Scalable Hierarchical Multitask Learning Algorithms for Conversion Optimization in Display Advertising

Amr Ahmed [*]
Google
amra@google.com

Abhimanyu Das [*]
Microsoft Research
abhidas@microsoft.com

Alexander J. Smola [*]
Carnegie Mellon University and Google
alex@smola.org

ABSTRACT

Many estimation tasks come in groups and hierarchies of related problems. In this paper we propose a hierarchical model and a scalable algorithm to perform inference for multitask learning. It infers task correlation and subtask structure in a joint sparse setting. Implementation is achieved by a distributed subgradient oracle and the successive application of prox-operators pertaining to groups and subgroups of variables. We apply this algorithm to conversion optimization in display advertising. Experimental results on over 1TB data for up to 1 billion observations and 1 million attributes show that the algorithm provides significantly better prediction accuracy while simultaneously being efficiently scalable by distributed parameter synchronization.

Categories and Subject Descriptors

G.3 [**Mathematics of Computing**]: Probability and Statistics; I.2.6 [**Artificial Intelligence**]: Learning

1. INTRODUCTION

In many cases data inference problems do not arise in isolation. That is, we usually encounter a range of *related* problems and there is considerable benefit in solving them jointly. This insight has been exploited repeatedly and it has led to algorithms commonly known as multitask learning techniques [4, 22, 12, 7, 18]. Applications, e.g. to massively multitasked spam filtering [20] show its practical importance. The key idea is that by solving related tasks we are able to learn more about an individual task.

In this paper we study the problem of conversion maximization in display advertising. That is, we focus on maximizing the occurrence of commercially relevant actions such as purchases, account creation, mailing list signups, etc. This involves estimating a user's propensity to perform such actions and to identify generally susceptible populations of users. The challenge here is that we have both a broad range of different advertisers and also a range of subtasks (views, clicks, conversions) that we wish to maximize.

[*]work done while the authors were at Yahoo! Research

WSDM'14, February 24–28, 2014, New York, New York, USA.
Copyright 2014 ACM 978-1-4503-2351-2/14/02 ...$15.00.
http://dx.doi.org/10.1145/2556195.2556264.

As is to be expected in computational advertising, the amount of data can be quite significant. Moreover, the data is not necessarily homogeneous. Tasks have wildly varying sizes, commensurate with the financial stake of the advertisers and the popularity of their product. Likewise, attributes are sparse and many occur only in a small number of contexts. This requires an effective inference approach.

Our work builds on well known multiple kernel learning [19] and collaborative filtering techniques namely that of effectively imposing a hyperprior on the regularization term. In doing so it is possible to cast multitask learning as a nontrivial convex optimization problem. See e.g. [23] for details. This strategy is then combined with a hierarchical model over task and subtask specific parameters. Furthermore, we impose structured sparsity along the lines of [5].

To solve the problem in practice we rely on a distributed subgradient oracle. Load-balancing is achieved by using consistent hashing for task distribution over processors and distributed variable aggregation to mitigate the latency and task restarts otherwise required in Hadoop MapReduce. That is, we use the variable distribution of [13] for storage. Subsequently we invoke a sequence of prox operators [5] to synchronize efficiently between local and global penalties. To summarize, our contributions are the following:

- We formulate the joint conversion, click and unattributed-conversion modeling problem in behavioral targeting as a large-scale hierarchical multitask learning problem and show that the convex multitask learning approach of [23] can be adapted to this setting.
- We design an efficient distributed implementation of the above algorithm that scales to Terascale data.
- Using a real-world, web-scale display advertising targeting data set and two smaller public datasets, we show the ability of our algorithm to significantly improve on the baseline modeling performance obtained by traditional single-task inference systems.

1.1 Challenges in Conversion Maximization

Recent trends in behavioral targeting and display advertising emphasize the importance of commercially relevant actions. That is, rather than user clicks, advertisers aim to maximize the number of conversions they receive. Conversions stand for purchases, account creation, or any other relevant action that an advertiser sees as particularly desirable. To obtain good performance, publishers tend to instrument their websites with embedded code which allows third parties to capture user transactions and generate user segments that are of high value for a particular advertiser [1]. These segments primarily contain users that are actually inclined to perform a transaction as opposed to a casual visit to the web site through an accidental click.

Conversions on an advertiser's web site are either "attributed" to their corresponding display ads based on advertiser-specific rules, such as the amount of time elapsed between the conversion time and the time that the ad was last shown to the user, or they are "unattributed" if they cannot be tied to a specific display ad. Past work [3] has shown the superiority of targeting platforms maximizing for attributed conversions to traditional solutions maximizing for clicks.

For conversion-oriented behavioral targeting the traditional approach has been to only consider attributed conversions. The corresponding inference problem for each advertising campaign is then solved independently, for instance, by fitting a Support Vector Machine or Logistic Regression model. This generates separate models based on user data for each campaign. However, a typical behavioral targeting platform optimizing for attributed conversions, henceforth simply referred as conversions, faces two core issues:

- There is a large volumes of user histories that need to be processed in a periodic fashion in order to perform inference over attributed conversions for a large number of ad campaigns. Processing activities of billions of users on a daily basis imposes many challenges such as how to build user profiles in an efficient way, and how to optimize multiple campaigns at the same time;
- When optimizing for each campaign separately we are likely to do poorly for infrequent campaigns. We therefore need to design algorithms that can deal with sparseness of attributed conversions in many campaigns. The absence of a sufficient number of labeled data for the inference tasks creates a major bottleneck against achieving tangible targeting performance improvement.

1.2 Multitask Learning

For each advertising campaign, we can formulate several other related inference tasks apart from conversion modeling: we can attempt to infer the likelihood of a click (which is typically a prerequisite for conversion) and to model the likelihood of unattributed conversions (the latter helps to identify similar users). These different inference problems of each campaign are likely to be correlated. Hence modeling them jointly should improve estimation performance.

Additionally, it is quite likely that there is significant correlations between inference tasks across different advertising campaigns. For example, if there exists advertising campaigns corresponding to different brands of cars, the conversion or click models for all these campaigns might be quite similar to each other. Thus, performing a joint inference over all these potentially-correlated tasks together might lead to better performance than solving these inference tasks separately. This is the basic premise of multitask learning.

The key difference to conventional multitask learning is that in our case there exists a *hierarchy* between tasks. That is, we expect that all tasks for a given campaign (e.g. selling car insurance for a particular company) have related specificity in terms of their user demographic. Hence it is only reasonable that the sets of features and preferences are shared between them. It is to be expected that joint feature selection should improve the performance of each estimator.

Strictly speaking, we have two somewhat related goals — one is to do well for *all* tasks, i.e. click, conversion, and unattributed conversion estimation. This is a symmetric setting where the goal is to use the task correlations to simultaneously improve the prediction performance of all the tasks. The other task is to perform well for conversion prediction while using the remaining data as *side information*. This is an asymmetric setting. While both tasks are rather related, they are subtly different in their performance criteria and in terms of the estimation problem.

In this paper, we formulate two different hierarchical multitask models for these settings: a *hierarchical* model for the symmetric setting, and an *attachment* model for the asymmetric setting. In the hierarchical model, we first define an inter-campaign correlation matrix on a root-level set of feature weights for each campaign. This is then used to derive feature weights for its conversions, clicks and unattributed conversions locally. In the attachment model, the inter-campaign correlation matrix is applied directly on the feature weights for the conversion model of each campaign.

There is ample literature covering the subject of multitask learning. However, in the context of our behavioral targeting problem, there are two objectives that a multitask learning algorithm should satisfy: it should be easily distributable and scale to thousands of campaigns and millions of features; it should extend to a multi-level task hierarchy. This makes the setting rather nontrivial in terms of efficient inference.

1.3 Approach

In this paper, we use a convex formulation approach for multitask learning. Its basic idea is described in several variants e.g. in the context of multitask learning [23], matrix inference [16] and multiple kernel learning [14]. Essentially, one imposes a penalty over the covariance matrices governing correlation between attribute vectors. This way we can ensure that primarily similar attribute sets and related attribute vectors are chosen.

In terms of conversion maximization this means that we use an inter-campaign covariance matrix to model the relationships between the various campaigns. Moreover, intra-campaign covariance matrices are used to model the relationships between clicks, conversions and unattributed conversions of each campaign. A matrix-variate normal prior is imposed on these covariance matrices. A joint optimization objective can be formulated for all the tasks by obtaining a maximum likelihood estimate of the covariance matrices and the per-task feature weights for all the tasks. As we will show later, this objective is convex. It can be solved in a distributed fashion using proximal subgradient methods, such as the Fast Iterative Shrinkage algorithm (FISTA) [6].

For both models, we use a scalable alternating subspace descent method for simultaneous inference of both the task correlation matrices and the feature weights. A key tool for achieving scalability will be to use a cluster of machines as a distributed subgradient oracle [17]. Since iterative thresholding algorithms like FISTA require a significant number of gradient computations (e.g. [6] report 100 steps), our platform must preserve state and data locality between iterations. This makes it unsuitable to a naive Hadoop MapReduce implementation. Instead, we employ a consistent hashing based synchronization algorithm. We apply our dis-

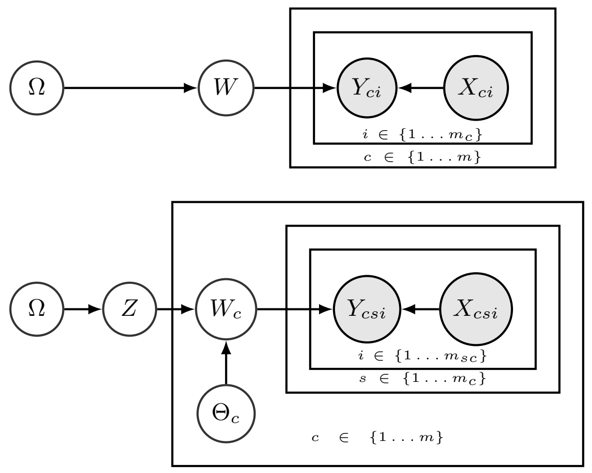


Figure 1: Top: Standard multitask learning using a Matrix-Variate distribution. Observations X_{ci} for campaign c receive labels Y_{ci}. These are assigned using campaign-specific weight vectors W_c. The latter are exchangeable but not independent, hence jointly drawn from some distribution $p(W|\Omega)$. Bottom: Hierarchical multitask learning. After drawing task specific parameters Z_c from an exchangeable but not independent distribution we draw subtask specific parameters W_s using a joint parameter Θ_c and Z_c. The rest remains unchanged.

tributed multitask learning framework to the conversion modeling problem described in [1] and we show how our system can improve the AUC performance significantly, when compared to individual conversion modeling of each ad-campaign.

2. MULTITASK LEARNING

2.1 Notation

We now cast the problem of campaign-specific estimation as a multitask learning problem. That is, we treat each campaign c as a task. In each such case we observe covariates (patterns) x_{ci} and our goal is to infer labels y_{ci}. For simplicity we assume that $x_{ci} \in \mathcal{X} = \mathbb{R}^d$ consists of d-dimensional vectors and moreover that $y_{ci} \in \mathcal{Y}$ are either binary $\mathcal{Y} = \{\pm 1\}$ for classification or real-valued $\mathcal{Y} = \mathbb{R}$ for regression. At a later stage (in section 3) we will assume that each campaign contains a number of subtasks s. The table below gives an overview of the symbols used:

$\mathcal{X}$	domain of observations (usually $\mathcal{X} = \mathbb{R}^d$)
$\mathcal{Y}$	domain of labels (usually $\mathcal{Y} = \{\pm 1\}$ or $\mathcal{Y} = \mathbb{R}$)
c	campaign index ($c \in \{1 \dots m\}$)
s	sub-campaign index ($s \in \{1 \dots n\}$)
j	observation index ($j \in \{1 \dots m_c\}$ or $j \in \{1 \dots m_{cs}\}$)
x	observation (x_{cj} or x_{csj})
X_c	set of observations for campaign c
y	observation (y_{cj} or y_{csj})
Y_c	set of labels for campaign c
w	parameter vectors (w_c or w_{cs})
W	stacked parameter vectors $W = \{\dots w_c \dots\}$
z	parameter vectors z_c for top level hierarchy
Z	stacked parameter vectors $Z = \{\dots z_c \dots\}$

Figure 1 captures the formal structure of the multitask learning problem. To capture interaction between covariates x_{cj}, campaigns c and associated labels y_{cj}, e.g. whether a particular user converted on an ad in a particular campaign at a particular occasion, we consider the issue of estimating $y|x, c$ for a large range of campaigns simultaneously. We denote by m the total number of campaigns and by m_c the number of observations per campaign. Formally we consider sets of covariates and labels indexed by a campaign c, denoted by $X^c = \{x_1^c, \dots, x_{n^c}^c\} \subseteq \mathcal{X}$ and $Y^c = \{y_1^c, \dots, y_{n^c}^c\} \subseteq \mathcal{Y}$. Here each pair (x_{cj}, y_{cj}) is drawn from some distribution $p(x, y|c) = p(y|x, c)p(x|c)$ of covariates and labels respectively. Finally, we denote by csi the combination of task, subtask, and coordinate, and by $\cdot$ the entire vector in the associated dimension. E.g. $w_{c \cdot i}$ denotes the vector over all subtasks s associated with c for coordinate i.

2.2 Objective

The inference problem is expressed either of risk minimization whenever we want to find a classifier which makes a small number of mistakes, or as one of maximizing the data likelihood. In the latter case we want to find parameters $W = \{w_1, \dots w_m\}$ such that the maximizes label likelihood:

$$p(Y|X, W) = \prod_{c=1}^{m} p(Y_c|X_c, w_c) = \prod_{c=1}^{m} \prod_{j=1}^{m_c} p(y_{cj}|x_{cj}, w_c) \quad (1)$$

Choices for $y_{cj}|x_{cj}, w_c$ are e.g.

$$p(y_{cj}|x_{cj}, w_c) = (2\pi\sigma^2)^{-\frac{1}{2}} e^{-\frac{1}{2\sigma^2}(y_{cj} - \langle x_{cj}, w_c \rangle)^2} \quad (2)$$

$$p(y_{cj}|x_{cj}, w_c) = \frac{1}{1 + e^{-y_{cj}\langle x_{cj}, w_c \rangle}} \quad (3)$$

for regression and classification respectively. A naive maximization of the conditional response likelihood $p(Y|X, W)$ leads to overfitting unless the model complexity is overly small or unless a suitable prior is used. Multitask learning models aim to address this problem by imposing a suitable prior distribution $p(W)$ on W which favors simple models and which exploits *correlation* between tasks. Consequently, instead of maximizing (1) one aims to find the Maximum-a-Posteriori (MAP) estimate of W via

$$\underset{W}{\text{maximize}}\, p(W) \prod_{c=1}^{m} p(Y_c|X_c, w_c) \quad (4)$$

The challenge is now to define models of $p(W)$ that are both computationally tractable and statistically meaningful. This hierarchical modeling imperative leads to multitask learning.

2.3 Multitask Prior

Our working assumption is that W is drawn from a matrix-variate distribution in such a way as to exploit correlations between the tasks, such as assuming that the tasks are more concentrated in a lower dimensional subspace. This is a reasonable assumption since there is no inherent order in which the tasks are laid out. One option is to choose a normal distribution as follows:

$$W \sim \mathcal{N}(0, \mathbf{1}_d \otimes \Omega) \text{ or equivalently } w_{\cdot i} \sim \mathcal{N}(0, \Omega) \quad (5)$$

for all coordinates i. The likelihood of $W|\Omega$ is given (up to constants) by

$$-\log p(W|\Omega) = \operatorname{tr} W\Omega^{-1}W^{\top} + d \log |\Omega| + c \quad (6)$$

It is straightforward to modify this by including a conjugate Wishart hyperprior on Ω. Unfortunately, the outcome is concave in W.

An alternative is to replace the log-barrier arising from a conjugate prior on Ω by a trace constraint and a positive semidefiniteness constraint. That is, we replace $\log|\Omega|$ by $\Omega \succeq 0$ and $\operatorname{tr}\Omega = 1$. This is used, e.g. in [23]. Such a modification leaves the eigenspace of the W-dependent part of the optimization problem unchanged. This leads to the following alternative:

$$\underset{W,\Omega}{\text{minimize}} \sum_c -\log p(Y_c|X_c, w_c) + \lambda \operatorname{tr} W\Omega^{-1}W^\top \tag{7a}$$

$$\text{subject to } \Omega \succeq 0 \text{ and } \operatorname{tr}\Omega = 1 \tag{7b}$$

The above formulation is convex in both W and Ω and can be solved using an efficient algorithm based on alternating subspace descent. For fixed Ω minimize (7) with respect to W. Subsequently, for fixed W, find the minimizer with respect to Ω. A simple constrained optimization problem shows that this can be found via

$$\hat{\Omega} = \frac{\left[W^\top W\right]^{-\frac{1}{2}}}{\operatorname{tr}\left[W^\top W\right]^{-\frac{1}{2}}} \tag{8}$$

This approach forms the baseline relative to which we will compare our proposed method.

3. HIERARCHICAL MULTITASK LEARNING

While the flat models presented in Section 2 can learn the correlation structure between tasks, they are not so easily amenable for distributed optimization because of the squared dependency between all the tasks. Fortunately, many large scale multitask problems possess a hierarchical structure that allows us to decompose them into tasks and subtasks. For example, in display advertising each advertiser can be regarded as a task (a campaign) within which we can define three subtasks as follows:

- Conversion prediction: estimate if the user will convert, i.e. perform a commercially relevant action, on the current display ad.
- Click prediction: predict if the user will click on the currently displayed ad.
- Unattributed conversion: historical data of users who converted on previous advertisements of the advertiser.

We use s to index the subtask. That is, rather than c we now use the tuple cs to index task and associated subtask, such as `(Coca Cola, clicks)`. All remaining notation is unchanged relative to the previous section.

In a nutshell we have two options for dealing with the hierarchical structure: firstly, we estimate the *joint* model for all tasks, subtasks and all campaigns. A second strategy is to solve the model for the primary subtask of conversion estimation exclusively and to use the associate (secondary) subtasks only as side-information. We will refer to the former as a *hierarchical* model and to the latter as *attachment* model. The key difference is in the following assumption:

- **Hierarchical Model:** We assume that for each task group there exists some parameter vector z_c, with $Z = \{z_1, \ldots z_m\}$ that specifies preferences per task

$$Z \sim \mathcal{N}(0, \mathbf{1}_d \otimes \Omega) \text{ or equivalently } z_{\cdot i} \sim \mathcal{N}(0, \Omega) \tag{9}$$

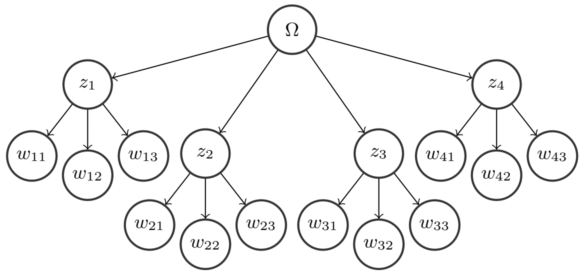
  Moreover, within each task, the distribution over subtasks is given by

$$w_{c \cdot i} \sim \mathcal{N}(1 \cdot z_{ci}, \Theta_c). \tag{10}$$

  This assumes that correlations within subgroups are decoupled.

- **Attachment Model:** Denote by $s = 1$ the primary subtask (conversion estimation). Instead of using z_c as an intermediary we couple the models directly via w_{c1} and use a hierarchical model on the remaining parameters. This amounts to

$$w_{\cdot 1i} \sim \mathcal{N}(0, \Omega) \text{ and } w_{c \cdot i} \sim \mathcal{N}(1 \cdot w_{c1i}, \Theta_c) \text{ for } s > 1.$$



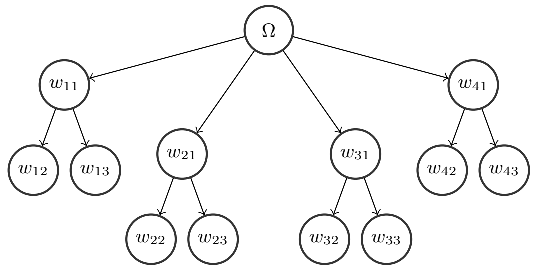


Figure 2: Top: Hierarchical dependency structure over parameter vectors for multitask learning. The intermediate parameter vector z_c encapsulates commonalities per task. Bottom: Attachment model. Here the conversion-specific parameters are directly coupled. For simplicity of the diagram we omitted Θ_c in both cases.

The diagram in Figure 2 describes the difference between both approaches for a rather simplistic structure of 4 tasks (in reality we may have millions of such tasks).

As previously discussed in Section 2, we again resort to a reformulation that uses a trace constraint and positive semidefiniteness rather than the log-barrier to restrict Ω and Θ_c. That is, instead of

$$\begin{aligned} & -\log p(W, Z|\Omega, \Theta) \\ = & \sum_c \frac{1}{2} \operatorname{tr}(w_{c\cdot} - 1 \cdot z_c)^\top (w_{c\cdot} - 1 \cdot z_c)\Theta_c^{-1} + m_{cs} \log|\Theta_c| \\ & + \frac{1}{2} \operatorname{tr} Z^\top \Omega^{-1} Z + m_c \log|\Omega| + c \end{aligned} \tag{11}$$

for the hierarchical model and analogous setting for the attachment model, we use the following objectives $L(W, Z, \Omega, \Theta)$:

$$L_{\text{hier}}(W, Z, \Omega, \Theta) \tag{12}$$
$$= \sum_c \frac{1}{2} \operatorname{tr}(w_{c\cdot} - 1 \cdot z_c)^\top (w_{c\cdot} - 1 \cdot z_c) \Theta_c^{-1} + \frac{1}{2} \operatorname{tr} Z^\top \Omega^{-1} Z$$

subject to $\operatorname{tr} \Theta_c \succeq 0$ and $\operatorname{tr} \Theta_c = 1$ and $\operatorname{tr} \Omega \succeq 0$ and $\operatorname{tr} \Omega = 1$ for the hierarchical model. Moreover, for the attachment model:

$$L_{\text{attach}}(W, \Omega, \Theta) \tag{13}$$
$$= \frac{1}{2} \operatorname{tr} w_{c\cdot}^\top w_{c\cdot} \Theta_c^{-1} + \frac{1}{2} \sum_i w_{\cdot 1i} \Omega^{-1} w_{\cdot 1i}^\top$$

subject to the same constraints as for L_{hier}. The only real difference is that we eliminated z and instead, we attach the model to w_{c1} directly. In either case this detaches the subtasks from the problem of joint task inference.

3.1 Structured Sparsity

A second aspect of multi-task learning is to use structured sparsity [5] to select relevant variables for an entire block of terms jointly rather than eliminating terms for each task individually. This is achieved by adding a mixed norm on the parameters W and Z to the optimization problem. We need some more notation first: the p norm of a vector $x \in \mathbb{R}^d$:

$$\|x\|_p^p := \sum_i |x_i|^p \text{ for } p < \infty \text{ and } \|x\|_\infty := \max_i |x_i| .$$

Moreover, the mixed norm of a matrix X, where we apply sparsity row-wise, is defined for $p, q \geq 1$ via

$$\|X\|_{p,q} := \left\| \|X_{1\cdot}\|_p , \ldots \|X_{d\cdot}\|_p \right\|_q . \tag{14}$$

Of particular interest is the $\|X\|_{2,1}$ norm, which attempts to eliminate entire rows of X at a time. Finally, we use the abbreviation $\|X\|_1 := \|X\|_{1,1}$ to denote the sum over absolute values in X. This leads to the following sparsity penalties for the hierarchical and attachment models respectively:

$$S_{\text{hier}}(W, Z) = \lambda_1 \|Z\|_1 + \lambda_2 \|Z\|_{2,1} + \lambda_1 \|W\|_1 + \lambda_2 \sum_c \|W_c\|_{2,1} \tag{15}$$
$$S_{\text{attach}}(W) = \lambda_1 \|W\|_1 + \lambda_2 \|W\|_{2,1} + \lambda_2 \sum_c \|W_c\|_{2,1} \tag{16}$$

The coefficients λ_1 and λ_2 govern the trade-off between generic sparsity and group sparsity. That is, for λ_2 there will be no correlation in sparsity patterns beyond what is obtained from data. For $\lambda_1 = 0$ we can assume that whenever any given $W_{csi} \neq 0$ then also all related $W_{cs'i}$ will not vanish.

3.2 Optimization Problems

We conclude this section by stating the two optimization problems that we will solve subsequently. The key ingredients are a likelihood function $-\log p(Y|X, W)$ which depends on the specific problem to solve, a simplified multitask learning penalty as defined by $L_{\text{hier}}(W, Z, \Omega, \Theta)$ and $L_{\text{attach}}(W, \Omega, \Theta)$ respectively, and a sparsity penalty as in $S_{\text{hier}}(W, Z)$ and $S_{\text{attach}}(W)$. We have the following for the hierarchical multitask model:

$$\begin{aligned} \underset{W,Z,\Omega,\Theta}{\text{minimize}} \; & \sum_{csj} -\log p(y_{csj}|x_{csj}, w_{cs}) + \frac{1}{2} \operatorname{tr} Z^\top \Omega^{-1} Z \\ & + \sum_c \frac{1}{2} \operatorname{tr}(w_{c\cdot} - 1 \cdot z_c)^\top (w_{c\cdot} - 1 \cdot z_c) \Theta_c^{-1} \\ & + \lambda_1 \|Z\|_1 + \lambda_2 \|Z\|_{2,1} \qquad \text{(17a)} \\ & + \lambda_1 \|W\|_1 + \sum_c \lambda_2 \|W_c\|_{2,1} \end{aligned}$$

$$\text{subject to } \Omega, \Theta_c \succeq 0 \text{ and } \operatorname{tr} \Omega = \operatorname{tr} \Theta_c = 1. \tag{17b}$$

Moreover, the attachment multitask model yields:

$$\begin{aligned} \underset{W,\Omega,\Theta}{\text{minimize}} \; & \sum_{csj} -\log p(y_{csj}|x_{csj}, w_{cs}) \qquad \text{(18a)} \\ & + \frac{1}{2} \sum_c \operatorname{tr} w_{c\cdot}^\top w_{c\cdot} \Theta_c^{-1} + \frac{1}{2} \sum_i w_{\cdot 1i} \Omega^{-1} w_{\cdot 1i}^\top \\ & + \lambda_1 \|W\|_1 + + \lambda_2 \|W\|_{2,1} + \sum_c \lambda_2 \|W_c\|_{2,1} \end{aligned}$$

$$\text{subject to } \Omega, \Theta_c \succeq 0 \text{ and } \operatorname{tr} \Omega = \operatorname{tr} \Theta_c = 1. \tag{18b}$$

4. INFERENCE

The optimization problems (17) and (18) are jointly convex in (W, Z, Ω, Θ) and (W, Ω, Θ) respectively. For practical optimization we resort to a Gauss-Southwell [15] approach of minimizing blocks of parameters at a time. In practice this means that we alternate between minimizing with respect to W, Z and Ω, Θ. This is known to converge to the globally optimal solution (albeit slowly on occasion). Issues of problem distribution and parallelization will be discussed in the next section.

4.1 Covariance Updates

Assume that we are given W, Z. In this case we may find optimal values for the psd matrices Ω, Θ_c using the derivation in (8) as follows:

$$\Omega = (\operatorname{tr} O)^{-1} O \tag{19}$$
$$\text{where } O = (ZZ^\top)^{-\frac{1}{2}} \qquad \text{(hierarchical)}$$
$$O = \left[\sum_i w_{\cdot 1i} w_{\cdot 1i}^\top \right]^{-\frac{1}{2}} \qquad \text{(attachment)}$$

Likewise, for Θ_c we have the updates

$$\Theta_c = (\operatorname{tr} T)^{-1} T \tag{20}$$
$$\text{where } T = \left[\sum_i (w_{c\cdot i} - z_{ci})(w_{c\cdot i} - z_{ci})^\top \right]^{-\frac{1}{2}} \qquad \text{(hierarchical)}$$
$$T = \left[\sum_i w_{c\cdot i} w_{c\cdot i}^\top \right]^{-\frac{1}{2}} \qquad \text{(attachment)}$$

This means that we can compute T entirely with only access to all subtask specific parameters s for a given campaign c. Hence, as long as it is possible to have all such data available on a single machine, we need not communicate the lower level of the hierarchy outside the machine.

4.2 Optimization with Sparsity Penalty

Next we need to discuss update steps in terms of W and Z. Recall that we imposed a mixed norm penalty on both terms such that we obtain group sparsity. Our strategy borrows from [5], and [10, Proposition 1]. Recall the structure of the penalties imposed by $\|\cdot\|_{2,1}$ and $\|\cdot\|_1$. They constitute a *hierarchy* over nonzero terms in W and Z respectively — the $(2,1)$-norm attempts to zero out the entire set of contributions for a given coordinate and the 1-norm ensures that even if we use an attribute, we only use it sparingly.

Given Ω, Θ the remainder of the problem is a convex *unconstrained* optimization problem. One of the algorithms recently proposed are of a structure resembling FISTA (Fast Iterative Successive Thresholding) [6]. In it one interleaves a gradient descent step with regard to the convex differentiable part of the objective with a thresholding step with regard to the sparsity penalty. That is, for the problem:

$$\underset{a}{\text{minimize}}\, f(a) + \lambda\Omega[a] \tag{21}$$

one performs the following steps (after initializing a_0)

$$b_{t+1} := a_t - \eta_t \partial_a f(a_t) \text{ and} \tag{22}$$

$$a_{t+1} = \underset{a}{\text{argmin}}\, \frac{1}{2t_t}\|a - b_{t+1}\|^2 + \lambda\Omega[a] \tag{23}$$

Here (22) is essentially a gradient descent step in f. The step (23) is commonly referred to as a prox-operator. The step size t_i is chosen such that t_i^{-1} majorizes the Lipschitz constant of $\partial_a f(a)$.

We discuss computing gradients with respect to the objective in Section 4.3 for the hierarchical model (the attachment model follows similarly). In this context we mean by $\mathcal{F}[Z, W]$ either the first two lines of (17) or of (18) with Ω and Θ fixed at this point. For now note that in our case $\Omega[a]$ decomposes into penalties applied per task. That is

$$\Omega[W, Z] = \sum_c \lambda_1 \left[\|z_c\|_1 + \sum_s \|w_{cs}\|_1\right] + \lambda_2 \left[\|z_c\|_2 + \sum_s \|w_{cs}\|_2\right] \tag{24}$$

Solving (23) can be carried out for each task c and for each z_c and w_c individually, hence it is amenable to easy distribution. Using [10, Proposition 1] one can see that performing successive prox operations with respect to the ℓ_1 norm and subsequently with respect to the ℓ_2 norm lead to an exact solution of (23). For instance, for z_c this means that we perform the following steps

$$z_c \leftarrow z_{ci} - t_i \partial_{z_c}\mathcal{F}[Z, W] \tag{25}$$

$$z_{ci} \leftarrow \operatorname{sgn} z_{ci} \max(0, |z_{ci}| - t_i\lambda_1) \tag{26}$$

and subsequently we threshold the entire vector via

$$z_c \leftarrow \frac{z_c}{\|z_c\|_2}\max(0, \|z_c\| - t_i\lambda_2) \tag{27}$$

In other words, first we perform gradient descent. Then all coefficients that are individually too small are eliminated and the remainder is shrunk. Finally, we perform shrinkage of the remainder in the direction of their unit vector. The objective is either that of the hierarchical or of the attachment model. Updates with respect to W are entirely analogous and therefore omitted.

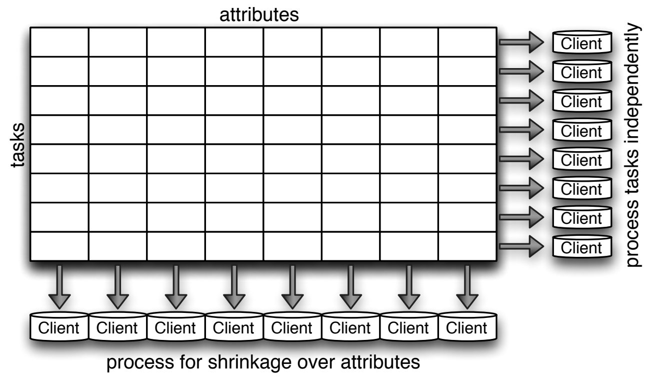

attributes
tasks
Client
process tasks independently
process for shrinkage over attributes

Figure 3: The parameter matrices W and Z are of size $\mathbb{R}^{d\times m}$. For gradient computation we need to have access to all parameters for a given task c on a given machine, hence the row-wise split. Subsequently, to perform shrinkage over attributes, we need all parameters pertaining to a feature (for all tasks) on a given machine. Load balancing for both of these tasks is achieved by consistent hashing.

4.3 Gradients

We complete our overview of optimization by discussing the gradient in terms of W and Z. As before, we limit ourselves to a discussion of the hierarchical model of (17). Since the reasoning required for (18) is essentially identical, we omit the details. Straightforward calculation yields

$$\partial_{z_{ci}}\mathcal{F}[Z, W] = \left[\Omega^{-1}Z_{\cdot i}\right]_c + \Theta_c^{-1}(1 \cdot z_{ci} - w_{c\cdot i}) \tag{28}$$

$$\partial_{w_{csi}}\mathcal{F}[Z, W] = \sum_j \partial_{w_{csi}} - \log p(y_{csj}|x_{csj}, w_{cs}) + \Theta_c^{-1}(w_{c\cdot i} - 1 \cdot z_{ci}) \tag{29}$$

As can be seen, again all gradients decompose in terms of tasks and subtasks respectively. We will exploit this for distributed optimization. The exact form of the gradient for $p(y_{csj}|x_{csj}, w_{cs})$ is straighforward to compute for both the regression and classification problem given the form of the conditional probability in (2) and (3) respectively.

5. DISTRIBUTED OPTIMIZATION

We now discuss how to implement a distributed optimization algorithm efficiently on a cluster of commodity workstations. As discussed previously, invoking steps (22) and (23), and updating Ω and Θ requires the following operations:

1. Compute partial subgradients of $\mathcal{F}[Z, W]$ for all campaigns with respect to Z and W.
2. Aggregate subgradients obtained from all instances and apply it to the model parameters.
3. Distribute coordinates (or subsets $S \in \mathcal{S}$ thereof) of the subgradients (or rather updated coordinates) to clients for application of the prox operator.
4. Invoke the prox operator.
5. Redistribute the results to the machines holding the campaign-specific data.

Since this is an iterative procedure with two barriers per iteration (send subgradients, return values) this sounds as if it were a good fit for MapReduce. Unfortunately, this approach suffers from inefficiencies inherent in the Hadoop implementation of MapReduce: context in the mappers is not preserved between iterations. Moreover, Hadoop communicates primarily via file I/O. Since the proposed algorithm can take tens of iterations and since we need to communicate parameters repeatedly, this means significant waste of resources by repeatedly having to initialize the state of the mappers. Hence we resort to a method discussed in [13, 2], namely to allocate the machines using Hadoop and then to establish an overlay communication network.

Algorithm 1 Distributed Optimization

```
1: for all $i = 1 \cdots p$ parallel do
2:     read data blocks from disk
3:     for all campaigns with $m(c) = i$ do
4:         compute subgradient $g_c$
5:     end for
6:     write $g_c$ to (key,value) store according to $m(i)$
7: end for
8: reach a barrier
9: for all $i = 1 \cdots p$ parallel do
10:    read $g_c$ from (key,value) store according to $m(i)$
11:    for all coordinates with $m(j) = i$ do
12:        solve the prox operator
13:    end for
14:    write to the (key,value) store according to $m(c)$
15:    Compute contribution to sufficient statistics of $\Omega$ and
       write it back to shared memory.
16: end for
17: reach a barrier
18: Read sufficient statistics of $\Omega$ and compute new value.
```

5.1 Data and Task Distribution

In the following we assume that we have p machines to process data. Recall that d denotes the number of attributes, i.e. $x_{csi} \in \mathbb{R}^d$ and that m denotes the number of campaigns. We use randomized load-balancing to determine which machine receives which portion of the data in both the data-bound and the parameter-bound part of the optimization procedure. This is achieved, e.g. by consistent hashing [11]:

$$m(c) = \operatorname*{argmin}_{m \in \mathcal{M}} h(m, c) \text{ and } m(i) = \operatorname*{argmin}_{m \in \mathcal{M}} h(m, i) \quad (30)$$

to assign machines from a machine pool $\mathcal{M}$ for campaigns c and coordinates i respectively.

Finally, data exchange is carried out in the form of a distributed (key,value) store. For reasons of practicality we used `memcached` as our reference implementation. This follows the design pattern of [21, 13] and it avoids file I/O for synchronization. Such a strategy is much more efficient than repeated invocations of Hadoop MapReduce.

5.2 Distributed Subgradient Oracle

By design, the subgradients of $\mathcal{F}[Z, W]$ and $\mathcal{F}[W]$ decompose into terms that are easily computable in a campaign-specific manner (terms related to the negative log-likelihood) and terms that are easily computable in a coordinate-specific manner (the penalties in terms of Ω, Θ and the sparsity penalties). Furthermore, only the former requires direct access to data, whereas the latter requires access to a given coordinate across all tasks. This means that we can compute gradients in two stages: a pass over data, as performed by the workers that have the data, a reshuffle of parameters, and a finalizing pass (plus prox step) in a coordinate-specific fashion.

Likelihood gradients: Since data is partitioned according to tasks c, subgradients with regard to w_{cs} are easily computed via

$$g_{cs} = \sum_{j=1}^{n_{cs}} \partial_{w_{cs}} - \log p(y_{csj} | x_{csj}, w_{cs}). \quad (31)$$

Next we compute gradients with respect to $\frac{1}{2} \sum_s w_{cs}^\top \Theta_c^{-1} w_{cs}$, i.e. we add $\Theta_c^{-1} w_{c\cdot}$ to g_{cs}. The analogous reasoning holds for $\sum_s (w_{cs} - z_c)^\top \Theta_c^{-1} (w_{cs} - z_c)$. These gradients are then redistributed according to Figure 3 such that all g_{csi} for a given coordinate i (ranging over all tasks and subtasks) are available on the same machine.

Multitask gradients: At this point we can compute coordinate specific parts as arising from the Ω-dependent terms on a per-coordinate basis. For this purpose we only need $Z_{\cdot i}$ or $w_{\cdot 1i}$, depending on whether we chose the hierarchical or attachment model respectively. We only need to read the weights corresponding to non-zero entries in Ω^{-1}.

5.3 Distributed Prox Operator and Covariance Estimation

The final step required is to solve the prox operator related to the $\|\cdot\|_1$ and $\|\cdot\|_{2,1}$ norms as these enforce sparsity. Whenever we have a fully hierarchical setting, Proposition 1 of [10] applies and we can simply perform prox steps bottom up in the process. Whenever this assumption is not satisfied, we may still iterate the prox operator to obtain a suboptimal solution. This suffices as a descent step, since optimization in Z and W is just a subroutine in the overall optimization scenario involving Θ and Ω. Note that the prox operator can be carried out in linear time — we only require computing norms of vectors and rescaling them.

The data exchange is completely analogous to the gradient computation, except that we now work on attributes rather than campaigns. After the prox operation we redistribute parameters back into a (key,value) storage. As before, this requires a barrier to ensure that up-to-date values are available on all workers for another pass through the data. Similar to the gradient computation phase, the read and write steps can be performed in parallel.

Finally we note that estimating Θ can be done locally in each worker however the sufficient statistics required to compute Ω (see 19) is distributed on a per-attribute basis. Thus we overlay this step with the prox-operator step. Each worker computes its contribution to the sufficient statistics using its assigned attributes. For example in the hierarchical model this reduces to computing a $C \times C$ matrix $\sum_{i \in m(i)} z_i z_i^\top$. After reaching a barrier, worker 0 then reads those partial sums and computes the new value for Z using (8) and then writes Ω back to a shared memory to be read by each worker for the next iteration. Alternatively each worker can read the sufficient statistics of Ω and compute the new value deterministically. Moreover, instead of using (8), we could use the graphical lasso estimation of [9] to get a sparse inverse covariance estimation of Ω^{-1} from its sufficient statistics. This sparse inverse covariance is desirable In distributed settings to minimize parameter movements when computing the multi-task gradient as it depends on the non-zero elements of the inverse covariance (i.e. Ω^{-1}).

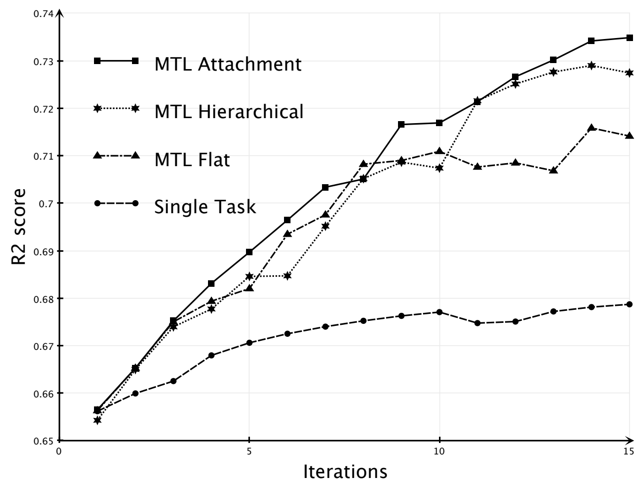


Figure 4: School Data: R^2 Performance for various multitask learning algorithms. Note the faster convergence and better performance of ATT-MTRL.

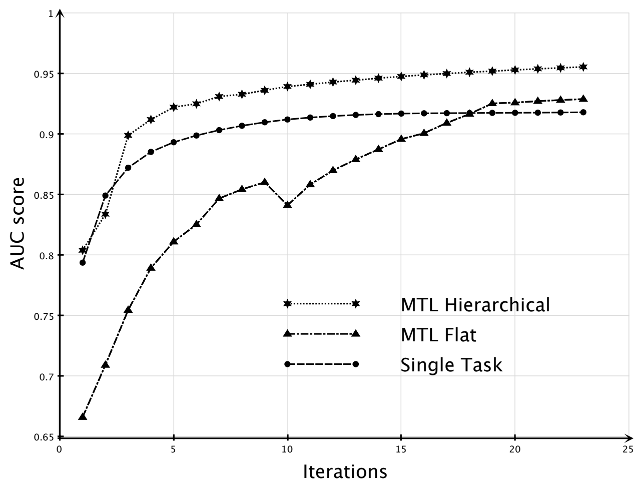


Figure 5: 20-newsgroup MTL: AUC Performance for various multitask learning algorithms. The hierarchical MTL algorithm starts off better than the flat algorithm and it consistently outperforms both single task and flat multitask learning.

6. EXPERIMENTS ON PUBLIC DATA

To establish the efficacy of our approach we report results on both two public datasets and one proprietary dataset. We make this choice since we are unable to share proprietary and financially relevant data outside Yahoo, yet at the same time we wish to provide the reader with some means of gaining insight into the working of the proposed algorithm. In other words, we show that the proposed algorithm improves on the state of the art and simultaneously that it scales to substantial problem sizes.

In terms of public datasets we choose two standard datasets: estimation of examination scores for students attending secondary schools in London and multi-task classification of the 20-newsgroup dataset. The algorithms we compare are the Hierarchical MTL (HIE-MTRL) and Attachment MTL (ATT-MTRL) algorithms, along with two baselines: the first one is the Single-task Learning (STL) algorithm which does not use multitask learning, and optimizes all the tasks independently. The second baseline is the Flat MTL (F-MTRL) algorithm of [23] that uses a matrix-variate normal prior on the task correlation matrix. This algorithm performs multitask learning, however it does not account for the hierarchical task and subtask structure. It "flattens" the task hierarchy and treats (task, subtask) as individual tasks and learns a joint covariance structure.

6.1 Student score estimation

This dataset has been used widely for studying multitask regression[1]. It consists of the exam scores of 15,362 students from 139 secondary schools in London during 1985, 1986 and 1987. Originally, the input consists of the year of the exam, four school-specific and three student-specific attributes. The goal is to estimate the exam scores for the students. Several papers [23, 4] evaluate multitask learning by treating the school-ids as attributes, but one could arguably treat this dataset as specifying a hierarchical task/subtask structure, where the school ID refer to tasks and the exam years for each school correspond to the subtasks of the school ID. Thus, there are a total of 139 tasks, with up to 3 subtasks for each task. We note that several tasks have only 2 tasks in the dataset.

We replace each categorical attribute with one binary variable for each possible attribute value as in [4] but remove the attributes corresponding to the exam years. As a result of this preprocessing, we have a total of 24 input attributes. We use a 66/34 split of the dataset to use as training instances and test instances, and report the average (over all years and schools) performance of our algorithms, on the test set. For our performance measure, we use the normalized inner product between the input score vector and the predicted score vector. This measure is proportional to the squared multiple correlation coefficient R^2, a normalized version of the regression error, defined as

$$R^2 := 1 - \frac{\sum_i (y_i - f_i)^2}{\text{Var}[y]}. \tag{32}$$

That is, it is the ratio between explained variance and total variance. See e.g. [8] for further details.

For F-MTRL, HIE-MTRL and ATT-MTRL we use 5-fold cross validation to determine the optimal value of the appropriate regularization constants and learning rate. Figure 6.1 plots the R^2 performance of the algorithms as optimization progresses. As can be seen in the graph, and as also reported by [23, 4], multitask learning provides a significant performance improvement. F-MTRL improves over the baseline by around 5%. However, by using the task-subtask hierarchy, our HIE-MTRL and ATT-MTRL obtain a further improvement from a score of around 0.71 (for F-MTRL) to almost 0.73. The performance for ATT-MTRL was slightly better than HIE-MTRL in this dataset.

6.2 Multi-Task Classification

The task at hand is multi-task classification of the 20-newsgroup dataset[2]. The goal here is to predict the newsgroup of a given post. The 20 news groups are arranged into a two-level hierarchy. The first level comprises 5 categories:

[1]http://www0.cs.ucl.ac.uk/staff/A.Argyriou/code

[2]http://qwone.com/ jason/20Newsgroups/

Table 1: 20-newsgroup analysis: AUC performance for single task and multitask algorithms on varying percentages of the data.

Fraction of data	STL	F-MTRL	HIE-MTRL
20%	0.755	0.775	**0.827**
40%	0.821	0.839	**0.881**
60%	0.875	0.893	**0.910**
80%	0.899	0.917	**0.932**
100%	0.918	0.931	**0.957**

politics, religion, recreational, science, and computers. Each category has 2-5 subcategories in the second level. We map categories to tasks and sub-categories to sub-tasks. The dataset comprises 18k documents and we followed the standard test/train split. We removed stop words and words appearing less than 10 times. Five-fold cross validation is used to determine the values of the regularization parameters for all models.

We measure classification accuracy using the AUC measure. Since in this dataset we do not have the notion of an anchor task, we only use the symmetric HIE-MTRL formulation. In Figure 5 we compare the performance of HIE-MTRL against the state of the art algorithm in [23] F-MTRL and against single-task baseline STL. As evident from figure HIE-MTRL improves over the F-MTRL baseline by around 2% points and the improvement was statistically significant. Moreover HIE-MTRL outperforms all other competing algorithms in terms of speed of convergence.

To see the effect of varying the training set size, we select different fractions of the data for each task to form the training set while keeping the test set fixed. As can be seen in Table 1 HIE-MTRL outperforms all other competing algorithms for a range of different sample sizes. Moreover, the improvement of HIE-MTRL over competing algorithms is more apparent when the training data size is small (a 5% improvement over the F-MTRL baseline using a 20% fraction of the training data).

7. OPTIMIZING DISPLAY ADVERTISING

7.1 Data

We collected 4 weeks of advertising data, i.e. impressions, clicks, and conversions, for a total of 1,468 advertising campaigns.[3] Each campaign is treated as a separate targeting task. 66% of the data is used for training, while the remaining 34% is used for scoring. The train/test split is performed using a reference time stamp (impressions before that time stamp used for training and afterwards impression for testing). Since the user profiles span 56 days of user history, each training/scoring example is preceded by at least 4 weeks of user events. This benchmark data set enables us to perform rigorous offline experiments. We count users based on the unique number of browser cookies (see table below).

days	users	features	campaigns	dataset size
56	10^9	934,000	630	1.4TB

We study the performance of our techniques compared to the baseline system developed in [3]. We mainly compare modeling performance in terms of the area under the ROC curve (AUC). Unless otherwise specified, all metrics are measured as conversion-weighted average of AUC across all campaigns in the benchmark set. We denote the conversion-weighted average of AUC as *Weighted AUC*.

[3]We note here that data from users that opted out of behavioral targeting were not collected.

We represent each user using features from both *active* and *passive* observations. Passive observations include viewing ads and visiting pages in which an action is not specifically required upon seeing the page. Active observations include issuing search queries and clicking ads in which users actually perform an action on the page.

Each advertising campaign has three subtasks:

- **Predicting conversions:** This sub task contains data that shows whether users converted on a given campaign. That is, it contains information whether they performed an advertiser-specified action such as purchasing a product or filling a form.
- **Predicting clicks:** This subtask contains data that shows whether users clicked on the ad of this campaign or not.
- **Prediction on auxiliary (unattributed conversion) data:** This subtask contains data that shows whether users converted on historic data on related campaigns of the same advertiser. This data is supplied by the advertiser.

We define the feature weight for a given user-(sub)campaign example to be the number of days (before showing the user the campaign ad) in which the feature appears. Our platform experiences a large variance of feature weights across our feature types thus making it hard to set a single count-threshold below which we consider the feature to be irrelevant. We thus rely on the learning algorithm to perform joint conversion optimization and feature selection.

7.2 Results

All experiments reported in this section were performed using 300 machines. We assess the performance both in terms of AUC accuracy and scalability of the algorithms. The attachment multitask learning algorithm (ATT-MTRL) significantly outperforms flat multitask and single task learning. Moreover, the results for hierarchical multitask learning (HIE-MTRL) were only slightly inferior to ATT-MTRL (thus we omit them for space limitations). This finding is consistent with our findings of Section 6.1. Note that ATT-MTRL also performs multitask feature selection, which is essential here due to the large feature space. We compare our performance with the baseline Single-task Learning, which optimizes for all the tasks and subtasks separately. We omit comparing with the F-MTRL for this task since the flat MTL requires flattening the task-subtask structure (2k tasks) which results in massive weight vector movements across machines and as such does not scale to this dataset (though in Table 4 we show the effect of introducing task and subtask covariance on the overall performance).

The parameters for all models were tuned on a validation set. In Table 2 we report the overall performance of the model against the baseline. As we can see, our model clearly outperforms the baseline. All improvements of our models over the baselines are statistically significant. Note that the task of conversion prediction is very difficult since positive examples are very rare.

Secondly, we quantify the effect of feature selection. For this purpose we select the top 10k, 30k and 50k features (using mutual-information measure) and use them in the STL. For comparison we run ATT-MTRL using conservative

Table 2: Attachment multitask performance.

AUC	STL	ATT-MTRL
all subtasks	0.658	**0.674**
conversions	0.629	**0.653**
auxiliary (unattributed)	0.677	**0.714**
clicks	0.662	**0.671**

Table 3: Feature selection effectiveness:

	Conversion AUC	features
STL + ℓ_2 + top features	0.606	10,000
STL + ℓ_2 + top features	0.609	30,000
STL + ℓ_2 + top features	0.607	50,000
ATT-MTRL (aggressive)	0.631	3,992
ATT-MTRL (conservative)	**0.653**	17,789

Table 4: Ablation study for ATT-MTRL.

AUC	conversions	all sub-tasks
L1	0.621	0.642
L1+L12	0.629	0.658
L1+L12+Θ	0.641	0.663
L1+L12+Θ+Ω	**0.653**	**0.674**

($\lambda_1 = 0.4, \lambda_2 = 10$) and aggressive ($\lambda_1 = 0.4, \lambda_2 = 25$) feature selection parameters. The results in Table 3 are reported in terms of the weighted average AUC measure.

Finally, in table 4 shows the contributions of the various components of the ATT-MTRL algorithm towards the learning performance. Ω refers to the task-correlation regularization, Θ refers to the sub-task correlation regularization, and $L1$ and $L12$ refer to per-campaign and multitask feature selection respectively. As seen from the figure, using multitask feature selection($L12$) leads to only a marginal improvement over single task feature selection($L1$). However, adding the multitask learning components lead to a significant improvement over using just $L1$ and $L12$ regularization. These results clearly show the importance of leveraging cross-campaign and cross-campaign-subtask information to improve the performance of campaigns with very few conversions, as opposed to the baseline techniques.

8. CONCLUSION

In this paper we addressed the problem of hierarchical multitask learning when tasks are organized in a hierarchy. We presented two convex formulations to this problem and showed that models that exploit the hierarchical structure outperformed flat models. Furthermore, we showed how to scale our models to a tera-scale advertising task. An advantage of our hierarchical formulation is the utilization of the task substructure for efficient parameter distribution that reduces parameter movements across machines. We validated our models on both public and private datasets with favorable performance.

9. REFERENCES

[1] A. Ahmed, M. Aly, A. Das, A. Smola, and T. Anastasakos. Web-scale multi-task feature selection for behavioral targeting. In *CIKM*, 2012.

[2] A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A. Smola. Scalable inference in latent variable models. In *Web Science and Data Mining (WSDM)*, 2012.

[3] M. Aly, A. Hatch, V. Josifovski, and V. K. Narayanan. Web-scale user modeling for targeting. In *WWW*, 2012.

[4] A. Argyriou, T. Evgeniou, and M. Pontil. Convex multi-task feature learning. *Machine Learning*, 73(3):243–272, 2008.

[5] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Optimization with sparsity-inducing penalties. *Foundations and Trends in Machine Learning*, 4(1):1–106, 2012.

[6] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[7] R. Caruana. Multitask learning. *Machine Learning*, 28:41–75, 1997.

[8] N. R. Draper and H. Smith. *Applied Regression Analysis*. John Wiley and Sons, New York, NY, 1981.

[9] J. Friedman, T. Hastie, and R. Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008.

[10] R. Jenatton, J. Mairal, G. Obozinski, and F. Bach. Proximal methods for hierarchical sparse coding. *Journal of Machine Learning Research*, 12:2297–2334, 2011.

[11] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Symposium on the Theory of Computing STOC*, pages 654–663, New York, May 1997. Association for Computing Machinery.

[12] T. R. Shultz and F. Rivest. Using knowledge to speed learning: A comparison knowledge-based cascade-correlation and multi-task learning. In *Proc. Intl. Conf. Machine Learning*, pages 871–878. Morgan Kaufmann, San Francisco, CA, 2000.

[13] A. J. Smola and S. Narayanamurthy. An architecture for parallel topic models. In *Very Large Databases (VLDB)*, 2010.

[14] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *Journal of Machine Learning Research*, 7:1531–1565, 2006.

[15] W. H. Southwell. Fitting data to nonlinear functions with uncertainties in all measurement variables. *Comput. J.*, 19(1):69–73, 1976.

[16] N. Srebro and A. Shraibman. Rank, trace-norm and max-norm. In P. Auer and R. Meir, editors, *Proc. Annual Conf. Computational Learning Theory*, number 3559 in Lecture Notes in Artificial Intelligence, pages 545–560. Springer-Verlag, June 2005.

[17] C. Teo, Q. Le, A. J. Smola, and S. V. N. Vishwanathan. A scalable modular convex solver for regularized risk minimization. In *Proc. ACM Conf. Knowledge Discovery and Data Mining (KDD)*. ACM, 2007.

[18] S. Thrun and J. O'Sullivan. Discovering structure in multiple learning tasks: the TC algorithm. In *Proc. Intl. Conf. Machine Learning*, pages 489–497. Morgan Kaufmann, 1996.

[19] M. Varma and B. R. Babu. More generality in efficient multiple kernel learning. In A. P. Danyluk, L. Bottou, and M. L. Littman, editors, *ICML*, volume 382 of *ACM International Conference Proceeding Series*, page 134. ACM, 2009.

[20] K. Weinberger, A. Dasgupta, J. Attenberg, J. Langford, and A. J. Smola. Feature hashing for large scale multitask learning. In L. Bottou and M. Littman, editors, *International Conference on Machine Learning*, 2009.

[21] J. Ye, J. Chow, J. Chen, and Z. Zheng. Stochastic gradient boosted distributed decision trees. In *CIKM*. ACM, 2009.

[22] K. Yu, V. Tresp, and A. Schwaighofer. Learning gaussian processes from multiple tasks. In *Proceedings of the 22nd International Conference on Machine Learning*, volume 119, pages 1012–1019. ACM, 2005.

[23] Y. Zhang and D.-Y. Yeung. A convex formulation for learning task relationships in multi-task learning. In *Uncertainty in Artificial Intelligence*, 2010.