JMLR: Workshop and Conference Proceedings 81–95

Transfer Learning by Kernel Meta-Learning

Fabio Aiolli AIOLLI@MATH.UNIPD.IT
Dept. of Mathematics, University of Padova, Via Trieste 63, 35121 Padova, Italy

Editor: I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver

Abstract

A crucial issue in machine learning is how to learn appropriate representations for data. Recently, much work has been devoted to *kernel learning*, that is, the problem of finding a good kernel matrix for a given task. This can be done in a semi-supervised learning setting by using a large set of unlabeled data and a (typically small) set of *i.i.d.* labeled data. Another, even more challenging problem, is how one can exploit partially labeled data of a source task to learn good representations for a different, but related, target task. This is the main subject of *transfer learning*.

In this paper, we present a novel approach to transfer learning based on kernel learning. Specifically, we propose a *kernel meta-learning* algorithm which, starting from a basic kernel, tries to learn chains of kernel transforms that are able to produce good kernel matrices for the source tasks. The same sequence of transformations can be then applied to compute the kernel matrix for new related target tasks. We report on the application of this method to the five datasets of the Unsupervised and Transfer Learning (UTL) challenge benchmark[1], where we won the first phase of the competition.

Keywords: transfer learning, kernel meta-learning, unsupervised learning, UTL challenge

1. Introduction

Transfer learning (Pan and Yang (2010), Caruana (1997)) shares some properties with semi-supervised learning: in both cases a large set of unlabeled data and a (generally far smaller) set of labeled data are available. However, in transfer learning, labeled data are only provided for a set of *source* tasks that are related, but different than the *target* task. In this paper, we assume all tasks are defined within a single domain, e.g. face recognition data, handwritten character recognition data, or textual data, just to name a few.

Kernel learning is a state-of-the-art paradigm for semi-supervised learning (Chapelle et al. (2006); Zhu and Goldberg (2009)). The goal of kernel learning is to learn a kernel matrix using available data (labeled and unlabeled) that optimizes an objective function that enforces the agreement between the kernel and the set of i.i.d. labeled data, e.g., by maximizing their alignment (Lanckriet et al. (2004)). On the other hand, unlabeled data are used to regularize the generated models by constraining the discriminant function to be smooth (that is, it should not vary too much on similar examples). However, in transfer learning, the distribution over the training/validation datasets (on examples and their labels) are generally different from the distribution over the target dataset, thus standard kernel learning methods do not directly apply.

1. http://clopinet.com/ul

© F. Aiolli.

In this paper, we explore kernel-based transfer learning and show we can indeed learn something for a task by exploiting other related tasks. In particular, rather than direct learning a kernel for a particular target task, we use source tasks to learn *how* a good kernel can be (algorithmically) generated for *any* task defined over the same domain. In a sense, we propose a *kernel meta-learner* (a learner which learns how to learn kernels from data). To the best of our knowledge this is a novel approach to kernel learning that seems promising for learning kernels and transferring knowledge in multi-task settings. Related work can be found in Pan and Yang (2010) and in the papers cited therein.

Using the technique presented in this paper, we won the first phase of the Unsupervised and Transfer Learning (UTL) challenge. Our algorithm was the best performing on three of five final competition datasets. Although we did not participate in the second phase of the challenge, experiments are presented in this paper demonstrating that the same approach can naturally be adapted to a pure transfer learning setting with competitive results.

Notation We first introduce notation used throughout the paper. Unless otherwise stated, we assume that a dataset is given as an $m \times n$ matrix $X \in \mathbb{R}^{m \times n}$ formed by m rows X_i, $i = 1, \ldots, m$, representing n-dimensional examples. We use the symbol $\circ$ as the Hadamard product (entry-wise) matrix multiplication. We also denote by $\mathbf{1}$ the column vector where each entry is set to 1, and $\mathbf{0}$ the null column vector, the dimensionality of which should be clear from the context they appear in.

Background In this paper, we mainly focus on positive semi-definite (PSD) matrices, that is the class of real matrices $K \in \mathbb{R}^{m \times m}$ such that $v^\top K v \geq 0$ for any real vector $v \in \mathbb{R}^m$. Given any representation $\{\mathbf{x}_i\}_{i=1,\ldots,m}$ for the examples, it is well known that the kernel matrix $K \in \mathbb{R}^{m \times m}$ formed by the dot products between examples, that is $K(i,j) = \mathbf{x}_i^\top \mathbf{x}_j$, is a PSD matrix. Additionally, for PSD matrices, it is always possible to perform the spectrum decomposition, $K = UDU^\top$, where U is an orthogonal matrix formed by the eigenvectors of K, and D is the diagonal matrix with the diagonal formed by the associated (non negative and decreasing) eigenvalues. So, we can always write $K = HH^\top$ where $H = UD^{\frac{1}{2}}$. As a result, any PSD matrix of order m can be seen as a kernel matrix for a dataset of m examples with the examples represented according to the matrix H. This highlights an important aspect of kernels, that any kernel matrix induces a representation of examples (e.g. by the matrix H) which only considers similarity relations between pairs of examples in the dataset. Furthermore, given a kernel matrix K, there can be infinitely many representations for the set of examples that have K as their kernel matrix.

Synopsis We briefly describe the UTL competition setting in Section 2. An algorithm for kernel meta-learning is described in Section 3 and the set of kernel transforms we have used for the challenge is described in Section 4. In Section 5, we discuss an adaptation of the basic algorithm for its use on the UTL challenge as well as some tricks that reduce the risk of overfitting. Additional post-challenge results are also reported. In Section 6, we propose a general strategy to learn the optimal sequence of transforms and present the results obtained on the UTL benchmark. Finally, in Section 7, we conclude the paper with some final considerations and subjective ideas concerning future work.

2. The UTL Challenge

The UTL challenge benchmark[2] contains data related to five different real-world, multiclass problems. For each of these domains, three datasets (development, valid, final) have been prepared using different subsets of the original problem classes. Thus, each dataset contains a sample of a subset of classes of the original domain. Multiple binary tasks were also defined on each of these datasets by splitting the classes in two parts in a number of different ways (obtaining positive and negative labels for the tasks). Here, we briefly describe the UTL challenge setting. Please, refer to (Guyon et al. (2011)) for more details.

2.1. The Competition

For each dataset, a data matrix represented as feature vectors (m examples in rows and n features in columns) was provided to the participants. The goal of the challenge was to produce a new kernel matrix $K \in \mathbb{R}^{m \times m}$ between examples such that the transformed representation would lead to good performance on supervised learning tasks defined over the valid and final datasets. The actual labels of the supervised tasks used by the organizers were unknown to the participants. The evaluation was made using cross validation by partitioning data several times into training and test sets. On each run, a simple (Hebbian) linear classifier defined on the training data (and the associated kernel matrix) is used to build the scoring function. The ranking produced is then evaluated in terms of the Area Under the ROC curve (AUC) and averaged over the random splits. The size of training data varies between 1 and 64 examples and the AUC is plotted on a log scale against the number of examples. Finally, the area under the learning curve (ALC) is used as the overall evaluation metric.

Note that the labels of the supervised tasks used for the evaluation were not available in the first or second phase. Additional labels (from the development set) were made available for transfer learning in the second phase only.

2.2. The Hebbian classifier

In this section, we give a brief description of the classifier used in the challenge. Let X be the matrix containing the vectorial representation of the examples (i.e. dataset as rows). For a given task defined over the set of examples X, the (Hebbian) linear classifier, or linear scoring function, $\mathbf{f} = X\mathbf{w} \in \mathbb{R}^m$ is constructed by setting $\mathbf{w} = X^\top \mathbf{y}$, $\mathbf{y} \in \mathbb{R}^m$, $\mathbf{y}_i = +\frac{1}{m_+}$ (resp. $\mathbf{y}_i = -\frac{1}{m_-}$) if the point X_i is positive (resp. negative) for the task, and $\mathbf{y}_i = 0$ if the point does not belong to the training set of the given task. The values m_- and m_+ represent the number of negative and positive training points, respectively.

Note that, the scoring function on the set of points X can be written as

$$\mathbf{f} = X\mathbf{w} = XX^\top \mathbf{y} = K\mathbf{y} = K_{tr}\mathbf{y}_{tr},$$

where $K = XX^\top$ is the kernel matrix, K_{tr} is the subset of kernel rows/columns corresponding to the training data of the task, and $\mathbf{y}_{tr}$ are the corresponding entries in $\mathbf{y}$. In other words, for any example X_i, $\mathbf{f}_i$ represents the difference between the average of kernels $K(i,+)$ across positive examples minus the average of kernels $K(i,-)$ across negative

2. http://www.causality.inf.ethz.ch/ul_data/DatasetsUTLChallenge.pdf

examples of the training set for the task. Thus, given a task, the score simply represents the algebraic difference of the similarities of an example with the centroids of positive and negative examples of the task.

This type of classifier has a number of nice properties. Firstly, the ranking of examples induced by the classifier does not depend on the scaling of the kernel matrix, that is, defining $K' = \alpha K$, with $\alpha > 0$ a scalar, does not change the ranking produced by the corresponding scoring function. Secondly, if we add a constant value to every entry of a kernel matrix, $K' = K + \beta \mathbf{1}\mathbf{1}^\top$, the values of the scoring function do not change. In fact, $\mathbf{f} = K'\mathbf{y} = K\mathbf{y} + \beta \mathbf{1}\mathbf{1}^\top \mathbf{y} = K\mathbf{y} = K_{tr}\mathbf{y}_{tr}$, since $\mathbf{1}^\top \mathbf{y} = 0$ by construction. These two properties make it possible to standardize the kernel without changing the ranking produced. Thus, an equivalent kernel taking values in $[0, 1]$ can be obtained by using the linear transformation $K' = (max(K) - min(K))^{-1}(K - min(K)\mathbf{1}\mathbf{1}^\top)$. We found this type of standardization useful as a preliminary step prior to discretization of the kernel matrix, a step that was required before the submission to the challenge server.

Finally, this linear classifier can also be seen as a strongly regularized version of an SVM variant (see for example Aiolli et al. (2008) for details) and it makes this kind of classifiers suitable when very few training data are used.

3. The Kernel Meta Learning (KML) Algorithm

The basic idea of this paper is to learn a chain of kernel transformations that, starting from an initial kernel matrix, leads to a better kernel for a target dataset. For this, a set of available validation (or source) supervised tasks are used to train the learner. The expected output of this procedure is an unsupervised "algorithm", or a sequence of operations, to perform on a given dataset and related kernel matrix. It is important to stress that we are not interested in the kernel matrices computed by this algorithm (in fact those kernel matrices cannot be used over different tasks and data) but we mainly focus on the sequence of operations used to obtain them from data.

We propose a greedy algorithm starting with a seed kernel on the available source data, and iteratively transforming it so that each transform results in a kernel with improved performance on the available source tasks. Labeled data is used to find the optimal transformation parameters during each step. Our intuition is that, by keeping the number of parameters involved in this optimization small, the method should generalize well on new tasks. It should produce chains of kernel transformations that are suitable for other related tasks as well.

Many strategies can be used to implement the idea above and optimize the parameters of these kernel transformations, and in Section 6 we propose a general strategy that finds an optimal sequence of such transformations. In this section, we describe the initial strategy we used in the first phase of the UTL challenge.

Assume an initial set of m examples (a dataset) as rows in a matrix $X \in \mathbb{R}^{m \times n}$. Starting from a linear kernel matrix $K(1) = XX^\top$ computed on this dataset, each step $t = 1, \ldots, \bar{t}$ of the algorithm computes a new kernel matrix $K'(t)$ by transforming the kernel $K(t)$ using one of a set of given operators (possible sets of operators will be described in detail in the following section). The next (perturbed) kernel $K(t+1)$ will be produced as a convex combination of $K(t)$ and $K'(t)$, i.e. $K(t+1) = (1-a) * K(t) + a * K'(t)$. The combination

coefficient $0 \leq a \leq 1$ is determined by validating over the set of labeled examples. Once a good kernel has been obtained for the validation tasks, or a predefined number of steps is reached, the algorithm stops and outputs the optimal sequence of transformations along with their combination parameters. Then, when the computation of a kernel matrix for another (related) task is needed, the same (unsupervised) sequence of transformations can be applied on the new set of unlabeled data.

Note that, since in the UTL challenge the labeled examples for the validation set were not directly available, a raw (in fact manual) validation was performed on each step (i.e. by submitting kernels and looking at the validation set results). In addition to the general algorithm we have given here, its real application to the challenge and methods to reduce the risk of overfitting on validation tasks will be explained in detail in Section 5.

4. Kernel Transforms

For the algorithm in Section 3 to work we must use kernel transforms that do not require direct access to feature vectors. Fortunately, we can exploit the *kernel trick* (Schölkopf et al. (1999)), where pattern-pattern similarities can indirectly represent examples. An additional contribution of the present work is to characterize some types of data transformations that have this characteristic. Below, we describe in detail the four classes of kernel transformations we used in the UTL challenge. In particular, we consider: a subset of affine transformations, transformations of the kernel spectrum, polynomial transformations, and an algorithmic transformation based on Hierarchical Agglomerative Clustering (HAC). All the proposed transforms will produce PSD matrices when they are applied to PSD matrices.

4.1. Affine Transformations: Centering and Normalization in Feature Space

Consider a kernel matrix $K \in \mathbb{R}^{m\times m}$ and any matrix $X \in \mathbb{R}^{m\times n}$, such that $K = XX^\top$. Here, we show how a set of affine transformations on the rows $\{X_i\}_{i=1,\ldots,m}$ of X can directly be performed by transforming the kernel matrix and hence they are suitable for use by our algorithm. Specifically, if we take any transformation of the form:

$$X'_i = \beta_i X_i + \gamma^\top X \tag{1}$$

with $\beta \in \mathbb{R}^m$, $\gamma \in \mathbb{R}^m$, then it can be shown that the following holds:

$$K' = X'X'^\top = (\beta\mathbf{1}^\top) \circ K \circ (\mathbf{1}\beta^\top) + \mathbf{1}\gamma^\top K + K\gamma\mathbf{1}^\top + (\gamma^\top K\gamma)\mathbf{1}\mathbf{1}^\top.$$

Hence, every linear transformation like this can be given as a kernel transformation. Well-known instances of this class of transformations are briefly described next.

A first simple transformation is the centering of examples in feature space. In this case, we have $X'_i = X_i - \frac{1}{m}\sum_{j=1}^m X_j$, which corresponds to the setting in Eq. 1 when $\beta = \mathbf{1}$ and $\gamma = -\frac{1}{m}\mathbf{1}$. By applying the formula above, we get the well known centering kernel transformation (see Shawe-Taylor and Cristianini (2004)),

$$K_c = K - \frac{1}{m}\mathbf{1}\mathbf{1}^\top K - \frac{1}{m}K\mathbf{1}\mathbf{1}^\top + \frac{1}{m^2}(\mathbf{1}^\top K\mathbf{1})\mathbf{1}\mathbf{1}^\top. \tag{T_c}$$

Similar transforms exist for other kinds of data processing. Pattern normalization, for example, corresponds to the same class of transformations given above with $\beta_i = 1/\sqrt{K(i,i)}$

and $\gamma = \mathbf{0}$. In this case, $K'(i,i) = X'_i X'^{\top}_i = ||X'_i||^2 = 1$ and $\text{trace}(K') = m$, the number of examples in the set.

We notice some interesting facts about kernel centering. If $K = XX^{\top}$ is centered, then for every $\alpha \in \mathbb{R}^+$ we have $\alpha K = \alpha XX^{\top} = (\sqrt{\alpha}X)(\sqrt{\alpha}X)^{\top}$, which is also centered. Furthermore, the sum of centered kernels $K' = \sum_{j=1}^{s} K_j$ is also a centered kernel since it can be seen as the kernel obtained by concatenating the individual feature space representations, $X'_i = [X_{i,1}, X_{i,2}, \ldots, X_{i,s}]$ and is clearly centered. From these two facts, it follows that any linear combination of centered kernels is also a centered kernel.

Similar results can be given for the trace of kernel matrices. For instance, any matrix obtained as a convex combination of matrices with the same trace t, will have trace t. Also, any convex combination of normalized kernels is a normalized kernel.

4.2. Kernel Spectrum Transformation

Another variety of valid kernels can be computed by modifying the spectrum of a given kernel matrix through a function with codomain in $\mathbb{R}^+$. Considering a positive semi-definite matrix (a kernel) K, this can always be written as $K = \sum_{i=1}^{m} \lambda_i \mathbf{u}_i \mathbf{u}_i^{\top}$ where $\{\lambda_1, .., \lambda_m | \lambda_i \geq 0, \lambda_i \geq \lambda_{i+1}\}$ are the eigenvalues (in decreasing order) and $\{\mathbf{u}_1, .., \mathbf{u}_m\}$ the corresponding eigenvectors of K. Our proposal here is to transform the eigenvalues via a function $\sigma(\lambda)$ taking values in $\mathbb{R}^+$, i.e.

$$K_\sigma = \sum_{i=1}^{m} \sigma(\lambda_i) \mathbf{u}_i \mathbf{u}_i^{\top}. \qquad (T_\sigma)$$

Different types of *spectrum transformation* functions can be defined on the spectrum of the eigen-decomposition of a kernel matrix. The two we have used for the UTL challenge[3] are:

Step: $\sigma(\lambda) = 1$ when $\lambda \geq \epsilon\lambda_1$, $0 \leq \epsilon \leq 1$, and $\sigma(\lambda) = 0$ otherwise. This transformation corresponds to the principal directions.

Power: $\sigma(\lambda) = \lambda^q$ where $q \geq 0$. This transformation corresponds to exponentiating the kernel matrix, i.e. $K' = K^q$.

Interestingly, the effect obtained when using *Power* as the spectrum transformation function is related to a softer version of the (kernel) PCA on the features of the kernel K. In fact, after performing an orthogonalization of the features, the weight of components having small eigenvalues are relatively reduced (when $q > 1$) or increased (when $q < 1$). Note that the complexity of the obtained kernel decreases with q, as the higher we set q, the smaller will be the number of significant directions we are using. Viceversa, when q tends to 0, the transformed matrix tends to the identity matrix and the data has orthogonal representations. This transform also has interesting connections with diffusion maps (Coifman and Lafon (2006)).

In order to better analyze the effect of this transform, we consider the spectral decomposition of the matrix $K = UDU^{\top}$, where U contains the eigenvectors $\{\mathbf{u}_j\}_{j=1,\ldots,m}$ as

3. Note that, since the kernel obtained after this transformation is not normalized, consistently with other transformations presented in this paper, a subsequent normalization of the obtained kernel can be performed. In fact, this is what we did in the challenge.

columns and D is a diagonal matrix with the eigenvalues of K in decreasing order. This decomposition exposes the new representations of the examples. Specifically, we have

$$\mathbf{x}'_i = [\sqrt{\sigma(\lambda_1)}\mathbf{u}_{1i}; \ldots; \sqrt{\sigma(\lambda_m)}\mathbf{u}_{mi}].$$

It is apparent that the above corresponds to a reweighting of the components. When q is large, more emphasis is given to the most important components and when q is small, all the components have more equal importance.

Note that when this transformation is used on a centered kernel matrix, the transformed matrix is also centered since all features are scaled by the same amount, while the trace changes according to trace(K') $= \sum_{i=1}^{m} \sigma(\lambda_i)$.

4.3. Polynomial-Cosine Kernel Transformation

So far, the proposed transformations do not modify the original feature space, rather they perform only linear transformations of the feature vectors. An alternate, non linear, way to transform the feature vectors is to apply a polynomial transformation. For this, we propose the following kernel transformation:

$$K_\pi(i,j) = \frac{(\cos(\mathbf{x}_i, \mathbf{x}_j) + u)^p}{(1+u)^p} = \frac{1}{(1+u)^p}\left(\frac{K(i,j)}{\sqrt{K(i,i)}\sqrt{K(j,j)}} + u\right)^p \qquad (T_\pi)$$

where $p \in \mathbb{N}$ and $u \geq 0$. Specifically, with this transformation, the original kernel is used to compute cosine similarity in feature space, and a polynomial transformation is computed on the result. Finally, the normalization term makes $K_\pi(i,i) = 1$ for every i. It is easy to show that this is a valid kernel using the closure properties of kernels (see e.g. Shawe-Taylor and Cristianini (2004)). One effect of the polynomial kernel, and this transformation in particular, is to further deemphasize similarities of dissimilar examples.

4.4. Algorithmic Transformations: a Kernel based on HAC

A clustering resulting from an unsupervised algorithm can be used to define a kernel. This is another way to exploit a similarity or kernel function to generate a new kernel. Hierarchical Agglomerative Clustering (HAC) is a popular method for clustering data. It starts by treating each pattern as a singleton cluster, subsequently, merging pairs of clusters until a single cluster contains all patterns. To do this, it requires (i) a pattern-pattern similarity function and (ii) a merge strategy that decides which pair of clusters to merge based on a cluster-cluster similarity function. In our case, the kernel to be transformed is used as the pattern-pattern similarity matrix. Popular cluster-cluster similarity measures are often based on *single-linkage*, *complete-linkage*, or *average-linkage* strategies. In the single-linkage (resp. complete-linkage) strategy, the similarity between two clusters is defined as the similarity of the most (resp. least) similar members. In the average-linkage strategy, the similarity between two clusters is defined as the average of similarities between all members of the two clusters.

A generalization of these measures can be defined as:

$$S(c_1, c_2) = \frac{\sum_{\mathbf{x}_i \in c_1, \mathbf{x}_j \in c_2} K(i,j) \cdot e^{\eta K(i,j)}}{\sum_{\mathbf{x}_i \in c_1, \mathbf{x}_j \in c_2} e^{\eta K(i,j)}}, \qquad (2)$$

where $\eta \in \mathbb{R}$. The single-linkage strategy corresponds to $\eta = +\infty$, the complete-linkage strategy corresponds to $\eta = -\infty$, and the average-linkage strategy is obtained by setting $\eta = 0$.

We now propose a kernel defined on the basis of agglomerative clustering. Let $C_t \in \mathbb{R}^{m \times m}, t \in \{1, \ldots, m\}$, be the matrix with binary entries such that $C_t(i,j) = 1$ whenever the examples i and j are in the same cluster at the t-th agglomerative step, and 0 otherwise. Clearly, $C_1 = I$, as this refers to the initial clustering where each example represents a different cluster, and $C_m = \mathbf{1}\mathbf{1}^\top$ is the matrix with all entries set to 1, as in this case there is a single cluster. Finally, the HAC kernel can be defined by:

$$K_h = \frac{1}{m} \sum_{t=1}^{m} C_t. \qquad (T_h)$$

In this way $K_h(i,j)$ represents the fraction of times the examples i and j are assigned to the same cluster in the HAC agglomerative process. It is a valid kernel since $K_h(i,j) = \frac{1}{m}\mathbf{x}_i^\top \mathbf{x}_j$, where $\mathbf{x}_i$ is one possible representation of the i-th example consisting of a binary vector having a component for each node of the dendrogram generated through the agglomerative process. Specifically, we have $\mathbf{x}_{is} = 1$ whenever the node s belongs to the dendrogram path starting from the root and ending on the leaf corresponding to the example i. It is also possible to show this kernel is proportional to the depth in the HAC dendrogram of the *Lowest Common Ancestor* (LCA) of the two examples.

4.5. Other kernel transforms

We conclude this section with additional examples of classes of transformations that were not used in the challenge but could be studied and added in the future.

A first interesting transform that can be used is $K' = KAK$ with A an opportune PSD matrix. It is possible to show that KPCA (Schölkopf et al. (1998)) is an instance of this transformation obtained by setting $A = U_k D_k^{-1} U_k^\top$, where U_k are the first k columns of the spectral decomposition of K, and $D_k \in \mathbb{R}^{k \times k}$ is the associated submatrix of D. Moreover, for any $n > 0$ and $C \in \mathbb{R}^{m \times n}$, the matrix $C^\top K C$ is positive definite and thus, this transformation can be used by our algorithm. Another easy transform can be obtained by using the RBF kernel $K(i,j) = \exp(-\beta(K(i,i) + K(j,j) - 2K(i,j)))$ with $\beta > 0$.

5. Adaptation of the KML algorithm to the UTL challenge and Results

In the first phase of the UTL challenge, labeled examples were not directly available and the algorithm described in Section 3 was not applicable as it is. So, we adapted the algorithm by simplifying the validation procedure to make it practical to be performed manually. As a side effect, this simplification reduced the effective hypothesis space and also the danger of overfitting. More specifically, we fixed the order of application of the transforms (based on their "complexity", from low to high complexity transforms) and limited the set of possible parameter choices. Moreover, a transformation was only accepted (i.e. $a > 0$) if it improved the ALC on validation significantly. In this way we avoided overtuning parameters. In this section we give additional details about the above-mentioned criteria.

The kernel centering transform, T_c, was only performed at the very beginning (i.e. data preprocessing) and only when its application improved the ALC on validation over a raw linear kernel. After that, the other transforms were validated one by one, starting from T_σ, then T_π, and finishing with T_h. This procedure was then iterated until there was no significant ALC improvement.

The T_σ transformation was validated with parameter $q = 0.2 \cdot q_0, q_0 \in \{1, \ldots, 10\}$. In order to decrease the actual number of values to try, we assumed a convex behavior of the $ALC(q)$ curve and limited ourselves by performing a binary search for the best q. The combination coefficient assigned to this type of transform was always a binary value, i.e. $a \in \{0, 1\}$, meaning the transform was simply *accepted* or not. This choice was made depending on whether the resulting improvement was considered significant.

The T_π transformation was validated with parameters $p \in \{1, \ldots, 6\}$, and $u \in \{0, 1\}$. We started with $p = 1$ and increased the value until the ALC on validation could not be further improved. Even in this case, the combination coefficient was chosen to be binary, depending on the significance of the real ALC improvement on the validation set.

Finally, the T_h transformation was validated with parameter $\eta \in \{-10, 0, +10\}$. However, we noticed that $\eta = 0$ was almost always the best choice. Moreover, since in this case the transformed kernel is expected to be fairly different than the original, the combination coefficient has been more accurately validated by doing a binary search on the set of values $a = 0.05 \cdot a_0, a_0 \in \{0, \ldots, 20\}$.

In Table 1, detailed results obtained at post-challenge time are reported. In particular, we used the same validation performed during the challenge in order to investigate possible overfitting. Notice that in some (very few) cases, the obtained results can slightly differ from the official results reported for the challenge due to minor differences[4]. The results confirm that the method is quite robust to overfitting. Although this behavior was absolutely expected, it could not be given as granted during the challenge.

6. A general strategy for Transfer Learning

As described in previous sections, kernel validation was performed manually for the challenge. In this section, we propose a general (and automatic) strategy for use when binary labels for the source tasks are available. Specifically, validation performs a greedy search over the space of transform sequences. We assume access to a predefined and finite set of transforms.

6.1. The TKML strategy: The Algorithm

The pseudo-code in Algorithm 1 describes a general strategy to find the optimal set of transformations according to a given notion of accuracy on source tasks. This procedure will be provided with a set of source datasets, $\mathcal{X}$, and a set of binary tasks, $\mathcal{L}$, defined over the source data. Solely to simplify notation, we assume each source dataset has the same number of binary tasks defined over it. However, it is trivial to adapt it to the case where

4. In particular, during the challenge some transforms were (erroneously) applied to the integer matrix used for submission instead of applying them to the correct real valued kernel matrix to transform.

Table 1: Details of the validation procedure performed by the KML algorithm on the five datasets of the UTL challenge (Phase 1). *RawVal* is the ALC result obtained by the linear kernel on the validation set. *BestFin* is the ALC result obtained by the best scoring competitor algorithm (its final rank in parenthesis). For each dataset, the ALC on validation and the ALC on the final datasets are presented. Note that only those transformations accepted by the algorithm ($a > 0$) are reported with their *optimal* parameters.

AVICENNA	RawVal: 0.1034 BestFin: 0.217428(2)	Val ALC	Fin ALC
T_c	k.s1.t0c1n0	0.124559	0.172279
$T_\sigma(q = 0.4), a = 1$	k.s1.t0c1n0.q04n	0.155804	0.214540
$T_\pi(p = 6, u = 1), a = 1$	k.s1.t0c1n0.q04n.p6u1	0.165728	0.216307
$T_h(\eta = 0), a = 0.2$	k.s1.t0c1n0.q04n.p6u1.d0.a02	0.167324	0.216075
$T_\sigma(q = 1.4), a = 1$	k.s1.t0c1n0.q04n.p6u1.d0.a02.q1_4n	0.173641	**0.223646**(1)
HARRY	RawVal: 0.6264 BestFin: **0.806196**(1)	Val ALC	Fin ALC
T_c	k.s2.t0c1n0	0.627298	0.609275
$T_\pi(p = 1, u = 0), a = 1$	k.s2.t0c1n0.p1u0	0.604191	0.678578
$T_h(\eta = 10), a = 1$	k.s2.t0c1n0.p1u0.d10	0.861293	0.716070
$T_\sigma(q = 2), a = 1$	k.s2.t0c1n0.p1u0.d10.q2n	0.863983	0.704331(6)
RITA	RawVal: 0.2504 BestFin: 0.489439(2)	Val ALC	Fin ALC
T_c	k.s3.t0c1n0	0.281439	0.462083
$T_\pi(p = 5, u = 1), a = 1$	k.s3.t0c1n0.p5u1	0.293303	0.478940
$T_h(\eta = 0), a = 0.4$	k.s3.t0c1n0.p5u1.d0.a04	0.309428	**0.495082**(1)
SYLVESTER	RawVal: 0.2167 BestFin: **0.582790**(1)	Val ALC	Fin ALC
$T_\sigma(\epsilon = 0.00003), a = 1$	k.s4.t0c0n0.e000003n	0.643296	0.456948(6)
TERRY	RawVal: 0.6969 BestFin: 0.843724(2)	Val ALC	Fin ALC
T_c	k.s5.t0c1n0	0.712477	0.769590
$T_\sigma(q = 2), a = 1$	k.s5.t0c1n0.q2n	0.795218	0.826365
$T_h(\eta = 0), a = 0.95$	k.s5.t0c1n0.q2n.d0.a095	0.821622	**0.846407**(1)

the number of tasks per dataset varies. Finally, we assume the existence and access to a finite set of predefined transformations in $\mathcal{T}$.

The algorithm maintains a priority queue of transformed kernels and additional information about them: the sequence of transformations that have already been applied, and the evaluation of the kernel produced (e.g. the ALC observed on the associated source tasks). The priority of an element of the queue is defined as a function of the ALCs obtained on the associated tasks (e.g. their average). On each iteration, an element of the queue (i.e. a kernel set, the list of applied transforms, and the evaluation of the transformed kernels) is extracted and all transformations available in the set $\mathcal{T}$ are applied, in turn, to the current kernels. After each transform has been applied, the obtained kernels are evaluated and inserted into the queue if it increases the relative performance.

At this point, it is important to note that the effectiveness of this algorithm depends very much on the type, and number, of transforms. In general, we can have up to $|\mathcal{T}|^v$

Input : $\mathcal{X} = \{X_1, \ldots, X_s\}$: A set of source data matrices $X_i \in \mathbb{R}^{m \times n}$
$\mathcal{L} = \{L_1, \ldots, L_s\}$: A set of source binary tasks $L_i \in \mathbb{R}^{m \times q}$
$\mathcal{T} = \{T_1, \ldots, T_k\}$: A set of k transforms

$\mathcal{K} = \{K_i = X_i X_i^\top\}_{i=1,\ldots,s}$ (current set of kernel matrices)
$BestEval = \texttt{Evaluate}(\mathcal{K}, \mathcal{L})$ (best evaluation so far)
$Q = [(\mathcal{K}, [], BestEval)]$ (priority queue)
$W^* = []$ (optimal list of transforms so far)

while not empty Q **do**
  $(\mathcal{K}, W, E) = Q.\texttt{Dequeue}()$
  if $E > BestEval$ **then**
    $BestEval = E$
    $W^* = W$
  end
  foreach $T_i \in \mathcal{T}$ **do**
    $\mathcal{K}' = \texttt{Transform}(\mathcal{K}, T_i)$
    $Eval = \texttt{Evaluate}(\mathcal{K}', \mathcal{L})$
    if $\texttt{AcceptTransform}(Eval, E)$ **then**
      $Q.\texttt{Enqueue}((\mathcal{K}', [W|T_i], Eval))$
    end
  end
end

Output: W^* : Optimal list of transforms

Algorithm 1: General Search Strategy for Transfer kernel Meta-learning (TKML).

different sequences of length v. This dimension represents the size of the hypothesis space. As hypothesis space size increases, we can expect higher accuracies of the optimal sequence generated by the algorithm. However, there is also the danger of overfitting the source data, producing sequences that will not generalize well to other data and tasks.

One way to keep the size of the hypothesis space small, while maintaining good performance, is to utilize the granularity of the available transformations. In particular, we can use a coarse-grained set $\mathcal{T}$ containing more complex transformations (i.e. transformations which are composition of other simpler transformations). We will give an example of application of this criterium in the following experiments.

6.2. TKML strategy: Experimental Setting

At the end of the challenge, we were provided with labels for both the validation datasets and for the transfer datasets for all problems. We now present additional experiments based on these new datasets. Specifically, we were curious to see if adding transfer labels to the validation process could improve results. For each challenge problem, we used only a subsample of the transfer set and corresponding labels, with size equal to the validation and final datasets.

We applied the general strategy described earlier to find the optimal set of transformations. The evaluation of a sequence of transforms (function Evaluate() in Algorithm 1) is performed by averaging the ALCs obtained on the tasks defined on the two source datasets. The acceptance condition (function AcceptTransform() in Algorithm 1) verifies whether adding a new kernel transform improves the performance of the sequence.

As already stated, a crucial factor is the choice of transforms, $\mathcal{T}$. After preliminary experiments performed on validation datasets and considering the criteria presented in Section 6.1, we have chosen the small set of transforms given in Table 2. Note that most of these transforms actually consist of sequences of simpler transforms. We also have performed experiments using a larger set of transforms and a finer selection of the transformations. As expected, the algorithm tended to overfit some source tasks in this case.

6.3. TKML strategy: Results

The initial seed kernel for all five challenge datasets is the linear kernel $K = XX^\top$, centered and scaled to trace m, the number of examples of the set (i.e. *trace standardization*). The validation and transfer datasets have been used as source datasets. The results for the five datasets are as follows (refer to Table 2 for a detailed description):

- AVICENNA: T_2, T_3, T_1, T_3 for 3 times;
- HARRY: T_2 for 3 times, T_3, T_2, T_3 for 3 times;
- RITA: T_3 for 4 times, T_1, T_3 for 6 times, T_1, T_3;
- SYLVESTER: T_5, T_3 for 10 times;
- TERRY: T_1 for 2 times, T_3 for 3 times, T_4.

Table 2: The five transforms (the set $\mathcal{T}$ of the TKML strategy) used in our experiments.

1	Spectral $\sigma(\lambda) = \lambda^{\sqrt{2}}$ (a) Centering (T_c) and trace standardization (b) Spectral Transform with $\sigma(\lambda) = \lambda^{\sqrt{2}}$ (c) Normalization
2	Spectral $\sigma(\lambda) = \lambda^{1/\sqrt{2}}$ (a) Centering (T_c) and trace standardization (b) Spectral Transform with $\sigma(\lambda) = \lambda^{1/\sqrt{2}}$ (c) Normalization
3	Polynomial Transform with $p = 2$
4	HAC Transform with $d = 0$
5	Step Spectral (a) Centering (T_c) and trace standardization (b) Spectral Transform with $\sigma(\lambda_i) = 1$ whenever $i \leq 5$, 0 otherwise (c) Normalization

Table 3: Results obtained with the TKML strategy of Section 6 and comparison with Phase 1 and Phase 2 challenge results. All the results refer to the final datasets. ALC($\mathcal{T}_0$) corresponds to the ALC obtained by the algorithm we used in the challenge and described in Section 5. ALC($\mathcal{T}_1$) corresponds to the ALC obtained by the TKML strategy with the set of transformations in Table 2. In parenthesis the rank we would have obtained in final challenge results for Phase 1 and 2. Finally, the last two columns report the best ALC obtained by our competitors in Phase 1 and 2.

DATASET	ALC($\mathcal{T}_0$)	ALC($\mathcal{T}_1$)	BestALC Phase 1	BestALC Phase 2
AVICENNA	0.223646	0.221256 (1,2)	0.218265	**0.227307**
HARRY	0.704331	0.815374 (1,2)	0.806196	**0.861933**
RITA	0.495082	**0.507535** (1,1)	0.489439	0.502948
SYLVESTER	0.456948	0.547636 (3,4)	0.582790	**0.593283**
TERRY	0.846407	**0.849543** (1,1)	0.843724	0.843724

In Table 3, the results on the final datasets are reported. Interestingly, there is a clear improvement when additional source tasks are used. Sometimes this improvement is dramatic, as in HARRY and SYLVESTER, two datasets where the strategy used in the challenge particularly suffered from overfitting. The new strategy is very competitive with other challenge participant entries as it gives the best results on the RITA and TERRY datasets, and the second best results for AVICENNA and HARRY datasets.

7. Final remarks

We conclude the paper with some final considerations about the proposed paradigm and future work.

Suitability of the method for Transfer Learning. The method proposed in this paper seems particularly well suited for transfer learning tasks, as it tries to learn a set of (unsupervised) kernel transformations. On the other hand, standard methods for semi-supervised learning typically optimize an objective function which needs of i.i.d. labeled data. We advocate that learning a sequence of data transformations should make the obtained solution depend more on the domain and less on the particular tasks used for optimization.

Needs of few labeled data. The method is expected to require very few labeled examples, since the labels are used only for validation. Although the ALC measure clearly depends on particular tasks for which it is computed, in the UTL challenge, the ALC measure is computed averaging over several binary tasks. This characteristic is important to prevent possible overfitting with respect to each particular task.

Generality of the method. The method is able to combine very different kinds of kernels that individually perform well on varied domains. For example, the HAC based kernel seems to be suited for data with some structure. In particular, this transform has been crucial to obtain the best result on TERRY, a dataset related to textual data. Another example, exponentiating the kernel matrix (via the eigenvalues obtained by its spectral decomposition) produces an orthogonalization of the features together with a reweighting of its components. This transform turned out to be very useful for all the challenge datasets. Specifically, its contribution is apparent when data lay on a subspace of reduced dimensions, or when a decorrelation of the features, as *whitening* (Duda et al. (2000)) for example, can be beneficial. Finally, as briefly discussed in this paper, additional kernel transforms can be plugged into the algorithm proposed.

Low computational burden. The computational requirements mainly depends on SVD computations needed for the T_σ transform. Interestingly, note that computing the T_σ transform with different parameters requires only a single SVD. In fact, let $K = UDU^\top$ be the SVD decomposition of K, then, any different transform with exponent q can be obtained by a matrix multiplication $K' = HH^\top$ where $H = UD^{\frac{q}{2}}$ (since D is diagonal, this last computation does not affect the computational complexity of the transform and it is fast to compute). We used Scilab[5] for the linear algebra routines, such as the SVD computation and matrix manipulation. C++ has been used for the computation of the HAC based kernel and for combining kernels.

Future works on Transfer Learning. A study about the connection between ours and other paradigms, such as deep learning, will be the subject of future work. In fact, our method can be considered a sort of 'deep kernel learning' similar in principle to a deep learning architecture where kernel transformations correspond to the different levels of a deep neural network. However, unlike deep learning, we do not use explicit representation of data as examples are represented on the basis of similarities with other examples. In the near future, we plan to investigate whether this method can be extended to transfer knowledge across varied domains. It would be interesting to define a metric between chains of transformations obtained in such a setting. This metric can be used to decide what type of knowledge could be transferred from one domain to another based on domain similarity.

5. http://www.scilab.org/

Acknowledgments

This work was supported by ATENEO 2009/2011 "Methods for the integration of background knowledge in kernel-based learning algorithms for the robust identification of biomarkers in genomics". We warmly thank the challenge organizers, and Isabelle Guyon in particular, for their support, Dav Zimak and the anonymous reviewers for their useful comments and suggestions.

References

Fabio Aiolli, Giovanni Da San Martino, and Alessandro Sperduti. A kernel method for the optimization of the margin distribution. In *International Conference on Artificial Neural Networks (ICANN)*, pages 305–314, 2008.

Rich Caruana. Multitask learning. In *Machine Learning*, pages 41–75, 1997.

O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006. URL http://www.kyb.tuebingen.mpg.de/ssl-book.

R.R. Coifman and S. Lafon. Diffusion maps. *Applied and Computational Harmonic Analysis: Special issue on Diffusion Maps and Wavelets*, 21:5–30, 2006.

R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000. ISBN 0471056693.

I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D.W. Aha. Unsupervised and transfer learning challenge. In *International Joint Conference on Neural Networks (IJCNN)*, pages 793–800, 2011.

Gert R. G. Lanckriet, Nello Cristianini, Peter L. Bartlett, Laurent El Ghaoui, and Michael I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knoweledge and Data Engineering*, 22(10):1345–1359, October 2010.

B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors. *Advances in Kernel Methods—Support Vector Learning*. MIT Press, Cambridge, MA, 1999.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, July 1998.

John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521813972.

Xiaojin Zhu and Andrew B. Goldberg. *Introduction to Semi-Supervised Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2009.