Convex Multi-Task Learning by Clustering

Aviad Barzilai
baviad@tx.technion.ac.il
Department of Electrical Engineering
The Technion, Haifa, Israel

Koby Crammer
koby@ee.technion.ac.il
Department of Electrical Engineering
The Technion, Haifa, Israel

Abstract

We consider the problem of multi-task learning in which tasks belong to hidden clusters. We formulate the learning problem as a novel convex optimization problem in which linear classifiers are combinations of (a small number of) some basis. Our formulation jointly learns both the basis and the linear combination. We propose a scalable optimization algorithm for finding the optimal solution. Our new methods outperform existing state-of-the-art methods on multi-task sentiment classification tasks.

1 Introduction

We focus on the problem of multi-task learning, where there are many tasks, which can be combined naturally into groups. An example of such a setting is predicting sentiment of product reviews. Different reviewers may use the same wording or superlatives to describe various levels of satisfaction and therefore hidden groups may exist in which similar wording is used to convey a different meaning. One reviewer may write "An OK product" to convey highest sentiment, while another may use the same exact language to describe a product she was not satisfied with. The latter reviewer may express high-sentiment by writing "great product" or "best product", while the former may write "bad product" about a product he was not happy with. Additionally, there are numerous products, and product categories or types, and typically there are few reviews on most products. The problem is, given a large set of reviews, written by many users on numerous products, how should one build a sentiment predictor?

Appearing in Proceedings of the 18th International Conference on Artificial Intelligence and Statistics (AISTATS) 2015, San Diego, CA, USA. JMLR: W&CP volume 38. Copyright 2015 by the authors.

There are two extreme approaches. In the first approach, a single predictor is built based on all the data. Although large amounts of data are used, such a predictor would suffer from the internal bias caused by the differences between the reviewers and product types. An alternative approach is to build an individual sentiment predictor per reviewer or per product (or both). A tailored predictor is expected to work well when trained on large amounts of data for each reviewer and product, unfortunately, in most cases, there are only a few samples per reviewer and product.

In this work we propose an alternative intermediate approach - build an individual predictor for each task, yet restrict all such predictors to be based on a small set of possible atoms. We formulate the problem as jointly clustering tasks and building individual sentiment predictors based on the clusters. We cast the problem as an optimization problem, which is not-convex, and thus may involve many local solutions. We manipulate the optimization problem, converting it into a min-max problem, which, with additional relaxation, yields a convex optimization problem. We derive two specific formulations, one based on hinge-loss (SVM) and the other based on the logistic loss. We derive efficient optimization procedures, and evaluate our algorithm on sentiment analysis tasks, with various reviewers and product types. Our study shows the superiority of our proposed algorithm over baselines and algorithms designed for a similar setting.

2 Related Work

In recent years, there has been a large body of work on multi-task learning and domain adaptation. In domain adaptation it is assumed that there is a single task to be performed but data may come from few domains or sources, and additionally, there is only unlabeled data from the task of interest. In this work we focus on the complementary problem of multi-task learning, where there could be many tasks, some of them very different from each other, and additionally, there is a small amount of labeled-data for each task. We focus on an approach in which models for all tasks are constructed simultaneously.

A well studied line of work is learning a model per task, yet with some constraint or regularization relating the models. Argyriou et al. [3] proposed to learn a model per task, yet all models should lie on a small dimensional subspace. Obozinski et al. [23] took a similar approach, with a different constraint, that all models share a small set of features. Evgeniou et al. [7] used similar modeling, requiring that all models will be close to each other. Gu and Han [8] proposed a related approach for learning a single task by partitioning the data into clusters, each locally linearly separable. A classifier is finally learned as a combination of a global model and per-cluster model. Jacob et al [14] proposed combining penalties on the distance between and within clusters, along with a global penalty, which they relax and solve in primal form. Zhong and Kwok [28] proposed to learn two sets of models to be combined in prediction, the first captures the shared structure and the second captures variations specific to each task. The works of both Ando and Zhang [2], and of Amit et al. [1], are closer to our work. They learn by finding a linear transformation of inputs, which is used by all tasks, and then one model per task, working on the output of the shared representation.

There are numerous works on Bayesian models and algorithms for multi-task learning, mainly by sharing a prior, which can be used to relate tasks [13, 27]. Other works [4, 18, 26] use Gaussian process predictors. Recent work has suggested using complex linear combination classifiers when classes have distinct subpopulations [9, 12] and classification trees for identifying interesting subgroups in the data [19].

A direction closer to ours is to both group tasks and learn a model per such group. The SHAMO algorithm [6] clusters tasks, and learns a model per cluster. SHAMO groups tasks into K clusters with a K-means like approach. In each cluster, all tasks share the same linear classifier and grouping between tasks is performed by iteratively learning classifiers for the current groups and updating the groups based on the cross task errors of the classifiers. When learning the task models, SHAMO uses a random permutation over the dataset. This permutation affects the learning result as SHAMO converges to a local minimum. Kang et al [15] use similar modelling, but cast learning as a mixed integer program.

A related algorithm to our suggested formulation is the GO-MTL algorithm [16][1]. GO-MTL assumes there are a small number of K latent basis tasks and each per-task classifier is built as a linear combination of the basis. The authors propose a logistic regression based implementation to which we compare our work. GO-MTL was shown [16] to have state-of-the-art performance, and thus we do not compare to other algorithms it outperforms.

The work of Maurer et al. [20] proposed to learn a (large) dictionary of base models, and set each model to be a linear combination (called code elements) of a small number of dictionary elements. The optimization problem they develop is not convex and solved by alternating minimization over the dictionary and code.

To conclude, in this work we focus on task clustering in the context of sentiment analysis, similar to the approach suggested in previous work ([6]). We suggest a formulation that simultaneously learns a task dictionary and code vectors ([16]), and derive a convex formulation and optimization method.

3 Problem Setting

There are T tasks, where each task t is associated with a distribution D_t over inputs $(\mathbf{x}, y)$ where $\mathbf{x} \in \mathbb{R}^M$ and $y \in \mathcal{Y}$. In this work we focus on binary classification and assume $\mathcal{Y} = \{+1, -1\}$. We assume to have N samples from all tasks $S = \{(\mathbf{x}_i, y_i, t_i)\}$ where $t_i \in \{1 \ldots T\}$. We further assume that an instance was generated by first sampling its task identity t_i from a fixed unknown multinomial distribution P and thereafter the input and label $(\mathbf{x}_i, y_i)$ were sampled from D_t. In this work we focus on linear classifiers, where a model $\boldsymbol{w}_t \in \mathbb{R}^M$ is associated with a task t. Given an input $\mathbf{x}_i$ belonging to task t_i, its predicted label is defined to be $\text{sign}(\boldsymbol{w}_{t_i} \cdot \mathbf{x}_i)$. We denote by $W \in \mathbb{R}^{M \times T}$ the matrix of all T linear models. That is, the model $\boldsymbol{w}_t$ associated with task t is the tth column of the matrix W. The goal of a learning algorithm is to generate a matrix W given a sample S.

4 Convex Multi-Task Clustering and Learning

Our primary assumption is that the T tasks can be clustered into K task-clusters, which we formulate by factoring the parameter matrix W into a product of two matrices: $F \in \mathbb{R}^{M \times K}$ and $G \in \mathbb{R}^{K \times T}$. The tth column of the matrix G, denoted by $\boldsymbol{g}_t \in \mathbb{R}^K$ associates task t with one of K clusters. For example, if the kth element of $\boldsymbol{g}_t$ is one, and all other elements of $\boldsymbol{g}_t$ are zero, we would say that t is associated with cluster k. The kth column of the matrix F, denoted by $\boldsymbol{f}_k \in \mathbb{R}^M$, is a (base) model associated with cluster k. A pair of matrices F, G defines T linear classifiers with $W = FG$, or $\boldsymbol{w}_t = F\boldsymbol{g}_t$.

[1]We thank the authors for sharing their code.

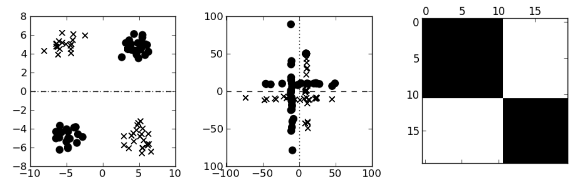

Figure 1: Sample results on the XOR (left) and PLUS (middle) datasets. Samples are divided into 20 tasks, each assigned to one of two groups. The right figure displays the structure found by the algorithm where each row/column represents a task and each cell represents the correlation between tasks. The figure shows that the method successfully identifies the two groups.

Given a labeled sample $S = \{(\mathbf{x}_i, y_i, t_i)\}$ a common practice in learning a linear model (F, G) is to perform structural risk minimization (SRM), picking a model that is simple (in some sense) and performs well on the training data. Learning is then being cast as an optimization problem,

$$C \sum_{i=1}^{N} \ell\left(FG, (\mathbf{x}_i, y_i, t_i)\right) + R(F, G) \ ,$$

where the first term is the empirical error evaluated using a non-negative loss function $\ell\left(W, (\mathbf{x}_i, y_i, t_i)\right)$. We work with functions that evaluate a model by applying the appropriate task model t_i on the input, that is,

$$\ell\left(W, (\mathbf{x}_i, y_i, t_i)\right) = \ell\left(\boldsymbol{w}_{t_i}, (\mathbf{x}_i, y_i)\right) \ .$$

We focus now on the hinge loss defined by $\ell\left(\boldsymbol{w}_{t_i}, (\mathbf{x}_i, y_i)\right) = \max\{0, 1 - (\boldsymbol{w}_{t_i} \cdot \mathbf{x}_i) y_i\}$. Later, in Sec. 7 we will generalize our method to other loss functions, and the logistic loss in particular. The second term is a regularization that penalizes pairs (F, G) according to their complexity, and the constant $C > 0$ is a tradeoff parameter. A common regularization function is the squared Euclidean norm, here applied on the base models, $R(F, G) = \frac{1}{2}\|F\|^2$. To summarize, our learning problem is written as the following optimization problem, where we write the hinge loss with slack variables,

$$\begin{aligned} \min_{F,G,\{\xi_i\}} \quad & \frac{1}{2}\|F\|^2 + C \sum_{i=1}^{N} \xi_i \qquad (1) \\ \text{s.t.} \quad & \xi_i \geq 0 \ , \\ & \xi_i \geq 1 - (F\boldsymbol{g}_{t_i}) \cdot \mathbf{x}_i y_i \qquad \text{for } i = 1 \dots N \\ & \boldsymbol{g}_t \in \{0,1\}^K, \ \ \|\boldsymbol{g}_t\|_2 = 1 \qquad \text{for } t = 1 \dots T \end{aligned}$$

The two last constraints ensure that the matrix G is a proper clustering matrix.

We proceed by computing the dual of (1) over F and $\{\xi_i\}$ by reducing it to SVMs objective. We denote by $\boldsymbol{f} \in \mathbb{R}^{MK}$ the column vector which is the stacking of the columns of the matrix F, that is, $\boldsymbol{f}^\top = \left(\boldsymbol{f}_1^\top \dots \boldsymbol{f}_K^\top\right)$. Additionally, given an input $\mathbf{x}_i \in \mathbb{R}^M$ and an assignment vector $\boldsymbol{g}_{t_i} \in \mathbb{R}^K$ we define $\mathbf{z}_i \in \mathbb{R}^{MK}$ to be the column vector composed of K blocks of dimension M. The kth block is defined to be the kth element of $\boldsymbol{g}_{t_i}$ times $\mathbf{x}_i$, that is, $\mathbf{z}_i^\top = \left((\boldsymbol{g}_{t_i})_1 \mathbf{x}_i^\top \dots (\boldsymbol{g}_{t_i})_K \mathbf{x}_i^\top\right)$. Note that the new inputs are implicit functions of the cluster allocation matrix $\mathbf{z}_i = \mathbf{z}_i(G, \mathbf{x}_i)$, and that $\mathbf{z}_i \cdot \mathbf{z}_j = (\mathbf{x}_i \cdot \mathbf{x}_j)(\boldsymbol{g}_{t_i} \cdot \boldsymbol{g}_{t_j})$.

By construction we have that, $(F\boldsymbol{g}_{t_i}) \cdot \mathbf{x}_i = \boldsymbol{f} \cdot \mathbf{z}_i$. Thus, we rewrite (1) as,

$$\begin{aligned} \min_{\boldsymbol{f},G,\{\xi_i\}} \quad & \frac{1}{2}\|\boldsymbol{f}\|^2 + C \sum_{i=1}^{N} \xi_i \qquad (2) \\ \text{s.t.} \quad & \xi_i \geq 0 \ , \ \xi_i \geq 1 - \boldsymbol{f} \cdot \mathbf{z}_i y_i \quad \text{for } i = 1 \dots N \\ & \boldsymbol{g}_t \in \{0,1\}^K \ , \ \|\boldsymbol{g}_t\|_2 = 1 \quad \text{for } t = 1 \dots T \end{aligned}$$

The last optimization problem defined over $\boldsymbol{f}$ and $\{\xi_i\}$ can be identified with the primal problem of SVM [5] with inputs $\{(\mathbf{z}_i, y_i)\}$, and thus it is equivalent to,

$$\begin{aligned} \min_{G} \max_{\{\alpha_i\}} \quad & \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)(\boldsymbol{g}_{t_i} \cdot \boldsymbol{g}_{t_j}) \quad (3) \\ \text{s.t.} \quad & \alpha_i \in [0, C] \quad \text{for } i = 1 \dots N \\ & \boldsymbol{g}_t \in \{0,1\}^K \ , \ \|\boldsymbol{g}_t\|_2 = 1 \quad \text{for } t = 1 \dots T \end{aligned}$$

and the optimal solution satisfies $\boldsymbol{f} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{z}_i$.

We define a new matrix variable $E = G^\top G \in \mathbb{R}^{T \times T}$ which is interrupted as a cluster association matrix. It is a PSD block-diagonal matrix of rank K with elements in $\{0,1\}$. We have that $E_{t,s} = 1$ iff tasks t and s belong to the same task cluster. We re-write (3) in terms of the new matrix E,

$$\begin{aligned} \min_{E} \max_{\{\alpha_i\}} \quad & \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)(E_{t_i,t_j}) \quad (4) \\ \text{s.t.} \quad & \alpha_i \in [0, C] \quad \text{for } i = 1 \dots N \\ & E \succeq 0 \ , \ E \in \{0,1\}^{T \times T} \ , \ rank(E) = K \\ & E_{t,t} = 1 \quad \text{for } t = 1 \dots T \end{aligned}$$

Now consider the case of two clusters where E is of rank 2 and has two diagonal blocks. If we know that the data should have balanced clusters, then we would favor the case of two similarly sized blocks rather than one large block and one small block. We propose to add a penalty defined to be the number of non-zero elements in E, that is $\sum_{t,s} E_{t,s}$. This constraint would favor the former case of two balanced clusters. Thus we add to the objective the term $\kappa \sum_{t,s} E_{t,s}$, for some, non-negative parameter $\kappa \geq 0$.

At this point we remove the rank constraint and allow the optimal rank to be learned by the algorithm. We will later show that clustering is still obtained. The resulting problem is not convex due to the constraint $E \in \{0,1\}^{T\times T}$, which we thus relax to $E \in [0,1]^{T\times T}$. The resulting optimization problem is,

$$\begin{aligned} \min_{E}\max_{\{\alpha_i\}} \quad & O(E,\alpha) && (5)\\ \text{s.t.} \quad & \alpha_i \in [0,C] \quad \text{for } i = 1\dots N\\ & E \succeq 0\ ,\ E \in [0,1]^{T\times T}\\ & E_{t,t} = 1 && \text{for } t = 1\dots T \end{aligned}$$

$$O(E,\alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j(\mathbf{x}_i\cdot\mathbf{x}_j)(E_{t_i,t_j}) + \kappa\sum_{t,s}E_{t,s}\ .$$

The final formulation is a saddle-point convex problem in both α and E, which follows similar arguments applied in other contexts [17, 25, 22]. Since the algorithm is based on **a sa**ddle **p**oint formulation for multi-task learning, we call it *asap-mt*.

We illustrate the power of our formulation using two synthetic datasets shown in the left and middle panels of Fig. 1. Both datasets contain 20 tasks, where the first 11 tasks are assigned with one linear classifier and the remaining 9 tasks with another. There are 4 training examples from each task, two of which with positive label and two with negative label. In the first dataset (left panel) the two classifiers are the x-axis, in one case the top half-space contains exactly inputs with positive label, and in the other case the points with negative label. That is, the two classifiers are $(0,1)$ and $(0,-1)$. In the middle data set (middle panel) the two classifiers are the y-axis and the x-axis, that is $(1,0)$ and $(0,1)$. The right panels shows the matrix E learned by our algorithm (for both datasets, after sorting the matrix). Clearly the algorithm identified that there are two tasks, and also identified correctly what tasks are associated. In other words, the algorithm combined the twenty training sets of size 4 to 2 training sets of size 44 and 36 respectively. In some cases, such as the provided synthetic datasets, the algorithm identifies hard clusters. In other cases, it is possible to obtain hard clusters by decomposing the matrix E into $G^\top G$ and performing additional clustering or thresholding over the matrix G.

5 Optimization

We solve (5) using a gradient-projection algorithm over the convex function (defined over E),

$$D(E) = \max_{\{\alpha_i\}} O(E,\alpha) \text{ s.t. } \alpha_i \in [0,C] \ \text{ for } i = 1\dots N\ .$$

It is a convex function since it is a point-wise maximum over convex functions. Additionally, the matrix with (i,j)th elements $(\mathbf{x}_i\cdot\mathbf{x}_j)(E_{t_i,t_j})$ is convex, since the matrix E is convex. The function $D(E)$ can be computed by solving the optimization problem which essentially is the dual of SVMs. There are numerous algorithms for solving it, such as SDCA [24].

Given some matrix E^t we compute the gradient $\nabla_E D(E^t) = \nabla_E O(E^t,\alpha^t)$ where $\alpha^t = \arg\max_\alpha O(E^t,\alpha)$ such that $\alpha_i \in [0,C]$, and perform a gradient step $Z^t \leftarrow E^t - \delta\nabla D(E^t)$. Since $D(E^t)$ is symmetric in E^t the gradient and Z^t are symmetric as well. The algorithm then defines the next matrix E^{t+1} to be the projection of Z^t over the constraint set,

$$\mathcal{E} = \left\{E : E \succeq 0, E \in [0,1]^{T\times T}, E_{t,t} = 1 \text{ for } t = 1\dots T\right\}.$$

The projection is performed by sequentially projecting over each constraint individually with an additional correction step [10] that ensures the convergence of the projection onto all constraints. Projecting a symmetric matrix Z over the PSD cone is performed by setting all the negative eigenvalues of Z to zero. Projecting the matrix Z over the constraint $Z \in [0,1]^{T\times T}$ is performed by clipping elements larger than one, to one, and lower than zero, to zero. Finally, projecting the matrix Z on the set of matrices with ones on the diagonal is performed by setting the diagonal elements to one. This sequence of projections is performed until convergence. The projection algorithm is summarized in lines 10-19 of Alg. 1.

The gradient-projection steps are performed until a convergence criteria is met. The algorithm is summarized in lines 1-9 of Alg. 1. We empirically found that normalizing all features to be within $[0,1]$ improves performance and therefore added an additional feature-wise normalization step to our implementation.

Given E, SDCA [24] converges to an ϵ optimal solution in $O((N+C)\log(\frac{1}{\epsilon}))$ for a smooth loss. Each gradient step takes $O(N^2)$ time, and a run of the projection loop takes $O(T^3+T^2+T) = O(T^3)$. Thus if the function perform n iterations in the projection function, and m iterations overall, the total time is $O(m\times((N+C)\log(\frac{1}{\epsilon})+N^2+nT^3))$. The dominating factor here is nT^3 as few iterations are not enough, and n is typically large. It is important to note that, in practice, alpha is not computed from scratch but rather from the previously found alpha, which empirically becomes increasingly faster as the optimization progresses. Our analysis in the paper is theoretical and intended as a worst case upper bound.

We have additionally experimented with a one-shot algorithm summarized in Alg. 2, which performs two iterations of solving α with an intermediate step for solving E. Its run time is faster than the algorithm

Algorithm 1 Gradient Projection Optimization Algorithm

```
 1: Input: $S = \{(\mathbf{x}_i, y_i, t_i)\}_{i=1}^N$
 2: Parameters: $C, \kappa, \delta > 0$
 3: Initialize: $E^0 = I \in \mathbb{R}^{T\times T}$ , $t = 1$
 4: repeat
 5:     Set $Z^t \leftarrow E^{t-1} - \delta\nabla D_E(E^{t-1})$
 6:     Set $E^t \leftarrow \text{PROJECT}_{\mathcal{E}}(Z^t)$
 7:     Set $t \leftarrow t + 1$
 8: until $\|E^{t-1} - E^t\|$ is smaller than tolerance
 9: return $E^t$

10: function $\text{PROJECT}_{\mathcal{E}}$(P)                         ▷ Projects $P$ on the set $\mathcal{E}$
11:     repeat
12:         Set $[U, \Lambda] \leftarrow \text{Eig}(P)$                      ▷ $P = U\Lambda U^\top$
13:         Set $Q = R \leftarrow U\ \text{Diag}(\max(0, \Lambda_{1,1}), \ldots, \max(0, \Lambda_{T,T}))\ U^\top$   ▷ Project on the PSD cone, set Q=R
14:         Set $Q_{t,s} \leftarrow \min\{1, \max\{0, Q_{t,s}\}\}$ for $t, s = 1\ldots T$   ▷ Project on box contraints
15:         Set $Q_{t,t} \leftarrow 1$ for $t = 1\ldots T$              ▷ Project on all diagonal elements equal one
16:         Set $P \leftarrow P - (R - Q)$                                   ▷ Correction step [10]
17:     until change in P is smaller than tolerance
18:     return P
19: end function
```

Algorithm 2 Single-shot Algorithm

```
 1: Input: $S = \{(\mathbf{x}_i, y_i, t_i)\}_{i=1}^N$
 2: Parameters: $C, \kappa, \delta > 0$
 3: Initialize: $E^0 = I \in \mathbb{R}^{T\times T}$
 4: Set $\alpha^0 = \arg\max_\alpha O(E^0, \alpha)$ such that $\alpha_i \in [0, C]$
 5: repeat
 6:     Set $Z^t \leftarrow E^{t-1} - \delta\nabla_E O(E^{t-1}, \alpha^0)$
 7:     Set $E^t \leftarrow \text{PROJECT}_{\mathcal{E}}(Z^t)$
 8:     Set $t \leftarrow t + 1$
 9: until $\|E^{t-1} - E^t\|$ is smaller than tolerance
10: Set $\alpha^* = \arg\max_\alpha O(E^*, \alpha)$ such that $\alpha_i \in [0, C]$
11: return $E^*, \alpha^*$
```

that solves the problem exactly, and in some cases its additional error compared to the first algorithm is small.

6 Illustration

We illustrate the properties of our algorithm using product reviews from Amazon. Each review is composed of one or more paragraphs, and an additional numeric score with one to five stars. We omitted reviews with three stars. We downloaded about 1,000 reviews from 32 product domains, and represented each review using token bi-grams.

In our first experiment we generated 192 tasks as follows. We divided the reviews of each domain into two sets with 480 reviews in each set. A review in the first set was assigned with a positive label $y = +1$ iff the product it covered was rated with five stars. Otherwise, if the product was rated with four stars or less, it was assigned with a negative $y = -1$ label. A similar process was performed in the second set, except here, a positive label was assigned to reviews with two or more stars, and a negative label only to reviews with a product that was rated with a single star. Each of these sets was partitioned randomly to three sets each with 160 reviews. From these 160 reviews, 64 were used for training, 64 for evaluation and 32 for tuning. To conclude, we generated $T = 32 \times 2 \times 3 = 192$ tasks, with $N = 192 \times 64 = 12,288$ training examples. We removed all features that appeared less than 2 times in the combined training sets, yielding bag-of-words representation in dimension of $M = 34,008$. Note, there are two "super" tasks, $32 \times 3 = 96$ tasks where only reviews with five stars were considered positive, and 96 tasks where only reviews with one star were considered negative.

We executed the algorithm with parameters chosen to illustrate the algorithm's best non-trivial solution (the trivial solution is when all tasks belong to the same cluster). The matrix E is shown in the top-left panel of Fig. 2, clearly there are two blocks each with 92 tasks, and corresponding exactly to the two ways the labels were generated from product star-scores. From E we also computed the matrix G satisfying $G^\top G = E$. The matrix appears in the second panel from the left of Fig. 2, where each column corresponds to one of the tasks. To better visualize the result, we display only the top four rows of the G matrix as all other values are near zero. From the plot we ob-

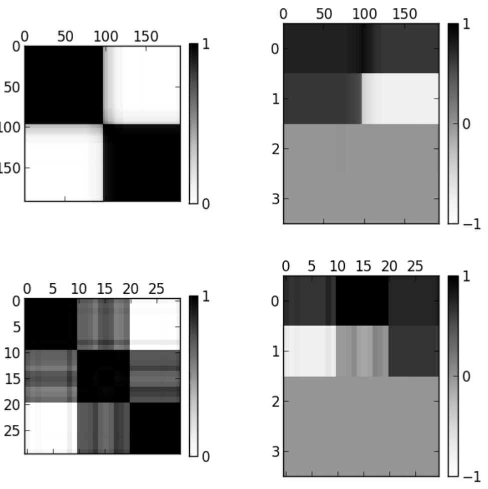

Figure 2: Results obtained by running on a sentiment dataset with 32 domains and two thresholds (top panel pair) and on a single domain dataset with three thresholds (bottom panel pair). Each panel pair shows the clustering matrix learned (E, left column) and its corresponding coefficients matrix (G, right panel).

serve that there are only two non-zero rows. The first row is fixed to +1 across all tasks, while the second row alternates between +1 and −1, corresponding to the tasks in which only a score of one star is considered a negative review (i.e. most reviews are positive) versus tasks in which only a score of five stars is considered a positive review. In other words, we have $\boldsymbol{g}_t \in \{(+1, +1, 0 \ldots 0), (+1, -1, \ldots 0)\}$. Thus, there are two possible classifiers, one is $\boldsymbol{f}_1 + \boldsymbol{f}_2$ (when most reviews are positive) and the other is $\boldsymbol{f}_1 - \boldsymbol{f}_2$ (when most reviews are negative).

We present in Tab. 1 the features with highest (positive) weight and lowest (negative) weight for $\boldsymbol{f}_1$ (two left columns) and $\boldsymbol{f}_2$ (two right columns). Clearly, most of the ten features with highest weight contain words with positive sentiment, e.g. *best*, *love* and *great*. Additionally, more features with negative weights are with negative sentiment than positive. Recall that the actual classifiers are either, the sum of these two vectors, which gives stronger bias towards positive label, or the difference, which gives higher bias towards negative label. For example, in the former case the weight of the word *good* is $0.22 - 0.15 = 0.07 > 0$ (see third and second columns, second line). On the contrary, in the later case the weight of the same word, *good*, is now $-0.15 - 0.22 < 0$. Similarly, the weight of the word *great* is now $0.16 - 0.29 < 0$ (third column, first line, and first column third line). There are less words with positive sentiment that have positive weights, such as *The best*, *I Love* and *Excellent*.

We repeated the above experiment, but with a slightly different setting. First, we took only reviews of a single product (or domain) - namely shoes. Second, we split the reviews into three sets and not two. In the first set, as above, only instances with five stars were associated with a positive review. In the third set, also as above, only instances with one star were associated with a negative review. Both sets are as before. The second set had balanced labeling, reviews with four or five stars were considered positive, while reviews with one or two stars were considered negative. The second set therefore overlaps with both the first and the third sets. Finally, each of these groups was partitioned randomly into ten sets. To conclude, here we had $1 \times 3 \times 10 = 30$ tasks, with three underlying tasks. As before, the training set is of size 64, the test of size 64 and validation set of size 32. Since we have less tasks and all come from the same domain, we used only features that appeared five times or more, ending with $8,813$ features.

The matrix E is shown in the left-bottom panel of Fig. 2, clearly there are three blocks each with 10 tasks, and corresponding exactly to the three ways the labels were generated from product star-scores. Note that the second task group (middle threshold) is similar to the other two tasks in close to equal amount. Typical value in the block of the first line and second column is 0.68, and the third line and second column is 0.75. (Typical value in the block of the first line and third column is 0.02). That is, the algorithm detected that the classification task based on the middle threshold lies between the other tasks. This is further observed in the matrix G shown in the right panel of Fig. 2. The first two features of the vectors $\boldsymbol{g}_t$ corresponding to the three task types are, $[0.69, 0.73]$, $[1, 0]$ and $[0.75, -0.67]$. The weight vector for each task is a linear combination of $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ with the above weights. All tasks are using the first vector $\boldsymbol{f}_1$ with positive weights. The first task adds the second vector, the middle one ignores it, and third subtracts it. That is, again, the task with middle threshold has about the average classification vector of the other two tasks. Finally, the words of $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ appear in Tab. 2. As expected there are domain unique sentiment words (comfortable) as well as general words (great, love, good, disappointed, not buy). Interestingly, There are no features in $\boldsymbol{f}_2$ with very low values (negative far from zero). It is still under investigation.

7 Logistic regression

Our approach can be generalized to other loss methods, using the same steps detailed in Sec. 4. Specifically, we derive our algorithm with the logistic loss function [11], instead of the hinge loss used in SVM.

$\boldsymbol{f}_1(+)$		$\boldsymbol{f}_1(-)$		$\boldsymbol{f}_2(+)$		$\boldsymbol{f}_2(-)$	
BEST	(0.16)	NOT	(-0.27)	GREAT	(0.29)	?	(-0.11)
LOVE	(0.16)	GOOD	(-0.15)	GOOD	(0.22)	NEG_BUY	(-0.11)
GREAT	(0.16)	BETTER	(-0.11)	LOVE	(0.16)	!-!	(-0.09)
THE-BEST	(0.11)	DISAPPOINTED	(-0.10)	VERY	(0.14)	NEG_!	(-0.09)
!	(0.09)	?	(-0.10)	NICE	(0.14)	NEG_EVEN	(-0.09)
EXCELLENT	(0.07)	BAD	(-0.09)	WELL	(0.11)	WASTE	(-0.09)
I-LOVE	(0.07)	NO	(-0.09)	BEST	(0.17)	DO-NOT	(-0.08)
PERFECT	(0.07)	!-!	(-0.08)	MORE	(0.10)	JUNK	(-0.08)
AWESOME	(0.07)	DOTS	(-0.08)	WORKS	(0.09)	WORST	(-0.08)
YOU-NOT	(0.07)	PRETTY	(-0.08)	PERFECT	(0.09)	BAD	(-0.08)

Table 1: features with highest (positive) weight and lowest (negative) weight for $\boldsymbol{f}_1$ (two left columns) and $\boldsymbol{f}_2$ (two right columns) when running on multiple domains

$\boldsymbol{f}_1(+)$		$\boldsymbol{f}_1(-)$		$\boldsymbol{f}_2(+)$	
GREAT	(0.17)	NOT	(-0.18)	NOT	(0.15)
COMFORTABLE	(0.16)	DISAPPOINTED	(-0.06)	VERY	(0.10)
!	(0.13)	BACK	(-0.05)	SHOE	(0.10)
THEY-ARE	(0.09)	OF-THE	(-0.05)	SHOES	(0.07)
LOVE	(0.09)	#	(-0.05)	SIZE	(0.07)
WEAR	(0.06)	UNCOMFORTABLE	(-0.05)	GOOD	(0.06)
WELL	(0.05)	ON-THE	(-0.05)	LIKE	(0.06)
GOOD	(0.05)	ORDERED	(-0.04)	#	(0.06)
VERY-COMFORTABLE	(0.05)	THEY-WERE	(-0.04)	GREAT	(0.05)
BEST	(0.05)	NEG_BUY	(-0.04)	COMFORTABLE	(0.05)

Table 2: features with highest (positive) weight and lowest (negative) weight for $\boldsymbol{f}_1$ (two left columns) and $\boldsymbol{f}_2$ (right column) when running on a single domain. The negative words of $\boldsymbol{f}_2$ were omitted as they had low weights

We begin with the general formulation, similar to (1)

$$\min_{F,G,\{\xi_i\}} \frac{1}{2}\|F\|^2 + C\sum_{i=1}^{N}\log(1+\exp(-(F\boldsymbol{g}_{t_i})\cdot\mathbf{x}_i y_i))$$
$$\text{s.t.} \quad \boldsymbol{g}_t \in \{0,1\}^K,\ \|\boldsymbol{g}_t\|_2 = 1 \quad \text{for } t = 1\dots T. \qquad (6)$$

Moving to the dual form [21] and applying the steps depicted in Sec. 4 we obtain the primal-dual form,

$$\min_{E}\max_{\{\alpha_i\}} \ O(E,\alpha) \qquad (7)$$
$$\text{s.t.} \quad \alpha_i \in [0,C] \quad \text{for } i = 1\dots N$$
$$E \succeq 0\ ,\ E \in [0,1]^{T\times T}\ ,$$
$$E_{t,t} = 1 \quad \text{for } t = 1\dots T\ ,$$

where we define $H(a) = a\log(a) + (C-a)\log(C-a)$ and,

$$O(E,\alpha) =$$
$$\kappa\sum_{t,s}E_{t,s} - \sum_i H(\alpha_i) - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j(\mathbf{x}_i\cdot\mathbf{x}_j)E_{t_i,t_j}$$

We solve the new formulation using methods detailed in Sec. 5. The main difference is the computation of the gradient. We call this variant which is based on the logistic-loss *asap-mt-lr* and the one based on the SVM is called *asap-mt-svm*.

8 Experiments

We evaluated our suggested approach in the context of sentiment analysis using an extensive array of datasets. We evaluated six algorithms all together. Our two baselines are single task learning (STL) and independent task learning (ITL). In STL all samples are considered to originate from a single learning task. We learn a single SVM classifier, with a linear kernel, for all samples. In ITL no task relation is modeled. A single linear SVM classifier is learned for each task independently of the other tasks. Gradient descent was used to solve SVM's objective. We evaluated SHAMO [6] and GO-MTL [16], the former is closest in spirit to our approach, while the latter has been shown to achieve state-of-the-art results in a similar setting.

Our comparison includes two types of datasets - single domain and multiple domain. In the single domain dataset, all reviews originated from a single Amazon category such as books. In the multiple domain, each review may belong to one of a few categories. Within each domain, the samples are split into multiple sub tasks. The sentiment of each review is classified based on a score threshold, as previously explained in Sec. 6, and multiple threshold classification tasks may be included in one dataset. The number of tasks in a single dataset equals: domains × splits × thresholds. We summarize the structure of each dataset in Tab. 6. Parameters of all algorithms are chosen using a development split of the dataset, thereafter the same parameters are used when testing. Both GO-MTL and ASAP-MT require two learning parameters, for the matching ℓ_1 and ℓ_2 norms. GO-MTL requires an additional parameter of K for the number of expected basis vectors, a hidden parameter in our method. The number of iterations for learning GO-MTL can also be tweaked and we select the best number of iterations using the development set. SHAMO requires the number of clusters K and the learner parameter (SVM in our case), while ITL and STL require only a single parameter each - the one of SVM. We report the average task accuracy.

Tab. 3 summarizes the results of running on six domains of the sentiment dataset, where each time only a single domain, and two thresholds, are used. We observe that STL performs worst, as it learns a single model for all tasks, then SHAMO and ITL perform about the same. ITL overfits as models are learned with smaller amount of data, while SHAMO is nonconvex in nature. GO-MTL performs second best, as was also observed previously [16]. Our algorithms, ASAP-MT-SVM and ASAP-MT-LR, perform the best on four out of the six domains, where the latter method is slightly better than the former (the former obtains better results on the development set, which suggests that the difference in results is due to parameter selection). Our algorithm groups tasks into clusters (as SHAMO and GO-MTL), yet it is convex (as STL and ITL), thus it combines the best of both worlds. Tab. 4 summarizes similar results for four thresholds. This problem is harder, as there are four "super" tasks, and not two as before. Indeed the accuracy values in this table are in general lower than in the previous ones, yet the trend remains, ASAP-MT obtains the best results in four out of six domains and with one additional result on par with GO-MTL. Finally, Tab. 5 summarizes the results of running on different variations of the sentiment dataset, each containing multiple domains, and the trend remains, STL is worst, then ITL and SHAMO (slightly better), GO-MTL, and finally, ASAP-MT-SVM and ASAP-MT-LR (best in two out of three).

DOMAIN	GOMTL	SHAMO	ITL	STL	MT-SVM	MT-LR
BOOKS	75.3	72.9	75.3	65.9	75.3	**77.6**
SHOES	75.5	72.4	74.5	63.3	76.3	**77.1**
SOFTWARE	**77.9**	76.0	75.0	62.5	75.0	76.0
SPORTS	75.8	73.2	75.3	65.4	**77.3**	76.3
TOYS	76.0	**80.7**	75.8	68.0	75.0	75.5
VIDEOS	74.7	72.7	75.0	59.6	75.5	**75.8**

Table 3: Average task accuracy when running on sentiment datasets, with a single domain, and using two thresholds. Six algorithms are compared: GO-MTL [16], SHAMO [6], single task learning (STL), independent task learning (ITL), as well as, ASAP-MT-SVM and ASAP-MT-LR.

DOMAIN	GOMTL	SHAMO	ITL	STL	MT-SVM	MT-LR
BOOKS	71.5	72.3	70.0	65.6	71.9	**72.7**
SHOES	76.5	71.5	72.3	64.8	74.4	**76.9**
SOFTWARE	71.0	70.0	69.6	66.0	74.2	**74.2**
SPORTS	**75.2**	73.8	72.7	64.8	74.2	**75.2**
TOYS	**75.6**	71.5	74.8	64.6	74.8	74.8
VIDEOS	71.0	71.3	69.2	59.6	**74.2**	72.9

Table 4: Average task accuracy when running on sentiment datasets, with a single domain, and four thresholds.

SUBSET	GOMTL	SHAMO	ITL	STL	MT-SVM	MT-LR
PAIR	72.6	73.5	71.2	66.0	**75.7**	75.4
MEDIUM	**78.4**	70.7	70.7	67.5	76.7	76.7
LARGE	79.4	74.8	71.9	69.9	77.0	**79.5**

Table 5: Average task accuracy when running on sentiment datasets with multiple domains. There are three group types: pair (two domains), medium (6 domains) and large (12 domains).

TYPE	DOMAINS	NSP	NTH	TASKS	TRAIN	TEST
TAB. 3	1	6	2	12	384	384
TAB. 4	1	3	4	12	240	480
PAIR	2	3	4	24	480	960
MEDIUM	6	3	4	72	1440	2880
LARGE	12	3	4	144	2880	5760

Table 6: Number of domains, splits, thresholds, tasks, training and testing samples of the presented datasets.

9 Summary

We proposed a multitask learning approach, that performs both clustering of tasks and learning a model per task. It can also be thought of as learning a dictionary and constructing models by combining (a small number of) dictionary elements. As opposed to most similar models, it is convex. Our empirical study showed that our method, ASAP-MT-SVM/LR, outperforms state-of-the-art methods, such as GO-MTL and SHAMO. We plan to derive generalization bounds for our formulation, analyze the convergence properties of the algorithms, and develop formulations for more complex problems.

Acknowledgments: This work was supported by a Marie Curie Reintegration Grant PIRG06-GA-2009-256497 and by the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI).

References

[1] Yonatan Amit, Michael Fink, Nathan Srebro, and Shimon Ullman. Uncovering shared structures in multiclass classification. In *ICML*, 2007.

[2] Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853, 2005.

[3] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Convex multi-task feature learning. *Machine Learning*, 73(3):243–272, 2008.

[4] Edwin V. Bonilla, Felix V. Agakov, and Christopher K. I. Williams. Kernel multi-task learning using task-specific features. In *AISTATS*, 2007.

[5] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.

[6] Koby Crammer and Yishay Mansour. Learning multiple tasks using shared hypotheses. In *NIPS*, 2012.

[7] Theodoros Evgeniou, Charles A. Micchelli, and Massimiliano Pontil. Learning multiple tasks with kernel methods. *Journal of Machine Learning Research*, 6:615–637, 2005.

[8] Quanquan Gu and Jiawei Han. Clustered support vector machines. In *AISTATS*, 2013.

[9] Peter Hall, Yingcun Xia, and Jing-Hao Xue. Simple tiered classifiers. *Biometrika*, 100(2):431–445, 2013.

[10] Didier Henrion and Jérôme Malick. Projection methods in conic optimization. In *Handbook on Semidefinite, Conic and Polynomial Optimization*, pages 565–600. Springer, 2012.

[11] David W Hosmer Jr and Stanley Lemeshow. *Applied logistic regression*. John Wiley & Sons, 2004.

[12] Hanwen Huang, Yufeng Liu, and JS Marron. Bidirectional discrimination with application to data visualization. *Biometrika*, 99(4):851–864, 2012.

[13] Hal Daumé III. Bayesian multitask learning with latent hierarchies. In *UAI*, pages 135–142, 2009.

[14] Laurent Jacob, Francis Bach, and Jean-Philippe Vert. Clustered multi-task learning: A convex formulation. In *NIPS*, 2008.

[15] Zhuoliang Kang, Kristen Grauman, and Fei Sha. Learning with whom to share in multi-task feature learning. In *ICML*, 2011.

[16] Abhishek Kumar and Hal Daume III. Learning task grouping and overlap in multi-task learning. *arXiv preprint arXiv:1206.6417*, 2012.

[17] Gert R. G. Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El Ghaoui, and Michael I. Jordan. Learning the kernel matrix with semidefinite programming. *J. Mach. Learn. Res.*, 5:27–72, December 2004.

[18] Neil D. Lawrence and John C. Platt. Learning to learn with the informative vector machine. In *ICML*, 2004.

[19] Wei-Yin Loh. Improving the precision of classification trees. *The Annals of Applied Statistics*, pages 1710–1737, 2009.

[20] Andreas Maurer, Massimiliano Pontil, and Bernardino Romera-Paredes. Sparse coding for multitask and transfer learning. *arXiv preprint arXiv:1209.0738*, 2012.

[21] Thomas P Minka. A comparison of numerical optimizers for logistic regression. Technical report, Carnegie Mellon University, 2003.

[22] Behnam Neyshabur, Yury Makarychev, and Nathan Srebro. Clustering, hamming embedding, generalized lsh and the max norm. *arXiv preprint arXiv:1405.3167*, 2014.

[23] Guillaume Obozinski, Ben Taskar, and Michael I. Jordan. Joint covariate selection and joint subspace selection for multiple classification problems. *Statistics and Computing*, 20(2):231–252, 2010.

[24] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss. *Journal of Machine Learning Research*, 14(1):567–599, 2013.

[25] Linli Xu, James Neufeld, Bryce Larson, and Dale Schuurmans. Maximum margin clustering. In *NIPS*, 2004.

[26] Kai Yu, Volker Tresp, and Anton Schwaighofer. Learning gaussian processes from multiple tasks. In *ICML*, 2005.

[27] Shipeng Yu, Volker Tresp, and Kai Yu. Robust multi-task learning with *t*-processes. In *ICML*, 2007.

[28] Wenliang Zhong and James Kwok. Convex multitask learning with flexible task clusters. *arXiv preprint arXiv:1206.4601*, 2012.