# Web-Scale Multi-Task Feature Selection for Behavioral Targeting

Amr Ahmed[2], Mohamed Aly[1], Abhimanyu Das[1],
Alexander J. Smola[2], Tasos Anastasakos[1] *
[1]Yahoo! Research, 4301 Great America Parkway, Santa Clara, CA 95054
[2]Google Research, 1600 Amphitheatre Parkway, Mountain View, CA 94043
amahmed@cs.cmu.edu, {aly,abhidas,tasos}@yahoo-inc.com, alex@smola.org

## ABSTRACT

A typical behavioral targeting system optimizing purchase activities, called *conversions*, faces two main challenges: the web-scale amounts of user histories to process on a daily basis, and the relative sparsity of conversions. In this paper, we try to address these challenges through feature selection. We formulate a multi-task (or group) feature-selection problem among a set of related tasks (sharing a common set of features), namely advertising campaigns. We apply a group-sparse penalty consisting of a combination of an $\ell_1$ and $\ell_2$ penalty and an associated fast optimization algorithm for distributed parameter estimation. Our algorithm relies on a variant of the well known Fast Iterative Thresholding Algorithm (FISTA), a closed-form solution for mixed norm programming and a distributed subgradient oracle. To efficiently handle web-scale user histories, we present a distributed inference algorithm for the problem that scales to billions of instances and millions of attributes. We show the superiority of our algorithm in terms of both sparsity and ROC performance over baseline feature selection methods (both single-task $L1$-regularization and multi-task mutual-information gain).

## Categories and Subject Descriptors

G.3 [**Probability And Statistics**]: Statistical Computing; I.2.6 [**ARTIFICIAL INTELLIGENCE**]: Learning

## Keywords

Sparsity, Feature Selection, Large-scale Learning, Behavioral Targeting

*The first three authors had equal contribution toward this work. [2] Contributed to this work while affiliated with Yahoo! Research.


*CIKM'12,* October 29–November 2, 2012, Maui, HI, USA.


## 1. INTRODUCTION

A typical behavioral targeting platform optimizing for conversions, e.g. [10, 3], faces two core issues:1) the large volumes of user histories to be processed in a periodic fashion and 2) sparseness of conversions. 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. Conversion rarity is another issue further complicating the task of the platform as it requires parsimoniously mining the user historical online behavior. Though Yahoo!'s novel platform presented in [3] develops a variety of engineering and machine learning techniques to cope with these two issues, it is never enough. While an increased number of features is often beneficial, using the maximum number of features possible may not amount to the best engineering decision, especially because of consequently increased computational costs. An additional pain point for our platform is to optimize campaigns with very low conversion volumes (tens or hundreds). For these campaigns, conversion rarity represents a major bottleneck against achieving tangible advertising improvements.

In this paper, we resort to feature selection as a powerful technique to cope with both problems. There is ample literature covering this subject both from an algorithmic [5] and a statistical [6] aspect. Feature selection clearly copes with the large volumes of user histories to be processed. To cope with conversion sparseness problem we consider the *union* of all (possibly sparse) attribute sets of all advertisers to decide which ad to choose (e.g. by means of a generalized second price auction).

Our problem is a variant of the multi-task learning problem with the key difference that we are not necessarily interested in the best function space but rather in the minimal subset of attributes required for good joint performance. We refer to this problem as multi-task feature selection. The key tool we use in this context is that of convex optimization for sparsity. To summarize, our contributions are the following:

- We formulate feature selection in behavioral targeting as a large-scale sparse mixed-norm optimization problem and show that the recently proposed FISTA algorithm for non-smooth convex optimization can be applied to this setting.
- We design an efficient distributed implementation that scales to terascale-sized data sets comprising billions of instances and millions of features.
- Using a real-world web-scale display advertising tar-

geting data set, we show the ability of our algorithm to beat baseline with both per-campaign and cross campaign standard feature selection techniques in terms of modeling performance (using the ROC measure) while achieving a reduction of one to two orders of magnitude in actually used behavioral features.

## 2. PROBLEM FORMULATION

Our behavioral targeting problem can be formalized as follows. To capture the interaction between covariates $x$, campaigns $c$ and associated labels $y$ (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 assume that the input space of the covariates is $\mathbb{R}^d$, that is, each covariate $x_i^c$ for instance $i$ of campaign $c$ lies in a $d$-dimensional feature space. We also let $n$ denote the total number of campaigns.

Formally we consider sets of covariates and labels indexed by a campaign $c$, as denoted by

$$X^c = \{x_1^c, \ldots, x_{m^c}^c\} \subseteq \mathcal{X} \text{ and } Y^c = \{y_1^c, \ldots, y_{m^c}^c\} \subseteq \mathcal{Y}$$

Here each $(x_i^c, y_i^c)$ is drawn from some distribution $p(x,y|c)$ of covariates and labels respectively. Typically we choose $\mathcal{Y} = \{\pm 1\}$, in particular when dealing with a simple classification problem. The prediction problem can be expressed either as one of risk minimization (we want to come up with 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 a parameter $w = \{w^1, w^2, \ldots w^c\}$ such that

$$p(Y|X,w) = \prod_c p(Y^c|X^c,w^c) = \prod_c \prod_{i=1}^{m^c} p(y_i^c|x_i^c,w^c) \quad (1)$$

is large. Here, we can also view $w$ as a $n \times d$ matrix such that each $w^c$ is a $d$-dimensional row vector.

For the purpose of the present paper we pick a rather specific form of $p(Y^c|X^c,w^c)$, namely a log-linear model where

$$-\log p(Y^c|X^c,w^c) = \log\left(1 + e^{-Y^c\langle w^c, X^c\rangle}\right). \quad (2)$$

Rather than minimizing the negative log-likelihood, we cast the problem of finding $w$ as risk minimization. Here we minimize a penalized version of the aggregate risk

$$R[w] = \sum_i R^c[w^c] \quad (3)$$

$$\text{where } R^c[w^c] = \frac{1}{m^c}\sum_{i=1}^{m^c} l(x_i^c, y_i^c, w^c). \quad (4)$$

Here the function $l(x,y,w)$ is a loss function, which is typically convex in $w$. Note that by choosing the logistic loss $l(x,y,w) = \log\left(1 + e^{-y\langle w,x\rangle}\right)$ (which is what we use in our experiments), logistic likelihood maximization and risk minimization can be treated in the same framework.

The specific choice of a regularizer is the subject of Section 3. For the moment we abstractly define it as $\Omega[w]$. This means that the problem of Maximum a Posteriori (MAP) estimation for $w$ can be cast as convex minimization:

$$\underset{w}{\text{minimize}} \sum_c R^c[w^c] + \lambda\Omega[w]. \quad (5)$$

## 3. GROUP-SPARSE FEATURE SELECTION

Recall that we denote by $R^c[w]$ the empirical risk terms incurred by the individual campaigns $c$ and by $R[w]$ the aggregate risk as defined in (4). For the purpose of fast estimation we want to ensure that only a small number of attributes are used in *any* of the campaigns — after all, if we were to use a feature in even just one campaign we would need to compute it beforehand regardless. This is achieved by combining a penalty per entry $w_{ij}$ of the parameter matrix, with one per group of parameters for each feature (i.e, per column $\|[w]_j\|$). We thus obtain the problem of minimizing

$$L[w] := R[w] + \lambda_1 \|w\|_1 + \lambda_2 \sum_i \|[w]_i\|_2 . \quad (6)$$

The last term of the above expression also corresponds to the $l_{1,2}$ mixed-norm (see e.g. [9]) of the parameter matrix $w$. It is easy to see that $L[w]$ enhances sparsity of the solution. For instance, provided that at optimality any subgradient $|\partial_{w_{ic}} R[w]| \leq \lambda_1$ it follows immediately that $w_{ic} = 0$. Furthermore, if $\|\partial_{w_i} R[w]\| \leq \lambda_2$ then likewise the entire coefficient vector satisfies $[w]_i = 0$.

To solve the associated optimization problem we employ the Fast Iterative Shrinkage Algorithm (FISTA) of [5]. At its heart lies a proximal operator $\rho(c|\lambda)$ (see e.g. [7]), which solves a simplified version of the penalized risk minimization problem studied above, and a subgradient computation which we distribute for the benefit of scalability. For the sake of completeness we state the problem solved by the proximal operator explicitly (it is easier to derive it from scratch rather than specializing [4]).

**Lemma 1** *The mixed norm optimization problem* $\rho(c|\lambda)$

$$\underset{x}{\text{argmin}} \frac{1}{2}\|x-c\|_2^2 + \lambda_1 \|x\|_1 + \lambda_2 \sum_i \|[x]_i\|_2$$

*can be found by* $x = \rho(c|\lambda)$ *in the following algorithm*

$$x_{ij} \leftarrow \tau(c_{ij}, \lambda_1) \qquad \text{for all } i,j \quad (7)$$

$$x_i \leftarrow \tau(x_i, \lambda_2) \qquad \text{for all } i \quad (8)$$

$$(9)$$

*Here* $\tau(x,\lambda) = x\max(0, 1 - \|x\|_2^{-1}\lambda)$ *is a shrinkage map.*

PROOF. Taking subgradients of the various norms in the optimization problem we see that $\partial_x \|x\|_2 = \|x\|_2^{-1} x$ if $x \neq 0$ and $\partial_x \|x\|_2 = B_1$ otherwise (i.e. it is an element of the unit ball). Hence it follows that $x$ can be written as a sum of subgradients

$$x \in c - \left[\lambda_1 \partial_x \|x\|_1 + \lambda_2 \partial_x \sum_i \|[x]_i\|_2 + \lambda_3 \partial_x \|x\|_2\right]. \quad (10)$$

That is, $x$ satisfies the first order optimality conditions and we established the result. □

An analogous shrinkage result holds whenever we minimize $R[w]$ rather than $\frac{1}{2}\|x-c\|_2^2$. Moreover, the steps outlined above constitute the inner loop of the fast iterated shrinkage algorithm (FISTA) of [5].

### 3.1 Optimization Using the FISTA Algorithm

We now adapt FISTA to the problem at hand. The basic strategy is described in Algorithm 1. That is, the algorithm first computes the subgradient $g := \partial_w R[w]$. Subsequently

the gradient and the current (and past) parameter vector are used in the prox operator to obtain a new iterate of the parameter vector.

**Algorithm 1** Optimization Using the FISTA Algorithm.

**input:** Lipschitz constant $L$ for $R[w]$.
Set $z_0 = w_1 = 0 \in \mathbb{R}^{|C| \times d}$ and $t_1 = 1$
**for** $i = 1$ **to** $N$ **do**

$$g \leftarrow \partial_{w_i} R[w_i] \tag{11a}$$

$$z_i \leftarrow \rho((w_i - \frac{g}{L})|\frac{\lambda}{L}) \tag{11b}$$

$$t_{i+1} \leftarrow 0.5\left[1 + \sqrt{1 + 4t_i^2}\right] \tag{11c}$$

$$w_{i+1} \leftarrow z_i + \frac{t_i - 1}{t_{i+1}}(z_i - z_{i+1}) \tag{11d}$$

**end for**

In order to invoke Algorithm 1 we need a number of components: firstly we need a Lipschitz bound $L$ on $R[w]$. In our experiments, we use the logistic loss, for which it is known that the corresponding Lipschitz constant is 1. If the instances are bounded in terms of their norm by some $r \geq \|x\|$ we can easily obtain an (admittedly) conservative bound of $L \leq rm$. Whenever $R$ is already normalized in terms of the sample size this reduces to $L \leq r$. [5, Lemma 4.3] show that FISTA converges at rate $O(Lk^{-2})$ where $k$ is the number of iterations. Moreover, whenever this estimate of $L$ is too conservative we may use an alternative step size adaptive variant of FISTA.

In terms of practical constraints — we need to be able to compute $g$ efficiently in a distributed fashion and to distribute parts of $g$ to different machines in order to invoke the prox operator $\rho$. These are the computationally expensive parts of the algorithm. Fortunately $\rho((w_i - \frac{g}{L})|\frac{\lambda}{L})$ decomposes into individual attributes , each of which can be computed individually. Likewise, computing $g$ can be easily parallelized over different campaigns, each of which have their own parameter vector $w^c$. We discuss both parts in further detail in section 4.

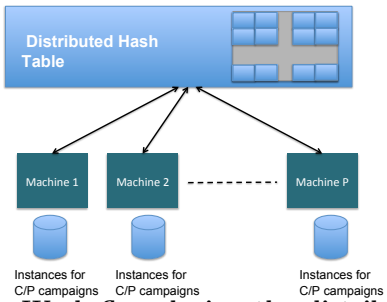


**Figure 1: Work flow during the distributed optimization algorithm. The weight matrix is broken into a set of PxP blocks and stored in a distributed hash table. The instances are partitioned among the machines. Each machine is assigned all instances that belong to C/P campaigns, where C is the number of campaigns and P are the number of machines.**

# 4. DISTRIBUTED OPTIMIZATION

Invoking Algorithm 1 involves the following main steps:

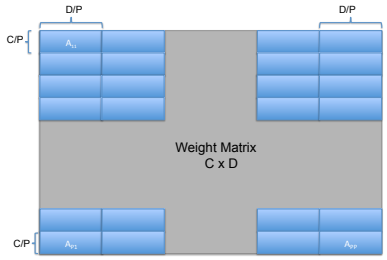


**Figure 2: Storage of the weight matrix. C is the number of campaigns, D is the number of features, and P is the number of machines. The weight matrix is divided into PxP blocks ($A_{1:P,1:P}$) each of which spans C/P campaigns and D/P features.**

1. Compute partial subgradients of $R[w]$ for all campaigns
2. Aggregate subgradients obtained from all instances.
3. Distribute coordinates (or subsets $S \in \mathcal{S}$ thereof) of the subgradients to different machines for application of the prox operator.
4. Invoke the prox operator and redistribute the results.

Naively this problem suggests the use of a MapReduce computation on Hadoop. Unfortunately, the latter is not well suited to computations that repeatedly use the same data as Hadoop mainly communicates via files which is undesirable since disk I/O overhead becomes comparable to the time to compute the results. Hence we resort to a method discussed in [1, 11], namely to allocate the machines using Hadoop and then the establish an overlay communication network.

Let $P$ be the number of machines, $D$ be the number of attributes, and $C$ be the number of campaigns. Thus the weight matrix to be learnt is of size $C \times D$. We first distribute the instances across machines, where each machine is assigned the instances (examples) of a set of $C/P$ campaings.These allocations do not change as the algorithm proceeds, thus minimizing data movement. As shown in Figure 2, the weight matrix is divided into $P \times P$ blocks each of which spans $C/P$ campaigns and $D/P$ attributes.This weight matrix divided into blocks can be stored across the Hadoop file system and these blocks can be exchanged between machines via file disk I/O operation. However, instead of writing these to disk and synchronizing processes via file I/O we synchronize them by storing the blocks in `memcached`. The latter, when used in storage rather than caching mode, allows us to spread the data evenly and to retain it in memory since it uses consistent hashing [8].

## 4.1 Distributed Subgradient Oracle

Computing subgradients of $R[w]$ requires the following operation

$$g = \sum_c \frac{1}{m^c} \sum_{i=1}^{m^c} \partial_w l(x_i^c, y_i^c, w^c) \tag{12}$$

$$= \Big(\dots, \underbrace{\frac{1}{m^c} \sum_{i=1}^{m^c} \partial_{w^c} l(x_i^c, y_i^c, w^c)}_{:=g^c}, \dots\Big). \tag{13}$$

In other words, the subgradient trivially decomposes into $\partial_{w^c} R^c[w^c]$. Hence it is advantageous to aggregate data from individual campaigns on each machine. This way most coordinates of the subgradient will vanish on each machine and only the composite $g$ be largely nonzero. Consequently we

only need to transmit smaller $g^c$ between the machines holding individual blocks. More specifically, each machine, say $i$, computes the gradient of the $C/P$ campaigns assigned to it. To do that, the machine first retrieves the current weight vector from the distributed hash table. This amounts to reading blocks $A_{i,1:P}$ from the distributed hash table. As evidence, this step can be performed in parallel across all machines. Once the machine obtains the current weight vector pertaining to its allocated campaigns, it does a single pass over the examples stored locally to compute the gradient and update the weight vector corresponding to its assigned campaigns. The next step now for machine $i$ is to write back the new weight vector to the distributed hash table. To do so, it first breaks its assigned weight into $P$ blocks and write them back to the hash table. Since each machine $i$ reads blocks $A_{i,1:P}$ and writes back blocks $A_{i,1:P}$, it is clear that these two steps can be executed in parallel across all the machines. After this step, the machine reach a barrier before proceeding to the next step. Similar to [2], we implemented a reverse-sense barrier algorithm that scales logarithmically with the number of machines.

## 4.2 Distributed Prox Operator

The final step required in addressing the distributed optimization problem of minimizing $L[w]$ is to solve the prox operator $\rho$. It decomposes along the set of attributes. Hence we simply need to solve each of the arising problems according to the algorithm described in Lemma 1 separately. Note that the latter can be done 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. Each machine, say machine $i$, reads blocks $A_{1:P,i}$ and solves the prox operator for a set of $D/P$ attributes. Once performed, it then breaks each of these attributes vectors (each of which is of dimension $C$) back into a set of $P$ blocks and writes them back to the distributed hash table. Similar to the gradient computation phase, this read and write steps can be performed in parallel, and followed by a barrier to ensure consistency of the weight matrix before moving to the next iteration.

## 4.3 Communication Complexity

As a consequence the (key,value) pairs are distributed uniformly over the $p$ machines involved. Hence the data exchange between any pair of machines is $O(1/p^2)$ — a fixed amount of data is distributed between $p$ machines. Each $O(1/p)$ sized block is distributed over $p$ machines. As long as all machines are located on the same switch this exhibits perfect scalability, i.e. it takes $O(1/p)$ time to process and synchronize. Finally, we obtain the following theorem:

**Theorem 2** *In order to process $m$ instances with $D$ dimensions from $C$ campaigns on $P$ processors we need $O\left(\frac{mD+\eta CD}{P}\right)$ time. Here $\eta$ is the ratio between network transfer and data processing speed.*

PROOF. To process the gradients associated with $mD$ data on $P$ machines costs $O(mD/P)$ time. Exchanging chunks costs $O(\eta CD/P)$ time. Evaluating the prox operator costs $O(CD/P)$ time. Since we may safely assume that the network is the limiting factor relative to processing the latter does not matter. We ignore log factors associated with network congestion and collision. □

**Algorithm 2** Distributed Optimization

**for all** $i = 1 \cdots P$ do **parallel do**
  Read blocks $A_{i,1:P}$
  **for all** $c = (i-1) * \frac{C}{P} + 1 \cdots i * \frac{C}{P}$ **do**
    compute subgradient $g_c := \partial_{w_c} R[w]$.
  **end for**
  Write back to the hash table blocks $A_{i,1:P}$.
**end for**
Reach a barrier.
**for all** $i = 1 \cdots P$ do **parallel do**
  Read blocks $A_{1:P,i}$
  **for all** $d = (i-1) * \frac{D}{P} + 1 \cdots i * \frac{D}{P}$ **do**
    solve the prox operator
  **end for**
  Write back to the hash table blocks $A_{1:P,i}$.
**end for**
Reach a barrier.

| Days | Users | Features | Campaigns | Size |
|---|---|---|---|---|
| 56 | 5.2B | 834K | 1468 | 2.143TB |

**Table 1: Basic statistics of the data used.**

# 5. EXPERIMENTS

To model the performance of our algorithm, we build targeting models for display advertising campaigns based on the targeting system for conversion-optimization presented in [3]. Table 1 presents some statistics about our data set.

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.

To assess the performance of our proposed sparse multi-task feature selection technique (denoted as MTFS), we start by applying standard feature selection baseline techniques to our data set. The first baseline that we use to compare our performance is a per-campaign L1-regularized learning model. In addition, we also use as a second baseline, a cross-campaign greedy information gain heuristic. More specifically, the set of baseline feature selection techniques that we compare against, are:

- Independent Per-Campaign $\ell_1$ Feature Selection: In

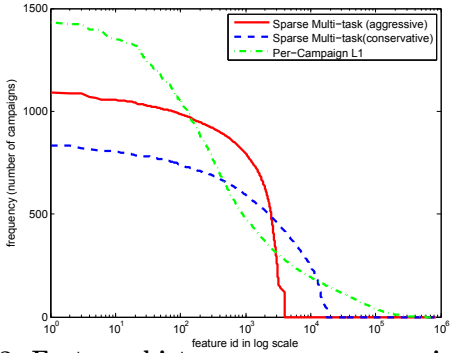


**Figure 3: Features histogram across campaigns. The X-axis represents the rank index of features under each method and the Y-axis gives the number of cmapigns having this feature in their model when trained using this method.**

this baseline, we apply logistic regression with $\ell_1$ regularization to each campaign separately.

- Global Cross-Campaign Mutual Information (MI) Gain: In this baseline, we compute the mutual information between features and labels on a per-campaign basis. Then, for each feature, we average all per-campaign mutual information values to compute the *average global mutual information* value across all campaigns. We use this latter to sort features cross campaign and take the top-k list. We apply logistic regression using $\ell_2$ regularization. (*Note that performing* $\ell_1$ here didn't enhance the performance, since the number of features are already small after the filtering step based on MI).

All Parameters for all models were tuned over a validation set. For our model we report two runs with conservative ($\lambda_1 = 0.4, \lambda_2 = 10$)and aggressive ($\lambda_1 = 0.4, \lambda_2 = 25$ )feature selections to give more insight about its mechanics. All results are reported in terms of the weighted average ROC measure * 100 across all campaigns.

As can be seen from Table 3, our MTFS group feature selection models for both conservative and aggressive sparsity levels, outperform the ROCs for not only the Cross-Campaign Information Gain Models, but also the Per-Campaign Information L1 Feature Selection Model.

In Figure 3, we show the histogram of feature counts in the trained model of different techniques. As evident, MTFS significantly reduces the total number of features which results in compact models that can be loaded in memory as well as trained more frequently and thus reduces the latency to predict user conversions It is quite striking that performing feature selection over each campaign independently results in 500K global features yet performs poorly compared to the merely 17K global features discovered by our joint training algorithm. This is largely due to the heavy tail distribution that occurs when features are not selected jointly (see Figure 3). Moreover, [3] showed that merely cutting features based on their frequency in the trained models, did NOT improve the accuracy of those models.

Table 2 shows the performance comparison of the different techniques for campaigns with less than 100 and 500 conversions, respectively. The numbers clearly show the superiority of our algorithm in leveraging cross-campaign information to improve the performance of campaigns with very few conversions, as opposed to the baseline techniques.

Finally, concerning the running time, the algorithm converges after only a few iterations ($< 10$) and the time consumed by the I/O was negligible to the time consumed by the gradient computation step. Finally, it should be noted that the gradient computation step just requires a single pass over the data, which is comparable to the amount of time required to train each model separately.

| Method | C<100 | C<500 |
|---|---|---|
| Global-MI: top 50K + L2 | 53.2 | 53.8 |
| Per-Campaign L1 | 50 | 50.1 |
| MTFS (conservative) | **60.3** | **62.7** |
| MTFS (aggressive) | **57.3** | **61.3** |

**Table 2: Modeling performance (ROC) comparison for campaigns with few conversions.**

| Method | ROC | Size |
|---|---|---|
| Global-MI: top 10K + L2 | 56.6 | 10,000 |
| Global-MI: top 30K + L2 | 56.9 | 30,000 |
| Global-MI: top 50K + L2 | 56.7 | 50,000 |
| Per-Campaign L1 | 54.5 | 588,975 |
| MTFS (conservative: $\lambda_1 = 0.4, \lambda_2 = 10$) | **61.1** | 17,789 |
| MTFS (aggressive: $\lambda_1 = 0.4, \lambda_2 = 25$) | **59.3** | 3,992 |

**Table 3: Modeling performance and resulting feature set sizes for the different feature selection techniques.**

## 6. CONCLUSION AND FUTURE WORK

In this paper we addressed the problem of multi-task feature selection for behavioral targeting across different advertising campaigns. We formulated the problem as a mixed-norm optimization problem and devised a novel and efficient distributed inference algorithm that scales to billions of instances and millions of attributes. We empirically showed that our algorithm not only reduces the number of overall features needed for classifications but also improves performance, especially for campaigns with sparse conversions, over models that perform individual task-level optimization. In future work, we intend to model the effect of hierarchical sparsity constraints for groups of features.

## 7. REFERENCES


[1] Amr Ahmed, Mohamed Aly, Joseph Gonzalez, Shravan Narayanamurthy, and Alexander J. Smola. Scalable inference in latent variable models. In *WSDM*, 2012.

[2] Amr Ahmed, Yucheng Low, Mohamed Aly, Vanja Josifovski, and Alexander J. Smola. Scalable distributed inference of dynamic user interests for behavioral targeting. In *KDD*, 2011.

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

[4] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Optimization with sparsity-inducing penalties. *CoRR*, abs/1108.0775, 2011.

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

[6] Emmanuel Candes and Terence Tao. Decoding by linear programming. *IEEE Trans. Info Theory*, 51(12):4203–4215, 2005.

[7] S. Ji and J. Yi. An accelerated gradient method for trace norm minimization. In *ICML*, 2009.

[8] 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.

[9] Guillaume Obozinski, Martin Wainwright, and Michael Jordan. High-dimensional union support recovery in multivariate regression. In *NIPS*, 2008.

[10] S. Pandey, M. Aly, A. Bagherjeiran, A. Hatch, P. Ciccolo, A. Ratnaparkhi, and M. Zinkevich. Learning to terget: What works for behavioral targeting. In *CIKM*, 2011.

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