Multi-Domain Learning by Confidence-Weighted Parameter Combination[*]

Mark Dredze MDREDZE@CS.JHU.EDU
Human Language Technology Center of Excellence
Johns Hopkins University
Baltimore, MD 21211
Alex Kulesza KULESZA@CIS.UPENN.EDU
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
Koby Crammer KOBY@EE.TECHNION.AC.IL
Department of Electrical Engineering
The Technion
Haifa 32000, Israel

Abstract

State-of-the-art statistical NLP systems for a variety of tasks learn from labeled training data that is often domain specific. However, there may be multiple domains or sources of interest on which the system must perform. For example, a spam filtering system must give high quality predictions for many users, each of whom receives emails from different sources and may make slightly different decisions about what is or is not spam. Rather than learning separate models for each domain, we explore systems that learn across multiple domains. We develop a new multi-domain online learning framework based on parameter combination from multiple classifiers. Our algorithms draw from multi-task learning and domain adaptation to adapt multiple source domain classifiers to a new target domain, learn across multiple similar domains, and learn across a large number of disparate domains. We evaluate our algorithms on two popular NLP domain adaptation tasks: sentiment classification and spam filtering.

1. Introduction

Machine learning algorithms typically learn a single task using training data that are representative of data observed at test time. Formally, training and test data are usually assumed to be drawn from the same distribution and labeled with the same decision function. In this setting, the learning algorithm can infer properties of the distribution and labels from observed data.

Reality is often more complex. Learning systems deployed on real world problems encounter multiple sources of data for which there may be different feature distributions and different decision functions. A computational advertising system, for example, may rank ads for queries that originate from many different countries, are in many different languages, and cover a variety of product domains. A system trained on all queries together, agnostic with respect to such properties, may benefit from having a large quantity of data on which to learn. However, it may also be that data

[*]. Preliminary versions of the work contained in this article appeared in the proceedings of the conference on *Empirical Methods in Natural Language Processing* (Dredze and Crammer, 2008). The original publication is available at `http://www.springerlink.com/content/a780497676803861/`. Please cite this paper as "Mark Dredze, Alex Kulesza, Koby Crammer. Multi-domain learning by confidence-weighted parameter combination. *Machine Learning*, 2009."

sources have conflicting properties that reduce the performance of a single model trained in this fashion. In this case, it would be preferable to train separate systems. In the end, both approaches (single and separate) are inadequate; data sources typically share some common characteristics and behaviors and differ in others. A single system obscures differences and separate systems ignore similarities. A hybrid approach is in order.

The challenge of multiple domains is particularly relevant to natural language processing (NLP) since statistical language systems require examples carefully labeled by trained annotators. The Penn Treebank (Marcus et al., 1993), one of the most widely used corpora for NLP applications, contains articles sampled from the Wall Street Journal. Since many systems are trained exclusively on this corpus, they may perform poorly on other domains or genres, such as biomedical texts (Lease and Charniak, 2005; McClosky and Charniak, 2008). The problem becomes worse for informal text, such as blogs and emails. New vocabulary items and writing styles confuse these systems.

Consider a multi-user spam filter, which must give high quality predictions for new users (without new user data), learn on multiple users simultaneously, and scale to thousands of accounts. While a single classifier trained on all users would generalize across current users and extend to new ones, it would fail to learn user-specific preferences. Alternatively, separate classifiers would capture user-specific behaviors but not generalize across users.

Responding to these challenges has been work in transfer learning, including domain adaptation, or genre shift, and multi-task learning. In transfer learning, knowledge from a source task or domain is transferred to a different but related task or domain. This is especially practical when there is plentiful data for one problem but limited or no labeled data for the problem of interest. For example, a face recognition system trained on a large number of photographs taken indoors could be transfered to outdoor photographs where few examples are available. In computational biology, mRNA splicing depends on the organism, and information learned from one organism could be adapted to a related organism (Schweikert et al., 2008). In general, the transfer from one domain to another may rely on small amounts of annotated data and/or large samples of unlabeled data, both from the target domain. However, common approaches typically focus on a one time transfer involving significant computational resources.

We focus on *multi-domain learning*, which combines properties of domain adaptation and multi-task learning in a flexible way. Like domain adaptation, we allow that different domains may be drawn from different underlying distributions (Blitzer et al., 2007b; Jiang and Zhai, 2007a). For example, one user may receive emails about engineering while another about art, differing in their distribution over features. Like multi-task learning, we also allow that the prediction function itself may change from domain to domain (Evgeniou and Pontil, 2004; Dekel et al., 2006). For example, one user may enjoy some emails that another user considers spam. Multi-domain learning unifies these two approaches by learning across both distributions and behaviors.

In this work, our goal is to model prediction tasks across a variety of data sources, using combined data to learn common behaviors while discriminating domain specific behaviors and learning them separately. We focus on NLP multi-domain learning problems. We also consider the standard adaptation setting, where there is no target domain data whatsoever. We extend our multi-domain learning algorithm to handle a large number of disparate domains, where common behaviors may not be common to every data source. Our central method is to intelligently *combine* classifiers trained in different ways to take advantage of their various strengths. For multi-domain learning, we combine domain-specific classifiers with a shared classifier learned across all domains. In the adaptation setting, we combine source domain classifiers for use in a new target domain. For large numbers of disparate domains, we learn which domain-specific and shared classifiers to combine.

We base our algorithms on the recently introduced Confidence-Weighted (CW) learning (Dredze et al., 2008; Crammer et al., 2008), an online learning algorithm for linear classifiers that incorporates confidence about each parameter into the update. Parameter confidence enables a more informed approach to combination, and we improve over the state of the art performance of CW algorithms by learning across data from multiple domains. This paper builds on previously published work

(Dredze and Crammer, 2008) by considering several new alternative formulations for multi-domain learning algorithms. We introduce two new objectives and show that previous work can be viewed as an approximation of one of these objectives. Additionally, we provide further evaluations and analysis of previously published methods.

We proceed with a formalization of the online multi-domain learning setting. After a review of CW classification, we propose a new approach to multi-domain learning through parameter combination. We first use this approach in the standard domain adaptation setting on several datasets. We then derive several generalizations of our approach to the multi-domain learning setting. This approach is also extended to handle learning from many disparate domains. Finally, we propose a method by which our approach can scale to many domains, where learning each domain individually becomes prohibitive. We conclude with a review of several related problems: domain adaptation, transfer learning and multi-task learning, focusing on NLP research.

2. Multi-Domain Learning

With efficiency in mind, we focus our efforts on binary online multi-domain learning algorithms, which process a single instance from a single domain at a time. Online algorithms are widespread in NLP because they are simple to implement and scale to very large datasets. Our algorithms will processes multiple streams of data from many different sources, transferring knowledge between domains with each update. Information transfer is achieved through a shared model that applies to all domains. Domain specific models are combined with the shared models for prediction and learning. In this section we use θ to refer generically to the entire collection of shared and domain specific parameters.

Online learning proceeds in rounds. On round t, the learner receives an instance $\boldsymbol{x}_t \in \mathbb{R}^N$ from domain d_t, drawn according to the fixed domain specific distribution D_{d_t}. The learner makes a prediction $\hat{y}_t \in \{-1, +1\}$ based upon its current model parameters θ_{t-1}, then receives the true label $y_t \in \{-1, +1\}$, determined by the domain specific labeling function $f_{d_t}(\boldsymbol{x}_t)$. Finally, the learner updates its prediction parameters to θ_t based on this new information.

To lighten notation, we will generally omit the subscripts t when describing updates that take place during a single round. We use tildes to indicate the parameter values just before the start of the round, and unadorned symbols to indicate the updated values. For example, on round t, $\tilde{\theta} \equiv \theta_{t-1}$ and $\theta \equiv \theta_t$.

3. Confidence-Weighted Linear Classifiers

Confidence-weighted (CW) linear classification (Dredze et al., 2008; Crammer et al., 2008) is an online learning algorithm for linear classifiers. CW maintains a probabilistic measure of parameter confidence, which may be useful in combining parameters from different domain distributions. We summarize CW learning to familiarize the reader.

Parameter confidence is formalized by a Gaussian distribution over weight vectors with mean $\boldsymbol{\mu} \in \mathbb{R}^N$ and diagonal covariance $\Sigma \in \mathbb{R}^{N \times N}$. The values μ_i and $\Sigma_{i,i}$ represent knowledge of and confidence in the parameter for feature i. The smaller $\Sigma_{i,i}$, the more confidence we have in the mean parameter value μ_i. (Using a full matrix Σ models second order feature interactions, which can be useful in many settings. However, in NLP tasks, where N is typically very large, it is impractical to maintain N^2 parameters. Therefore, we follow the approach of Dredze et al. (2008) and consider diagonal covariance matrices, which only require N additional parameters as compared with a standard linear model.)

For binary classification, a CW model predicts the highest probability label,

$$\arg\max_{y \in \{\pm 1\}} \Pr_{\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)} \left[y(\boldsymbol{w} \cdot \boldsymbol{x}) \geq 0 \right] \ .$$

$\boldsymbol{x}_t$	Instance on round t
$\hat{y}_t$	Prediction on round t
y_t	Label on round t
d_t	Domain on round t
$\boldsymbol{\mu}_t$	The mean of the distribution on round t
Σ_t	The covariance matrix of the distribution on round t
μ_i^z, Σ_i^z	The $\boldsymbol{\mu}$ and Σ value for the ith parameter of classifier z
$\boldsymbol{w}$	The weight vector drawn from the model distribution
$\tilde{\boldsymbol{\mu}}, \tilde{\Sigma}$	Old model parameters before update
$\boldsymbol{\mu}, \Sigma$	New model parameters after update
$\boldsymbol{\mu}^c, \Sigma^c$	Combined model parameters
$\boldsymbol{\mu}^z, \Sigma^z$	Individual model parameters
b_i^z	The weight given to the ith parameter of classifier z in the combination
c_i^z	The normalized weight b_i^z such that $c_i^z = b_i^z / \sum_{z \in Z} b_i^z$
ϕ	The free parameter for CW, defined as $\phi = \Phi^{-1}(\eta)$
λ_1, λ_2	The weights for shared and domain specific parameters ($\lambda_2 = 2 - 2\lambda_1$)
k	The number of underlying shared parameters

Table 1: Notation and definitions.

The Gaussian distribution over parameter vectors $\boldsymbol{w}$ induces a univariate Gaussian distribution over the score $S = \boldsymbol{w} \cdot \boldsymbol{x}$ parameterized by $\boldsymbol{\mu}$, Σ and the instance $\boldsymbol{x}$: $S \sim \mathcal{N}\left(\mu, \sigma^2\right)$, with mean $\mu = \boldsymbol{\mu} \cdot \boldsymbol{x}$ and variance $\sigma^2 = \boldsymbol{x}^\top \Sigma \boldsymbol{x}$. In practice, rather than sample a weight vector from the distribution, we first compute the expected weight vector, which for Gaussian distributions is simply the mean parameters $\boldsymbol{\mu}$, and use it to make predictions with the familiar rule: $\text{sign}(\boldsymbol{\mu} \cdot \boldsymbol{x})$. (A discussion of this rule's relationship to other prediction rules can be found in (Crammer et al., 2008).)

The CW learning algorithm is inspired by the Passive Aggressive (PA) update (Crammer et al., 2006), which ensures a positive margin on the current example while minimizing parameter change. CW replaces the Euclidean distance used in the PA update with the Kullback-Leibler (KL) divergence over Gaussian distributions. It also replaces the minimal margin constraint with a minimal probability constraint: with some given probability $\eta \in (0.5, 1]$ a drawn classifier will assign the correct label. This strategy yields the following objective to be solved on each round of learning:

$$\begin{aligned} (\boldsymbol{\mu}, \Sigma) = \arg\min\ & \mathbb{D}_{\mathrm{KL}}\left(\mathcal{N}(\boldsymbol{\mu}, \Sigma) \parallel \mathcal{N}\left(\tilde{\boldsymbol{\mu}}, \tilde{\Sigma}\right)\right) \\ \text{s.t. } & \Pr\left[y\left(\boldsymbol{w} \cdot \boldsymbol{x}\right) \geq 0\right] \geq \eta \ , \end{aligned}$$

where $\left(\tilde{\boldsymbol{\mu}}, \tilde{\Sigma}\right)$ are the parameters from the previous round and $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$. The constraint ensures that the resulting parameters will correctly classify $\boldsymbol{x}$ with probability at least η. For convenience we write $\phi = \Phi^{-1}(\eta)$, where Φ is the cumulative function of the normal distribution. The constraint can be written as:

$$y(\boldsymbol{\mu} \cdot \boldsymbol{x}) \geq \phi\sqrt{\boldsymbol{x}^\top \Sigma \boldsymbol{x}} \ .$$

While the square root appears to make the constraint non-convex, Crammer et al. (2008) show that this objective is in fact convex in a modified parameter space. However, replacing the standard deviation term with the variance (i.e., removing the square root) has been shown to be a simple and effective approximation (Dredze et al., 2008) and we use that constraint here as well.

The resulting optimization problem has an additive closed form solution:

$$\begin{aligned} \boldsymbol{\mu} &= \tilde{\boldsymbol{\mu}} + \alpha y \tilde{\Sigma} \boldsymbol{x} \\ \Sigma^{-1} &= \tilde{\Sigma}^{-1} + 2\alpha\phi \boldsymbol{x}\boldsymbol{x}^\top \ . \end{aligned}$$

We denote the current signed margin as $M = y(\tilde{\boldsymbol{\mu}} \cdot \boldsymbol{x})$ and the variance of the margin as $V = \boldsymbol{x}^\top \tilde{\Sigma} \boldsymbol{x}$. We can solve for α using the quadratic equation, yielding the closed form

$$\alpha = \frac{-(1+2\phi M) + \sqrt{(1+2\phi M)^2 - 8\phi\,(M - \phi V)}}{4\phi V} \ .$$

α is only positive when the constraint is violated, so updates are only computed when $\alpha > 0$.

Each update changes the feature weights $\boldsymbol{\mu}$ with an additive update similar to passive-aggressive. The noticeable difference is the inclusion of Σ, which weighs the update to $\boldsymbol{\mu}$. When a parameter has small Σ (high confidence) the update to this parameter is smaller in favor of other lower confidence parameters. The name confidence weighted comes from this behavior. Since $\alpha >= 0$ for all updates, Σ is non-increasing. Every time the algorithm updates a parameter, the confidence in that parameter increases.

Our choice of CW classifiers in this work is due to their parameter confidence values, which provide an informed way to combine parameters from different classifiers. Additionally, since we require per-parameter confidence estimates, other confidence based classifiers that only provide confidence in the resulting predictions are not suitable for this setting.

It should be noted that there are many techniques for combining the output of multiple classifiers for ensemble learning or mixture of experts. Kittler et al. (1998) provide a theoretical framework for combining classifiers. Some empirical work has considered adding versus multiplying classifier output (Tax et al., 2000), using local accuracy estimates for combination (Woods et al., 1997), and applications to NLP tasks (Florian et al., 2003). However, these papers consider combining classifier output for prediction. In contrast, we consider *parameter* combination for both prediction and learning.

4. Multi-Classifier Parameter Combination

The basis of our approach to multi-domain learning is to combine the parameters of CW classifiers from separate domains while respecting their confidence estimates. A combination method takes a collection Z of CW classifiers, each parameterized by its own mean and variance parameters $\{(\boldsymbol{\mu}^z, \Sigma^z)\}_{z \in Z}$ and produces a single combined classifier $(\boldsymbol{\mu}^c, \Sigma^c)$. A simple technique would be to average the mean parameters of classifiers into a new weight vector, which is a common method for combining linear classifiers. However, this ignores the difference in feature distributions and confidence learned by each model. Consider for example a case in which the weight associated with some word in a source classifier has a value near 0. This could suggest that the word is rare, but it might also be that it is simply neutral for prediction (like the work "the"). The information captured by Σ allows us to distinguish between these two cases: high variance indicates a lack of confidence in the value of the weight vectors because of a small number of examples (first case), and low variance indicates that the value of the weight is based on significant evidence. We favor combinations sensitive to this distinction.

Since CW classifiers are Gaussian distributions, we formalize classifier parameter combination as finding a new distribution that minimizes the weighted divergence to a set of given distributions:

$$(\boldsymbol{\mu}^c, \Sigma^c) = \arg\min_{\boldsymbol{\mu}^c, \Sigma^c} \sum_{z \in Z} \mathbb{D}(\mathcal{N}(\boldsymbol{\mu}^c, \Sigma^c) || \mathcal{N}(\boldsymbol{\mu}^z, \Sigma^z) \ ; \ \boldsymbol{b}^z) \ ,$$

where the classifier specific importance-weights $\boldsymbol{b}^z \in \mathbb{R}_+^N$ can used to inject additional information into the combination, as discussed later. Since we use diagonal Σ, the distance between each individual component is independent of the others. Thus the objective breaks down into a sum of distances between each set of parameters:

$$\mathbb{D}(\mathcal{N}(\boldsymbol{\mu}^c, \Sigma^c) || \mathcal{N}(\boldsymbol{\mu}^z, \Sigma^z) \ ; \ \boldsymbol{b}^z) = \sum_{i=1}^{N} b_i^z \mathbb{D}(\mathcal{N}(\mu_i^c, \Sigma_i^c) || \mathcal{N}(\mu_i^z, \Sigma_i^z)) \ ,$$

where μ_i^z is the $\boldsymbol{\mu}$ value for the ith parameter of classifier z and Σ_i^z is the Σ value for the ith parameter of classifier z (the ith diagonal value).

We consider two choices for $\mathbb{D}$. First, $\mathbb{D}$ can be a generalized squared Euclidean (ℓ_2) distance between the parameters:

$$\mathbb{D}(\mathcal{N}(\boldsymbol{\mu}^c, \Sigma^c)||\mathcal{N}(\boldsymbol{\mu}^z, \Sigma^z); \boldsymbol{b}^z) = \frac{1}{2}\sum_{i=1}^{N} b_i^z \left[(\mu_i^c - \mu_i^z)^2 + (\Sigma_i^c - \Sigma_i^z)^2\right] \ .$$

We solve for $(\boldsymbol{\mu}^c, \Sigma^c)$ by setting the derivative of the objective to 0. For μ_i^c, we get

$$\sum_{z \in Z} b_i^z(\mu_i^c - \mu_i^z) = 0 \quad \Rightarrow \quad \mu_i^c = \sum_{z \in Z} c_i^z \mu_i^z$$

where we let $c_i^z = b_i^z / \sum_{z \in Z} b_i^z$ for convenience. Similarly,

$$\Sigma_i^c = \sum_{z \in Z} c_i^z \Sigma_i^z \ .$$

Note that this is a (weighted) average of parameters; however, in this more general form we can add confidence information through the choice of $\boldsymbol{b}^z$, as discussed later.

The second option is to set $\mathbb{D}$ to be the KL-divergence, which, since Σ is diagonal, can be decomposed into the sum of KL-divergences between univariate Gaussians:

$$\begin{aligned}
&\mathbb{D}(\mathcal{N}(\boldsymbol{\mu}^c, \Sigma^c)||\mathcal{N}(\boldsymbol{\mu}^z, \Sigma^z); \boldsymbol{b}^z) \\
&= \sum_{i=1}^{N} b_i^z \mathbb{D}_{\mathrm{KL}}(\mathcal{N}(\mu_i^c, \Sigma_i^c)||\mathcal{N}(\mu_i^z, \Sigma_i^z)) \\
&= \sum_{i=1}^{N} \frac{b_i^z}{2}(\log(\Sigma_i^z/\Sigma_i^c) + \Sigma_i^c/\Sigma_i^z + (\mu_i^c - \mu_i^z)^2/\Sigma_i^z - 1)
\end{aligned}$$

Again, we solve by setting the derivative to 0. For μ_i^c this yields

$$\sum_{z \in Z} \frac{b_i^z}{\Sigma_i^z}(\mu_i^c - \mu_i^z) = 0 \quad \Rightarrow \quad \mu_i^c = \left(\sum_{z \in Z} \frac{c_i^z}{\Sigma_i^z}\right)^{-1} \sum_{z \in Z} \frac{c_i^z}{\Sigma_i^z}\mu_i^z \ .$$

In words, the combined mean parameters are a weighted average of the individual mean parameters, with weights proportional to $\frac{c_i^z}{\Sigma_i^z}$. For Σ_i^c we get

$$\sum_{z \in Z} b_i^z(-1/\Sigma_i^c + 1/\Sigma_i^z) = 0 \quad \Rightarrow \quad \Sigma_i^c = \left(\sum_{z \in Z} c_i^z/\Sigma_i^z\right)^{-1} \ ,$$

which is a weighted harmonic mean of the individual confidence parameters.

While each parameter is naturally weighed by its variance in the KL version, we can also explicitly encourage this sort of behavior (which favors high confidence values) by setting $b_i^z = a - \Sigma_i^z \geq 0$, where a is the initialization value for Σ_i^z. This will prove helpful when the KL is difficult to compute, since we can achieve similar intuition using the simpler ℓ_2 combination. We call this type of $\boldsymbol{b}$-weighting "variance," as opposed to uniform ($b_i^z = 1$). We therefore have two combination methods (ℓ_2 and KL) and two weighting methods (uniform and variance).

5. Datasets

For evaluation we selected two domain adaptation datasets: spam (Jiang and Zhai, 2007a) and sentiment (Blitzer et al., 2007b).

		Apparel	Books	DVD	Kitchen	Elect.	Music	Video
Train	All Src	12.32	16.85	13.65	13.65	15.00	18.20	17.00
	Target	12.02	18.95	17.40	14.40	14.93	18.30	19.27
	Best Src	14.12	22.95	17.30	15.52	15.52	20.75	19.43
	Average Src	21.15	25.76	21.89	22.88	23.84	24.19	25.78
ℓ_2	Uniform	14.03	19.58	15.53	16.68	18.75	18.38	17.13
	Variance	**13.18**	**18.63**	**13.73**	15.10	17.37	**17.83**	**16.25**
KL	Uniform	13.50	19.53	14.48	14.78	17.45	18.10	16.33
	Variance	13.48	19.05	14.15	**14.02**	**16.82**	18.22	16.42

Table 2: Test error for multi-source adaptation on sentiment data over 10-fold cross validation. Combining classifiers improves over selecting a single classifier a priori (*Average Source*).

Spam The ECML/PKDD spam data contains a single task with multiple users. The goal is to classify an email as either spam or ham (not-spam). Each of the users may have slightly different preferences and features. The task contains two sets of data, one with three users (task A) and one with 15 (task B). Each user has a collection of spam and ham email messages, where an email message is represented as a bag-of-words. We used 700 training messages for each user in set A and 100 training messages for each user in set B. For both sets we selected 300 test emails for each user.

Sentiment The goal of the sentiment task is to classify a product review as being either positive or negative in its evaluation about the product. Product reviews are drawn from user product feedback on Amazon.com. Reviews are selected from seven product types: apparel, books, dvds, electronics, kitchen appliances, music and videos. Each review is represented by features of unigram and bigrams counts.

In addition to having each of the seven product types as a separate domain, we created different datasets by mixing the different domains and modifying the decision boundary. The decision boundary—the boundary between positive and negative reviews—was modified by using the ordinal rating of each instance (1-5 stars) and excluding boundary instances, i.e., positive reviews are above 2 starts while negative are below 2 stars, etc. We created four sets of data through this mixing process.

- **All** - The 7 Amazon domains, one for each product category.
- **Books** - 3 domains all drawn from the books product reviews, with each using a different decision boundary for positive and negative examples (2, 3 and 4 stars.)
- **DVDs** - This is the same as the *Books* dataset (different decision boundaries) but DVD reviews are used instead.
- **Books+DVDs** - A combination of the data used in the *Books* and *DVDs* data above. This gives six domains with varying decision boundaries.

The *All* dataset captures the typical domain adaptation scenario, where each domain has the same decision function but different features. *Books* and *DVDs* have the opposite problem: the same features but different classification boundaries. *Books+DVDs* combines both issues into a single task. For every experiment with the sentiment data we use 1500 training instances and 100 test instances per domain.

6. Multi-Domain Adaptation

We begin by examining the typical domain adaptation scenario, but from an online perspective since learning systems often must adapt to new users or domains quickly and with no training data. For

		User 0	User 1	User 2
Train	All Src	3.85	3.57	3.30
	Target	1.80	3.17	2.40
	Best Source	4.80	4.28	3.77
	Average Source	8.26	6.91	5.75
ℓ_2	Uniform	5.25	4.53	4.75
	Variance	4.63	**3.80**	**4.60**
KL	Uniform	4.53	4.23	4.93
	Variance	**4.32**	3.83	4.67

Table 3: Test error for multi-source adaptation on spam data over 10-fold cross validation. Combining classifiers improves over selecting a single classifier a priori (*Average Source*).

example, a spam filter with separate classifiers trained on each user must also classify mail for a new user. Since other users' training data may have been deleted or kept private, the existing classifiers must be combined for the new user.

We combine the existing user-specific classifiers into a single new classifier for a new user. Since nothing is known about the new user (in particular, their decision function), every source classifier could be relevant. However, feature similarity—possibly measured using unlabeled data—can be used to weigh source domains. Specifically, we combine the parameters of each classifier according to their confidence using the combination methods described above.

We evaluated the four combination strategies—ℓ_2 vs. KL, uniform vs. variance—on spam and sentiment data. For each evaluation, a single domain was held out for testing while separate classifiers were trained on each source domain, i.e., no target training. Source classifiers were then combined and the combined classifier evaluated on the test data (400 instances) of the target domain. This evaluates the ability of a classifier to adapt from several observed domains to a new unobserved domain.

A separate classifier was trained for 5 online iterations over one of the domains in the training data. (5 iterations seemed to ensure convergence.) The resulting classifiers were then combined and tested on the target domain. We repeated each experiment using 10-fold cross validation. Additionally, the CW parameter ϕ was tuned by selecting a single randomized split of the data for each each experiment.

We include several baselines:

- **Target-** Train on the held out target domain. This indicates the possible performance if we obtained labeled data for the unobserved target domain.
- **All Source-** Train a single classifier on all of the source domains. This is a useful strategy if we could maintain all observed source data and then retrain when informed of a new domain.
- **Best Source-** Select the closest known domain to the target domain by selecting the best performing classifier from classifiers trained individually on each domain. This is the best possibility with omniscience, i.e., knowing what the performance of each available classifier will be on the new data.
- **Average Source-** Take the average performance of each of the existing source domain classifiers applied to the target domain. This is the expected real world performance of randomly selecting a source classifier for the new target domain.

While at least one source classifier achieved high performance on the target domain (*Best Source*), the correct source classifier cannot be selected without target data and selecting a random source classifier yields high error. In contrast, a combined classifier almost always improved over the best source domain classifier (tables 2 and 3). We observe that some of our results improve over the best training scenario, i.e., combinations can be better than performance using target domain data. This is likely due to the increase in training data available by using multiple domains. We note with interest that our combinations sometimes improve over *All Source* (for Music and Video). This may be due to a regularization effect of training separate models.

The ℓ_2 methods performed best, yielding the best classifier on 7 out of 10 combinations. Clearly, classifier parameter combination is an effective way of obtaining good classifiers without prior knowledge of the target domain.

7. Learning Across Domains

In addition to adapting to new domains, multi-domain systems should learn common behaviors across domains. Naively, we can assume that the domains are either sufficiently similar to warrant one classifier or different enough for separate classifiers. The reality is often more complex. Instead, we maintain shared and domain-specific parameters and combine them for learning and prediction. So far we have seen the value of parameter confidence combinations for prediction. In this section we investigate their usefulness for learning.

Multi-task learning aims to learn common behaviors across related problems, a similar goal to multi-domain learning. The primary difference is the nature of the domains/tasks. In our setting, while the prediction function for each domain may differ, we seek to perform the same task (eg. spam filtering) on each domain, i.e., the possible outputs for each domain are the same (spam/ham.) In contrast, multi-task settings have several different tasks (different outputs) over the same inputs (features.) The commonality between these two settings is that the different prediction functions are related and we seek to adapt a multi-task approach to our multi-domain setting by using our classifier combination techniques.

We seek to learn domain specific parameters guided by shared parameters. Dekel et al. (2006) followed this approach for an online multi-task algorithm, although they did not have shared parameters and assumed that a training round comprised an example from each task. Evgeniou and Pontil (2004) achieved a similar goal by using shared parameters for multi-task regularization. Specifically, they assumed that the weight vector for problem d could be represented as $w^c = w^d + w^s$, where w^d are task specific parameters and w^s are shared across all tasks. In this framework, all tasks are close to some underlying mean w^s and each one deviates from this mean by w^d. Their SVM style multi-task objective minimizes the loss of w^c and the norm of w^d and w^s, with a tradeoff parameter allowing for domain deviance from the mean. The simple domain adaptation algorithm of feature splitting used by Daumé (2007) is a special case of this model where the norms are equally weighted.

In the case of SVM learning, the appropriate combination is to take the sum of the parameters. Since the resulting prediction and update are independent of the scaling of the weights, an average is unnecessary. However, the combination methods we considered above are more appropriate for the CW setting. We can define a similar objective to Evgeniou and Pontil (2004) by replacing the summation of parameters with a combination method using a distance function $\mathbb{D}$ and recast the SVM objective (minimize the norm of the parameters) in the CW framework.

Unlike Evgeniou and Pontil (2004), we work in the online setting, where examples arrive in rounds. We wish to derive an online update for parameters $(\boldsymbol{\mu}^z, \Sigma^z)$ in terms of the current example $(\boldsymbol{x}, y)$ and the old parameter values. We want to guarantee that after each iteration the combined distribution yields a correct prediction for $\boldsymbol{x}$ with high probability, while also making the smallest possible change from the existing parameters $(\tilde{\boldsymbol{\mu}}^z, \tilde{\Sigma}^z)$. We consider two different ways of formalizing the latter objective.

First, we can directly minimize the sum of KL-divergences across the base classifiers; in other words, given current parameters $\{(\tilde{\boldsymbol{\mu}}^z, \tilde{\Sigma}^z)\}_{z\in Z}$, minimize

$$\sum_{z\in Z} \mathbb{D}_{\mathrm{KL}}(\mathcal{N}(\boldsymbol{\mu}^z, \Sigma^z)||\mathcal{N}(\tilde{\boldsymbol{\mu}}^z, \tilde{\Sigma}^z))$$

subject to the prediction constraint. This can be thought of as distribute then update, since the combination is first distributed to the individual parameters and then an update is computed. Alternatively, we can minimize the KL-divergence of the combination distribution itself, namely

$$\mathbb{D}_{\mathrm{KL}}(\mathcal{N}(\boldsymbol{\mu}^c, \Sigma^c)||\mathcal{N}(\tilde{\boldsymbol{\mu}}^c, \tilde{\Sigma}^c)) \ .$$

This can be thought of as update then distribute, since the update is compute using the combined distribution, and then distributed to each individual parameter. Since the combination distribution is derived from other parameters, we then need to decide how to change those underlying parameters to achieve the resulting desired combination distribution. We can do this in some cases by making an average update, or more carefully by minimizing the KL-divergences of the base classifiers subject to the combination result. We discuss these options more carefully in the context of specific combination rules.

We call this framework Multi-Domain Regularization (MDR). The algorithm maintains a set of shared parameters and a set of domain specific parameters for each domain. The domain specific parameters are used for each individual domain and the shared parameters are used on all domains. On each round, the algorithm receives instance $\boldsymbol{x}$ and domain d for which it creates a combined classifier $(\boldsymbol{\mu}^c, \Sigma^c)$ using the shared and domain specific parameters. A prediction is issued using the standard linear classifier prediction rule $\mathrm{sign}(\boldsymbol{\mu}^c \cdot \boldsymbol{x})$. The shared and domain specific parameters are then updated using one of the objectives above, which are further developed in the next section.

For each algorithm we introduce in the next section we consider two evaluation settings. First, we evaluate each algorithm using a single pass over a stream of instances from multiple domains, simulating a real world setting. In this setting algorithms that adapt quickly to multiple domains perform better since they make the most out of seeing each example once. We also consider the standard batch setting, where online algorithms are run multiple times over the data. We use 10 online training iterations. In this setting, baseline methods will improve since they can examine every example multiple times, reducing the need for learning extra information across domains. As before, the CW parameter ϕ is optimized on a single randomized run for each dataset. Each experiment is run over 10 random divisions of the data into train and test splits. We report results as error on the held out test data.

In addition to evaluating our multi-domain learning methods, we evaluate the performance of a single classifier trained on all domains (*Single*) and a separate classifier trained on each domain (*Separate*). We also include results for *Feature Splitting* based on the work of Daumé (2007). We note that in the online setting, for equally weighted parameter sets the method of Evgeniou and Pontil (2004) reduces to Feature Splitting. Therefore, we can compare our algorithms to this method to determine the improvements gained by using more advanced classifier combination methods in this setting.

Table 4 shows the results of a single pass over the training data for this baseline on the spam and sentiment data. Batch setting results are shown in Table 5. In almost every case, feature splitting improves over a single or separate classifiers. Therefore, we will compare algorithms in the next section to feature splitting. We also note that for the *All* dataset for spam, which contains reviews from many domains, the most effective strategy is to train a single classifier. For sentiment data feature distributions change between domains but the decision functions remain mostly the same. Therefore, the most effective strategy is to combine all data into a single set. We will observe the ability of our multi-domain learning algorithms to adapt to this setting by learning behaviors in the shared parameters.

	Spam		Sentiment			
	Task A	*Task B*	*Books*	*DVD*	*Books DVD*	*All*
Single	2.11	6.60	17.77	17.40	18.10	**14.34**
Separate	2.93	8.64	19.67	19.87	18.70	16.96
Feature Splitting	**1.94**	**5.51**	**9.97**	**9.70**	**9.05**	14.73

Table 4: Results show average test error across 10 runs in a true online setting. Each dataset contains multiple domains. The results show the error achieved by training a single classifier or separate classifiers for each domain. The feature splitting method of Daumé (2007) performs the best since it can balance between shared and domain specific behaviors.

	Spam		Sentiment			
	Task A	*Task B*	*Books*	*DVD*	*Books DVD*	*All*
Single	1.54	5.09	17.87	17.43	18.15	13.76
Separate	1.90	7.07	19.73	20.30	18.95	16.06
Feature Splitting	**1.38**	**4.21**	**7.67**	**6.97**	**7.02**	**13.37**

Table 5: Results for the same tests as in Table 4 but run as a batch algorithm with 10 online training iterations.

7.1 Base Classifier Divergence

Given instance features $\boldsymbol{x}$ and correct label y, we want to compute the solution to the following optimization, where weight vector $\boldsymbol{w}^c$ is drawn from the combination distribution $\mathcal{N}(\boldsymbol{\mu}^c, \Sigma^c)$ as defined in Sec. 4.

$$\begin{aligned}\min_{\{(\boldsymbol{\mu}^z,\Sigma^z)\}_z} \quad & \sum_{z\in Z} \mathbb{D}_{\mathrm{KL}}(\mathcal{N}(\boldsymbol{\mu}^z,\Sigma^z)||\mathcal{N}(\tilde{\boldsymbol{\mu}}^z,\tilde{\Sigma}^z)) \\ \text{s.t.} \quad & \Pr\left[y(\boldsymbol{w}^c\cdot\boldsymbol{x})>0\right]\ge\eta \end{aligned} \qquad (1)$$

The constraint can be rewritten

$$y(\boldsymbol{\mu}^c\cdot\boldsymbol{x}) \ge \phi\sqrt{\boldsymbol{x}^T\Sigma^c\boldsymbol{x}}$$

where $\phi = \Phi^{-1}(\eta)$ and Φ is the Gaussian cumulative distribution function. We drop the square root to get a variance constraint, as described in Sec. 3.

The Lagrangian of the resulting optimization is

$$\sum_{z\in Z} \mathbb{D}_{\mathrm{KL}}(\mathcal{N}(\boldsymbol{\mu}^z,\Sigma^z)||\mathcal{N}(\tilde{\boldsymbol{\mu}}^z,\tilde{\Sigma}^z)) - \alpha(y(\boldsymbol{\mu}^c\cdot\boldsymbol{x}) - \phi\boldsymbol{x}^T\Sigma^c\boldsymbol{x}) \ .$$

Since all Σ are diagonal and μ_i^c and Σ_i^c depend only on the values of μ_i^z and Σ_i^z for various z, we can decompose the entire Lagrangian into the following sum of N terms.

$$\sum_{i=1}^{N}\left[\sum_{z\in Z}\frac{1}{2}(\log(\tilde{\Sigma}_i^z/\Sigma_i^z) + \Sigma_i^z/\tilde{\Sigma}_i^z + (\mu_i^z-\tilde{\mu}_i^z)^2/\tilde{\Sigma}_i^z - 1) - \alpha(y\mu_i^c x_i - \phi\Sigma_i^c x_i^2)\right] \qquad (2)$$

We now take the derivative with respect to μ_i^z.

$$\frac{1}{\tilde{\Sigma}_i^z}(\mu_i^z-\tilde{\mu}_i^z) - \alpha y\frac{\partial\mu_i^c}{\partial\mu_i^z}x_i$$

Here we assume, as is the case for both of our combination methods, that $\partial\Sigma_i^c/\partial\mu_i^z = 0$. Setting the Lagrangian derivative to zero yields the update rule

$$\mu_i^z = \tilde{\mu}_i^z + \alpha y \tilde{\Sigma}_i^z \frac{\partial \mu_i^c}{\partial \mu_i^z} x_i \ .$$

This rule is like the single-domain CW update in that low values of $\tilde{\Sigma}_i^z$ (high confidence) lead to smaller parameter updates, while less confident parameters undergo larger adjustments. Here, however, there is the additional factor $\partial\mu_i^c/\partial\mu_i^z$, which roughly suggests that parameters contributing more to the combination should receive larger updates.

Performing the same calculations for Σ_i^z yields

$$\frac{1}{2}(-1/\Sigma_i^z + 1/\tilde{\Sigma}_i^z) - \alpha \left(y \frac{\partial \mu_i^c}{\partial \Sigma_i^z} x_i - \phi \frac{\partial \Sigma_i^c}{\partial \Sigma_i^z} x_i^2 \right)$$

and the corresponding update is

$$\frac{1}{\Sigma_i^z} = \frac{1}{\tilde{\Sigma}_i^z} + 2\alpha\phi \frac{\partial \Sigma_i^c}{\partial \Sigma_i^z} x_i^2 - 2\alpha y \frac{\partial \mu_i^c}{\partial \Sigma_i^z} x_i \ .$$

The partial derivates $\frac{\partial \mu_i^c}{\partial \mu_i^z}$, $\frac{\partial \Sigma_i^c}{\partial \Sigma_i^z}$ and $\frac{\partial \mu_i^c}{\partial \Sigma_i^z}$ depend on the chosen combination method.

7.1.1 ℓ_2 Distance

When combination is performed using ℓ_2 distance, we have the following partial derivatives.

$$\frac{\partial \mu_i^c}{\partial \mu_i^z} = c_i^z \quad , \quad \frac{\partial \Sigma_i^c}{\partial \Sigma_i^z} = c_i^z \quad , \quad \frac{\partial \mu_i^c}{\partial \Sigma_i^z} = 0$$

This simplifies the update rules to

$$\begin{aligned} \mu_i^z &= \tilde{\mu}_i^z + \alpha y \tilde{\Sigma}_i^z c_i^z x_i \\ \frac{1}{\Sigma_i^z} &= \frac{1}{\tilde{\Sigma}_i^z} + 2\alpha\phi c_i^z x_i^2 \ . \end{aligned}$$

Following the update, the new combination parameters are given by

$$\begin{aligned} \mu_i^c &= \sum_{z \in Z} c_i^z \mu_i^z \\ &= \sum_{z \in Z} c_i^z (\tilde{\mu}_i^z + \alpha y \tilde{\Sigma}_i^z c_i^z x_i) \\ &= \tilde{\mu}_i^c + \alpha y x_i \sum_{z \in Z} {c_i^z}^2 \tilde{\Sigma}_i^z && (3) \\ \Sigma_i^c &= \sum_{z \in Z} c_i^z \Sigma_i^z \\ &= \sum_{z \in Z} c_i^z \left(\frac{1}{\tilde{\Sigma}_i^z} + 2\alpha\phi c_i^z x_i^2 \right)^{-1} && (4) \end{aligned}$$

	Spam		Sentiment			
	Task A	*Task B*	*Books*	*DVD*	*Books DVD*	*All*
Feature Splitting	**1.94**	**5.51**	**9.97**	9.70	9.05	**14.73**
Eq (1): ℓ_2 Uniform	3.30	6.67	11.03	**8.93**	**8.48**	15.40
Eq (1): ℓ_2 Variance	3.43	8.09	10.67	10.53	14.45	15.09

Table 6: Results show average test error across 10 runs in a true online setting. Each dataset contains multiple domains. In the majority of the datasets, the feature splitting baseline beats the MDR objective (equation (1)). An analysis showed that this MDR objective led to over-fitting of the data.

	Spam		Sentiment			
	Task A	*Task B*	*Books*	*DVD*	*Books DVD*	*All*
Feature Splitting	**1.38**	**4.21**	**7.67**	**6.97**	**7.02**	**13.37**
Eq (1): ℓ_2 Uniform	2.82	6.45	11.00	10.30	8.73	15.33
Eq (1): ℓ_2 Variance	3.13	6.98	10.17	10.43	13.17	15.01

Table 7: Results for the same tests as in Table 6 but run as a batch algorithm with 10 online training iterations.

We solve for α by treating the constraint as an equality (which holds by the KKT conditions at optimum unless $\alpha = 0$). This yields the following.

$$\begin{aligned}
& y(\boldsymbol{\mu}^c \cdot \boldsymbol{x}) = \phi \boldsymbol{x}^T \Sigma^c \boldsymbol{x} \\
\Rightarrow \;& y \sum_{i=1}^{N} \left(\tilde{\mu}_i^c + \alpha y x_i \sum_{z \in Z} {c_i^z}^2 \tilde{\Sigma}_i^z \right) x_i = \phi \sum_{i=1}^{N} \sum_{z \in Z} c_i^z \left(\frac{1}{\tilde{\Sigma}_i^z} + 2\alpha\phi c_i^z x_i^2 \right)^{-1} x_i^2 \\
\Rightarrow \;& \sum_{i=1}^{N} \left[y \tilde{\mu}_i^c x_i + x_i^2 \sum_{z \in Z} \left(\alpha {c_i^z}^2 \tilde{\Sigma}_i^z - \phi c_i^z \left(\frac{1}{\tilde{\Sigma}_i^z} + 2\alpha\phi c_i^z x_i^2 \right)^{-1} \right) \right] = 0
\end{aligned}$$

We can solve this equation using a line search on α. A similar approach was taken by Dredze et al. (2008) in computing an exact solution for a diagonalized CW update.

Unfortunately, it is not possible to obtain a similar form for a KL-divergence combination update. Since the combination for $\tilde{\mu}_i^c$ relies on $\tilde{\Sigma}_i^z$, the updates for both $\boldsymbol{\mu}^c$ and Σ^c are interdependent. There are several simplifying assumptions that help to reduce the complexity of the update, such as assuming only two sets of parameters—as in our case here—and replacing the value of $\tilde{\Sigma}_i^z$ in the combination of $\tilde{\mu}_i^c$ with a constant, thereby reducing the dependency between the parameters. Still, these approaches were insufficient and a KL update depends on solving a system of equations. Since we favor fast online methods, we did not pursue this update.

7.1.2 Evaluation

We evaluated objective (1) using both uniform and variance weightings. Results are shown in Table 6 for the online setting and in Table 7 for the batch setting. In general, uniform weighting performs better than variance and we carry this method forward for comparison. Unfortunately, the new MDR objective outperforms feature splitting on only two out of six datasets. In the batch setting, feature splitting does better every time. Upon examination, we observed that the algorithm behaved aggressively and seemed to over-fit the training data. These results are similar to those obtained by Dredze et al. (2008) who used a line search to solve an exact CW diagonal method and observed over-fitting. In their case an exact solution was too aggressive and a relaxation produced superior performance. Additionally, the aggressive nature of the algorithm reduced the values of Σ quickly, which may have lead to numerical instability in estimating solutions for α using a line search. This problem could be addressed by finding a closed form MDR update. We consider both of these issues in the next section.

7.2 Combined Classifier Divergence

An alternative update method is to minimize the change in *combined* classifier distribution rather than the underlying base classifier distributions. This objective computes a regular CW update and then distributes it to the combination's underlying parameters. We compute the solution to the following optimization.

$$\begin{aligned}\min_{\boldsymbol{\mu}^c,\Sigma^c} \; & \mathbb{D}_{\mathrm{KL}}(\mathcal{N}(\boldsymbol{\mu}^c,\Sigma^c)||\mathcal{N}(\tilde{\boldsymbol{\mu}}^c,\tilde{\Sigma}^c)) \\ \text{s.t. } & \Pr\left[y(\boldsymbol{w}^c\cdot\boldsymbol{x})>0\right]\geq\eta \end{aligned} \tag{5}$$

The constraint can be rewritten as before, and dropping the square root yields the following Lagrangian.

$$\mathbb{D}_{\mathrm{KL}}(\mathcal{N}(\boldsymbol{\mu}^c,\Sigma^c)||\mathcal{N}(\tilde{\boldsymbol{\mu}}^c,\tilde{\Sigma}^c)) - \alpha(y(\boldsymbol{\mu}^c\cdot\boldsymbol{x}) - \phi\boldsymbol{x}^T\Sigma^c\boldsymbol{x}) \ .$$

This is exactly Eq. (10) of Dredze et al. (2008). Following their derivation we get the optimal solution, and manipulation yields the update rule

$$\boldsymbol{\mu}^c = \tilde{\boldsymbol{\mu}}^c + \alpha y\tilde{\Sigma}^c\boldsymbol{x} \quad , \quad (\Sigma^c)^{-1} = (\tilde{\Sigma}^c)^{-1} + 2\alpha\phi\boldsymbol{x}\boldsymbol{x}^T \ ,$$

where

$$\hat{\alpha} = \frac{-(2\phi M+1)+\sqrt{(2\phi M+1)^2-8\phi(M-\phi V)}}{4\phi V} \ , \tag{6}$$

and

$$\alpha = \max(\hat{\alpha},0) \ .$$

7.2.1 ℓ_2 Distance

The question now remains: how should we update $(\boldsymbol{\mu}^z,\Sigma^z)$ so that the desired (updated) combination distribution is achieved? A principled approach calls for a new objective to determine how to distribute the update to the combination's underlying parameters. We can minimize the change in the base parameters subject to the constraint that they achieve the desired combination. For the ℓ_2 combination, this yields:

$$\begin{aligned}\min_{\{(\boldsymbol{\mu}^z,\Sigma^z)\}_z} \; & \sum_{z\in Z}\mathbb{D}_{\mathrm{KL}}(\mathcal{N}(\boldsymbol{\mu}^z,\Sigma^z)||\mathcal{N}(\tilde{\boldsymbol{\mu}}^z,\tilde{\Sigma}^z)) \\ \text{s.t. } \forall i: \; & \sum_{z\in Z} c_i^z\mu_i^z = \mu_i^c = \tilde{\mu}_i^c + \alpha y\tilde{\Sigma}_i^c x_i \\ & \sum_{z\in Z} c_i^z\Sigma_i^z = \Sigma_i^c = ((\tilde{\Sigma}_i^c)^{-1}+2\alpha\phi x_i^2)^{-1} \end{aligned} \tag{7}$$

The Lagrangian is given by

$$\begin{aligned}\sum_{i=1}^{N}\Bigg[&\sum_{z\in Z}\frac{1}{2}(\log(\tilde{\Sigma}_i^z/\Sigma_i^z)+\Sigma_i^z/\tilde{\Sigma}_i^z+(\mu_i^z-\tilde{\mu}_i^z)^2/\tilde{\Sigma}_i^z-1) \\ &-\beta_i\left(\sum_{z\in Z}c_i^z\mu_i^z-\tilde{\mu}_i^c-\alpha y\tilde{\Sigma}_i^c x_i\right)-\gamma_i\left(\sum_{z\in Z}c_i^z\Sigma_i^z-((\tilde{\Sigma}_i^c)^{-1}+2\alpha\phi x_i^2)^{-1}\right)\Bigg]\end{aligned} \tag{8}$$

	Spam		Sentiment			
	Task A	*Task B*	*Books*	*DVD*	*Books DVD*	*All*
Feature Splitting	1.94	**5.51**	9.97	9.70	9.05	14.73
Eq (1): ℓ_2 Uniform	3.30	6.67	11.03	8.93	**8.48**	15.40
Eq (5): ℓ_2 Uniform	**1.89**	5.55	10.17	9.37	8.77	14.89
Eq (5): ℓ_2 Variance	2.12	5.72	**8.93**	**8.73**	12.37	**14.36**

Table 8: Results show average test error across 10 runs in a true online setting. Each dataset contains multiple domains. The new objective (5) improves on the majority of the datasets compared with feature splitting. We also list the best performing objective (1) for comparison.

	Spam		Sentiment			
	Task A	*Task B*	*Books*	*DVD*	*Books DVD*	*All*
Feature Splitting	**1.38**	**4.21**	**7.67**	6.97	**7.02**	**13.37**
Eq (1): ℓ_2 Uniform	2.82	6.45	11.00	10.30	8.73	15.33
Eq (5): ℓ_2 Uniform	1.54	4.46	8.13	**6.93**	7.55	13.46
Eq (5): ℓ_2 Variance	1.43	4.52	9.13	9.50	12.60	13.53

Table 9: Results for the same tests as in Table 8 but run as a batch algorithm with 10 online training iterations.

Taking the derivatives of the Lagrangian gives the update rules

$$\mu_i^z = \tilde{\mu}_i^z + \beta_i \tilde{\Sigma}_i^z c_i^z$$
$$(\Sigma_i^z)^{-1} = (\tilde{\Sigma}_i^z)^{-1} - 2\gamma_i c_i^z \ .$$

Plugging into the constraints and solving gives

$$\begin{aligned} & \sum_{z \in Z} c_i^z (\tilde{\mu}_i^z + \beta_i \tilde{\Sigma}_i^z c_i^z) = \tilde{\mu}_i^c + \alpha y \tilde{\Sigma}_i^c x_i \\ \Rightarrow \ & \sum_{z \in Z} \beta_i {c_i^z}^2 \tilde{\Sigma}_i^z = \alpha y \tilde{\Sigma}_i^c x_i \\ \Rightarrow \ & \beta_i = \frac{\alpha y \tilde{\Sigma}_i^c x_i}{\sum_{z \in Z} {c_i^z}^2 \tilde{\Sigma}_i^z} \end{aligned} \tag{9}$$

and

$$\sum_{z \in Z} c_i^z ((\tilde{\Sigma}_i^z)^{-1} - 2\gamma_i c_i^z)^{-1} = ((\tilde{\Sigma}_i^c)^{-1} + 2\alpha\phi x_i^2)^{-1} \ .$$

While we can now obtain a closed form solution for several parameters, γ still requires a line search. Note that if $c_i^z = 1/M$, we have $\beta_i = \alpha y M x_i$ and the update for μ_i^z is $\tilde{\mu}_i^z + \alpha y \tilde{\Sigma}_i^z x_i$.

As in the previous section, we were unable to obtain an update form suitable for a line search when using KL combination.

7.2.2 Evaluation

We evaluated objective (5) solved by the secondary step (7). Results are shown in Table 8 for the online and Table 9 for the batch setting. This new objective improves over the approach from the previous section. Additionally, the results for sentiment *All* come close to the results for a single classifier, indicating that this method effectively shifts learning to the shared parameters. First computing an update and then distributing it to the parameters seems to have an inherent regularization effect that reduces over-fitting seen in the previous algorithm. Still, this MDR algorithm does not improve over the baseline on every dataset. We still desire a closed form solution.

7.3 Averaged Update for Combined Classier Divergence

While the previous sections have explored two objectives that give different approaches to our problem, we have yet to derive a closed form solution for either. Additionally, these updates cannot be extended to KL combinations, which limits their ability to extend to different combination methods. Finally, these two objectives did not improve over the baselines in every case.

The average performance of these methods may be related to the lack of a closed form solution. Dredze et al. (2008) found that closed form solutions, even when they were approximations for the intended objective, performed better than exact solutions which were optimized using inexact methods (e.g., line search.) Additionally, they observed that the exact diagonalization methods converged faster than the approximate methods, indicating that the exact solution was over-fitting the data. These results favored the use of closed form solutions over inexact methods, even if the objective was an approximation to the intended objective.

We face the same situation in our application. Solving the objectives outlined so far has failed to field a closed form solution and the resulting methods were unable to consistently improve over the baseline. Instead, we seek a similar solution to Dredze et al. (2008): we simplify the objective to obtain a closed form solution. This result will increase the speed of the algorithm and may improve the performance.

We begin our derivation of a closed form solution by returning to objective (5). We observe that when using the ℓ_2 combination, the combination parameters are simple weighted (normalized) sums of the base parameters. Before, we sought to minimize the change of the combined parameters, which in turn minimizes the change in each of the underlying parameters. The question is how to distribute responsibility for this change? Which set of underlying parameters should change more and which less? Solving this problem turned out to be difficult and was without a closed form solution. Instead, we propose to assign equal responsibility for the update to each of the underlying parameters: they should change equally. The result is that we make a simple averaged update to the underlying parameters.

To make this averaged update, we simply use the update presented in Dredze and Crammer (2008):

$$\begin{aligned}
\mu_i^z &= \tilde{\mu}_i^z + (\mu_i^c - \tilde{\mu}_i^c) \\
&= \tilde{\mu}_i^z + \alpha y \tilde{\Sigma}_i^c x_i \\
(\Sigma_i^z)^{-1} &= (\tilde{\Sigma}_i^z)^{-1} + \left[(\Sigma_i^c)^{-1} - (\tilde{\Sigma}_i^c)^{-1}\right] \\
&= (\tilde{\Sigma}_i^z)^{-1} + 2\alpha\phi x_i^2 \qquad (10)
\end{aligned}$$

However, by using α as defined in (6) we ensure that the responsibility for fulfilling the constraint is spread across both sets of parameters. Additionally, we can follow the same derivation for KL and obtain the same closed form solution, where μ_i^c and Σ_i^c are obtained using a KL combination. We simply plug the parameters resulting from this combination into (10).

7.3.1 Evaluation

Given our previous observations, we expect the average update (10) to be less susceptible to over-fitting and possible numerical estimation errors. Results for the average update using ℓ_2 and KL as well as uniform and variance weightings are shown in Table 10 for the online setting and in Table 11 for the batch setting compared to the previous objective and baseline methods. These updates

	Spam		Sentiment			
	Task A	*Task B*	*Books*	*DVD*	*Books DVD*	*All*
Single	2.11	6.60	17.77	17.40	18.10	14.34
Separate	2.93	8.64	19.67	19.87	18.70	16.96
Feature Splitting	1.94	5.51	9.97	9.70	9.05	14.73
Eq (1): ℓ_2 Uniform	3.30	6.67	11.03	8.93	8.48	15.40
Eq (5): ℓ_2 Uniform	1.89	5.55	10.17	9.37	8.77	14.89
Avg: ℓ_2 Uniform	**1.61**	**5.16**	**6.63**	7.97	8.60	14.20
Avg: ℓ_2 Variance	2.22	7.73	14.87	14.83	17.83	14.44
Avg: KL Uniform	1.96	5.43	8.37	**7.07**	**8.43**	14.56
Avg: KL Variance	1.89	5.32	16.77	17.00	18.03	**13.79**

Table 10: Results show average test error across 10 runs in the true online setting. Each dataset contains multiple domains. The average update improves over solving an additional minimization problem for objective (5). Additionally, it improves on all datasets compared with the baselines.

	Spam		Sentiment			
	Task A	*Task B*	*Books*	*DVD*	*Books DVD*	*All*
Single	1.54	5.09	17.87	17.43	18.15	13.76
Separate	1.90	7.07	19.73	20.30	18.95	16.06
Feature Splitting	1.38	4.21	7.67	6.97	7.02	13.37
Eq (1): ℓ_2 Uniform	2.82	6.45	11.00	10.30	8.73	15.33
Eq (5): ℓ_2 Uniform	1.54	4.46	8.13	6.93	7.55	13.46
Avg: ℓ_2 Uniform	**1.37**	4.86	5.97	6.13	**6.35**	**12.97**
Avg: ℓ_2 Variance	1.68	7.30	12.17	11.27	17.27	14.27
Avg: KL Uniform	1.40	**3.79**	**5.30**	**6.10**	6.63	13.04
Avg: KL Variance	1.58	3.97	13.33	13.33	16.92	13.16

Table 11: Results for the same tests as in Table 10 but run as a batch algorithm with 10 online training iterations.

improve when compared both with previous methods and baselines in both batch and online settings. In each dataset the best results are obtained using one of the approximation methods. Additionally, we observe that each average method does almost as well as or better than the single classifier on the sentiment *All* dataset. Given a choice of which algorithm to deploy, MDR has the advantage of being able to learn across domains when domains differ and obtain similar results as a single classifier when all domains share the same behaviors.

8. Learning in Many Domains

A strong motivator for learning across domains are settings with large numbers of domains or users, where the aggregation of data has potential for improved methods. While we have so far considered settings with a small number of similar domains, we now extend our methods to a large number of domains. In doing so, we tackle two problems: learning divergent behaviors common to different subsets of the domains and scaling our methods to large numbers of domains. Consider a spam filter used by a large email provider, which filters billions of emails for millions of users. Suppose that spammers control many accounts and maliciously label spam as legitimate. Alternatively, subsets of users may share preferences. Since behaviors are not consistent across domains, shared parameters cannot be learned. Additionally, algorithms must scale to this setting, handling millions of users.

We begin by considering the common real-world problem of many divergent domains. Behaviors may be common to multiple users, but are likely not common to all users, such as the spam case above. The goal is to learn behaviors common to subsets of users with our MDR framework. Our approach is to group domains by their behavior and then learn these behaviors across the group. For example, discriminating spammer accounts from legitimate accounts would enable separate intra-group learning for each type. We learn such groups and simultaneously model parameters for them by augmenting the MDR framework.

Instead of a single set of MDR parameters shared across all domains, we use k sets of shared parameters, each shared across one of k learned groups. Behaviors common to a sub-group can be learned by the corresponding shared parameters. If the mapping between domains and groups is known beforehand, then we can treat these as k independent problems and use any of the objectives and updates from before. If there are many fewer shared parameters than domains, we can expect to benefit from such multi-domain learning.

However, when there is no known grouping of domains, we must learn the groups as we learn the model parameters. Intuitively, a domain should be mapped to a group with shared (group) parameters that correctly classify the examples from that domain. Hence, a group represents a collection of domains with similar behavior on the prediction task of interest. A common technique for learning such a mapping is the Weighted Majority algorithm (Littlestone and Warmuth, 1994), which weighs a mixture of experts (classifiers). In our setting, an expert is a set of shared parameters and we seek the set of shared parameters that gives the best predictions for a domain. However, since we require a hard assignment to a group rather than a mixture, the algorithm reduces to picking the shared classifier with the fewest errors (e) in predicting the domain. Thus, instead of maintaining an expert weight for each set of shared parameters, we keep a count of the number of known mistakes made by each shared classifier on each domain. For learning, the shared classifier with the fewest mistakes for a domain is selected for an MDR update. (Classifier ties are broken randomly.) An outline of the algorithm is given in Algorithm 1.

While we experimented with more complex techniques, this simple method worked well in practice. When a new domain is added to the system, it appears to take fewer examples to learn which shared classifier to employ than it does to learn a new domain-specific model from scratch. Note that this approach adds a free parameter k. It can be set using development data if needed; however, we observe that k can in practice be fixed to a large constant. Since only a single shared classifier is updated on each round, the algorithm will favor selecting a previously used group over a new one since it will have higher accuracy as long as the domain is similar to previously observed data. Therefore, the algorithm can adaptively use as many shared parameter sets as needed. This may not be optimal, but it is simple and frequently effective.

8.1 Evaluation

To evaluate this approach, we created a large number of varying domains using our spam and sentiment data.

- *Spam.* Six email domains were created by splitting each of the three task A users into two domains. One domain remained unchanged and the other had its label flipped, representing a malicious user. This yielded 400 train and 100 test emails for each of the six resulting domains. The optimal learner would create two groups, one for the unchanged labelings and the other for the malicious users.

- *Sentiment.* For sentiment, the book domain was split into three groups, each with a different binary decision boundary: star ratings of 2, 3 or 4. Each of these resulting three groups was split again into 8 groups, of which half then had their labels flipped. The result was 24 domains

Algorithm 1 Multi-Domain Regularization Across Many Domain

```
**Input:** $\phi \in [0, \infty], K, D$
**Initialize:**
  $\forall k: \quad \boldsymbol{\mu}_1^k = \mathbf{0}\ ,\ \Sigma_1^k = I$
  $\forall d: \quad \boldsymbol{\mu}_1^d = \mathbf{0}\ ,\ \Sigma_1^d = I$
  $\forall k, d: \quad e_d^k = 0$
**for** $t = 1, 2 \ldots$ **do**
  Receive example $\boldsymbol{x}_t$ from domain $d$
  Select group $k = \arg\min_{k'} e_d^{k'}$
  Create $\boldsymbol{\mu}_t^c, \Sigma_t^c$ from $\boldsymbol{\mu}_t^k, \Sigma_t^k$ (shared) and $\boldsymbol{\mu}_t^d, \Sigma_t^d$ (domain) using $\ell_2$ or KL
  Output the prediction $\hat{y}_t = \mathrm{sign}(\boldsymbol{\mu}_t^c \cdot \boldsymbol{x}_t)$
  Receive true label $y_t$
  **for** $k' = 1 \ldots K$ **do**
    $e_d^{k'} = e_d^{k'} + \mathbb{I}(y_t(\boldsymbol{\mu}_t^{k'} \cdot \boldsymbol{x}_t) < 0)$
  **end for**
  Compute update to $\boldsymbol{\mu}_t^k, \Sigma_t^k$ and $\boldsymbol{\mu}_t^d, \Sigma_t^d$ using (10)
**end for**
**Output:** Final $\{\boldsymbol{\mu}^k, \Sigma^k\}_{k=1}^K$ and $\{\boldsymbol{\mu}^d, \Sigma^d\}_{d=1}^D$
```

with varying decision boundaries and inverted labels. The same procedure was repeated for DVD reviews; for a decision boundary of 3 stars, 6 groups were created while decision boundaries of 2 and 4 stars were split into 3 groups each. One or two of the domains from each set had their labels flipped. The result was 12 DVD domains. DVD and Books together total 36 total domains with various decision boundaries, features, and inverted decision functions. Each domain had 300 train and 100 test instances.

We made one additional modification that improved performance. In defining their multi-task regularization objective, Evgeniou and Pontil (2004) include parameters λ_1, λ_2 to tradeoff between minimizing the norms of the shared parameters and task specific parameters. The λ values effectively encode a belief about the degree to which parameters are shared between tasks (in the same group). We added these hyperparameters to our averaged MDR update (10), yielding the modified rules:

$$\begin{aligned} \mu_i^k &= \tilde{\mu}_i^k + \lambda_1 \alpha y \tilde{\Sigma}_i^c x_i \\ (\Sigma_i^k)^{-1} &= (\tilde{\Sigma}_i^k)^{-1} + \lambda_1 2\alpha\phi x_i^2 \\ \mu_i^d &= \tilde{\mu}_i^d + \lambda_2 \alpha y \tilde{\Sigma}_i^c x_i \\ (\Sigma_i^d)^{-1} &= (\tilde{\Sigma}_i^d)^{-1} + \lambda_2 2\alpha\phi x_i^2 \end{aligned} \tag{11}$$

For convenience, we can rewrite $\lambda_2 = 2 - 2\lambda_1$, where $\lambda_1 \in [0, 1]$. We then optimize the single additional hyperparameter λ_1 on a single randomized run along with ϕ.

Each experiment was repeated using 10-fold cross validation with one online training iteration for each of our two many-domain datasets. Experiments were repeated for increasing settings of k. The results for the MDR averaged update (11) are shown in Figure 1. For both spam and sentiment data, adding additional shared parameters and simultaneously learning groupings of domains significantly reduces error, with further reductions as k increases. The end result is a 45% error reduction for spam and a 38% reduction for sentiment over the best baseline. While each task has an optimal k (about 5 for spam, 2 for sentiment), larger values still achieve comparable error, indicating the relative insensitivity of performance to the use of large k values.

We note some ties between the proposed method and certain Bayesian learning approaches: we have essentially described a hierarchy of model parameters, albeit a shallow hierarchy of depth 2. Observing that k can be treated as "large" also suggests non-parametric learning, which allows a

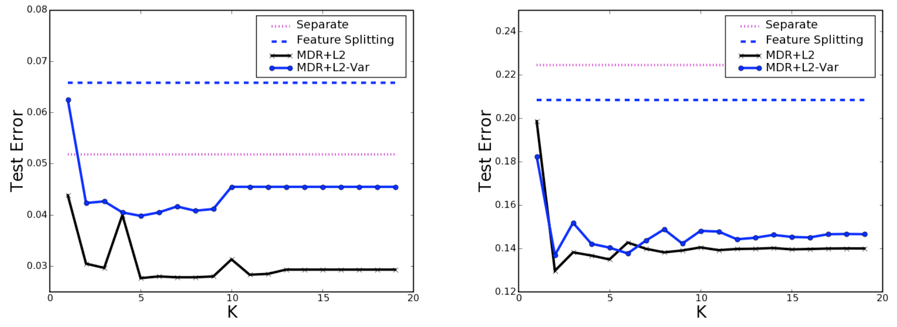

Figure 1: Test error over 10 randomized runs for learning across many domains - spam (left, 6 domains) and sentiment (right, 36 domains) - with MDR average update 11 using k shared classifiers. Increasing k improves performance as otherwise conflicting behaviors can be shared across domains. Note that as k increases, performance changes little, indicating that k need not be optimized but instead set to a large constant.

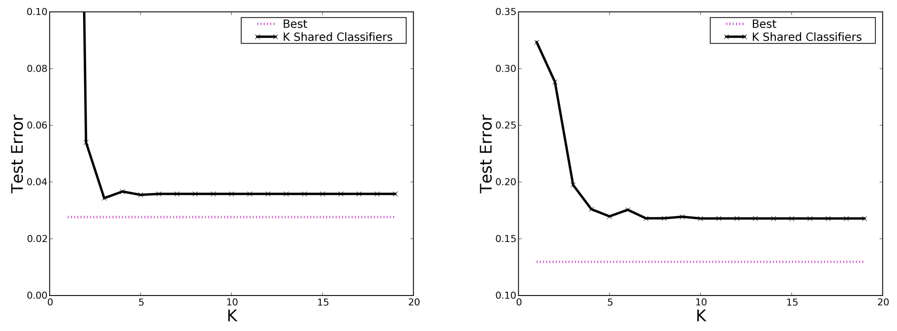

Figure 2: Test error over 10 randomized runs for learning across many domains - spam (left, 6 domains) and sentiment (right, 36 domains) - with MDR average update 11 using k shared classifiers and no domain specific parameters. As before, increasing k improves performance and that at some point performance changes little.

model to use a potentially unlimited number of parameters, depending on the data. Recent work by Daumé (2009) introduces a hierarchical Bayesian model for multi-task learning with applications to multi-domain learning. It is possible that our methods and these types of models can be combined to learn arbitrary hierarchies for complex multi-domain data. However, a significant challenge in this combination is the creation of a hierarchy in an online setting, where models predictions are required as the hierarchy is learned from a single instance at a time. This type of combination remains an open research area.

8.2 Scaling to Many Domains

While adding parameters clearly helps for many domains, it may be impractical to keep domain-specific classifiers for thousands or millions of domains. In this case, we can eliminate the domain-specific classifiers and rely on the k shared classifiers alone, but still learning the groupind of domains. We compare this approach to the best result from MDR above, again varying k. Figure 2 shows that losing domain-specific parameters hurts performance, but is still an improvement over baseline methods. Additionally, we can expect better performance as the number of similar domains increases. This may be an attractive alternative to keeping a very large number of parameters.

9. Related Work

Multi-domain learning intersects two areas of research: domain adaptation and multi-task learning. We first begin with a review of domain adaptation in NLP tasks and then review some related multi-task learning literature.

9.1 Domain Adaptation and Transfer Learning

Changing domains presents a difficult problem for NLP tasks since systems rely on a familiar vocabulary (features) for learning. Changing domains means both introducing new signals and removing reliable sources of information.

Several theoretical analyses have considered the domain adaptation problem. Ben-David et al. (2006) formalize the problem by defining two sources of adaptation error. First, feature distributions differ between two domains, meaning that test instances are unlikely to look like training examples. Since most NLP tasks use lexical items as features, this problem can be especially difficult. In general, this problem is addressed through the use of unlabeled target data since feature distributions can be measured and aligned without annotated examples. Second, decision functions differ between domains; some instance are labeled differently depending on the domain. Correcting this error relies on knowledge of the labeling function, which can only be gleaned from labeled target examples. Dredze et al. (2007) showed how domain adaptation for parsing is difficult when annotation guidelines differ for different domains. Blitzer et al. (2007a) prove bounds for both settings demonstrating how data from multiple domains can be mixed for learning on the target domain. Also of note is recent theoretical work on learning from multiple data sources (Mansour et al., 2009). This analysis shows the importance of using domain combinations that are sensitive to distributions. Our multi-domain algorithms take a similar approach since our combinations are sensitive to the feature distributions in each domain: more frequent features have lower variance and contribute more to the combined parameters.

In addition to the theoretical analyses, there has been significant empirical work on algorithms for domain adaptation and transfer learning. Several approaches rely on unlabeled data for overcoming feature distribution differences between domains. This approach can be thought of as semi-supervised learning, in which a learner trained on a small amount of (source) labeled data is augmented with a large amount of unlabeled (target) data. Following this theme, several semi-supervised techniques have been used for adaptation. Blitzer et al. (2006) introduced structural correspondence learning, which is based on a semi-supervised learning algorithm (Ando and Zhang, 2005) to align features from two domains based on common behaviors. Blitzer et al. (2007b) further showed significant improvements in sentiment classification domain adaptation. Additional improvements were shown when small amounts of target labeled data were available for learning. Others have learned representations for adaptation, such as Satpal and Sarawagi (2007) who create representations that maximize learning accuracy and minimize domain differences, which follows a theoretical result obtained by Ben-David et al. (2006). Do and Ng (2006) learn a parameter function, of which one benefit is better transfer between domains.

Another manner in which representations can aid adaptation is through careful structuring of features. For example, NLP applications make frequent use of feature conjunctions and back-off features, creating a natural feature hierarchy. Features higher in the hierarchy are more general and can appear in more domains. Arnold et al. (2008) create a learning method that favors such features to yield more generalizable models. Alternatively, Dai et al. (2009) align features in each domain so that learning can spread from features observed in labeled data to new target features. Rather than rely on aligning or structuring representations, useful feature statistics can be learned from unlabeled data. Second order features, which require covariance estimation, model feature interactions useful for learning. Raina et al. (2006) obtain these statistics from data from related problems. Similarly, self taught learning creates representations from unlabeled data sampled without domain restrictions (Raina et al., 2007). These representations may collapse similar domains, enabling adaptation. Another approach relying on unlabeled target domain data is instance weighting, whereby instances in source domains are given more weight during discriminative training when they closely match observed feature distributions in the target domain (Jiang and Zhai, 2007a,b).

A related setting, transfer learning leverages out of domain data to improve learning on in domain examples since out of domain data often indicates important features and appropriate parameter values. A common approach is to regularize a target problem based on out of domain learning. Chelba and Acero (2004) use a source domain classifier as a regularizer for a target domain with limited data. Marx et al. (2008) take a similar approach and define several priors for maximum entropy learning using data from related domains or users. Alternatively, Dai et al. (2007) use a generative model to learn a supervised classifier, and then employ an EM algorithm on unlabeled data to re-estimate target domain weights. Daumé and Marcu (2006) defines a graphical mixture model to learn differences between domains. Our approach can be considered similar to these methods, in that we rely on a shared set of parameters to transfer beliefs about parameter values across domains. Our approach can also be viewed as regularization following Dekel et al. (2006), who define a multi-domain online learning algorithm that regularizes updates using multiple domains. However, their algorithm assumes an instance from each domain for every update.

Some have modeled transfer as a covariance shift problem, which has been widely studied in machine learning. Covariance shift assumes that data from both domains have the same decision function but domain data points have been shifted. Bickel et al. (2007) model this shift for adaptation by learning a kernel logistic regression classifier. Similarly, Bickel et al. (2009) assume a change in prior over examples and derive a procedure for resampling weights that match the available examples to ones in the target domain. Changes in priors affect NLP tasks like word sense disambiguation, which relies on a prior over word senses for each domain. Re-estimating these priors for each domain improves disambiguation after adaptation (Chan and Ng, 2006).

9.2 Multi-Task Learning

Most transfer learning algorithms consider a one time transfer between problems, whereby a system is adapted to a new domain. However, in this paper we consider multi-domain learning, which learns across many domains at the same time and transfer happens continuously throughout learning. A similar setting is that of multi-task learning (Caruana, 1997), also known as inductive transfer. In multi-task learning, many related tasks are learned at once since information relevant to one task may be relevant to other related tasks. Like multi-domain learning, multi-task learning can improve accuracy on all related tasks. The two settings are related: multi-task learning has many tasks and a single dataset and multi-domain learning has a single task and many datasets.

As discussed above, our algorithm is based on Regularized Multi-Task Learning (Evgeniou and Pontil, 2004). In particular, Daumé (2007) considered a special case of this algorithm and applied it in a multi-domain learning scenario, showing results comparable to many of the more complex techniques considered above. Recent work has generalized this approach to a hierarchy of tasks, which is learned using Bayesian methods (Daumé, 2009). Dekel et al. (2006) gave a similar online approach but did not use shared parameters, instead assuming that instances from each domain were provided on each round. Others have taken a more direct approach to finding useful features, such as feature selection based on data from multiple related tasks (Obozinski et al., 2006). Additionally, some multi-task work clusters tasks by similarity (Bakker and Heskes, 2003). In particular, Thrun and O'Sullivan (1998) first group tasks and then learning across tasks within a grouping. We take a similar approach for learning on many disparate domains. We cluster our domains into groups (online) and share parameters between co-clustered domains. A related idea is presented by Abernethy et al. (2007), who examine prediction with expert advice across many online problems at once. The intuition is that a quality expert is good across many problems. This is similar to our assumption about features across multiple domains; a good feature can be predictive across multiple related domains.

10. Conclusion

We have explored several multi-domain learning settings using CW classifiers and a combination methods. For domain adaptation, our approach creates a better classifier for a new target domain than selecting a random source classifier a priori. For learning across domains, we have considered several formulations for combining domain specific and shared parameters based on confidence sensitive combinations. Our approach reduces error on several datasets by sharing learned information. Many practical problems may have subgroups of domains which share common behaviors but not share behaviors across all domains. We extended our approach to handle this disparate domain setting by increasing the number of shared parameters. Furthermore, we have shown how the same approach can be scaled to many domains with a small performance reduction, an important consideration for large scale learning settings. These scenarios are all realistic for NLP systems in the wild.

This work raises some new questions about multi-domain learning. First, when learning on a large number of disparate domains, can hierarchical clustering or a known hierarchical structure be used to better select shared parameters? Our algorithms are not restricted to two sets of parameters per update and perhaps knowledge can be shared in a hierarchical structure. Second, how can prior knowledge about domain similarity be included in the combination methods? We include two simple methods for weighting parameters: uniform and variance. Additional methods can be based on observed similarities on unlabeled data or prior knowledge. Third, we considered cases where domains could be clustered into sub-groups in which common behaviors are learned. Our clustering, and those considered by others (Thrun and O'Sullivan, 1998; Bakker and Heskes, 2003) assume a clustering of domains. However, domains may have certain classes of features whose behaviors could form overlapping subgroups. We plan to explore these scenarios in future work.

References

Jacob Duncan Abernethy, Peter Bartlett, and Alexander Rakhlin. Multitask learning with expert advice. Technical Report UCB/EECS-2007-20, EECS Department, University of California, Berkeley, Jan 2007.

Rie Ando and Tong Zhang. A framework for learning predictive structure from multiple tasks and unlabeled data. *Journal of Machine Learning Research (JMLR)*, 6:1817–1853, November 2005.

Andrew Arnold, Ramesh Nallapati, and William W. Cohen. Exploiting feature hierarchy for transfer learning in named entity recognition. In *Association for Computational Linguistics (ACL)*, 2008.

B. Bakker and T. Heskes. Task clustering and gating for bayesian multi–task learning. *Journal of Machine Learning Research (JMLR)*, 4:83–99, 2003.

Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

Steffen Bickel, Michael Brückner, and Tobias Scheffer. Discriminative learning for differing training and test distributions. In *International Conference on Machine Learning (ICML)*, 2007.

Steffen Bickel, Christoph Sawade, and Tobias Scheffer. Transfer learning by distribution matching for targeted advertising. In *Advances in Neural Information Processing Systems*, pages 145–152, 2009.

John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2006.

John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jenn Wortman. Learning bounds for domain adaptation. In *Advances in Neural Information Processing Systems (NIPS)*, 2007a.

John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Association for Computational Linguistics (ACL)*, 2007b.

Rich Caruana. Multitask learning. *Machine Learning*, 28:41–75, 1997.

Yee Seng Chan and Hwee Tou Ng. Estimating class priors in domain adaptation for word sense disambiguation. In *Association for Computational Linguistics (ACL)*, 2006.

Ciprian Chelba and Alex Acero. Adaptation of maximum entropy classifier: Little data can help a lot. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2004.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research (JMLR)*, 7:551–585, 2006.

Koby Crammer, Mark Dredze, and Fernando Pereira. Exact confidence-weighted learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.

Wenyuan Dai, Gui-Rong Xue, Qiang Yang, and Yong Yu. Transferring naive bayes classifiers for text classification. In *American National Conference on Artificial Intelligence (AAAI)*, 2007.

Wenyuan Dai, Yuqiang Chen, Gui-Rong Xue, Qiang Yang, and Yong Yu. Translated learning: Transfer learning across different feature spaces. In *Advances in Neural Information Processing Systems (NIPS)*, pages 353–360, 2009.

Hal Daumé. Frustratingly easy domain adaptation. In *Association for Computational Linguistics (ACL)*, 2007.

Hal Daumé. Bayesian multitask learning with latent hierarchies. In *Uncertainty in Artificial Intelligence (UAI)*, 2009.

Hal Daumé and Daniel Marcu. Domain adaptation for statistical classifiers. *Journal of Artificial Intelligence Research (JAIR)*, 26(101-126), 2006.

Ofer Dekel, Philip M. Long, and Yoram Singer. Online multitask learning. In *Conference on Learning Theory (COLT)*, 2006.

Chuong B. Do and Andrew Ng. Transfer learning for text classification. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

Mark Dredze and Koby Crammer. Online methods for multi-domain learning and adaptation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2008.

Mark Dredze, John Blitzer, Partha Pratim Talukdar, Kuzman Ganchev, Joao Graca, and Fernando Pereira. Frustratingly hard domain adaptation for parsing. In *Shared Task - Conference on Natural Language Learning - CoNLL 2007 shared task*, 2007.

Mark Dredze, Koby Crammer, and Fernando Pereira. Confidence-weighted linear classification. In *International Conference on Machine Learning (ICML)*, 2008.

Theodoros Evgeniou and Massimiliano Pontil. Regularized multi-task learning. In *Conference on Knowledge Discovery and Data Mining (KDD)*, 2004.

Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang. Named entity recognition through classifier combination. In *Conference on Computational Natural Language Learning (CONLL)*, 2003.

Jing Jiang and ChengXiang Zhai. Instance weighting for domain adaptation in nlp. In *Association for Computational Linguistics (ACL)*, 2007a.

Jing Jiang and ChengXiang Zhai. A two-stage approach to domain adaptation for statistical classifiers. In *Conference on information and knowledge management (CIKM)*, 2007b.

J. Kittler, M. Hatef, R.P.W. Duin, and J. Matas. On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):226–239, 1998.

Matthew Lease and Eugene Charniak. Parsing biomedical literature. In *International Joint Conference on Natural Language Processing (IJCNLP)*, 2005.

N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994.

Y. Mansour, M. Mohri, and A. Rostamizadeh. Domain adaptation with multiple sources. In *Advances in Neural Information Processing Systems*, 2009.

M.P. Marcus, M.A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of English: the penn treebank. *Computational Linguistics*, 19(2):313–330, 1993.

Zvika Marx, Michael T. Rosenstein, Thomas G. Dietterich, and Leslie Pack Kaelbling. Two algorithms for transfer learning. In *Inductive Transfer: 10 years later*, 2008.

David McClosky and Eugene Charniak. Self-training for biomedical parsing. In *Association for Computational Linguistics (ACL)*, 2008.

Guillaume Obozinski, Ben Taskar, and Michael Jordan. Multi-task feature selection. In *ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.

R. Raina, A. Battle, H. Lee, B. Packer, and A.Y. Ng. Self-taught learning: transfer learning from unlabeled data. In *International Conference on Machine Learning (ICML)*, pages 759–766, 2007.

Rajat Raina, Andrew Ng, and Daphne Koller. Constructing informative priors using transfer learning. In *International Conference on Machine Learning (ICML)*, 2006.

Sandeepkumar Satpal and Sunita Sarawagi. Domain adaptation of conditional probability models via feature subsetting. In *European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2007.

Gabriele Schweikert, Christian Widmer, Bernhard Schölkopf, and Gunnar Rätsch. An empirical analysis of domain adaptation algorithms for genomic sequence analysis. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.

David M. J. Tax, Martijn van Breukelen, Robert P. W. Duina, and Josef Kittler. Combining multiple classifiers by averaging or by multiplying? *Pattern Recognition*, 33(9):1475–1485, September 2000.

S. Thrun and J. O'Sullivan. Clustering learning tasks and the selective cross–task transfer of knowledge. In S. Thrun and L.Y. Pratt, editors, *Learning To Learn*. Kluwer Academic Publishers, 1998.

Kevin Woods, W. Philip Kegelmeyer Jr, and Kevin Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19 (4):405–410, 1997. ISSN 0162-8828. doi: http://doi.ieeecomputersociety.org/10.1109/34.588027.