History Dependent Domain Adaptation

Allen Lavoie
Rensselaer Polytechnic
lavoia@cs.rpi.edu

Matthew Eric Otey
Google Pittsburgh
otey@google.com

Nathan Ratliff
Google Pittsburgh
ratliffn@google.com

D. Sculley
Google Pittsburgh
dsculley@google.com

Abstract

We study a novel variant of the domain adaptation problem, in which the loss function on test data changes due to dependencies on prior predictions. One important instance of this problem area occurs in settings where it is more costly to make a new error than to repeat a previous error. We propose several methods for learning effectively in this setting, and test them empirically on the real-world tasks of malicious URL classification and adversarial advertisement detection.

1 Introduction

In typical domain adaptation problem settings, the loss function used at testing time may differ in some way from the loss function that is known during training [1]. In this paper, we study a domain adaptation problem with an interesting historical structure in which *tomorrow's loss function depends on the predictions given today*.

A historical dependency in the loss function can occur when we have a classification system that makes predictions day after day on similar input data, and for which classification errors are costly to correct in terms of time or effort. In this case, it is important not only to predict with good accuracy, but also to *avoid making new errors* with respect to previous predictions. The nightmare scenario we wish to avoid is a system that achieves 99% accuracy, but which makes errors on a different 1% of the data each day. After a hundred days the cumulative error rate would be 100%.

The problem of history dependent domain adaptation occurs frequently in practice, but it has not been previously studied formally to our knowledge. An example of this is Google's system for learning to detect adversarial advertisements [5]. This system is responsible for protecting users by detecting and blocking advertisements that may cause the user some form of harm. In this system, automated classification is used as a first line of defense, with significant human effort used to correct errors so that the total system error is driven to zero. Any error from the automated classifier requires human effort to correct, but the correction is effectively permanent. Thus, the key to reducing human cost is for the automated classifier to avoid making new errors over time while still achieving high accuracy.

This paper aims to make the following contributions. We define the problem of history dependent domain adaptation, formalizing the trade-off between performance and hypothesis stability, and present several methods for learning within this setting. Our study is grounded in two real-world classification problems where history dependent domain adaptation is needed: malicious URL detection [4] and adversarial advertisement classification [5]. We present experimental results on data from these domains showing that, over time, history dependent domain adaptation can be used to dramatically reduce the incidence of new prediction errors with little or no loss in classical predictive performance measures such as the area under the ROC curve (AUC).

2 Defining the Problem: When New Errors are Especially Costly

Classical domain adaptation problems deal with situations where the test and training data are drawn from different distributions [2]. Here we are concerned with deployed classification systems that make repeated predictions on a set of entities with a time-varying distribution. Importantly, the loss function for classifying these entities has a historical dependency on the system's prior predictions.

Formally, assume our system uses a prediction function $h_t : \mathbb{R}^n \to \{-1,\ 1\}$, which may differ with each time step t. The model is applied to an entity $e \in E_t$, which is represented by a vector $\mathbf{x}_t \in \mathbb{R}^n$ at time step t. This vector may vary over time. For example, if the entity is a particular URL, its content may change or its metadata may vary over time, causing the representation vector to differ. The set of entities E_t classified by $h_t(\cdot)$ may grow over time. That is, $E_{t-1} \subseteq E_t$.

In this setting, it is clearly possible that the system classifies the same entity differently over time. That is, it is possible that $h_t(e) \neq h_{t-1}(e)$. This could be caused by changes in the representation vector for e, such that $\|\mathbf{x}_t - \mathbf{x}_{t-1}\| > 0$, or it could be caused by changes in the prediction model used by $h(\cdot)$, which may be introduced by training updates on additional or altered training data.

Of course, if we had full label feedback on all predictions at time $t-1$, learning a good model at time t would be easy. In this case, we could simply define and optimize a loss function of the form:

$$\min_{h_t} L(h_t, D_t) + \alpha \sum_{e \in D_{t-1}} I(h_t(e) \neq y_e \wedge h_{t-1}(e) = y_e).$$

Here $L(h, D_t)$ is some standard classifier loss function to be minimized, I is the indicator function, and y_e is the true label of e. It essentially penalizes $h_t(\cdot)$ for making false positive predictions not made by $h_{t-1}(\cdot)$.[1]

However, in real-world situations, it is likely that at time t we do not have full label feedback for all predictions made at time $t-1$. For large-scale deployed systems, collecting ground-truth labels for all examples may be prohibitively expensive, and the labels that are available may take time to collect. Therefore we need to prevent making new errors while being unsure which of the previous classifications were correct.

3 Methods for Reducing New Errors

In this section, we present several methods for learning with history dependent domain adaptation. For this initial study, we use the classical linear support vector machine (SVM) as the baseline method. Recall that a linear SVM may be learned by solving the following optimization problem:

$$\min_{\mathbf{w}} \quad L(\mathbf{w}, D) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

Here $L(\mathbf{w}, D)$ is hinge-loss. This can be solved efficiently for large-scale problems using stochastic gradient descent variants [6]. Each of the methods we propose will vary this basic linear SVM.

Model Averaging As a baseline approach, we can treat history dependent domain adaptation as a temporal smoothing problem, and take a smoothed average of predictions from our different models. At time $t+1$ with historical weight vectors w_i, $h_{t+1}(x) = \langle \frac{1}{t+1} \sum_{i=1}^{t+1} w_i, x \rangle$.

Warm-Start Online Learning with Small Fixed Step Size We can use the prior weight vector $\mathbf{w}_t$ rather than the null vector as the initial hypothesis for stochastic gradient descent. Taking a limited number of steps with fixed step size η will restrict the amount that $\mathbf{w}_{t+1}$ can diverge from $\mathbf{w}_t$ [3].

Adding a Nearness Constraint Similarly, we can add history dependency to our training by adding a hard constraint that $\mathbf{w}_{t+1}$ be close to its predecessor in the weight space. We constrain $\mathbf{w}_{t+1}$ to lie within an L2 ball of radius δ, where $\mathbf{w}_t$ is the center of the ball. This is implemented efficiently via projected gradient descent [7], with the following constraint added to the linear SVM optimization problem:

$$\text{require: } \|\mathbf{w}_{t+1} - \mathbf{w}_t\| \leq \delta.$$

Prediction Regularization We may also enforce nearness not in weight space, as above, but in the prediction space by regularizing the new model's predictions back towards those of the old model.

[1] This greedy heuristic is an important approximation since the full optimization over hypothesis sequences is often intractable and would entail retrospectively modifying past hypotheses.

This ensures that a penalty is paid for diverging from the prior model's predictions, which will be traded off against the benefit given for reducing classification loss or model complexity by the parameter α. We experimentally analyze two such techniques. First, we add a quadratic penalty to the loss proportional to $\sum_{\mathbf{x} \in D_{t+1}} (\langle \mathbf{w}_{t+1}, \mathbf{x} \rangle - \langle \mathbf{w}_t, \mathbf{x} \rangle)^2$.

The second method treats the previous model predictions as additional (weighted) examples, adding a new collection of hinge loss terms (one for each $\mathbf{x}$) proportional to $\sum_{\mathbf{x}} \max\{0,\ 1 - h_t(x) w_{t+1}^T x\}$.

4 Experiments on Real-World Problems

4.1 Experimental Set Up

Our goal in these experiments is to simulate the performance of each method from Section 3 in the setting of a large, deployed system over time. We experiment on two large real-world data sets that have the temporal qualities described in Section 2.

Adversarial Advertisement Classification Data The `adversarial-ads` data is a large sample from 170 days of Google's proprietary adversarial advertisement classification data from the system described in [5]. For the evaluation purposes in this simulation, we define time intervals as periods of one week, and at each period train on the previous 100 days of data. (this does not reflect the actual production process). Entities are individual advertisements, and are represented by a sparse, high dimensional feature set described in detail in [5]. The classification task is to label each advertisement as *adversarial* (positive) or *non-adversarial* (negative).

At each time step t, a new model is trained on the labeled data from the prior time steps and evaluated on data from the current time step. Entities are randomly selected as *training* or *test* entities, so that a training entity from a prior time step is never used as a test entity in the current time step. For experimental purposes, all of the data used is labeled.

Malicious URL Identification Data To complement our experiments on private data, we also run tests on a public data set that is qualitatively similar. The `malicious-url` data set contains roughly 2.4 million examples over a 120 day date range [4]. Here time intervals are 10 days, and during each interval we train on the previous 20 days of data. The task is to classify URLs as *malicious* (positive) or *non-malicious* (negative), where malicious URLs may contain malware or similarly harmful data [4]. This data set is similar to the `adversarial-ads` in that it is relatively large and uses a sparse, high dimensional feature space.

4.2 Evaluation Metrics

Our goal is to achieve good classification performance and reduce the incidence of new errors over time. Thus we report two separate performance metrics that should be considered together when evaluating success. The first metric is the traditional AUC metric for evaluating classifier performance. For both data sets, we report the percent difference from a baseline control of naively retraining each model from scratch every time period. The second metric we report is cumulative unique false positive rate (CUFP). This is defined at time t as total number of unique entities that have ever been wrongly marked as positives by any hypothesis $h_1(\cdot), ..., h_t(\cdot)$, divided by the total number of unique entities. As with AUC, we report results relative to the control baseline.

4.3 Results

Figure 4.2 summarizes the results of our experiments. From left to right, top to bottom, the subplots depict the AUC and CUFP rates for the `adversarial-ads` problem, and the AUC and CUFP rates of the `malicious-urls` problem, respectively. [2]

Of the methods we compared, *constrained*, *average*, and *warm start* performed the best with *constrained* and *average* generally edging out *warm start*. *hinge* and *squared* showed a slight benefit on the URL problem, but little, if any, overall improvement on the Adversarial Advertisers problem. On both problems, none of the methods performed significantly different than the control in terms of AUC.

[2] Error bars rendered the plots unreadable, but on average, they spanned approximately 2% and 12% for the `adversarial-ads` AUC and CUFP plots from upper bound to lower bound, respectively, and spanned .1% and 11% for the `malicious-url` AUC and CUFP plots, respectively. In particular, no method performed significantly different than the control with respect to AUC on the `adversarial-ads` problem.

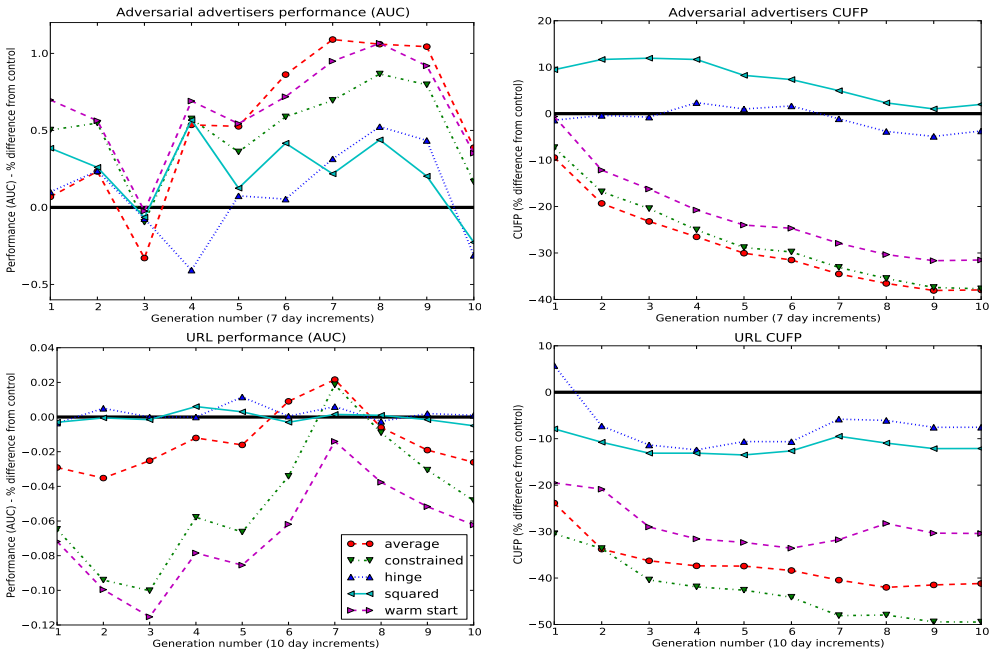

Figure 1: **Experimental Results**. Top: `adversarial-ads`, bottom: `malicious-urls`, left: AUC performance, right: cumulative false positives (CUFP). Each performance metric is represented relative to the *control* model, shown in each plot (relative to itself) as the zero line.

5 Conclusions

We proposed a number of techniques to mitigate new errors made between successive model retrainings in real-world machine learning systems. Interestingly, some of the simplest and most intuitively appealing methods performed the best in our experiments, reducing a measure of cumulative false positives without adversely affecting the overall system performance. We are currently collecting a more extensive suite of experimental results, and developing theoretical results to better characterize the behavior of these algorithms.

References

[1] J. A. D. Bagnell. Robust supervised learning. In *Proceedings of AAAI*. American Association for Artifical Intelligence, June 2005.

[2] H. Daumé III and D. Marcu. Domain adaptation for statistical classifiers. *Journal of Artificial Intelligence Research*, 26(1):101–126, 2006.

[3] M. Herbster, M. K. Warmuth, and P. Bartlett. Tracking the best linear predictor. *Journal of Machine Learning Research*, 1:281–309, 2001.

[4] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Identifying suspicious urls: an application of large-scale online learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009.

[5] D. Sculley, M. Otey, M. Pohl, B. Spitznagel, J. Hainsworth, and Y. Zhou. Detecting adversarial advertisements in the wild. In *Proceedings of the Seventeenth Conference on Knowledge Discovery and Data Mining*, 2011.

[6] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 116. ACM, 2004.

[7] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *In Twentieth International Conference on Machine Learning*, 2003.