Stable Coactive Learning via Perturbation

Karthik Raman KARTHIK@CS.CORNELL.EDU
Thorsten Joachims TJ@CS.CORNELL.EDU
Department of Computer Science, Cornell University, Ithaca, NY, USA

Pannaga Shivaswamy PANNAGA@RESEARCH.ATT.COM
AT&T Research, San Francisco, CA, USA

Tobias Schnabel TBS49@CORNELL.EDU
Fachbereich Informatik, Universitaet Stuttgart, Stuttgart, Germany

Abstract

Coactive Learning is a model of interaction between a learning system (e.g. search engine) and its human users, wherein the system learns from (typically implicit) user feedback during operational use. User feedback takes the form of preferences, and recent work has introduced online algorithms that learn from this weak feedback. However, we show that these algorithms can be unstable and ineffective in real-world settings where biases and noise in the feedback are significant. In this paper, we propose the first coactive learning algorithm that can learn robustly despite bias and noise. In particular, we explore how presenting users with slightly perturbed objects (e.g., rankings) can stabilize the learning process. We theoretically validate the algorithm by proving bounds on the average regret. We also provide extensive empirical evidence on benchmarks and from a live search engine user study, showing that the new algorithm substantially outperforms existing methods.

1. Introduction

A growing number of interactive systems use machine learning to adapt their models to different environments, different users, or different user populations. Examples of such systems range from search engines and recommender systems, to personal assistants and autonomous robots. Ideally, these system should learn directly from their users in a manner that is unobtrusive, robust, and efficient.

A model of such learning processes is Coactive Learning (Shivaswamy & Joachims, 2012), combining a boundedly rational model of user behavior with an online learning model that formalizes the goal of learning. In particular, Coactive Learning models the interaction between the user and a learner using weaker assumptions about the user feedback than in standard supervised learning. At each step, the learner (e.g. search engine) receives a context (e.g. query) for which it predicts an object (e.g. ranking, say $[d_1, d_2, d_3, d_4, ...]$). This object is presented to the user. If this object is suboptimal, the user responds with a slightly improved object, but not necessarily the optimal object as typically assumed in supervised learning. This means the user merely provides a preference, which can typically be inferred from implicit feedback (e.g., clicks on d_2 and d_4 imply that the user would have preferred the ranking $[d_2, d_4, d_1, d_3, ...]$). The goal of learning is to minimize regret, which is the cumulative suboptimality of predictions over the life of the learning system.

While learning algorithms exist for the Coactive Learning model (Shivaswamy & Joachims, 2012), we show in this paper that they can perform poorly in the presence of noise. To overcome this problem, we propose a new learning algorithm that is robust to noise and performs well even in an agnostic setting. Our algorithm – called the *Perturbed Preference Perceptron* – produces greatly improved generalization performance both in simulation experiments, as well as in a live user study on an operational search engine. Furthermore, we prove regret bounds for this algorithm that characterize its behavior and provide explicit guidance for its application in practice, especially for ranking problems in search and recommendation.

Proceedings of the 30^{th} International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

2. Related Work

Our work follows the coactive learning model proposed in (Shivaswamy & Joachims, 2012), which we discuss in Section 3. Feedback in coactive learning lies between the Multi-Armed Bandit problem (Auer et al., 2002b;a; Flaxman et al., 2005) (payoff only for selected action) and the Expert-Advice problem (Cesa-Bianchi & Lugosi, 2006; Zinkevich, 2003) (payoff for all actions). However, the coactive learner never observes absolute payoffs, but merely a preference between two actions. This aspect of preference feedback is similar to the dueling bandits model (Yue et al., 2009; Yue & Joachims, 2009), but the algorithm choses both actions in the dueling bandits model, while the user and algorithm chose one action each in the coactive learning model.

Coactive learning also differs from other preference learning problems. For example, in ordinal regression (Crammer & Singer, 2001) a training example (x, y) provides an absolute rank y. Ranking with pairwise preferences (Herbrich et al., 2000; Freund et al., 2003; Chu & Ghahramani, 2005) is another popular problem. However, existing algorithms require an iid sample in a batch setting, while coactive learning works with no-iid data in an online setting. Listwise approaches to ranking (see (Liu, 2009)) differ from coactive learning as they require the *optimal* ranking for a query, not just a preference between typically suboptimal rankings. Partial monitoring games (Bartók et al., 2010) also differ from coactive learning, as they require that loss and feedback matrices are revealed to the learning algorithm. Furthermore, partial monitoring games have no explicit notion of context that is available at the beginning of each round.

The work in this paper is based on perturbing the output of a predictor for improved feedback. In information retrieval, this idea has been proposed for at least two purposes. First, search results from two retrieval functions are interleaved (Chapelle et al., 2012) to elicit unbiased user preferences. Second, the "FairPairs" perturbation strategy (Radlinski & Joachims, 2006) was proposed for de-biasing click data in search. We use the FairPair idea, and provide the first exploration and learning algorithm for this type of feedback.

3. Coactive Learning Model

The coactive learning model, as proposed in (Shivaswamy & Joachims, 2012), is used in the rest of this paper. At each iteration t, the user states a context $\mathbf{x}_t$ (e.g., query) and the learning algorithm makes a prediction $\mathbf{y}_t \in \mathcal{Y}$ (e.g., ranking) in response. The user draws some utility $U(\mathbf{x}, \mathbf{y})$ from this prediction,

Algorithm 1 Preference Perceptron.

Initialize $\mathbf{w}_1 \leftarrow \mathbf{0}$
for $t = 1$ **to** T **do**
  Observe $\mathbf{x}_t$
  Present $\mathbf{y}_t \leftarrow \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}} \mathbf{w}_t^\top \phi(\mathbf{x}_t, \mathbf{y})$
  Obtain feedback $\bar{\mathbf{y}}_t$
  Update: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \phi(\mathbf{x}_t, \bar{\mathbf{y}}_t) - \phi(\mathbf{x}_t, \mathbf{y}_t)$

and provides an improved prediction $\bar{\mathbf{y}}_t \in \mathcal{Y}$ as feedback. Denoting the optimal prediction for iteration t as $\mathbf{y}_t^* = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}} U(\mathbf{x}_t, \mathbf{y})$, the quality of the users' feedback $\bar{\mathbf{y}}_t$ in response to $\mathbf{y}_t$ is characterized as *expected α-informative*,

$$\mathbf{E}_{\bar{\mathbf{y}}_t}[U(\mathbf{x}_t, \bar{\mathbf{y}}_t)] \geq U(\mathbf{x}_t, \mathbf{y}_t) + \alpha(U(\mathbf{x}_t, \mathbf{y}_t^*) - U(\mathbf{x}_t, \mathbf{y}_t)) - \xi_t. \quad (1)$$

The expectation is under $\mathbf{P}_{\mathbf{x}_t}[\bar{\mathbf{y}}_t | \mathbf{y}_t]$. The definition characterizes by how much the feedback provided, in expectation, is an α-factor improvement over the presented object relative to the maximum possible improvement $U(\mathbf{x}_t, \mathbf{y}_t^*) - U(\mathbf{x}_t, \mathbf{y}_t)$, while allowing for slack ξ_t. Characterizing the feedback from boundedly rational users through Eq. (1) is sensible: a boundedly rational user may satisfice and not search the full space $\mathcal{Y}$ for the optimal $\mathbf{y}^*$ (captured by α), and may make imperfect assessments of utility (captured by ξ_t).

We define the (average) **regret** of a learning algorithm after T iterations as:

$$REG_T = \frac{1}{T} \sum_{t=1}^{T} \left(U(\mathbf{x}_t, \mathbf{y}_t^*) - U(\mathbf{x}_t, \mathbf{y}_t) \right). \quad (2)$$

The goal of a coactive learning algorithm is to minimize regret. In the rest of this paper, we assume a linear model of utility $U(\mathbf{x}, \mathbf{y}) = \mathbf{w}_*^\top \phi(\mathbf{x}, \mathbf{y})$, where $\mathbf{w}_* \in \mathbf{R}^N$ is an unknown vector. Here, $\phi(\mathbf{x}, \mathbf{y}) \in \mathbf{R}^N$ represents the joint feature vector of context $\mathbf{x}$ and object $\mathbf{y}$. We assume that this vector is bounded, *i.e,* $\forall \mathbf{x}, \mathbf{y}; \|\phi(\mathbf{x}, \mathbf{y})\|_{\ell_2} \leq R$. Note that true utility U and weight vector $\mathbf{w}_*$ are never revealed to the learning algorithm. They are only used in their evaluation.

This paper focuses on coactive learning for ranking. However, the model itself is more general and has applications in machine translation, robotics, etc.

4. The Instability Problem

The *Preference Perceptron* (Shivaswamy & Joachims, 2012) is a simple algorithm for coactive learning. Nevertheless, it can be shown to have tight regret bounds when the user feedback has no noise. However, we will show in the following subsection that it can fail catastrophically in noisy environments.

The Preference Perceptron (Algorithm 1) maintains a weight vector $\mathbf{w}_t$ which is typically initialized to $\mathbf{0}$. At each time step t, the algorithm observes the context $\mathbf{x}_t$ and presents an object $\mathbf{y}_t$ that maximizes $\mathbf{w}_t^\top \phi(\mathbf{x}_t, \mathbf{y})$ over $\mathbf{y} \in \mathcal{Y}$. The algorithm then observes user feedback $\bar{\mathbf{y}}_t$ and updates the weight vector $\mathbf{w}_t$ in the direction $\phi(\mathbf{x}_t, \bar{\mathbf{y}}_t) - \phi(\mathbf{x}_t, \mathbf{y}_t)$.

Theorem 1 *(Shivaswamy & Joachims, 2012) The expected average regret of the preference perceptron can be upper bounded, for any $\alpha \in (0, 1]$ and any $\mathbf{w}_*$ as*

$$\mathbf{E}[REG_T] \leq \frac{1}{\alpha T} \sum_{t=1}^{T} \xi_t + \frac{2R\|\mathbf{w}_*\|}{\alpha\sqrt{T}}. \qquad (3)$$

The above bound is tight in the noise-free case and does not make any assumptions, as any user behavior can be characterized by appropriate values of α and ξ_t.

In this paper, we focus on rankings of documents $\mathcal{D} = \{d_1, d_2, ..., d_n\}$ as outputs $\mathbf{y} \in \mathcal{Y}$. For such rankings, we construct the feature vector $\phi(\mathbf{x}, \mathbf{y})$ as the discounted sum $\phi(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \gamma_i \phi(\mathbf{x}, \mathbf{y}^{(i)})$ of document feature vectors $\phi(\mathbf{x}, d)$, where $\mathbf{y}^{(i)}$ is the i-th document in the ranking. The γ_i are decreasing position discounts, such that sorting by document utility $U(\mathbf{x}, d) = \mathbf{w}^\top \phi(\mathbf{x}, d)$ provides a ranking of maximum $U(\mathbf{x}, \mathbf{y})$ for a given $\mathbf{w}$.

4.1. Instability: User Study on Search Engine

We implemented the Preference Perceptron on the full-text search engine of `arxiv.org`, constructing feedback rankings $\bar{\mathbf{y}}_t$ from $\mathbf{y}_t$ by moving the clicked documents to the top (more details in Section 7). Unfortunately, the Preference Perceptron did not learn a good ranking function in this online experiment, and Figure 1 shows the comparison against a hand-tuned baseline using Interleaving (Chapelle et al., 2012). The black line shows that the Preference Perceptron (*i.e., PrefP[top]*) only barely improves over the baseline (a value of 1 would indicate equivalence).

Figure 2 gives some insight into this disappointing performance. It shows that the learned rankings do not stabilize and that the learning process oscillates. In particular, even after thousands of updates, the top 10 documents of the same query before and after 100 update steps only overlap by 4 documents on average.

Figures 1 and 2 also show (in red) the behavior of the algorithm we introduce in this paper, the *Perturbed Preference Perceptron for Ranking* (*3PR*). Note that it achieves substantial improvements over the baseline and that it does not oscillate.

4.2. Instability: Illustrative Example

Why did the Preference Perceptron oscillate? Consider the following toy problem, where the goal is to learn rankings using the feature vector construction from above. In this toy example, document utility is independent of the context $\mathbf{x}$ and only document d_1 has utility $U(\mathbf{x}, d_1) = 1$, all others have utility -1. Feature vectors $\phi(\mathbf{x}, d)$ have 2 binary features that exactly reflect utility (*i.e.,* $\phi(\mathbf{x}, d_1) = [1, 0]$ while $\forall i \in [2, n] : \phi(\mathbf{x}, d_i) = [0, 1]$). Now let us consider the following simple user model: Each iteration, users view the current $\mathbf{y}_t$ (i.e., documents ranked by $\mathbf{w}_t^\top \phi(\mathbf{x}_t, d)$). They examine each $\mathbf{y}^{(i)}$ in order, click the first one they deem to have utility 1, and then stop. However, users being an imperfect judge of utility, make each $+1/-1$ judgment with only 80% accuracy. We construct the feedback ranking $\bar{\mathbf{y}}_t$ from $\mathbf{y}_t$ by swapping the clicked document into rank 1.

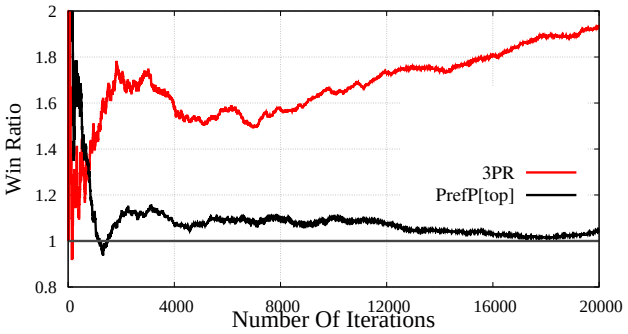

Figure 1. Results of the user study showing the ratio of wins versus the hand-tuned baseline.

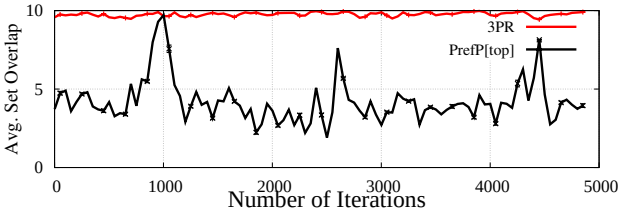

Figure 2. Number of common results in the top 10 for the same query using two different models that are 100 learning iterations apart(*i.e.,* $\mathbf{w}_t$, $\mathbf{w}_{t+100}$). Results are binned over intervals of size 50 and averaged over 100 random queries.

Let us analyze the behavior of the Preference Perceptron on this toy example. In fact, let us assume that the algorithm is initialized with the weight vector $\mathbf{w}_1 = [1, -1]$, which correctly ranks d_1 first. If the user correctly clicks $\mathbf{y}^{(1)}$, the Preference Perceptron makes no change to $\mathbf{w}_t$. However, whenever the user selects an incorrect $\mathbf{y}^{(i)}$ below (for which there is a $\sim$20% chance), the first weight decreases and the second weight increases. Eventually, the ranking will "flip" and d_1 will move to the *last* position. Even if the system eventually recovers from this catastrophic failure, the same sequence of events will lead to d_1 being placed at the bottom again. Thus, the system oscillates.

The gravity of the problem can be seen in the following simulation results. For $n = 10$ documents and DCG

discounting for γ_i (see Section 6), the average rank of d_1 within the first 1000 iterations of the Preference Perceptron is 9.36 (1 is best, 10 is worst). In fact, $\mathbf{y}^{(1)}$ is in the worst position for most of the time steps, since it takes a low-probability event of 0.2^9 to correct the ranking, but a high-probability event of 0.2 almost immediately flips it back. Note that "averaging" does not fix this oscillation problem, since it is not a result of unbiased noise. In fact, an Averaged Perceptron (Collins, 2002) showed an average rank of 9.37.

5. Stable Coactive Learning

How can we prevent these oscillations to ensure convergence and improve regret? The key problem in the toy example from above is that the user feedback incurs large slack ξ_t in (1) when d_1 is in the top position – even though it is perfectly α-informative without slack in all other cases. We now develop the Perturbed Preference Perceptron to handle this bias in the feedback and guarantee stability.

To motivate the algorithm, consider what happens if we run the Preference Perceptron, but present the user a *perturbed* ranking where, with 50% probability, we swap the top two documents. Even for the optimal weight vector $\mathbf{w}^*$, note that feedback on the perturbed ranking is now expected α-informative without slack under the given user model. This stabilizes the learning process, since preferences now often reinforce $\mathbf{w}^*$ – namely whenever the relevant document d_1 is at rank two and the user clicks on it. Running the simulation from Section 4.2 using the perturbed rankings greatly improves the average rank of $\mathbf{y}^{(1)}$ from 9.36 to 2.08.

5.1. Perturbed Preference Perceptron

Following the idea of using perturbation to combat feedback bias, Algorithm 2 defines the *Perturbed Preference Perceptron*. It is analogous to the conventional Preference Perceptron with two changes. First, the algorithm accepts a subroutine $Perturb(\hat{\mathbf{y}}_t)$ for perturbing the object $\hat{\mathbf{y}}_t = \operatorname{argmax}_{\mathbf{y}\in\mathcal{Y}} \mathbf{w}_t^\top \phi(\mathbf{x}_t, \mathbf{y})$. Second, since a perturbed object $\mathbf{y}_t$ is presented to the user, the user's preference feedback – and the subsequent update – is relative to $\mathbf{y}_t$, not $\hat{\mathbf{y}}_t$.

5.2. Theoretical Analysis

We now characterize the regret of the Perturbed Preference Perceptron as a function of the perturbation strategy. The following theorem bounds the expected regret of the Perturbed Preference Perceptron in terms of two quantities. First, consider *expected α-informativeness* of the user feedback analogous to Eq. (1),

Algorithm 2 Perturbed Preference Perceptron.

Input: $Perturb(\cdots)$, $GetFeedback(\cdots)$
  $\mathbf{w}_1 \leftarrow \mathbf{0}$ {Initialize weight vector}
  for $t = 1$ **to** T **do**
    Observe $\mathbf{x}_t$
    Compute $\hat{\mathbf{y}}_t \leftarrow \operatorname{argmax}_{\mathbf{y}\in\mathcal{Y}} \mathbf{w}_t^\top \phi(\mathbf{x}_t, \mathbf{y})$
    $\mathbf{y}_t \leftarrow Perturb(\hat{\mathbf{y}}_t)$ {Perturb Object}
    Present $\mathbf{y}_t$
    Obtain feedback $\bar{\mathbf{y}}_t \leftarrow GetFeedback(\mathbf{y}_t)$
    Update: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \phi(\mathbf{x}_t, \bar{\mathbf{y}}_t) - \phi(\mathbf{x}_t, \mathbf{y}_t)$

$$\mathbf{E}_{\bar{\mathbf{y}}_t,\mathbf{y}_t}\Big[\mathbf{w}_*^\top \phi(\mathbf{x}_t, \bar{\mathbf{y}}_t)\Big] - \mathbf{E}_{\mathbf{y}_t}\Big[\mathbf{w}_*^\top \phi(\mathbf{x}_t, \mathbf{y}_t)\Big] \geq \alpha\left(\mathbf{w}_*^\top \phi(\mathbf{x}_t, \mathbf{y}_t^*) - \mathbf{E}_{\mathbf{y}_t}\Big[\mathbf{w}_*^\top \phi(\mathbf{x}_t, \mathbf{y}_t)\Big]\right) - \xi_t. \tag{4}$$

Note that the feedback $\bar{\mathbf{y}}_t$ is relative to the perturbed $\mathbf{y}_t$, and that expectation is taken over perturbations.

Second, consider *affirmativeness* w.r.t. a perturbed $\mathbf{y}_t$,

$$\mathbf{E}_{\bar{\mathbf{y}}_t,\mathbf{y}_t}\Big[\mathbf{w}_t^\top \phi(\mathbf{x}_t, \bar{\mathbf{y}}_t)\Big] - \mathbf{E}_{\mathbf{y}_t}\Big[\mathbf{w}_t^\top \phi(\mathbf{x}_t, \mathbf{y}_t)\Big].$$

Affirmativeness reflects the relationship between noise in the user feedback and noise from perturbation relative to the current model $\mathbf{w}_t$. Positive affirmativeness indicates that the user feedback typically confirms the ordering based on the current $\mathbf{w}_t$, while negative affirmativeness indicates the opposite. Based on these two quantities, we state the following regret bound.

Theorem 2 *The expected average regret of Algorithm 2 for a perturbation strategy satisfying the following bound on the average affirmativeness,*

$$\frac{1}{T}\sum_{t=1}^{T}\left(\mathbf{E}\Big[\mathbf{w}_t^\top \phi(\mathbf{x}_t, \bar{\mathbf{y}}_t)\Big] - \mathbf{E}\Big[\mathbf{w}_t^\top \phi(\mathbf{x}_t, \mathbf{y}_t)\Big]\right) \leq \Delta, \tag{5}$$

can be upper bounded as

$$\mathbf{E}[REG_T] \leq \frac{1}{\alpha T}\sum_{t=1}^{T}\xi_t + \frac{\sqrt{4R^2 + 2\Delta}\,\|\mathbf{w}_*\|}{\alpha\sqrt{T}}. \tag{6}$$

All proofs are provided in the supplementary material. Note that the average affirmativeness defined in (5) is a quantity that can be estimated by the learning algorithm, implying a *dynamic* strategy that determines how to perturb. Note further that in the bound Δ is always zero in the absence of perturbation, which recovers the conventional Preference Perceptron and its regret bound as a special case. The above bound can be substantially tighter than that of the conventional Preference Perceptron, since it allows trading-off between Δ and $\sum \xi_t$. In the toy example from above perturbation reduced $\sum \xi_t$ to zero at a modest increase in Δ.

We also state two corollaries that give bounds on the regret w.r.t. an additive/multiplicative bound on the amount of perturbation.

Corollary 3 *The expected average regret of Algorithm 2 for a perturbation strategy satisfying*

$$\frac{1}{T}\sum_{t=1}^{T}\left(\mathbf{w}_t^\top\phi(\mathbf{x}_t,\hat{\mathbf{y}}_t)-\mathbf{E}\left[\mathbf{w}_t^\top\phi(\mathbf{x}_t,\mathbf{y}_t)\right]\right)\leq\Omega, \quad (7)$$

can be upper bounded as

$$\mathbf{E}[REG_T]\leq\frac{1}{\alpha T}\sum_{t=1}^{T}\xi_t+\frac{\sqrt{4R^2+2\Omega}\,\|\mathbf{w}_*\|}{\alpha\sqrt{T}}. \quad (8)$$

Corollary 4 *The expected average regret of Algorithm 2 for a perturbation strategy satisfying*

$$\forall t:\mathbf{E}\left[\mathbf{w}_t^\top\phi(\mathbf{x}_t,\mathbf{y}_t)\right]\geq(1-\beta)\mathbf{w}_t^\top\phi(\mathbf{x}_t,\hat{\mathbf{y}}_t) \quad (9)$$

for $0\leq\beta\leq 1$, can be upper bounded as

$$\mathbf{E}[REG_T]\leq\frac{1}{\alpha T}\sum_{t=1}^{T}\xi_t+\frac{\beta R\|\mathbf{w}_*\|}{\alpha}+\frac{\sqrt{2(4-\beta^2)}R\|\mathbf{w}_*\|}{\alpha\sqrt{T}}.$$

Corollary 3 follows immediately from Theorem 2, and Corollary 4 follows the structure of the proof in (Raman et al., 2012) for (unperturbed) coactive learning with approximate inference.

The bounds presented above not only provide a theoretical sanity check for Algorithm 2, but also give explicit guidelines for designing effective perturbation strategies that we will exploit in Section 6.

6. Perturbed Preference Perceptron for Ranking

Ranking is one of the most common learning tasks for online systems, since it is the basis for search and recommendation. These systems are ideally suited for coactive learning, since they can easily sense user interactions that provide (noisy) feedback. We now develop perturbation and feedback strategies for the Perturbed Preference Perceptron that ensure stable learning of ranking functions.

For a perturbed ranking $\mathbf{y}$, let $\bar{\mathbf{y}}$ be a feedback ranking that is derived from interactions (e.g., clicks) in $\mathbf{y}$. Our goal is a perturbation and feedback strategy such that $\bar{\mathbf{y}}$ fulfills Eq. (4) with large α and small ξ. Let us consider some properties such a strategy should have.

First, it is desirable to perturb *uniformly* throughout the ranking, so that any user experiences the same amount of perturbation no matter how deep they explore. Second, we would like to make only *local* perturbations to minimally alter the ranking. Third, the construction of the feedback ranking $\bar{\mathbf{y}}$ should be *robust* to noisy clicks, limiting the increase in ξ in Eq. (4).

Algorithm 3 Perturbation and feedback for the Perturbed Preference Perceptron for Ranking (3PR).

Function FORMPAIRS()
With prob 0.5: **return** $(\{1,2\},\{3,4\},\{5,6\}\cdots)$
else: **return** $(\{1\},\{2,3\},\{4,5\},\{6,7\}\cdots)$

Function PERTURB($\hat{\mathbf{y}},p$)
$\mathbf{y}\leftarrow\hat{\mathbf{y}}$ {Initialize with top-scoring ranking}
$Pairs\leftarrow FORMPAIRS()$
for $i=0\cdots len(Pairs)$ **do**
  $\{j,j+1\}\leftarrow Pairs[i]$ {Get Pair}
  With prob p:
    swap($\mathbf{y}[j]$,$\mathbf{y}[j+1]$); swap($Pairs[i][0]$,$Pairs[i][1]$)
return ($\mathbf{y}$, $Pairs$)

Function GET-FEEDBACK($\mathbf{y}, clicks, Pairs$)
$\bar{\mathbf{y}}\leftarrow\mathbf{y}$ {Initialize with presented object}
for $i=0\cdots len(Pairs)$ **do**
  $\{j_{upper},j_{lower}\}\leftarrow Pairs[i]$ {Get Pair}
  if $\mathbf{y}[j_{lower}]\in clicks$ AND $\mathbf{y}[j_{upper}]\notin clicks$ **then**
    swap($\bar{\mathbf{y}}[j_{upper}]$, $\bar{\mathbf{y}}[j_{lower}]$)
return $\bar{\mathbf{y}}$

These desiderata naturally lead to the perturbation and feedback strategy in Algorithm 3, which follows the FairPairs method proposed in (Radlinski & Joachims, 2006). The top-scoring ranking $\hat{\mathbf{y}}$ (e.g., $\hat{\mathbf{y}}=[d_1,d_2,d_3,d_4,d_5,d_6,...]$) is split into adjacent pairs of documents (e.g., $[(d_1,d_2),(d_3,d_4),(d_5,d_6),...]$), and each pair is flipped with probability p to produce the perturbed ranking $\mathbf{y}$ (e.g., $\mathbf{y}=[(d_2,d_1),(d_3,d_4),(d_6,d_5),...]$). Whenever the user clicks on the bottom document of a pair, top and bottom document are swapped to produce the feedback ranking $\bar{\mathbf{y}}$ (e.g., for clicks on $\{d_1,d_4,d_6\}$ in $\mathbf{y}$, we construct $\bar{\mathbf{y}}=[(d_1,d_2),(d_4,d_3),(d_6,d_5),...]$). We call Algorithm 2 using the functions from Algorithm 3 the *Perturbed Preference Perceptron for Ranking (3PR).*

We now establish regret bounds for the 3PR algorithm, using the joint feature map $\phi(\mathbf{x},\mathbf{y})$ for queries $\mathbf{x}$ and rankings $\mathbf{y}$ described in Section 4. In particular, we use position-discounting factors $\gamma_i=\frac{1}{\log_2(i+1)}$ as in the DCG metric (Manning et al., 2008).

Proposition 5 *The 3PR with swap prob. p has regret:*

$$\leq\frac{\sum_{t=1}^{T}\xi_t}{\alpha T}+\frac{p(1-\frac{\gamma_2}{\gamma_1})R\|\mathbf{w}_*\|}{\alpha}+\frac{\sqrt{2(4-p^2(1-\frac{\gamma_2}{\gamma_1})^2)}R\|\mathbf{w}_*\|}{\alpha\sqrt{T}}.$$

On the one hand, the *3PR* algorithm provides the first exploration strategy with a regret bound for FairPair feedback. On the other hand, the regret bound implies

that the swapping of pairs does not need to necessarily be "fair" (i.e., $p = 0.5$). For example, consider a **dynamic** swap strategy that, at iteration t, determines its perturbation based on the cumulative affirmativeness $R_t = \sum_{i=1}^{t-1} \mathbf{w}_i^\top \phi(\mathbf{x}_i, \bar{\mathbf{y}}_i) - \mathbf{w}_i^\top \phi(\mathbf{x}_i, \mathbf{y}_i)$ and the maximum perturbation $D_t = \mathbf{w}_t^\top \phi(\mathbf{x}_t, \hat{\mathbf{y}}_t) - \mathbf{w}_t^\top \phi(\mathbf{x}_t, \mathbf{y}'_t)$, where $\mathbf{y}'_t$ is the ranking obtained by swapping all pairs in $\hat{\mathbf{y}}_t$. Note that D_t is an apriori bound on the maximum affirmativeness of the user feedback at iteration t. Based on these observable quantities, we propose the following dynamic adaptation rule for the swap probability with the following regret bound.

Proposition 6 *For $\Delta \geq 0$, dynamically setting the swap prob. of $3PR$ to be $p_t \leq max(0, \frac{\Delta \cdot t - R_t}{D_t})$ has regret:*

$$\leq \frac{1}{\alpha T} \sum_{t=1}^{T} \xi_t + \frac{\|\mathbf{w}_*\|}{\alpha \sqrt{T}} \sqrt{4R^2 + 2\Delta + (\gamma_1 - \gamma_2) R \sqrt{\frac{4R^2 + 2\Delta}{T}}}.$$

7. User Study on Search Engine

To investigate the real-world effectiveness of the *3PR* algorithm compared to the conventional Preference Perceptron (*PrefP*), we performed a user study (already alluded to in Section 4.1) on the full-text search engine of `arxiv.org`. Results were collected in two subsequent runs, one for each method. We used a query-document feature vector $\phi(\mathbf{x}, d)$ of length 1000, which included various query-dependent features (e.g., query-title match) and query-independent features (i.e. the age of a document).

Users coming to the search engine were randomly assigned one of two groups with equal probability. For users assigned to the learning group, we used the clicked documents of their query to construct the feedback rankings as described previously (i.e. move clicked documents to top for *PrefP*, and paired feedback with swap probability 0.5 for *3PR*). For users assigned to the evaluation group, the ranking induced by the current weight vector was compared to a baseline ranking that was generated with manually selected weights. We employed Balanced Interleaving (Joachims, 2002; Chapelle et al., 2012) for this comparison, which is a paired and blind test for eliciting a preference between two rankings. We record how often a user prefers a learned ranking over the baseline (i.e. wins a pairwise comparison). Both learning algorithms were initialized to start with the weights of the baseline ranker.

Figure 1 shows the results of the experiment, plotting the win/loss ratio of each learning method over the baseline. While *PrefP* initially performs well, its win ratio eventually hovers only slightly above 1. The *3PR* method, on the other hand, converges to a win-ratio of 1.9, which is large (and highly significant according to a Binomial Sign Test) compared to the experiments in (Chapelle et al., 2012). Finally, Figure 3 shows the average affirmativeness Δ from Theorem 2. It shows that Δ is positive and stabilizes, indicating that the amount of perturbation was appropriate.

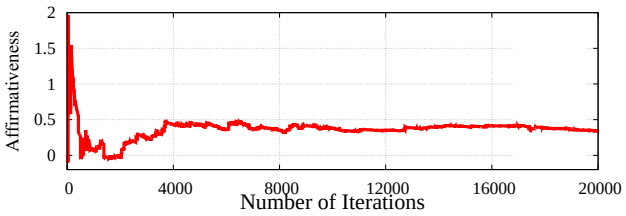

Figure 3. Average affirmativeness of $3PR$ in user study.

8. Experiments on Benchmark Data

To get more detailed insights into the empirical performance of the proposed methods, we also conducted offline experiments on benchmark datasets.

First, we use the Yahoo! learning to rank dataset (Chapelle & Chang, 2011) (abbreviated *Websearch*), which consists of roughly 28k queries and 650k documents (*i.e.,* URLs). For each query-url pair in the dataset, there is a joint feature vector $\phi(\mathbf{x}, d)$ of 700 features and a relevance rating in the range 0-4. In each iteration, the system is given a query and presents a ranking. In total, the system was run for over 28k iterations. All our results are averaged over 20 different runs (by randomizing the query stream order).

Second, we simulate two news recommendation tasks, using the RCV1 (Lewis et al., 2004) and the 20 Newsgroups datasets (abbreviated *News*). The RCV1 corpus contains over 800k documents that each belong to one or more of 103 topics, while the News dataset contains 20k documents that each belong to one of 20 topics. We used TFIDF features, with a total feature set of size 3k for RCV1 and 1k for News[1]. In these experiments, we simulated user interests by equating users with single topics. The algorithms were run for 50K iterations for RCV and 10K for News (by cycling through the data), and the results are averaged over all users (i.e. topics).

We assume the following model of user interaction. The user scans the ranking from the top down to the tenth result and clicks on up to five results. To study the stability of the different algorithms, clicks are corrupted by noise. For RCV1 and News, a user goes down the ranking and clicks on relevant documents, but with η chance of incorrectly assessing the relevance of a document ($\eta = 0.2$). On the search dataset, the user's relevance assessment are corrupted by adding independent Gaussian noise ($\sigma = 1$) to the true relevance of

[1]Feature selection as per the max. class χ^2 metric similar to (Lewis et al., 2004).

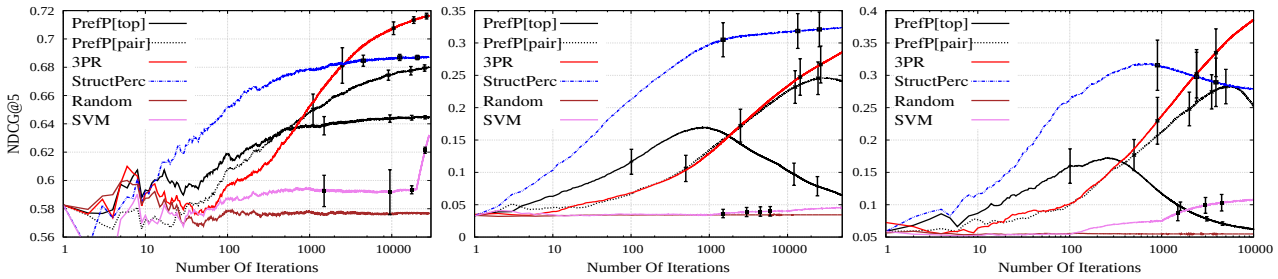

Figure 4. Learning curves for all algorithms on Websearch (left), RCV1 (middle), and News (right).

each document; the user then clicks on the 5 documents with highest (corrupted) relevance in the top 10.

8.1. What is the Generalization Performance of the Perturbed Preference Perceptron?

First, let us compare the *3PR* against alternative algorithms, including the conventional Preference Perceptron where clicked documents are moved to the top of the feedback ranking (*PrefP[top]*). We also consider a variant of the conventional Preference Perceptron that uses the same paired feedback as *3PR*, but has swap probability zero (*PrefP[pair]*).

To compare with a regularized batch learner, a ranking SVM with move-to-top feedback was trained at (10,100,1k,10k,20k) iterations using a setup similar to (Shivaswamy & Joachims, 2012). Between training steps, the current predictor is used to present rankings. For this experiment, we retrospectively pick the best C value (per run) and report the NDCG@5 corresponding to that C (*i.e.,* we are biasing in favor of the SVM).

As a (rough) upper bound, we consider a *Structured Perceptron* (Collins, 2002) that is trained with the optimal $\mathbf{y}^*$ without added noise. This simulates clean and exhaustive expert feedback, which is typically unobtainable in practice. As a lower bound, we report the performance of uniformly random lists of documents.

The results for this experiment are shown in Figure 4. It can be seen that the *3PR* achieves a significantly higher NDCG@5 compared to other online algorithms *PrefP[top]* and *PrefP[pair]* at the end of the runs. In fact, *PrefP[top]* fails catastrophically on two of the datasets like in the toy example from Section 4.2. *PrefP[pair]* is more stable, but shows similar deterioration as well. An interesting extension could be the combination of aggressive move-to-top feedback in early iterations with more conservative *3PR* updates later.

Due to the biased training data that violates the iid model, the SVM performs poorly. We conjecture that more frequent retraining would improve performance, but be orders of magnitude more computationally expensive (especially with realistic model selection).

Table 1. NDCG@5 of presented and perturbed rankings after maximum number of iterations.

	Websearch	RCV1	News
Presented $\mathbf{y}$	.717 ± .002	.286 ± .028	.386 ± .035
Predicted $\hat{\mathbf{y}}$	.723 ± .002	.291 ± .028	.397 ± .035

The Structured Perceptron learns faster than *3PR*, but despite receiving much stronger training data (optimal $\mathbf{y}^*$ without feedback noise) its eventual performance is worse than *3PR* on two tasks. This may be surprising at first glance. However, it is known that Perceptron-style algorithms tend to work less well on multiclass/structured problems without good linear fit, and that they can even degenerate (Lee et al., 2004; Chen & Xiang, 2006). Intriguingly, the *3PR* seems less affected by this problem.

8.2. How does the Perturbed Ranking Compare to the Optimal Prediction?

In the case of $3PR$, the algorithm first computes the argmax ranking $\hat{\mathbf{y}}$ but then presents the perturbed ranking $\mathbf{y}$. While the previous section showed the NDCG@5 of the presented rankings $\mathbf{y}$, Table 1 shows the NDCG@5 for both $\mathbf{y}$ and $\hat{\mathbf{y}}$. As expected, the presented rankings are of slightly lower quality than $\hat{\mathbf{y}}$ due to perturbation. However, this small loss in quality leads to a big gain in the overall learning process in the long run — as demonstrated by the poor performance of $PrefP[pair]$.

8.3. How much Perturbation is Needed?

While complete lack of perturbation leads to divergence, it is unclear whether a swap probability of 0.5 is always optimal. Intuitively, we expect that with low noise, smaller perturbation suffices to achieve high performance, while at higher noise levels, perturbation probabilities need to be higher to overcome the noise.

Figure 5 explores the effect of different perturbation rates p in Alg. 3 on the performance of the *3PR*. It appears that a swap probability of more than 0.5 usually hurts. While 0.5 typically performs reasonably well, 0.25 produces the best performance on RCV1.

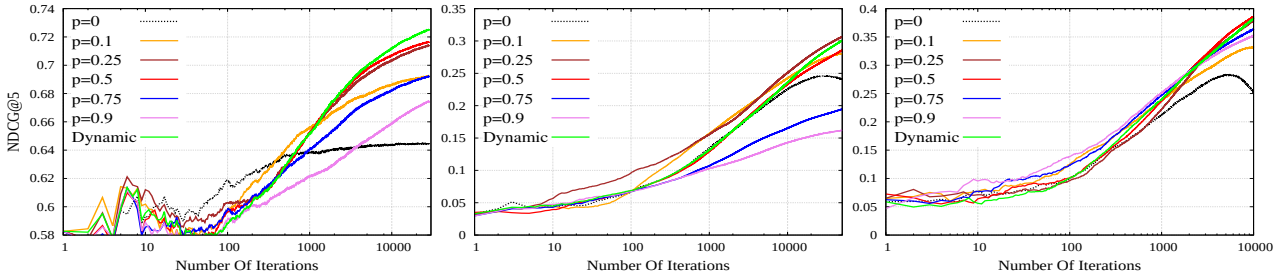

Figure 5. NDCG@5 of the *3PR* algorithm for different swap probabilities and the dynamically adapted swap probability on Websearch (left), RCV1 (middle), and News (right).

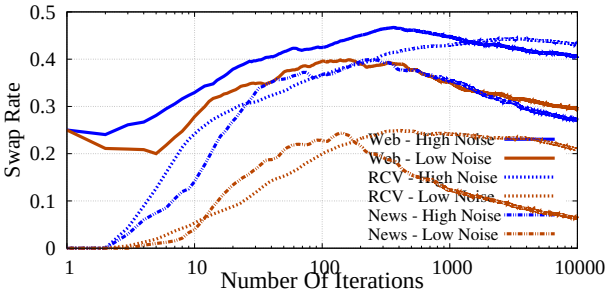

Figure 6. Change in average swap probability of the dynamic method with $\Delta = 0$ low and high feedback noise.

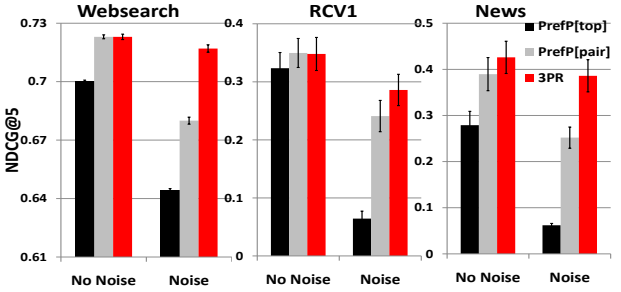

Figure 7. Performance of *PrefP[top]*, *PrefP[pair]* and *3PR* at the max. no. of iterations with and w/o feedback noise.

8.4. Can we Automatically Adapt the Perturbation Rate?

Ideally, we would like to automatically select an appropriate swap probability. Note that this does not need to be a single fixed number, but can change over the learning run. Proposition 6 defined such a perturbation strategy that accounts for the current affirmativeness and adjusts the swap probability to optimize the regret bound in Theorem 2. The results of this dynamic strategy using $\Delta = 0$ are also included in Figure 5. As we see from the figure, the method is able to adjust the swap rates to achieve performance among the best.

Figure 6 shows how the swap probability chosen by the dynamic strategy varies. It can be observed that the swap probability first increases and then eventually decreases to exploit more often. When changing the noisiness of the user feedback, we find that the strategy automatically accounts for larger noise by increasing the swap rate relative to the low noise setting.

8.5. How does Noise Affect the Perturbed Preference Perceptron?

Our motivation for the *3PR* algorithm was the inability of *PrefP[top]* to handle the noise in the feedback it encountered in the user study. Therefore, all benchmark experiments we reported included feedback noise as described in the beginning of Section 8. But how does the *3PR* algorithm perform without added noise?

Figure 7 compares the performance of *3PR* to that of *PrefP[top]* and *PrefP[pair]* with and without user feedback noise. Even with no feedback noise, *3PR* outperforms *PrefP[top]* and is at least comparable to *PrefP[pair]*. Furthermore, the performance of *3PR* declines much less when noise is introduced, as compared to the other algorithms.

Note that "*no noise*" is somewhat of a misnomer. While we did not add any noise, even the expert provided ratings probably contain some amount of noise. Moreover, any feedback that cannot be explained by a linear model appears as noise to the algorithm, which is likely to be a substantial source of noise in any real-world application. The *3PR* algorithm handles this gracefully.

9. Conclusions

We presented the Perturbed Preference Perceptron, an online algorithm for learning from biased and noisy preferences in the coactive learning model. Unlike existing methods, the algorithm is stable and does not oscillate. The key idea lies in a controlled perturbation of the prediction, and we give theoretical regret bounds that characterize the behavior of the new algorithm. Focusing on learning to rank, we develop perturbation strategies for ranking and show that the new algorithm substantially outperforms existing methods in an online user study, as well as in benchmark experiments. This work was supported in part by NSF Awards IIS-0905467, IIS-1142251, and IIS-1247696.

References

Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002a.

Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. The non-stochastic multi-armed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002b.

Bartók, Gábor, Pál, Dávid, and Szepesvári, Csaba. Toward a classification of finite partial-monitoring games. In *ALT*, pp. 224–238, 2010.

Cesa-Bianchi, N. and Lugosi, G. *Prediction, learning, and games*. Cambridge University Press, 2006.

Chapelle, O. and Chang, Y. Yahoo! learning to rank challenge overview. *JMLR - Proceedings Track*, 14: 1–24, 2011.

Chapelle, O., Joachims, T., Radlinski, F., and Yue, Yisong. Large-scale validation and analysis of interleaved search evaluation. *ACM Transactions on Information Systems (TOIS)*, 30(1):6:1–6:41, 2012.

Chen, D. and Xiang, D. The consistency of multicategory support vector machines. *Adv. Comput. Math*, 24(1-4):155–169, 2006.

Chu, W. and Ghahramani, Z. Preference learning with gaussian processes. In *ICML*, 2005.

Collins, M. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *EMNLP*, 2002.

Crammer, K. and Singer, Y. Pranking with ranking. In *NIPS*, 2001.

Flaxman, A., Kalai, A. T., and McMahan, H. B. Online convex optimization in the bandit setting: gradient descent without a gradient. In *SODA*, 2005.

Freund, Y., Iyer, R. D., Schapire, R. E., and Singer, Y. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969, 2003.

Herbrich, R., Graepel, T., and Obermayer, K. Large margin rank boundaries for ordinal regression. In *Advances in Large Margin Classifiers*. MIT Press, 2000.

Joachims, T. Optimizing search engines using clickthrough data. In *KDD*, 2002.

Lee, Yoonkyung, Lin, Yi, and Wahba, Grace. Multicategory support vector machines. *Journal of the American Statistical Association*, 99(465):67–81, 2004.

Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. RCV1: A new benchmark collection for text categorization research. *JMLR*, 5:361–397, 2004.

Liu, T-Y. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3, March 2009.

Manning, C., Raghavan, P., and Schütze, H. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

Radlinski, F. and Joachims, T. Minimally invasive randomization for collecting unbiased preferences from clickthrough logs. In *AAAI*, pp. 1406–1412, 2006.

Raman, K., Shivaswamy, P., and Joachims, T. Online learning to diversify from implicit feedback. In *KDD*, 2012.

Shivaswamy, P. and Joachims, T. Online structured prediction via coactive learning. In *ICML*, 2012.

Yue, Y. and Joachims, T. Interactively optimizing information retrieval systems as a dueling bandits problem. In *ICML*, 2009.

Yue, Y., Broder, J., Kleinberg, R., and Joachims, T. The k-armed dueling bandits problem. In *COLT*, 2009.

Zinkevich, M. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*, 2003.