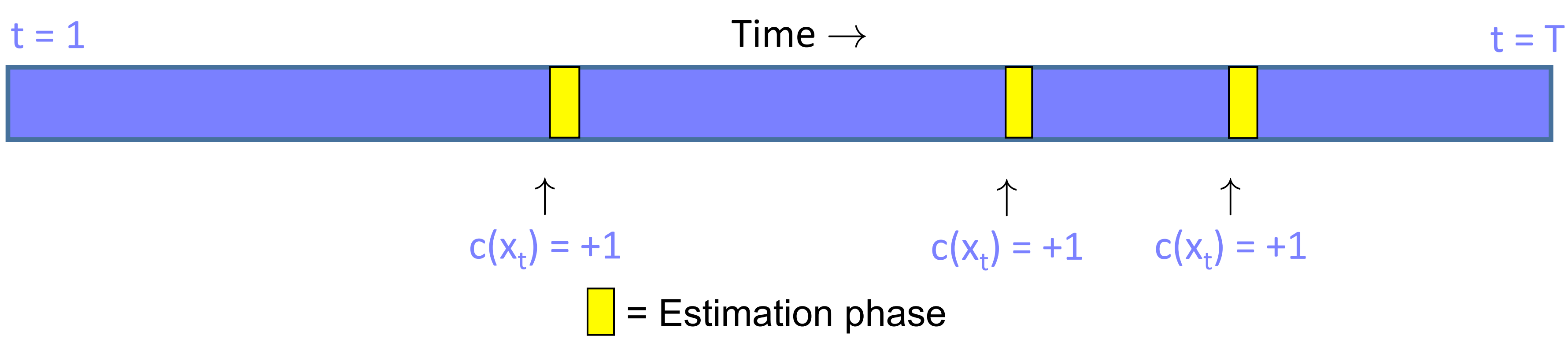
# UCB Algorithm with Event Classifier

- As time goes on, we want $c \approx f$.

- So how do we …
  - … determine that a prediction was wrong?
  - … represent classifier $c$, and improve $c$ after a mistake?
  - … bound the impact of mistakes on regret?

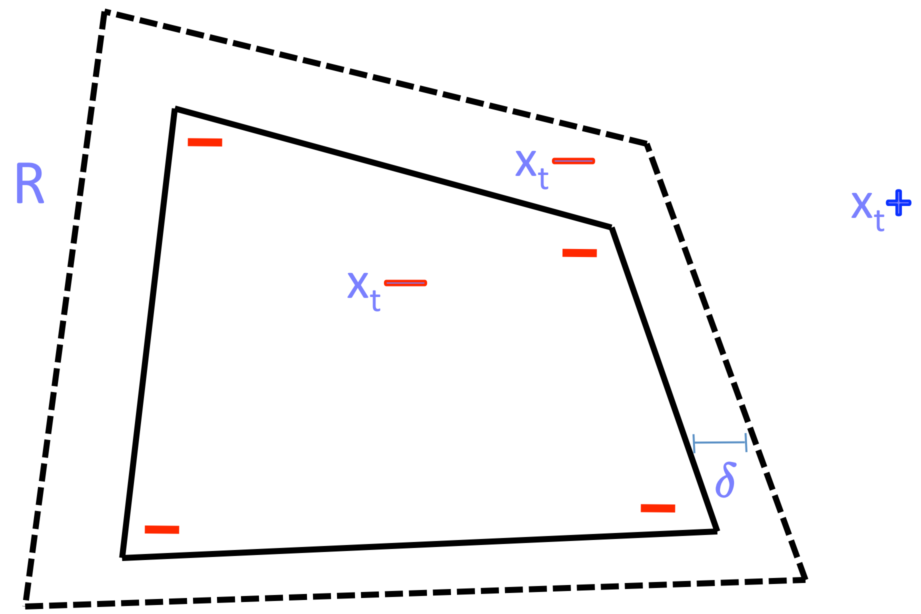
# Determining That a Prediction Was Wrong

- If an event occurred, then click probability $p_d$ of some document $d$ must have changed.
- For each $d$, we can estimate $p_d$ by repeatedly displaying $d$, and recording its empirical click frequency.
- So to check whether $c(x_t) = +1$ was a <u>false positive</u>:
  - Spend a little time immediately after the prediction to estimate $p_d$ for all $d$.



# Classifier $c$

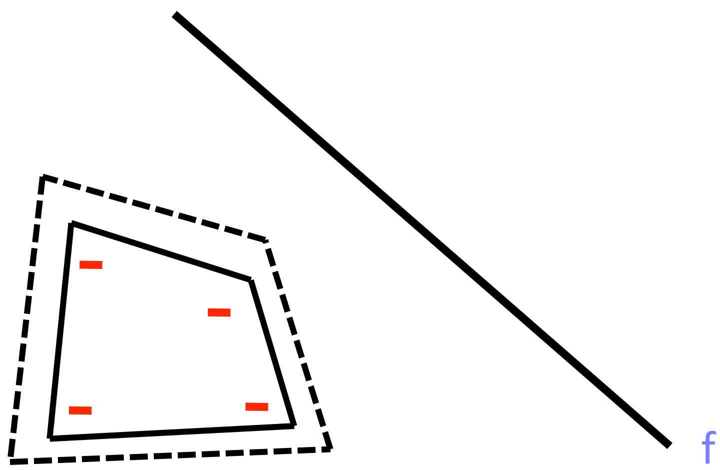
- Let $R$ be the "$\delta$-extended" convex hull of previous <u>false positives</u>.




- We predict $c(x_t) = -1$ <u>if and only if</u> $x_t \in R$

# Bounding Mistakes

- Fact: We will never predict a false negative!
  - Our negative predictions are <u>very</u> conservative.



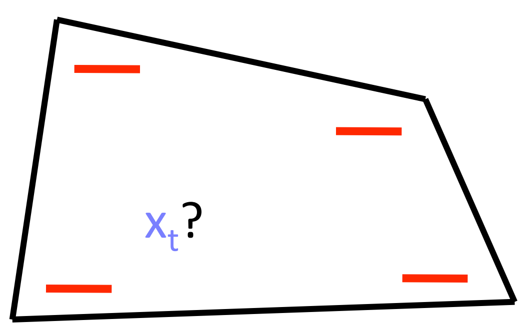

- Fact: Every time we predict a false positive, the convex hull grows.
  - So we can bound the number of false positives.

# Technicalities

- Technically, due to complications such as:
  - Noise
  - Multiple events occurring in a brief time span.

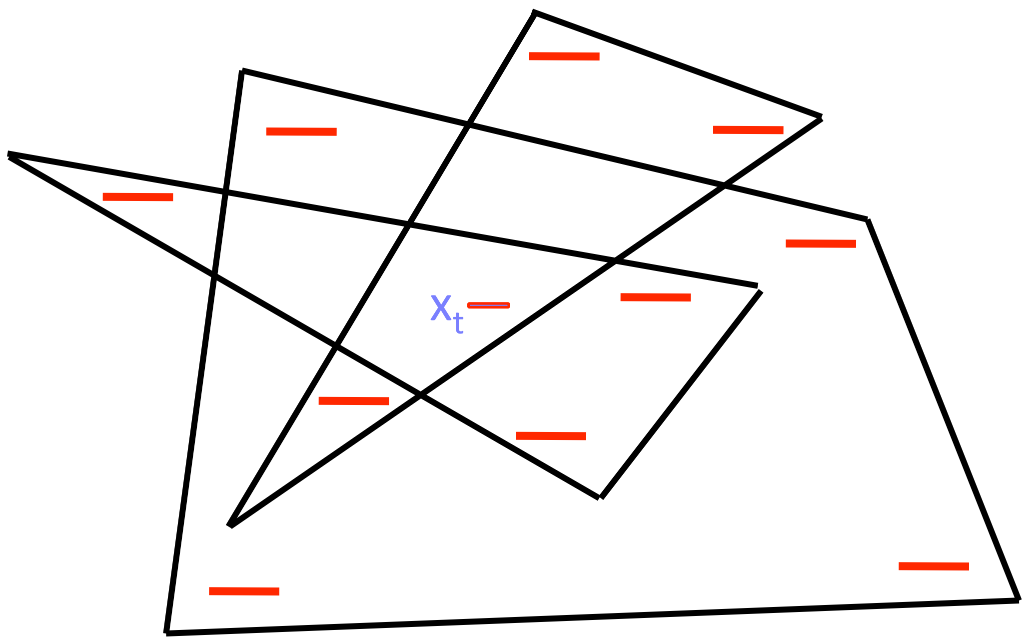
  we can never be <u>certain</u> that a prediction is a false positive.

- Hence, we can never be <u>certain</u> that an example inside the convex hull is really a negative.



# Technicalities

- So we maintain several convex hulls:




- And predict $c(x_t) = -1$ only if $x_t$ is inside all of them.
- We will be correct with high probability.

# UCB Algorithm with Event Classifier

- **Theorem:**
  Oops!
  $E_{UCB+Classifier}$[# of clicks] $\geq E_{Opt}$[# of clicks] – $O((1/\delta)^d\, k \log T))$

- where:
  - $\delta$ = Margin
  - d = Dimension of feature space
  - k = Number of events

- Intuition: Convex hulls grow very slowly, hence strong dependence on d.

# Another Lower Bound

- Theorem: For any algorithm, one of the following holds:

  1. $E_{Alg}$[# of clicks] $\geq$ $E_{Opt}$[# of clicks] – $\Omega((1/\delta)^d \log T)$, or

  2. $E_{Alg}$[# of clicks] $\geq$ $E_{Opt}$[# of clicks] – $\Omega(\sqrt{T}/(1/\delta)^d)$

- Intuition: Convex hull-based prediction necessary, because we can't afford to miss even one event (remember lower bound).

- Moral: You can have subexponential dependence on $d$, or logarithmic dependence on $T$, but never both.

# Experiment: Setup

- Used search log data from Microsoft Live Search.
- Used data for the query “liz claiborne”, from the year of her death.
- To conduct a realistic experiment, we just “replayed” the log data against our algorithms.
- We replayed the same year’s data five consecutive times.
  - So in this experiment, Liz dies five times.
  - This simulates multiple events.
- We use only $d = 3$ features, so we don’t suffer from exponential dependence on $d$.

# Results

| Method | Total Clicks Over Five Years |
|---|---|
| UCB | 49,679 |
| UCB+Classifer | 50,621 (+1.9%) |
| UCB+Oracle | 51,207 (+3.1%) |

# Recap

- New algorithms for …
  - … apprenticeship learning, which can output policies that outperform the expert being followed.
  - … web search, which efficiently provide relevant search results for queries whose meaning can abruptly change.
- Thanks!