The Offset Tree for Learning with Partial Labels

Alina Beygelzimer
IBM Research
beygel@us.ibm.com

John Langford
Yahoo Research
jl@yahoo-inc.com

Tong Zhang
Rutgers Statistics Department
tongz@rci.rutgers.edu

Abstract

We present an algorithm, called the offset tree, for learning in situations where a loss associated with different decisions is not known, but was randomly probed. The algorithm is an optimal reduction from this problem to binary classification. In particular, it has regret at most $(k-1)$ times the regret of the binary classifier it uses, where k is the number of decisions, and no reduction to binary classification can do better. We test the offset tree empirically and discover that it generally results in superior (or equal) performance, compared to several plausible alternative approaches.

1 Introduction

This paper is about a new learning setting, which can be thought of as an *offline* variant of the *contextual* k-armed bandit problem.

- The k-armed bandit problem [14] has been well studied [2, 3, 6]. In the basic setting, a learner repeatedly chooses one of k actions, and then learns the value of that action (but not the value of other actions). The goal is to compete with the best constant action over many rounds. The *contextual* k-armed bandit problem [3, 12] is a variant where the algorithm receives a feature vector x before choosing an action, and the goal is to compete with the best policy $\pi : x \to \{1, ..., k\}$ in some set of policies. The key difference between the bandit setting and the traditional online setting is that in the former, the information (label) is observed for the chosen action only.
- The *offline* nature of the problem is new here. Imagine that some existing policy (which we do not control) chooses actions and observes rewards. This process generates a training set of $(x, a, p(a), r)$ tuples where x is the feature vector, a is the action chosen, $p(a)$ is the probability that a was chosen, and r is the reward observed. The goal in the offline version of the problem is to learn a policy that performs well on future observations. Since the setting is offline, none of the online algorithms for k-armed bandit or k-armed contextual bandits are applicable.

Are offline contextual bandit problems interesting? We believe the answer is yes, as supported by the following applications:

1. **Medical Studies**. In a number of medical studies, patients are randomly allocated to treatment groups. The typical goal in these studies is to determine if treatment A is more effective than treatment B. Using the techniques described here, we can use the results of the study to build a policy for choosing between treatments A and B, *conditioned on the patient history*.
2. **Content Recommendation**. Many internet sites recommend ads, webpages, or news stories based on user history, search engine queries, or other observable quantities. If some policy chooses to display these items at random, a reward may be observed in terms of whether a user clicks on the item.

In both of the above settings, it may be tempting to simply use statistical techniques where we condition on the observations, and then use the empirical expected outcome for different actions. This approach scales very poorly with the amount of context information, because it requires exponentially many samples in the number of bits of context. As an example, conditioning on the search engine query might be possible for common queries, but a user's query sequence, together with the user's geographic location, is almost always unique.

Defining the problem If the policy generating the data is deterministic, then it is easy to come up with examples where it is impossible to predict which policy in a set of policies is preferred. Given the necessity of randomization, we make a simplifying assumption that the randomization is uniform. Although this assumption is often not justified in practice, this is not a significant limitation because the results can be easily generalized to nonuniform distributions. For example, one can use rejection sampling techniques to convert nonuniform distributions into uniform distributions. All our bounds still hold in this case, up to a constant depending on the ratio of the uniform distribution and the nonuniform randomizing distribution. One can also directly generalize the proofs to nonuniform distributions.

Here is a formal description of data generation:

1. Some unknown distribution $D(x, \vec{r})$ generates a feature vector x and a vector $\vec{r} = (r_1, r_2, ..., r_k)$, where for $i \in \{1, \ldots, k\}$, $r_i \in [0, 1]$ is the reward of the i-th action.
2. The offline policy chooses an action a uniformly at random.
3. The reward r_a is revealed.

The goal is to learn a policy $\pi(x)$ for choosing action a given x, with the goal of maximizing the expected reward. We call D a *partial label problem*.

Plausible approaches One method of solving a relatively unstudied problem is to reduce it to one of the problems we know how to solve.

One natural attempt is to regress on the reward given x and a, and then choose according to the largest predicted reward. However, this approach has regret (the difference between the incurred loss and the smallest achievable loss on the problem) that scales with the square root of the regressor's regret. (A detailed proof of this claim can be found in the appendix.) The rationale for bounding the regret of the policy on the original problem D in terms of the regret of the regression algorithm on the induced regression problem is to subtract off the inherent noise in both problems, directly bounding the excess loss due to suboptimal prediction. The notion of regret is formalized in later sections.

Another approach is to use the one-step reinforcement learning reduction [20]. Given a random distribution $p(a)$ over the actions, the basic idea is to create *importance-weighted* multiclass examples $(x, a, r_a/p(a))$, where r_a is the reward received from choosing action a. The created samples are then fed into an importance-weighted multiclass classification algorithm, with the output classifier used to make future predictions. It can be shown that when $p(a)$ is uniform, the resulting regret on the original partial label problem is bounded by k times the importance-weighted multiclass regret (see the appendix). When reduced to binary classification using the standard all-pairs reduction [7], we get a bound of $k(k-1)$ times the binary regret. Other multiclass to binary reductions can improve the dependence on k, but they all yield worse results than our approach.

Our contribution We present the Offset Tree algorithm for reducing the partial label problem to binary classification. The algorithm uses a trick to remove some of the uncertainty due to the lack of information. This trick is easiest to understand in the case of $k = 2$ choices, covered in Section 2. When the observed reward r_a of our choice a is low, we essentially pretend that the other choice a' was chosen and a different reward $r'_{a'}$ was observed. Precisely how this is done and why this works is driven by the analysis. This basic trick is composable in a binary tree structure for $k > 2$, as described in Section 3.

The Offset Tree reduction guarantees that for any binary classifier c, the regret of the corresponding policy π_c on the original partial label problem is bounded by the regret of c on the induced binary classification problem, times $k - 1$, where π_c is the policy generated by the reduction using c.

Section 4 proves that no reduction to binary classification has a tighter upper bound. This is the first nonconstant lower bound proved for learning reductions. Note that k-class classification can

be reduced to binary classification with a regret guarantee that does not depend on k. This contrast suggests that learning in this setting is fundamentally harder than supervised learning. Since the upper and lower bounds match perfectly, the Offset Tree algorithm is a perfect regret reduction.

An empirical comparison of the three approaches is presented in Section 5.

Prior work The problem considered here is a noninteractive version of the bandits problem with side information [19], which has been analyzed under various additional assumptions about how the world behaves [10, 13, 15, 17], including payoffs as a linear function of the side information [1, 2]. The Exp4 algorithm [3] has a nice assumption-free analysis. However, it is intractable when the number of policies we want to compete with is large, and it relies on careful control of the action choosing distribution.

Sample complexity results for policy evaluation in reinforcement learning [9] and contextual bandits [12] show that an Empirical Risk Minimization type algorithms can succeed in this setting. Compared to these results, (1) we show that it is possible to efficiently reuse existing, fully supervised learning algorithms without the need to modify them, and (2) we show how to best do this.

Transformations from partial label problems to fully supervised problems can be thought of as methods for dealing with sample selection bias [8], which are heavily studied in Economics and Statistics (leading to a Nobel prize in Economics in 2000).

2 The Binary Case

An *importance-weighted* k-class classification problem is defined by a distribution D over $X \times Y \times W$, where X is some feature space, Y with $|Y| = k$ is the label space, and $W \subset [0, \infty)$. The goal is to learn a classifier $c : X \to Y$ minimizing the *importance-weighted loss*

$$\ell(c, D) = \mathbf{E}_{(x,y,w)\sim D}[w \cdot \mathbf{1}\left(c(x) \neq y\right)],$$

given training examples of the form $(x, y, w) \in X \times Y \times W$, where w is the cost associated with mislabeling x, and $\mathbf{1}$ is the indicator function which is 1 when its argument is true, and 0 otherwise.

This section deals with the special case of $k = 2$. For simplicity, let the two choices be named 1 and -1. The key insight appears in step (1) of Algorithm 1, where importance weighted binary examples are formed. The offset of $1/2$ changes the range of importances, effectively derandomizing the induced problem.

A folk theorem (see [21]) says that for any importance-weighted distribution P, there exists a constant $\overline{w} = \mathbf{E}_{(x,y,w)\sim P}[w]$ such that for any predictor $h : X \to Y$,

$$\mathbf{E}_{(x,y,w)\sim P'}[\mathbf{1}(h(x) \neq y)] = \frac{1}{\overline{w}}\mathbf{E}_{(x,y,w)\sim P}[w \cdot \mathbf{1}(h(x) \neq y)],$$

where P' is the distribution over $X \times Y$ defined by

$$P'(x, y, c) = \frac{c}{\overline{w}}P(x, y, c),$$

marginalized over c. In other words, choosing h to minimize the error rate under P' is equivalent to choosing h to minimize the expected cost under P. Thus, we can convert the importance-weighted binary examples formed in step (1) of the algorithm into binary examples by re-weighting the underlying distribution according to the importances, and then apply any binary classification algorithm on the re-weighting training set. This conversion can be done using the Costing algorithm [21], which alters the underlying distribution using rejection sampling on the importance weights.

A binary learning algorithm Classify is used as a subroutine. Once importance-weighted binary examples are converted into binary examples, they are fed into Classify.

This way, Binary-Offset transforms partial label examples into binary examples of the form (x, y), where x is a set of feature values and y is a binary label. This process also implicitly transforms a distribution D defining the partial label problem into a distribution over binary examples. We denote this distribution by BO(D).

The *error rate* of a classifier $c : X \to \{1, -1\}$ on distribution BO(D) is defined as

$$e(c, \text{BO}(D)) = \mathbf{Pr}_{(x,y)\sim \text{BO}(D)}[c(x) \neq y].$$

Algorithm 1: Binary-Offset (binary learning algorithm Classify, partial label dataset S)

set $S' = \emptyset$
for each $(x, a, r_a) \in S$ **do**
  Form an importance weighted example

$$(x, y, w) = (x, \text{sign}\left(a\left(r_a - 1/2\right)\right), 2\left|r_a - 1/2\right|). \tag{1}$$

  Add (x, y, w) to S'.
return Classify(Costing(S')).

The corresponding *binary regret* is given by

$$\text{reg}_e(c, \text{BO}(D)) = e(c, \text{BO}(D)) - \min_{c'} e(c', \text{BO}(D)),$$

where the min is over all classifiers $c' : X \to \{1, -1\}$. Thus regret is the difference between the incurred loss and the lowest achievable loss on the same problem. The theorem stated below shows that the binary regret controls the partial label regret. Statements bounding regret are more desirable than statements bounding losses because regret analysis separates excess loss from unremovable noise, making the statement nontrivial even for noisy problems.

For the $k = 2$ partial label case, the policy that a classifier c induces is simply the classifier. The value of a policy π is defined as

$$\eta(\pi, D) = \mathbf{E}_{(x,\vec{r})\sim D}\left[r_{\pi(x)}\right],$$

and the regret of policy π is defined as

$$\text{reg}_\eta(\pi, D) = \max_{\pi'} \eta(\pi', D) - \eta(\pi, D).$$

We prove a theorem relating the regret of c on the partial label problem D to the regret of c on the induced binary distribution $\text{BO}(D)$. Note that the theorem is quantified over all classifiers, which includes the classifier returned by Classify in the last line of the algorithm.

Theorem 2.1. (Binary Offset Regret) For all 2-class partial label problems D and all binary classifiers c,

$$\text{reg}_\eta(c, D) \leq \text{reg}_e(c, \text{BO}(D)).$$

The proof is in the appendix.

3 The Offset Tree Reduction

The technique in the previous section can be applied repeatedly using a tree structure to give an algorithm for general k. Consider a maximally balanced binary tree on the set of k choices, which essentially implements a single elimination tournament on the choices, conditioned on a given observation x. Each internal node in the tree is associated with a binary classification problem. Working from the leaves toward the root, each internal node predicts which of its two inputs has the larger expected reward. For each node, $\text{TREE}(a)$ returns the binary label defining whether the action a comes from left or right subtrees. Each input is either a leaf or a winning choice from another internal node closer to the leaves.

At each node in the tree, the same importance weight offsetting technique is used as in the binary case described in Section 2.1. The tree structure can be constructed recursively: The root node contains all choices, or labels. Starting from the root, the set of labels at each internal node is recursively split in half, until singletons are reached.

The training algorithm, Offset-Tree, is given in Algorithm 2. The testing algorithm defining the predictor is given in Algorithm 3.

The theorem below gives an extension of Theorem 2.1 for general k. To simplify the analysis, we use a simple trick which allows us to consider only a single induced binary problem, and thus a single binary classifier c. The trick is to add the node index n as an additional feature into each

Algorithm 2: Offset-Tree (binary learning algorithm Classify, partial label dataset S)

Fix a binary tree T over the choices
for each internal node n in order from leaves to root **do**
  Let $S_n = \emptyset$
  for each sample $(x, a, r_a) \in S$ such that a is in the leaves of the subtree with root n and every internal node on the path from n to a predicts a **do**
    Let a' be the other action at node n
    If $r_a < 1/2$, add example $(x, \text{TREE}(a'), 2(1/2 - r_a))$ to S_n, else add $(x, \text{TREE}(a), 2(r_a - 1/2))$
  Let $c_n = \text{Classify}(\text{Costing}(S_n))$
return $\{c_n\}$

Algorithm 3: Offset-Test (classifiers $\{c_n\}$, unlabeled example x)

return unique action a for which every classifier c_n from leaf to root prefers the action.

importance weighted binary example created by the algorithm, and then train based upon the union of all the training sets.

The fact that classifiers are conditionally dependent on other classifiers closer to the leaves causes no problem since we quantify over *all* classifiers and thus we can regard the learned classifier as fixed. In practice, one can either use multiple classifiers (as we do in our experiments) or use iterative techniques for dealing with the cyclic dependence.

As before, a partial label problem D and the classifier c have a well-defined induced binary classification problem formed by simply drawing a partial label example and seeing which binary classification examples the reduction creates. We denote this induced distribution as $\text{OT}(D)$. Similarly, we denote the policy induced by the Offset-Test algorithm using the classifier c by OT_c. For the following theorem, the definitions of regret are from section 2.1.

Theorem 3.1. (Offset Tree Regret) *For all k-class partial label problems D, for all binary classifiers c,*

$$\begin{aligned}\text{reg}_\eta(\text{OT}_c, D) &\leq \text{reg}_e(c, \text{OT}(D)) \cdot \mathbf{E}_{(x,\vec{r})\sim D} \sum_{n(a,a')\in T} [\,|r_a - 1/2| + |r_{a'} - 1/2|\,] \\ &\leq (k-1)\,\text{reg}_e(c, \text{OT}(D)),\end{aligned}$$

where $n(a, a')$ ranges over the $(k-1)$ internal nodes in T, and a and a' are its inputs determined by c's predictions.

The result in this theorem is the best we know how to accomplish in the worst case. With a little bit of help, we can however do even better. Instead of using $1/2$, we can use an arbitrary threshold $t(x, a, a')$. The one that minimizes the regret bound turns out to be the median value of the reward for a and the reward for a' given x. This has an important implication when designing the tree: choices which tend to have similar rewards should generally be grouped together.

4 Lower Bound

This section shows that no method for reducing the partial label setting to binary classification can do better. First we formalize a learning reduction, which relies upon a binary classification oracle. The lower bound we prove below holds for *all* such learning reductions.

Definition 4.1. (Binary Classification Oracle) A binary classification oracle O is a (stateful) program that supports two kinds of queries:

1. **Advice**. An advice query $O(x, y)$ consists of a single example (x, y) where x is a feature set and $y \in \{1, -1\}$ is a binary label. It has no return. An advice query is equivalent to presenting a training example.

2. **Predict**. A predict query $O(x)$ is made with a feature set x. The return value is a binary label, either 1 or -1.

We typically analyze learning reductions with respect to expectations over examples. However, all learning reductions work on a per-example basis, and that representation is somewhat easier to work with here.

Definition 4.2. (Learning Reduction) A learning reduction is two (possibly randomized) algorithms R and R^{-1}.

1. The algorithm R takes a partially labeled example (x, a, r_a) and a binary classification oracle O as input. It forms a (possibly dependent) sequence of Advice queries $(x_1, y_1), (x_2, y_2), \ldots$.

2. The algorithm R^{-1} takes an unlabeled example x and a binary classification oracle O as input. It asks a (possibly dependent) sequence of Predict queries $x_1, x_2, x_3, ...$, and makes a prediction dependent only on the Oracle's predictions $f(y_1, y_2, \ldots)$.

Since R has access to the partially labeled example and R^{-1} does not, it's goal is to advise R^{-1} of which action to choose. The only means of communication it has is via the Binary Classification Oracle which adversarially chooses to answer evaluation queries incorrectly.

In this setting, the error rate of the binary classifier is defined for any input x as the fraction of advice queries $O(x_i, y_i)$ for which evaluation queries are incorrect: $O(x_i) \neq y_i$. More generally, when there is a distribution over x we take the expected value of this fraction.

Theorem 4.1. *For all reductions R, R^{-1}, there exists a partial label problem D and an oracle O such that*

$$\mathrm{reg}_\eta(R^{-1}(O), D) \geq (k-1)\,\mathrm{reg}_e(O, R(D)).$$

The proof is provided in the appendix.

5 Experimental Results

We compared the performance of Offset Tree to two other methods mentioned in the introduction. Ideally, this comparison would be with a data source in the partial label setting. Unfortunately, data of this sort is rarely available publicly, so we used many publicly available multiclass datasets and allowed queries for the reward (1 or 0 for correct or wrong) of only one value per example.

The first method for reducing a partial label problem to a well known learning problem is to regress on the rewards of different choices, either by adding the choice into the features or by learning a separate regressor for each choice, and then choosing the argmax of the predicted values.

The second method creates importance weighted multiclass examples (x, a, kr_a), where a is chosen uniformly at random from $\{1, \ldots, k\}$ and r_a is the reward of choosing a. The larger the reward, the higher the importance of predicting a on x. The created samples are then fed into an importance-weighted multiclass prediction algorithm, with the output classifier used to make predictions on the test set. (We remove the importances in the k-class problem using the Costing reduction [21], which can be understood as rejection sampling according to the weight values. The resulting multiclass problem is reduced to binary classification using the All-Pairs reduction [7], described in the introduction.) Further details about both methods and their analysis are provided in the appendix.

We performed the comparison on a number of publicly available multiclass datasets [16]. For all datasets, we report the average result over 10 random splits (fixed for all methods), with $2/3$ of the dataset used for training and $1/3$ for testing.

Figure 1 shows the error rates (in %) of Offset-Tree plotted against the error rates of Regression (left) and Importance-Weighting (right). Decision trees (J48 in Weka [18]) were used as a base binary learning algorithm for both Offset-Tree and Importance-Weighting. For the regression approach, we learned a separate regressor for each of the k choices. (A single regressor trained by adding the choice as an additional feature performed worse.) M5P and REPTree, both available in Weka [18], were used as base regression algorithms.

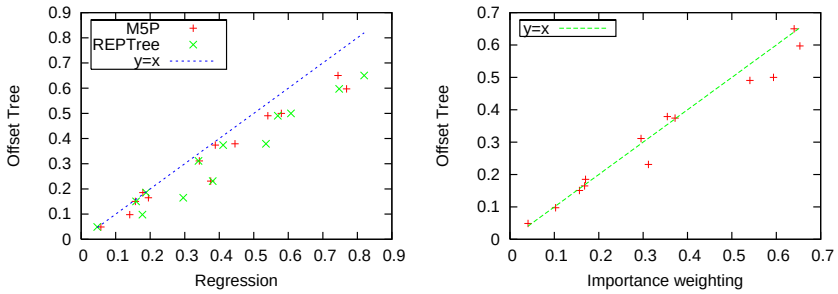

Figure 1: Error rates (in %) of Offset Tree versus the regression approach using two different base regression algorithms (left) and Offset Tree versus Importance Sampling (right) on several different datasets using decision trees as a base classifier learner.

Offset Tree clearly outperforms Regression, in some cases considerably. The advantage over Importance Weighting is moderate—often the performance is similar and occasionally it is substantially better.

We did not perform any optimization or parameter tuning. All datasets tested are included. Note that although some error rates appear large, we are choosing among k alternatives and thus an error rate of less than $1 - 1/k$ gives an advantage over random guessing. Dataset-specific test error rates are reported in Table 5.

	Properties			Single regressor		k regressors		
Dataset	k	Examples	Weighting	M5P	REPTree	M5P	REPTree	Partial Tree
ecoli	8	336	0.3120	0.5663	0.3376	0.3752	0.3811	0.2311
flare	7	1388	0.1565	0.1570	0.1685	0.1570	0.1592	0.1506
glass	6	214	0.5938	0.6662	0.5846	0.5800	0.6077	0.5000
letter	25	20000	0.3546	0.6974	0.5491	0.4456	0.5352	0.3790
lymph	4	148	0.2953	0.5267	0.4622	0.3422	0.3400	0.3114
optdigits	10	5620	0.1682	0.5426	0.4108	0.1948	0.2956	0.1649
page-blocks	5	5473	0.0407	0.0590	0.0451	0.0571	0.0465	0.0488
pendigits	10	10992	0.1029	0.2492	0.1840	0.1408	0.1774	0.0976
satimage	6	6435	0.1703	0.2027	0.1968	0.1787	0.1878	0.1853
soybean	19	683	0.6533	0.8824	0.7327	0.7688	0.7473	0.5971
vehicle	4	846	0.3719	0.6142	0.5665	0.3886	0.4114	0.3743
vowel	11	990	0.6403	0.9034	0.8919	0.7440	0.8198	0.6501
yeast	10	1484	0.5406	0.6626	0.5679	0.5406	0.5697	0.4904

6 Discussion

We have analyzed the information-theoretic (or prediction-theoretic) tractability of learning when we only know the outcome of one alternative from a set of k alternatives. The Offset-Tree approach has a worst-case dependence on $k - 1$ (Theorem 3.1), and no other reduction approach can provide a better guarantee.

The algorithms presented here show how to learn from one step of exploration. By aggregating information over multiple steps, we can learn good policies using any binary classification method. A straightforward extension of this method to deeper time horizons, is not compelling as $k - 1$ is replaced by k^T in the regret bounds. Due to the lower bound proved here, it appears that further progress solving reinforcement learning must come with additional assumptions.

References

[1] N. Abe, A. Biermann, and P. Long. Reinforcement learning with immediate rewards and linear hypotheses, *Algorithmica*, 37(4): 263–293, 2003.

[2] P. Auer. Using confidence bounds for exploitation-exploration tradeoffs, JMLR, 3: 397–422, 2002.

[3] Peter Auer, Nicol Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem, FOCS 322–331, 1995.

[4] P. Auer, N. Cesa-Bianchi, P. Fischer. Finite time analysis of the multi-armed bandit problem, *Machine Learning*, 47: 235–256, 2002.

[5] C. Elkan. The foundations of cost-sensitive learning, IJCAI 2001, 973–978.

[6] Even-dar, E., Mannor, S., and Mansour, Y. Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. JMLR 2006, 7, 1079-1105.

[7] T. Hastie and R. Tibshirani. Classification by pairwise coupling, NIPS 1997.

[8] J. Heckman. Sample Selection Bias as a Specification Error, Econometrica, Vol. 47, No. 1 (Jan., 1979), 153–161.

[9] M. Kearns, Y. Mansour, and A. Y. Ng. Approximate Planning in Large POMDPs via Reusable Trajectories, NIPS 2000.

[10] S. Kulkarni. On bandit problems with side observations and learnability, Allerton 1993, 83–92.

[11] J. Langford and A. Beygelzimer. Sensitive error correcting output codes, COLT 2005.

[12] J. Langford and T. Zhang. The Epoch-Greedy Algorithm for Contextual Multiarmed Bandits, NIPS 2007.

[13] S. Pandey, D. Agarwal, D. Chakrabati, V. Josifovski. Bandits for taxonomies: a model based approach, SDM 2007.

[14] H. Robbins. Some aspects of the sequential design of experiments, *Bulletins of the American Mathematical Society*, 58: 527–535, 1952.

[15] A. Strehl, C. Mesterham, M. Littman, and H. Hirsh, Experience-efficient learning in associative bandit problems, ICML 2006, 889–896.

[16] C. Blake and C. Merz, UCI Repository of machine learning databases. University of California, Irvine.

[17] C. C. Wang, S. Kulkarni, and H. Vincent Poor, Bandit problems with side observations, *IEEE Transactions on Automatic Control*, 50(5), 2005.

[18] I. Witten and E. Frank. Data Mining: Practical machine learning tools with Java implementations, 2000: http://www.cs.waikato.ac.nz/ml/weka/.

[19] M. Woodruff. A one-armed bandit problem with concomitant variates, JASA, 74 (368): 799–806, 1979.

[20] Bianca Zadrozny, Ph.D. Thesis, University of California, San Diego, 2003.

[21] B. Zadrozny, J. Langford, and N. Abe. Cost sensitive learning by cost-proportionate example weighting, ICDM 2003.