# Applying Online Multitask Learning (OMTL) Algorithms

**Sehwan Chung[1], Parker Howard[2], Gautam Thakur[2]**
[1] Department of Civil and Environmental Engineering
[2] Department of Electrical Engineering and Computer Science
University of Michigan

## Abstract

We explore Online Multitask Learning (OMTL), which allows for multiple tasks to be learned in a real-time setting. OMTL learns task weight vectors and adaptively learns task relatedness from the input data. This work examines previous work and implements the adaptive relatedness matrix in contrast to works that assume a fixed relationship between tasks. We explore an OMTL setting in this paper on data sets in a binary classification setting and present results along with baseline results.

## 1 Problem Statement

Multitask Learning (MTL) can be utilized in settings where access to data from multiple related tasks is available. The goal is to learn all tasks. These related tasks can be jointly learned, rather than through a single classifier, to create a generalized model. By jointly learning, generalization across all tasks can be improved. Furthermore, in the online setting where data is being streamed in real-time, Online Multitask Learning (OMTL) can be used for the classifier to learn classification parameters for multiple tasks. In the setting we explore, we learn $K$ weight vectors (one for each task), as well as a $K \times K$ task relatedness matrix. The approach of OMTL allows for $K$ (the number of tasks) different learning problems to be reduced to a single problem. This problem can then be defined by a single optimization problem. We will explore the OMTL setting for binary classifiers.

### 1.1 Motivation

In prior methodologies, task relatedness can be enforced or defined by a prior assumption. Task relatedness can vary over time, rendering this prior assumption ineffective and restrictive. By learning the task relatedness matrix in the online setting, we can capture this information to improve results.. In some cases, tasks can be entirely unrelated, and in which case, multiple instances of the same algorithm can be used on each task. It is often the case that data is related, and this can exploited. We explore the method in Saha [1] that proposes a method for learning task relatedness directly from the data. The method explores a way to learn a task relatedness matrix as well as weight vectors for individual tasks in the online setting. Uncorrelatedness and negative correlation are also captured by the task relatedness. Many prior works are limited by formation of task relatedness and can not capture uncorrelatedness. Learning a task relatedness matrix allows inter-task relations to provide information on incoming samples belonging to a task. The goal is for the adaptive task relatedness matrix to improve estimates of the weight vectors of the individual tasks. Online learning of the task relatedness matrix is framed as a Bregman divergence minimization problem [2]. The matrix is defined as a task covariance matrix, which is how this is true.

We explore the results of the OMTL algorithm from Saha [1]. The work assumes instances arriving sequentially. Results are then compared to baseline methods of the Perceptron based online multitask learning (CMTL) and BatchOpt. CMTL uses a fixed task interaction matrix, rather than an adaptive relatedness matrix, to find *K* Perceptron weight vectors. The BatchOpt algorithm as well uses an alternative task relationship matrix. The following section describes prior works in MTL, after which the adaptive approach for OMTL is explored.

## 2 Related Works

Prior work in the area of MTL largely differs in task relatedness assumptions and formulations. MTL is explored in Caruana [3]. Tasks are trained in parallel with a shared representation to leverage information in related tasks. This allows achievement of a goal of generalization performance improvement. Training signals are used as an inductive bias to improve performance. When using MTL for backpropagation, the shared information is utilized in a hidden layer shared by all tasks. Often multiple tasks arise naturally. For a medical diagnosis, different tests and measurements become features belonging to patients, which can become the tasks to be learned over.

A notion of task-relatedness is discussed in Ben-David [4]. For probability distributions over different tasks, a notion of F-relatedness is defined. Letting F be a set of transformations, we can say the probability distributions are F-related if there are transformations such that each probability transformation can be transformed to another.

OMTL is discussed in Dekel [5]. Online learning takes place in sequential rounds. At each round, in the multitask instance, K tasks are observed. A loss function is selected to calculate loss for each task. The global loss function is used to combine individual loss values, and the goal is to minimize this global loss function. A global loss function that simply sums all individual loss values treats tasks as independent. A global function that takes the maximum individual loss serves to find the worst mistake. For each time t, k instances are observed as $(\mathrm{x}_{t,1}, \ldots, \mathrm{x}_{t,k}) \in \mathbb{R}^{n_1} \times \ldots \times \mathbb{R}^{n_k}$. For each a, separate classifier is used. The weight vector used to define the *j'*th classifier at time t is $\boldsymbol{w}_{t,j} \in \mathbb{R}^{n_j}$. We define labels as $y_{t,j}$ for the *j'th* task at time t. Using the hinge loss function to penalize, we can denote the loss associated with the *j'*th task at time *t* to be

$$l_{t,j} = \max\{0, 1 - y_{t,j}\boldsymbol{w}_{t,j} \cdot \boldsymbol{x}_{t,j}\}$$

Losses are placed in vector $l_t$ and global loss is calculated as the norm $\| \boldsymbol{l}_t \|$. The algorithm updates weights for each *t* to minimize the global loss. Using a common loss function can cause certain tasks to become prioritized over others. Combining the loss functions into a global loss does not utilize the relationship information between tasks that is available in the data.

In Evgeniou [6], the idea of a multitask kernel is introduced. The *K* learning problems are reduced to a single problem by using a multitask kernel that accounts for co-regularization in the corresponding objective function. In Cavallanti [7], the task interaction matrix is defined as a $K \times K$ matrix *A*. *A* can be chosen in such a way to encode assumptions about the tasks. This work assumes a fixed task relationship matrix. The OMTL methods explored seek to expand on these by having the adaptively updated task relationship matrix.

## 3 Methods

We start by giving a background for the 3 baseline algorithms used in our analysis: Online Multitask Learning setting using a priori fixed Task-Relatedness Matrix (2 different approaches), as well as a brief about the method used for optimizing the batch setting in Multitask Learning. We then progress towards the variation of the general setting described above to account for the 3 OMTL algorithms that Saha [1] uses with an adaptively learned Task Relatedness Matrix as the novel approach.

The baseline algorithm using a fixed Task Relatedness Matrix is the Perceptron based Online Multitask Learning setting as described in Cavallanti et al. [7], which from hereon will be abbreviated as CMTL. An alternative approach to the same using a different Task Relatedness Matrix, which will be referred to as *IPL*.

The general goal (i.e., the output) out of all the algorithms is to learn either just the weight vectors $w_j$ corresponding to the different tasks (assuming a fixed Task-Relatedness Matrix in CMTL and *IPL*), or both the weight vectors and the Matrix **A** as well (like in the 3 novel OMTL approaches). The terms "task-interaction" and "task-relatedness" are used interchangeably.

We assume that the dataset consists of data coming from a pre-defined set of *K* different tasks to the learner in rounds, and the observation is made one sample at a time, in an adversarial fashion. Thus, the learner proceeds in rounds, and at a particular round *t*, receives a data point $x_{t,k}$ from the set and the task I.D. $k \in \{1,2,\ldots,K\}$. First, a linear classification prediction $\widehat{y_t} \in \{-1,1\}$ is obtained using the weight for that task, and then the true label $y_t \in \{-1,1\}$ is received. Based on the comparison of the two aforementioned labels, a decision is made to run an iteration of the learning algorithm. In all algorithms, **w** and **A** are updated only when a prediction mistake occurs. Furthermore, an incoming data sample is stored as a compound vector $\phi_t = (0,\ldots,0,x_{i_t},0,\ldots,0) \in R^{Kd}$. For consistency, the K various weight vectors are also compounded as $w_s^T = (w_{1,s}^T,\ldots,w_{K,s}^T) \in R^{Kd}$. Thus, s here denotes the number of updates to the weight vectors from the learning algorithm until round t ($s < t$).

### 3.1 CMTL

Following Cesa-Bianchi and Lugosi [8], we can view the CMTL algorithm as an optimization of the following loss function:

$$\arg\min_{w \in R^{Kd}} \left[ \frac{1}{2} \boldsymbol{w}^T (\boldsymbol{A} \otimes \boldsymbol{I}_d) \boldsymbol{w} + \sum_{1}^{t} \boldsymbol{l}_t(\boldsymbol{w}) \right] \tag{3.1}$$

where $l_t(w) = \left[\frac{1}{2} - y_t w^T \phi_t\right]_+$ is the hinge loss of the weight vector w at time t. **A** represents the Task-relatedness Matrix of the data. $(\boldsymbol{A} \otimes \boldsymbol{I_d})$ is the Kronecker product of A with the Identity matrix of the dimension of the feature vectors. Thus, in the above equation (3.1), the Kronecker product is the term that co-regularizes the compound weight vectors **w** and aims at bringing the individual task weight vectors closer to each other.

Thus, in CMTL, the **K** weight vectors are updated simultaneously using the fixed priori Task Relatedness Matrix for CMTL. We also can note that for this case, the learning rate (γ) of the algorithm can be determined from the entries of the interaction matrix to be used in the updates rules for each of the **K** Perceptron weights. The following fixed task interaction matrix is used:

$$\mathbf{A}^{-1} = \begin{bmatrix} 2 & 1 & \ldots & 1 \\ 1 & 2 & \ldots & 1 \\ \ldots & \ldots & \ldots & \ldots \\ 1 & 1 & \ldots & 2 \end{bmatrix}$$

Thus, for CMTL, we observe that **γ = (**2/**K**+1**)** for the same task **j = k**, and 1/**K**+1 for different task. This update scheme is reasonable. The interpretation is that it does a fixed constant update for the current task, as well as, does "half-updates" for the remaining K − 1 tasks because they are expected to be related to the current task.

We get the following update rule:

$$\boldsymbol{w_s} = \boldsymbol{w_{s-1}} + \boldsymbol{y_t}(\boldsymbol{A} \otimes \boldsymbol{I_d})^{-1} \boldsymbol{\phi_t} \tag{3.2}$$

As we had observed earlier, this equation represents the vectorized version for the entire task collection, using the above compound vectors. Thus, for individual task j, this reduces to:

$$\boldsymbol{w_{k,s}} = \boldsymbol{w_{k,s-1}} + \boldsymbol{y_t}(\boldsymbol{A_{k.k}})^{-1} \boldsymbol{x_t} \tag{3.2}$$

The above equations are the resultant update steps we use during the learning algorithm CMTL. This forms the baseline algorithm since it has the fixed task-relatedness Matrix and updates all tasks simultaneously.

# 4 Adaptive Relatedness Matrices

Our novel approaches for OMTL, as well as the second baseline algorithm BatchOpt, use a variable update for the **A.** The objective function in the equation (3.1) modifies in this case following Crammer et. Al [9] as:

$$\underset{\boldsymbol{w}\in\boldsymbol{R^{Kd}},\mathbf{A}\succ\mathbf{0}}{\arg\min}\left[\mathbf{D_w}(\mathbf{w}||\mathbf{w_s}) + \mathbf{D_A}(\mathbf{A}||\mathbf{A_s}) + \sum_{\mathbf{1}}^{\boldsymbol{t}} \boldsymbol{l}_{\mathrm{t}}(\boldsymbol{w})\right] \tag{4.1}$$

where $\mathbf{D_w(.||.)}$ and $\mathbf{D_A(.||.)}$ denote Bregman divergences. This cost function is inspired by the classical cost function formulations of online algorithms, where there is a balance between 'conservativeness' and 'correctiveness' during the update of the weight vectors. The matrix **A** is no longer a fixed matrix, and the matrix regularization penalty over **A** is added such that it stays close to the previous estimate of the interaction matrix, which is similar to a conservative update strategy.

In equation 4.1 above, objective function is generic and the $\mathbf{D_A(.||.)}$ term allows substituting any suitable divergence defined over positive definite matrices. The general form of matrix divergence between two positive definite matrices, for a strictly convex function $\boldsymbol{\phi}$:

$$\boldsymbol{D}_{\phi}(\boldsymbol{X},\boldsymbol{Y}) = \boldsymbol{\phi}(\boldsymbol{X}) + \boldsymbol{\phi}(\boldsymbol{Y}) + \boldsymbol{tr}((\boldsymbol{X}-\boldsymbol{Y})\boldsymbol{f}(\boldsymbol{Y})^{\boldsymbol{T}}) \tag{4.2}$$

where f(Y) is partial differential of $\boldsymbol{\phi}(\boldsymbol{Y})$. Based on different divergences we use in the paper, we get different update steps for matrix **A.**

## 4.1 BatchOpt

First, we describe the second baseline algorithm based on the variable A update. This follows the [Zhang and Yeung, 2010]. In the batch approach, first all K weight vectors are estimated, before computing A, and the process is repeated in an alternating fashion until convergence. The ideal step update for A for this algorithm turns out to be:

$$\boldsymbol{A_S} = \frac{(\boldsymbol{W_{s-1}^T W_{s-1}})^{\mathbf{1/2}}}{\boldsymbol{tr}((\boldsymbol{W_{s-1}^T W_{s-1}})^{\mathbf{1/2}})} \tag{4.3}$$

## 4.2 OMTL

After solving for $\mathbf{w_s}$, we fix it and solve for **A**. We can create $\mathbf{W}_{\boldsymbol{s}}$ as a $d \times K$ matrix by reshaping the $Kd \times 1$ vector $\mathbf{w_s}$. After fixing this, the objective function becomes

$$\underset{\boldsymbol{A}\succ 0}{\arg\min}\left[\frac{1}{2} tr(\boldsymbol{W_{s-1}} \boldsymbol{A} \boldsymbol{W_{s-1}^T} + \boldsymbol{D_A}(\boldsymbol{A}\|\boldsymbol{A}_{\mathrm{s-1}})\right] \tag{4.4}$$

The solution to this is

$$\mathbf{A}_{\mathrm{s}} = f^{-1}(f(\boldsymbol{A}_{s-1}) - \eta\ \mathrm{sym}(\nabla_A \tfrac{1}{2} tr(\boldsymbol{W}_{s-1}\boldsymbol{A}\boldsymbol{W}_{s-1}^T) \tag{4.5}$$

We define $\mathrm{f}(\boldsymbol{A}) = \nabla_A \phi(\boldsymbol{A})$, $\mathrm{f}^{-1}$ as the inverse of *f*, and $\mathrm{sym}(\boldsymbol{X}) = (\boldsymbol{X} + \boldsymbol{X}^{\mathrm{T}})/2$. The learning rate of the matrix $\boldsymbol{A}$ is $\eta$. The cases of LogDet divergence and von-Neumann divergences are considered for $\phi$, as well as framing $\mathbf{A}_{\mathrm{s}}$ as a covariance of weight vectors.

### 4.2.1 LogDet Divergence

For LogDet divergence, we have $\phi(\boldsymbol{X}) = \phi_{LD}(\boldsymbol{X}) = -log|\boldsymbol{X}|$. The LogDet divergence between two positive definite matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ is then defined as

$$D_{\phi_{LD}}(\boldsymbol{X},\boldsymbol{Y}) = tr(\boldsymbol{X}\boldsymbol{Y}^{-\mathbf{1}}) - log|XY^{-1}| - n \tag{4.6}$$

This gives $f(\boldsymbol{A}) = \nabla_{\mathrm{A}}\phi(\boldsymbol{A}) = -\boldsymbol{A}^{-1}$ and $f^{-1}(\boldsymbol{A}) = -\boldsymbol{A}^{-1}$. We can reduce (3.2) to

$$\mathbf{A}_s = \left((\boldsymbol{A}_{s-1}^{-1}) + \eta \ \mathrm{sym}(\boldsymbol{W}_{\mathrm{s\text{-}1}}^{\mathrm{T}}\boldsymbol{W}_{\mathrm{s\text{-}1}})\right)^{-1} \tag{4.7}$$

The OMTL algorithm based on the LogDet matrix divergence is referred to as OMTL_LogDet.

### 4.2.2 Von-Neumann Divergence

For von-Neumann divergence, we have $\phi(\boldsymbol{X}) = \phi_{VN}(\boldsymbol{X}) = tr(\boldsymbol{X}log\boldsymbol{X} - \boldsymbol{X})$. The von-Neumann divergence between two positive definite matrices **X** and **Y** is then defined as

$$D_{\phi_{LD}}(\boldsymbol{X}, \boldsymbol{Y}) = tr(\boldsymbol{X}log\boldsymbol{X} - \boldsymbol{Y}log\boldsymbol{Y} - \boldsymbol{X} + \boldsymbol{Y}) \tag{4.8}$$

This gives $f(\boldsymbol{A}) = \nabla_{\mathrm{A}}\phi(\boldsymbol{A}) = log(\boldsymbol{A})$ and $f^{-1}(\boldsymbol{A}) = \exp(\boldsymbol{A})$. We can reduce (3.2) to

$$\mathbf{A}_s = exp(log(\boldsymbol{A}_{\mathrm{s-1}}) - \eta \ \mathrm{sym}(\boldsymbol{W}_{\mathrm{s\text{-}1}}^{\mathrm{T}}\boldsymbol{W}_{\mathrm{s\text{-}1}})) \tag{4.9}$$

The OMTL method based on the von-Neumann divergence is referred to as OMTL_Von.

### 4.2.3 Covariance

In addition to the two divergence methods for updating **A**, we also use the covariance of the task weight vectors as an alternative. Using the task weight covariance is a natural way to estimate relationships between tasks. Prior works explore using the covariance to model relatedness when assuming a Gaussian prior. We define our update for **A** as

$$\boldsymbol{A}_s = \mathrm{cov}(\boldsymbol{W}_{\mathrm{s\text{-}1}}) \tag{4.10}$$

and call this method OMTL_Cov. Cov is defined as the covariance operation on a matrix.

## 4.3 Outline

While the running the algorithms with variable A update, we also have another model parameter in our setup called epoch, which represents the fraction of dataset points that we need to wait for before we start updating **A.** For all algorithms, we first keep **A** as an Identity Matrix, thus assuming Independent tasks. After the number of rounds crosses epoch, we either change **A** as per the CMTL equation in section 3 or apply incremental update functions on it for the OMTL cases. The reason for this procedure is that we do not want to start updating **A** with poorly learned weight vectors, which is usually the case for the initial runs. We show in the later section that epoch has significant effects on the algorithm performances.

Now that all the common concepts are introduced for our algorithms, we present a brief outline of the algorithm used in this setting for both the baselines and the OMTL approaches:

```
Pre-processing of Data
Initializing w and A according to algorithm; Select model parameters
    for s = 1 to T:
        receive (x_{s,k},k) and (y_s);
        generate ŝ = sgn{w_k^T x_{s,k}}
        if (ŷ_s != y_s):
                update w_s
                if s > EPOCH:
                        update A_s
                s = s+1
```

# 5 Evaluation

We conducted tests on both Synthetic and Real datasets to test the claims of the implemented paper [1] as well as study the effect of Model parameter selections on the learning results from the algorithms. We observed that our Synthetic data performs similar to the claims of the paper, and we identified a set of Real data that validates the claims of the paper. However, during our

evaluations, we also found some weaknesses of the implemented paper because the authors do not discuss the effect of task correlation on algorithm performance or the effect of learning rate. We also observed that the results are highly dependent on Model Parameter selection.

## 5.1 Experimental Setup

### 5.1.1 Datasets

As suggested in [1], we synthesized an artificial dataset as follows: First, we generated three weight vectors $w_1, w_2, w_3 \in \mathbb{R}^{10}$ such that $w_1 = -w_2$ and $w_3$ is uncorrelated with the other two. Then we generated three binary classification tasks of 100 data points each, so that we can evaluate the learned task correlation as well as classification accuracy.

After processing several real datasets including the spam dataset from our original proposal [12], we decided to evaluate a pair of Landmine datasets [11], due to computational and memory limitations. This pair of datasets virtually treated as separate, since the inherent properties of the data in the 2 sets resulted in important differences in our evaluation results as discussed below. The first dataset consists of data coming from 5 different tasks and contains 2538 data points. The second contains 10 tasks with a total of 5657 data points. We expand on the different results obtained from these datasets in subsequent sections.

## 5.12 Evaluation Method

The learning algorithms in our codes receive data in the online setting discussed. We executed our code with different settings for both the baseline as well as novel algorithm approaches, taking various parameter values for learning rate $\boldsymbol{\gamma}$ and value of the epoch parameter ranging from 0.1 to 1.0 with a step size of 0.1. We ran our code 20 times per one setting, calculated classification accuracy for each trial along with standard deviation.

## 5.2 Results

### 5.2.1 Synthetic Dataset

On evaluation of the artificial dataset, we observed that the OMTL algorithms significantly outperform the CMTL and BatchOpt as per the claims of [1]. Tables 1a and 1b below show the task relatedness of the learnt weights out of OMTL_LogDet and baseline CMTL algorithm after a single iteration over the dataset. We observe that as expected from the claims of the paper, CMTL is neither able to capture the uncorrelatedness between $\mathbf{w_1}$ and $\mathbf{w_3}$, nor the high negative correlation between the first 2 weights. On the other hand, the OMTL algorithm is able to capture the relatedness much closer to the true case. Furthermore, the prediction accuracies of OMTL_LogDet with a learning rate of $10^{-8}$ is **88.93%**, while that of CMTL in a similar setting is **69.33%.**

| | | |
|---|---|---|
| 1.0 | -0.906 | -0.33 |
| -0.906 | 1.0 | 0.21 |
| 0.3 | 0.21 | 1.0 |

| | | |
|---|---|---|
| 1.0 | -0.18 | 0.51 |
| -0.18 | 1.0 | 0.65 |
| 0.51 | 0.65 | 1.0 |

Table 1a and 1b: Task relatedness after single iteration for OMTL_LogDet(Left) and CMTL(Right)

### 5.2.2 Real-world Dataset

In an application where tasks are positively correlated with each other, it is possible that the CMTL method gives better classification accuracy than OMTL methods, thereby contradicting claims in [1]. Such situations were found to fit the assumption of CMTL algorithm. Hereby, we present two different sets of Landmine dataset, treating them as independent, to empirically support our reasoning.

For Set 1, which has five tasks, we achieved successful results as expected from the paper, i.e., the OMTL algorithms outperformed the CMTL and BatchOpt algorithm in terms of mean accuracy. This observation can further be corroborated by the learnt correlation of the weights as in Appendix section 1. The table below shows the mean accuracy calculated for all the algorithms at epoch = 0.5 (i.e., we wait for half the total set of data points before starting to update **A**), and we can see that OMTL algorithms can classify the tasks. We discuss dependence on model parameters as discussed in the next section. Here, we observe accuracy on full training, similar to [1].

| **LOGDET** | **VON** | **COV** | CMTL | BATCHOPT |
|---|---|---|---|---|
| **65.89(4.05)** | **63.3(5.57)** | **64.5(4.84)** | 61.2(6.73) | 62.6(3.30) |

Table 2: Mean accuracy (std dev) of the algorithms for 20 runs at EPOCH=0.5 on real dataset 1

The second set of data, with ten tasks, gave more interesting results and revealed weaknesses in the implemented paper. CMTL algorithm performed better than OMTL approaches for more of the results, contradicting the claims of the paper, though the difference in performance is not as significant as Set 1. The result can be explained as per the task correlations learnt as in Appendix section 2. As we can observe in table 2 below, CMTL gives better accuracy than OMTL at epoch = 0.5. We still observe that BatchOpt performed worse than the Online setting algorithms.

| LOGDET | VON | COV | **CMTL** | BATCHOPT |
|---|---|---|---|---|
| 57.98(1.97) | 58.91(4.27) | 57.2(4.47) | **59.09(**2.87) | 56.5(2.87) |

Table 3: Mean accuracy (std dev) of the algorithms for 20 runs at EPOCH=0.5 on real dataset 2

### 5.2.3 Model Parameter Selections

The effect of the learning rate($\gamma$) on results is not discussed in [1] and thus brings forth another weakness of the paper. We find the learning rate does not affect the CMTL algorithm, because we observe learning rates are fixed in that case. The learning rate for the novel OMTL approaches has a more significant effect on performance compared to other algorithms we dealt with as a part of coursework. This is because the Online setting does not permit repeated iterations, which we can have in a batch setting. For the synthetic dataset, the OMTL_LogDet algorithm had variable accuracy from 79.67% to 89.33% on decreasing $\gamma$ from $10^{-4}$ to $10^{-16}$. The real datasets also showed similar magnitudes of accuracy variability with a decrease in $\gamma$**.** While decreasing the rate in the aforementioned ranges helped the accuracies improve, the accuracies became almost constant or even decreased after $\gamma = 10^{-20}$. This confirms the fact that a very low learning rate can cause the algorithm to not have enough iterations as per the datasets to correctly learn the task weights.

Our observations about the effect of epoch were as per the claims of the paper for the real dataset results. We observed that changing epoch can significantly affect the final accuracies. However, if we wait too long, the learner may not be able to completely utilize the relatedness among the tasks in the weight update equations. This is similar to a K independent Perceptron algorithm for most of the duration. This is also what we observed in the datasets.

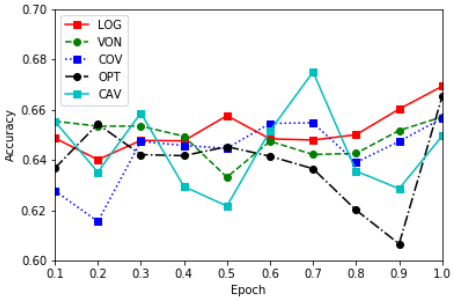


Figure 2: Effect of epoch on real dataset 1

Figure 2 illustrates the effect of epoch on the real dataset 1. We observe that for a majority, the performance of the OMTL algorithms (abbreviated as “LOG” , “VON” and “COV”) outperform both baselines. We also note that for all algorithms, there is a particular epoch that gives maximum mean accuracy as well as there is a decrease towards the latter half.

Figure 3 illustrates the effect of epoch on the real dataset 2. We observe that most of our observations are similar to the effect on dataset 1, except for the accuracy of CMTL outperforming the OMTL algorithms, as discussed earlier.

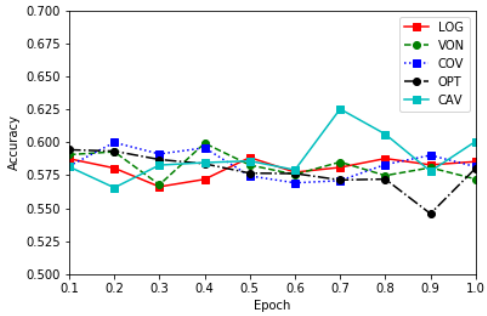


Figure 3: Effect of epoch on real dataset 2

## 6 Conclusion

Through this project, we studied the effects of using an adaptive Task-Relatedness Matrix in Online Multitask Learning algorithm compared to prior work which assumes a priori on the same. We also compared performance of the novel algorithms with the optimal use-case algorithm in a batch setting. While the claimed performance of the novel algorithms in the paper were validated through our synthetic dataset as well as one of our real dataset cases, further investigation revealed some weaknesses and caveats in the claims of the implemented paper. We noted a consistency in performance of the OMTL and CMTL algorithms over that of the BatchOpt algorithm. The CMTL algorithm may perform better than OMTL in a dataset with strong positively correlated weight vectors, as observed in our second Real dataset analysis. We also observed significant effects of model parameters on performance. While our results for varying epoch were consistent with the paper, we inferred that the lack of discussion of the effect of learning rate on the algorithm performance is another weakness of the paper.

## 7 Contributions and Major Obstacles

We had divided the code implementation into 3 major sections and each one of us was responsible for implementation of that section for all 5 algorithms. Further, Sehwan and Parker worked on the research of real datasets as well as the pre-processing required to read the data files for evaluation. Gautam and Parker worked on the synthetic dataset generation and evaluations. Gautam and Sehwan worked on the evaluations of the real datasets.

Difficulties with the project arose in finding appropriate datasets for testing. Many publicly available datasets, such as spam and sentiment, are sparse and have extremely high dimensionalities. This creates issues in computational time as well as preprocessing of data. By testing on the smaller landmine set and a synthetic dataset, we were able to overcome this. Future work would expand upon larger datasets.

## References


[1] A. Saha, P. Rai, H. Daumé III, and S. Venkatasubramanian. Online learning of multiple tasks and their relationships. In Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS'11), Ft. Lauderdale, Florida, 2011.

[2] K. Tsuda, G. Ratsch, and M. K. Warmuth. Matrix exponentiated gradient updates for on-line learning and ¨ Bregman projection. JMLR, 6:995–1018, 2005.

[3] R. Caruana. Multi–task learning. Machine Learning, 28: 41–75, 1997.

[4] S. Ben-David and R. Schuller. Exploiting task relatedness for multiple task learning. Proceedings of Computational Learning Theory (COLT), 2003.

[5] O. Dekel, P. M. Long, and Y. Singer. Online multitask learning. In Proceedings of the 19th Annual Conference on Learning Theory (COLT'06), pages 453–467, 2006.

[6] T. Evgeniou, C.A. Micchelli, and M. Pontil. Learning multiple tasks with kernel methods. J. Machine Learning Research, 6: 615–637, 2005.

[7] G. Cavallanti, N. Cesa-Bianchi, and C. Gentile. Linear algorithms for online multitask classification. Journal of Machine Learning Research, 11:2901–2934, 2010.

[8] N. Cesa-Bianchi and G. Lugosi. Prediction, Learning, and Games. Cambridge University Press, 2006.

[9] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y Singer. Online passive-aggressive algorithms. Journal of Machine Learning Research, 7:551–585, 2006.

[10] Y. Zhang and D.-Y. Yeung. A convex formulation for learning task relationships in multi-task learning. In UAI, 2010.

[11] http://www.ee.duke.edu/~lcarin/LandmineData.zip

[12] http://ecmlpkdd2006.org/challenge.html

# A Appendix

## A.1 Section I: Task Relatedness results for real dataset 1

The figure 1 below shows the task correlation matrix using the learnt weight vectors out of LOGDET algorithm. The individual task vectors in this dataset were found to have low correlation in some places as well negative correlation in others. Similar to the results of our synthetic dataset, the CMTL algorithm is not able to capture these changes through the weight update step as it enforces positive correlation between weight vectors in the assumed fixed Task Relatedness Matrix. That is the reason behind the poor performance of CMTL in comparison, which is as expected in the paper.

| | | | | |
|---|---|---|---|---|
| 1 | 0.69 | -0.25 | -.28 | 0.28 |
| 0.69 | 1 | -0.27 | -.25 | 0.4 |
| 0.25 | -0.27 | 1 | 0.7 | 0.4 |
| -.28 | -.25 | 0.7 | 1 | 0.4 |
| 0.28 | 0.5 | 0.4 | 0.4 | 1 |

Figure 1: Learnt task relatedness Matrix for Dataset 1

## A.2 Section II: Task Relatedness results for real dataset 2

Continuing our observations of the effect of task relatedness, we find that some individual task vectors in this dataset were highly and positively correlated as compared to the previous set, as can be seen in figure 2 below. These 4 tasks highlight why the assumptions of the paper regarding enhanced performance of the adaptive Task-relatedness Matrix is refuted in this particular dataset. As per the matrix **A** defined in Section 3 for CMTL, the fixed Task Relatedness of that matrix assumes that tasks are almost uniformly related to each other with a strong positive correlation and thus fits better here than the learned **A** using the OMTL approaches. We also note that CMTL does not give significantly better results, since there are some negatively correlated vectors as well and thus there is a balancing effect on the algorithm performance.

| | | | |
|---|---|---|---|
| 1.0 | 0.47 | 0.55 | 0.71 |
| 0.47 | 1.0 | 0.98 | 0.72 |
| 0.55 | 0.98 | 1.0 | 0.67 |
| 0.71 | 0.72 | 0.67 | 1.0 |

Figure 2: Learnt task relatedness Matrix for subset of Dataset 2