Multi-task learning:
Alternating Structure Optimization vs. Convex clustering

Charles Corbière, Hamza Cherkaoui

January 31, 2017

1 Introduction

The typical approach in statistics and machine learning is to learn one task at a time. Large problems are broken into small, reasonably independent subproblems that are learned separately and then recombined. However, using this methodology is counterproductive because it ignores a potentially rich source of information available in many real-world problems: the information contained in the training signals of other tasks drawn from the same domain.

Multi-task learning is an approach to inductive transfer where several related tasks are learned simultaneously, using a shared representation of what features are important for the tasks. Explicit a priori information on how the tasks are related is not needed. Indeed, the assumption is that the shared representation is rich enough to describe each task, and can be learned from the data. It tries to avoid overlearning and therefore improve generalization.

A few pratical situations may be adapted to multi-task learning. For example, a primary goal in medical decision making is to accurately, swiftly, and economically identify patients at high risk from diseases like pneumonia so they may be hospitalized [1]. In this problem the diagnosis of pneumonia has already been made. The goal is not to diagnose whether the patient has pneumonia, but to determine how much risk the illness poses to the patient. Using a database where all patients have been hospitalized and subject to prior and posterior measurement in hospital, we use those extra tasks for MTL in order to predict whether a person need to be hospitalized. Another example is prediction of student test results for a collection of schools, based on school demographics [2]. Instead of learning for each school separately, we consider this problem as a multi-task learning with each school as a task to learn.

As reviewed in our state-of-art, three point of view prevale in MTL : considering wheter all task are related, or considering they exhibit a group structure or considering there are some outliers taks. In this paper, we will focus on two technics from the first two assumption. More over, we will go further by studying a convex relaxation for each one in order to improve their effeciency.

2 Multi-task framework

We consider m learning tasks where each task $l \in \mathbb{N}_l$ is associated with a set of training data $\{(x_1^l, y_1^l), ..., (x_{n_l}^l, y_{n_l}^l)\} \in \mathbb{R}^d \times \mathbb{R}$ independently drawn from a distribution P_l. The goal is to learn m functions $\{f_1, f_2, ..., f_m\}$ such that $f_m(x_i^m) \approx y_i^m$. Note that the case m = 1 is the standard (single–task) learning problem.

In the context of learning linear functions for supervised classification or regression, this can be achieved by including a priori information about the weight vectors associated with the tasks, and how they are expected to be related to each other. For simplicity we assume that function f_l for the l^{th} task is a hyperplane, that is

$$f_l(x) = \langle w_l, x \rangle$$

. The generalization to nonlinear models could be done through the use of Reproducing Kernel Hilbert Spaces (RKHS) We denote $W = [w_1, ..., w_m] \in \mathbb{R}^{d \times m}$ as the weight matrix to be estimated. Given a loss function $l(\cdot, \cdot)$, the empirical risk is given by:

$$L(W) = \sum_{l=1}^{m} \frac{1}{n_l} \Big(\sum_{i=1}^{n_l} l(w_l^T x_i^l, y_i^l) \Big)$$

We study the following multi-task learning formulation: $\min_W L(W) + \Omega(W)$, where Ω encodes our prior knowledge about the m tasks.

3 Convex Alternating Structural Optimization

3.1 ASO algorithm

Alternating Structural Optimization is a regularization-based method assuming all tasks are related. More precisely, tasks are assumed to share a common feature space $\theta \in \mathbb{R}^{h \times d}$, where $h \leq \min(m, d)$ is the dimensionality of the shared feature space and θ has orthonormal columns, i.e., $\theta\theta^T = I_h$. Formally, the prediction function f can be expressed as:

$$\forall l \in 1, ...m, \quad f_l(x) = w_l^T x = u_l^T x + v_l^T \theta x$$

where $U = [u_1, ..., v_l, ...u_m]$ is the high-dimensional feature space and $V = [v_1, ..., v_l, ...v_m]$ the weight for the low-dimensional space based on θ.

When chosing square norm regularization $\Omega = \| \cdot \|^2$ to control the task relatedness among m tasks, this leads us to the ASO minimization problem [1]:

$$\begin{aligned} \min_{W, \{v_l\}, \theta} \quad & L(W) + \sum_{l=1}^{m} \alpha \|w_l - \theta^T v_l\|_2^2 \\ \text{s.t.} \quad & \theta\theta^T = I_h \end{aligned}$$

where α is the regularization parameter for task relatedness.

3.2 iASO algorithm

However, this problem has bad generalization performance. To improve it, we add another term which control the complexity of the predictor functions, consisting in an improved ASO (iASO) [2]:

$$\begin{aligned} \min_{W, \{v_l\}, \theta} \quad & L(W) + \sum_{l=1}^{m} \alpha \|w_l - \theta^T v_l\|_2^2 + \beta \|w_l\|_2^2 \\ \text{s.t.} \quad & \theta\theta^T = I_h \end{aligned}$$

Note that it reduces to the ASO algorithm by setting $\beta = 0$ and it reduces to m independent support vector machines by setting $\alpha = 0$.

3.3 cASO algorithm

Our two first algorithms have one main issue: even if we build L as a convex loss function, the problem formulation is non-convex due to orthonormal constraints on θ.
To address it, we relax the feasible domain into a convex set. Let's consider the actual set defined by: $\mathcal{M}_e = \{M | M = \theta\theta^T, \theta\theta^T = I_h, \theta \in \mathbb{R}^{d \times m}\}$. Boyd et. Vandenberghe, in [3], showed that the convex hull of $\mathcal{M}_e$ can be expressed as the convex set $\mathcal{M}_c$ given by

$$\mathcal{M}_c = \{M | tr(M) = h, M \preceq I, M \in \mathbb{S}_+^d\}$$

and each element in $\mathcal{M}_e$ is referred to as an extreme point of $\mathcal{M}_c$, which is the smallest convex set that contains $\mathcal{M}_e$. This leads us to the convex relaxed problem formulation cASO [2]:

$$\begin{aligned} \min_{W,M} \quad & L(W) + \Omega_{cASO}(W, M) \\ \text{s.t.} \quad & tr(M) = h, M \preceq I, M \in \mathbb{S}_+^d \end{aligned}$$

where $\Omega_{cASO}(W, M) = \alpha\eta(1+\eta)tr(U^T(\eta I + M)^{-1}U)$ with $\eta = \beta/\alpha \geq 0$

3.4 Computation of cASO

Like in a block coordinate descent method, we fix simultaneously W or M, while the other one can be optimized in terms of the fixed one.

3.4.1 Computation of W for a Given M

For that part, no particular problem, for a given M , the optimal W can be computed by solving the following problem:

$$\min_{W} \quad L(W) + \Omega_{cASO}(W, M)$$

where $\Omega_{cASO}(W, M) = \alpha\eta(1+\eta)tr(U^T(\eta I + M)^{-1}U)$.

3.4.2 Computation of M for a Given W

Theoretically, the minimization of M given W could be done the same way. However, computationally, this would be too expensive. We detail in the following an iterative efficient approach to find its optimal solution.

First, we decompose $W = P_1 \Sigma P_2$ its singular value decomposition, where $P_1 \in \mathbb{R}^{d\times d}$ and $P_2 \in \mathbb{R}^{m\times m}$ are orthogonal, and rank(W) = q with generally $q \leq m \leq d$ and where $\Sigma = diag(\sigma_1, ...\sigma_m)$ with the eigenvalues in decreasing order. Let's now consider the following optimization problem [3]:

$$\begin{aligned} \min_{\{\lambda_i\}_{i=1}^q M} \quad & \frac{\sigma_i^2}{\eta + \lambda_i} \\ \text{s.t.} \quad & \forall i \in \mathbb{N}_q, \sum_{i=1}^{q} \lambda_i = h, 0 \leq \lambda_i \leq 1 \end{aligned}$$

In [2], Ye et al. prove that the optimal solution to find an optimum M for a given W can be obtained by solving the previous problem. We can then construct our optimal $M^* = P_1 \Lambda^* P_1^T$ where $\Lambda^* = diag(\lambda_1^*, ..., \lambda_q^*, 0)$ made from the optimal solution from the previous optimization problem.

After having iterated until convergence, we can construct θ using the top h eigenvectors of M^* and still $V = \theta U$.

4 Convex clustered multi-task learning

In this method we introduce a penalization on the variance to cluster the tasks.
As previously, let's consider the dataset D_n, the weights matrix W of the m tasks, $W = [W_1, W_2, ..., W_m]$. Since we consider linear model, one task is done with $f_l(x) =< W_l, x >$.
We suppose that those m tasks can be clustered in C groups. Let $\overline{W}_c$ be the mean weights vector of the cluster c. Each tasks belonging to that cluster should be related, so each corresponding W_l should be closed to $\overline{W}_c$. Jacob and Bach, in [5], propose to force that by introducing variance penalties:

- $\Omega_{\text{mean}}(W) = n\|\overline{W}\|^2$: a global penalty

- $\Omega_{\text{between}}(W) = \sum_{c=1}^{r} \|\overline{W}_c - \overline{W}\|^2$ with $\overline{W} = \frac{1}{C}\sum_{c=1}^{C} \overline{W}_c$: a measure of between-cluster variance

- $\Omega_{\text{within}}(W) = \sum_{c=1}^{r} \sum_{i \in I(c)} \|\overline{W}_i - \overline{W}_c\|^2$: a measure of within-cluster variance, this term can be seen as a K-means approach

which lead to the full penalisation: $\Omega(W) = \epsilon_m \Omega_{\text{mean}}(W) + \epsilon_b \Omega_{\text{mean}}(W) + \epsilon_w \Omega_{\text{mean}}(W)$.

4.1 Notations

We will consider m task of linear regression over $\mathcal{X} = \mathbb{R}^d$. Note that we could also choose to do classification. We will consider a training set $(x_i, y_i)_{i=1,...,n}$, hence $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y} = \mathbb{R}$. Each tuple (x_i, y_i) is associated with a task $l \in \{1, ..., m\}$. We denote by $\mathcal{I}lt) \subset \{1, ..., n\}$ the set of indices for the task l.
As we have choose to work with linear model, each task is associated with a weight-vector w_l. We denote by $W = (w_1, ..., w_m)^T$ the associated $m \times d$ matrix. We denote by l the loss function on this regression problem associated with $(x_i, y_i)_{i=1,...,n}$, and L the associated risk. Hence we have: $L(W) = \frac{1}{n}\sum_{l=1}^{m} \sum_{\mathcal{I}(l)} l(x_i w_l^T, y_i)$
We also introduce some usefull matrices, $u = [1\ 1...1] \in \mathbb{R}^m$, $U = \frac{uu^T}{m} \in \mathbb{R}^{m \times m}$ and $\Pi = I - U$.

4.2 Problem formulation

As explained in [5], we will enforce our approach with a specific penalisation denoted by Ω, hence we consider the problem:

$$\min_{W \in \mathbb{R}^{m \times d}} \quad L(W) + \lambda \Omega(W)$$

Recall from recap section, we want to penalize how big the weight w_l are, the variance between each w_l within a cluster and the variance between the clusters. To do so, in a matrix notation, we define Ω as:

$$\Omega(W) = \epsilon_m tr(W^T U W) + \epsilon_b tr(W^T (M - U) W) + \epsilon_w tr(W^T (I - M) W)$$

with $M = E(E^T E)^{-1} E^T$. With r being the number of cluster, we denote E the $m \times r$ matrix encoding the cluster such that $E_{ij} = 1$ if (x_i, y_i) belong to task j and 0 if not. E should be optimized on the training set as we will see. If we put $\Sigma(M)^{-1} = \epsilon_m U + \epsilon_b (M - U) + \epsilon_w (I - M)$, we obtain:

$$\min_{W \in \mathbb{R}^{m \times d}} \quad L(W) + \lambda tr(W^T \Sigma(M)^{-1} W)$$

Here, we want to force the cluster to be 'compact', meaning that $\epsilon_W \geq \epsilon_B \geq \epsilon_M$. We also want not to give the cluster as a prior but rather to optimize it, hence:

$$\min_{W \in \mathbb{R}^{m \times d}} \min_{M \in \mathcal{M}_r} \quad L(W) + \lambda tr(W^T \Sigma(M)^{-1} W)$$

with $\mathcal{M}_r = \{M : M = E(E^T E)^{-1} E^T, E_{ij} \in 0, 1\}$. But rather than working with $\Sigma(M)$, we put $\Sigma = \Sigma(M)$ and define $\mathcal{S}_r = \{\Sigma(M) : M \in \mathcal{M}_r\}$, hence we have:

$$\min_{W \in \mathbb{R}^{m \times d}} \min_{M \in \mathcal{S}_r} \quad L(W) + \lambda tr(W^T \Sigma^{-1} W)$$

4.3 A convex relaxation

The set $\mathcal{S}_r = \{\Sigma(M) : M \in \mathcal{M}_r\}$ being not convex, we will optimize over a convex set of positive semi-definite matrices that contains $\mathcal{S}_r$.

Recall that we have denote $\Sigma(M)^{-1} = \epsilon_m U + \epsilon_b (M - U) + \epsilon_w (I - M)$. M only appear in the last two terms. One can prove: $\epsilon_b (M - U) + \epsilon_w (I - M) = \Pi(\epsilon_B M + \epsilon_W (I - M))\Pi$. If we denote by $\hat{M}$ the matrix $\Pi M \Pi$ and inject the previous equality in $tr(W^T \Sigma^{-1} W)$, we have:

$$tr(W^T \Sigma^{-1} W) = \epsilon_M tr(W^T U W)(W\Pi)^T (\epsilon_B \hat{M} \epsilon_W (I - \hat{M}))(W\Pi)$$

We choose to incorporate the first term in the risk, since it only rely on W. So we define:

$$L_c(W) = L(W) + tr(W^T U W)$$

Finally, we put $\Sigma_c^{-1} = \epsilon_B \hat{M} \epsilon_W (I - \hat{M})$, so our penalization becomes $tr((W\Pi)^T \Sigma_c^{-1} (W\Pi))$.

We sum up the constraint on $\hat{M}$ that was inherited from M (ie E):
$\hat{M} \geq U$, $0 \leq \hat{M} \leq \Pi$ and $tr(\hat{M}) = r - 1$.
A convex relaxation for this set is to choose $\Sigma_c \in \mathcal{S}_c$ with $\Sigma_c = \{\Sigma_c \in \mathcal{S}_+^m : \alpha I \leq \Sigma_c \leq \beta I, tr(\Sigma_c) = \gamma\}$ with $\alpha = \epsilon_M^{-1}$, $\beta = \epsilon_B^{-1}$ and $\gamma = (m - r + 1)\epsilon_W^{-1} + (r - 1)\epsilon_B^{-1}$, at least our problem become:

$$\min_{W \in \mathbb{R}^{m \times d}} \quad \min_{\Sigma_c \in \mathcal{S}_c} \quad L(W) + \lambda tr((W\Pi)^T \Sigma_c^{-1} (W\Pi))$$

4.4 Minimizing our problem

We now have a differentiable and convex problem. The only difficulty left is to optimize on W and Σ_c on the same time, one can formulate the problem as:

$$\min_{W \in \mathbb{R}^{m \times d}} \quad \left(L(W) + \min_{\Sigma_c \in \mathcal{S}_c} \lambda tr((W\Pi)^T \Sigma_c^{-1} (W\Pi)) \right)$$

In [5], Jacob and Bach present an algorithm to compute $\Sigma_c^* = \min_{\Sigma_c \in \mathcal{S}_c} \lambda tr((W\Pi)^T \Sigma_c^{-1} (W\Pi))$
Hence we can provide this general gradient decent algorithm:

$W = W_0$;
while *not convergence* **do**
  $\Sigma_c^* = \min_{\Sigma_c \in \mathcal{S}_c} \lambda tr((W\Pi)^T \Sigma_c^{-1} (W\Pi))$;
  $W = W - \eta \nabla_W (Lc(W) + \lambda tr((W\Pi)^T \Sigma_c^{*-1} (W\Pi)))$;
end

5 Experiments

In this section, we evaluate Alternating Structure Optimization, Convex Alternating Structure Optimization and Clustered Multi-Task Learning algorithms and compare them to a multi independent SVM on three regression problems using a toy generated dataset, School dataset and Sarcos dataset.

5.1 Datasets

First, to test our algorithms, we have created a toy dataset. We generated three clusters of three tasks each. The tasks are vectors of $\mathbb{R}^d$, $d = 12$. For each cluster, a center $\overline{w}_c$ was generated in $\mathbb{R}^{d-2}$, so that the three clusters are orthogonals. More precisely, each $\overline{w}_c$ had $(d-2)/2$ random features randomly drawn from $\mathcal{N}(0, \sigma_r^2), \sigma_r^2 = 900$, and $(d-2)/2$ zero features. Then, each tasks l was computed as $w_l + \overline{w}_{c(l)}$, where $c(l)$ was the cluster of l. w_l had the same zero feature as its cluster center, and the other features were drawn from $\mathcal{N}(0, \sigma_c^2), \sigma_c^2 = 16$. The last two features were non-zero for all the tasks and drawn from $\mathcal{N}(0, \sigma_c^2)$. For each task, 1 000 points were generated and a normal noise of variance $\sigma_n^2 = 150$ was added.

Our first real data set comes from the Inner London Education Authority. This data consists of examination records of 15362 students from 139 secondary schools. The goal is to predict the exam scores of the students based on features based on the student and the exam itself. Those features are represented as categorical variables using binary variables, so the total number of inputs for each student in each of the schools was 27. For computational reasons, we test our model on the first 27 schools.

SARCOS, our second dataset, relates to an inverse dynamics problem for a seven degrees-of-freedom SARCOS anthropomorphic robot arm[1]. The task is to map from a 21-dimensional input space (7 joint positions, 7 joint velocities, 7 joint accelerations) to the corresponding 7 joint torques. For computational reasons, we select 10 000 samples.

[1] http://www.gaussianprocess.org/gpml/data/

5.2 Implementation

We have chosen to implement our work in a Scikit-Learn BaseEstimator class style. Thanks to that implementation, we simply have to call a fit and a predict method on a dataset.

For the clustered multi-task learning, our main class ClusteredLinearRegression have many possible options. One can do regression or classification: the Frobenius loss function and the Logistic loss function are avaible within this class. One can choose to estimate the cluster, or give a prior cluster or even estimate without any cluster. All options are available during the initialization of the class. The optimization done during the fit is based on FISTA algorithm. The cluster matrice estimation is based on the algorithm provide in [5], we have re-use the implementation of Laurent Jacob and had some minor modification.

On ASO and cASO, we use l-bfgs-b method to optimize our weight matrix W given a shared low dimensional matrix θ.

5.3 Model selection and evaluation metrics

On this experiment, we focus on regression problem. As a consequence, we can think of the Mean Squared Error to evaluate the error made by our model. But this error is not standardized and doesn't allow us to compare models. Thus, we prefered to use the normalized root Mean Squared Error (nrMSE) defined as follow:

$$nrMSE = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2}}{y_{max} - y_{min}}$$

where $f(x_i)$ correspond to the task l attached to x_i

We run our model for each dataset varying four test size proportion. For each run, we made 5 random splits of the data into training and we measured the generalization performance using nRMSE. The score computed correspond to the mean and the variance of those five measures

5.4 Results

For computational reasons, we have pre-fit our hyper-parameter for each dataset and for each model using a grid search method. We compute a plot for the following test size percentage : $\{0.30, 0.40, 0.50, 0.60\}$. The results are presented for each dataset on the following plots.

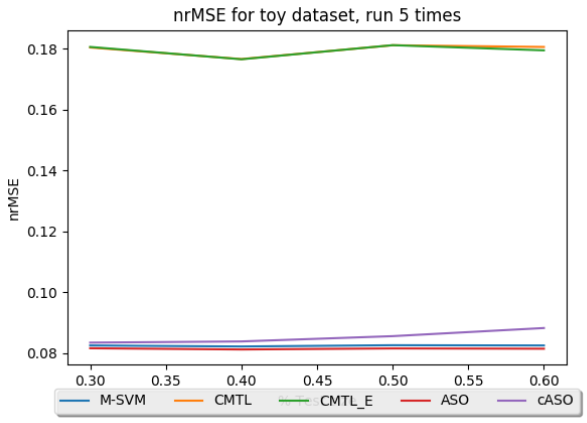

Figure 1: *nrMSE for toy dataset, run 5 times, with 2000 samples, pre-fit hyper-parameter*

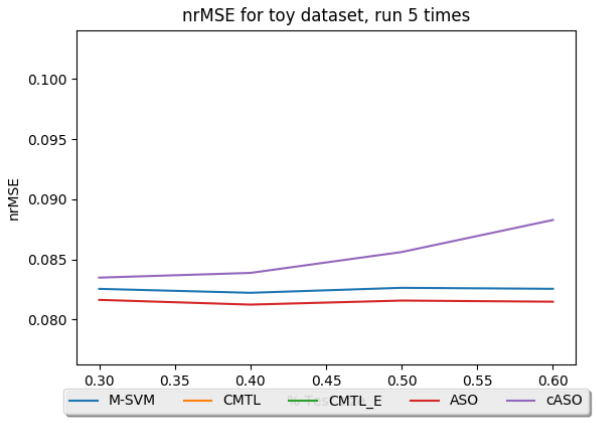

Figure 2: *Zoomed nrMSE on ASO, cASO and M-SVM for toy dataset*

First of all, when computing on our toy dataset, we observe that CMTL performs less than SVM, even when given the true clusters (CMTLE). One way to explain it is all the hyper parameters to fit. Due to computational cost, we prefit the guessed number of clusters. On the other side,

we can appreciate the ASO algorithm which is the most accurate predictor here. cASO seems to perform less than ASO. Indeed, we have to remember that cASO is a convex relaxation of ASO and does not minimize the same problem, except for a certain configuration [2].

Now, on school dataset, we observe the same comportement than in the toy dataset, except for cASO. Indeed, this time cASO performs better than SVM. The school data has the particularity to have differents samples for each task, when they were shared among all tasks for toy dataset (and also for sarcos dataset). As a consequence, the training set is smaller than usual. We can appreciate here the effect of linking all those tasks to allow a better performance via ASO and cASO. CMTL seems to improve when the test size increase but the important variance doesn't allow us to conclude.

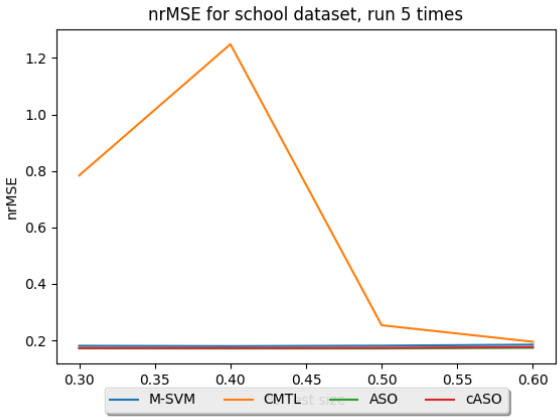

Figure 3: *nrMSE for School data, run 5 times, with 28 schools, prefit*

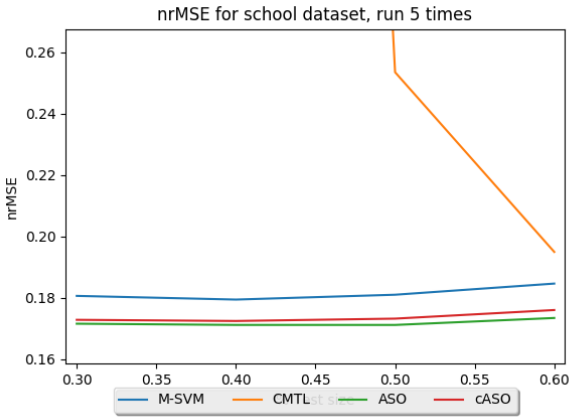

Figure 4: *Zoomed nrMSE on ASO, cASO and M-SVM for School data*

Finally, the SARCOS dataset is the one presenting best results for Clustered multi-task learning. We also still have ASO as the best modelisation of our tasks. Remembering that SARCOS consist of predicting a seven degrees-of-freedom robot arm, we can easily understand the closeness of our tasks.

On time matters, ASO is the fastest algorithm. cASO is slightly slower due to the double optimization problem he is solving on each iteration. On the other side, CMTL running time, as multiple SVM, in those experiments can be up to 5 minutes as we run five times in a row, that's why we chose to sample our total dataset.

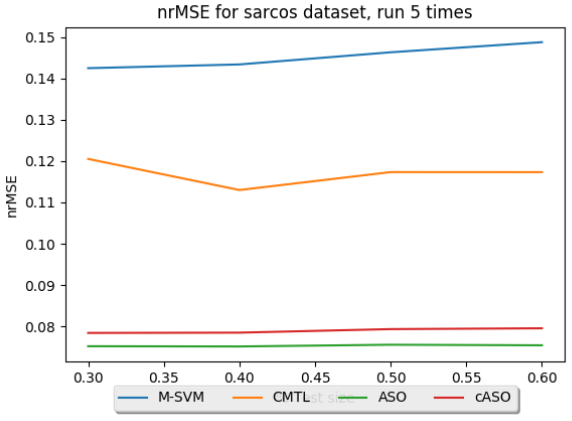

Figure 5: nrMSE for Sarcos data, run 5 times, with 1000 samples, prefit

6 Conclusion

We presented in this paper two methods for multi-task learning: one assuming all tasks share a low-dimensional structure and the other assuming taks are clustered. Further, we presented convex relaxation for both algorithm in order to help computation.

On our three dataset, Alternating Structure Optimization showed the best performances. Altough we were expecting the contrary, ASO performed better in term of score and computation time than Convex Alternating Structure Optimization. The reasons can be a bad implementation

of cASO. Plus, we're dealing here with "small" datasets here, between 1000 and 5000 samples, so ASO is not affected yet by its computation limits.Clustered Multitask Learning had more constrast results. As cASO, we could explain it by a really hard implementation. However, CMTL shouldn't be put aside as we had some difficulties here because of computation limitation. As expected, AS

To go further, we could actually establish an equivalence relationship between cASO and CMTL [4]. More precisely, we could prove that the minimization problem of ASO and CMTL are equivalent if the cluster number in K-means equals to the size of the shared low-dimensional feature space. Finally, our code is well structured as in scikit-learn and could be re-used for other datasets [2].

References

[1] R.Ando et al. A Framework for Learning Predictive Structures from Multiple Tasks and Unlabeled Data. *JMLR*, 2005.
[2] J.Chen et al. A Convex Formulation for Learning Shared Structures from Multiple Tasks. *ICML*, 2009.
[3] S.Boyd et L.Vandenberghe. Convex optimization. *Cambridge University Press*, 2004
[4] J.Zhou et al. Clustered Multi-Task Learning Via Alternating Structure Optimization. *NIPS*, 2011.
[5] L.Jacob, J-P.Vert, F.Bach. Clustered Multi-Task Learning: a Convex Formulation. *NIPS*, 2008

[2] https://github.com/chcorbi/MultiTaskLearning