TL by Kernel Meta Learning

Fabio Aiolli
University of Padova (Italy)

Kernels

- Given a set of m examples, a kernel

$$K = \Phi\Phi^\top$$

is a positive semi-definite (PSD) matrix

- Φ is the matrix where the features of examples in a (possibly infinite) feature space are accommodated as rows.

- Well known facts about kernels and PSD matrices:
  - The matrix K always diagonalizable
  - Every PSD matrix is a kernel

$$K = UDU^\top = (UD^{1/2})(UD^{1/2})^\top = HH^\top$$

UTL Challenge Terminology

- Domain (or problem):
  - we have 5 different domains in the UTL challenge (A,H,R,S,T)
  - each of them is a multi-class problem
- Dataset:
  - classes in the domain and the associated examples has been assigned to one of 3 datasets (development, valid, final)
  - each dataset contains a subset of examples for a subset of the classes of the domain
- Task:
  - multiple binary tasks are defined on each dataset by partitioning in different ways the classes of each dataset in two parts (positive and negative) for that task

UTL Challenge - Scoring

- AUC: measures the goodness of the ranking produced
- ALC: measures how good the ranking is when very few (1-64) examples are given as training set

- To make the scoring less dependent from particular tasks, given a dataset, the ALC obtained for different tasks on the same dataset are averaged

UTL Challenge – The Classifier

- Let X contain the set of examples in a dataset
- Linear classifier: $w = X^\top Y$ where $Y_i \in \{-\frac{1}{m_-}, +\frac{1}{m_+}\}$
- Scoring function: $f_X = Xw = \underbrace{(XX^\top)}_{K} Y$
- For each example, $f(x)$ represents the difference between the average values of $K(x, \cdot)$ with positive and negative examples

$$f(x) = \mathrm{AVE}_{z \in +} K(x, z) - \mathrm{AVE}_{z \in -} K(x, z)$$

The Ideal kernel

- Very Good Representation + Poor Classifier
  - Good Classification
- Poor Representation and Very Good classifier
  - Poor Classification

- The IDEAL kernel: $K^*(x_i, x_j) = y_i y_j$
  - In this case, even the simple classifier used in the challenge would give a perfect classification!
  - Clearly we do not have label information on all the patterns!
- The STUPID kernel: $K(x_i, x_j) = k$

Kernel Learning Idea

- Learn K by maximizing a kind of agreement between available data labels and the obtained classifier output (e.g. by maximizing alligment, or minimizing SVM dual value)
- Tipically this is made by searching for a convex combination of predefined kernel matrices

- SDP in general!
- More importantly it needs of *i.i.d.* samples! Then, not directly applicable to the challenge.

Kernel Meta-Learning

- Kernel learning focuses on learning a kernel suitable for a given target task in a given dataset
- Instead, we propose to focus on HOW a good kernel can be algorithmically learned starting from the data, "independently" from the particular dataset!

- Kernel meta-learner:

  a learner which learns how to learn kernels from data!

- The basic intuition: if two datasets are related, then an algorithm producing good kernels for a source dataset should be able to produce good kernels for the target dataset too!

KML Algorithm

- Learn a chain of kernel transformations able to transform a basic (seed) kernel defined on a source dataset into a good kernel for the same dataset
- Validation on available labeled data will guide the search for which transformations to use

$$
\begin{aligned}
&K_{X_S}(0) \\
&K_{X_S}(1) = \tau_1(K_{X_S}(0)) \\
&K_{X_S}(2) = \tau_2(K_{X_S}(1)) \\
&\cdots \\
&K_{X_S}(t) = \tau_t(K_{X_S}(t-1))
\end{aligned}
$$

- Then, apply the same transformation chain to the target dataset, starting from the seed kernel on target data

$$K_{X_T} = \tau_p(\tau_{p-1}(\cdots \tau_2(\tau_1(K_{X_T}(0)))))$$

- Start from a seed kernel
  - E.g. $K_X(1) = XX^\top$

- On each step $t = 1, \ldots, \bar{t}$
  - Compute $K'_X(t)$ by transforming $K_X(t)$ using an operator $\tau := S^+ \rightarrow S^+$ (more details in the following)
  - Next transformed kernel is obtained by a convex combination of $K_X(t)$ and $K'_X(t)$ i.e.

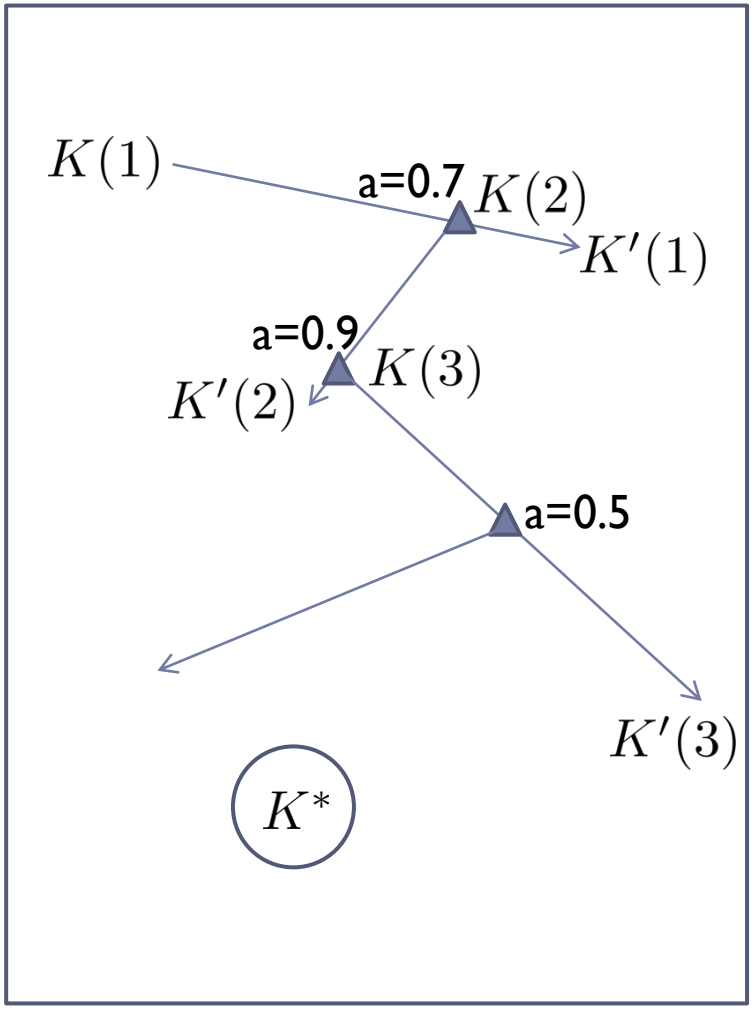
$$K'_X(t+1) = (1-a)K_X(t) + aK'_X(t)$$

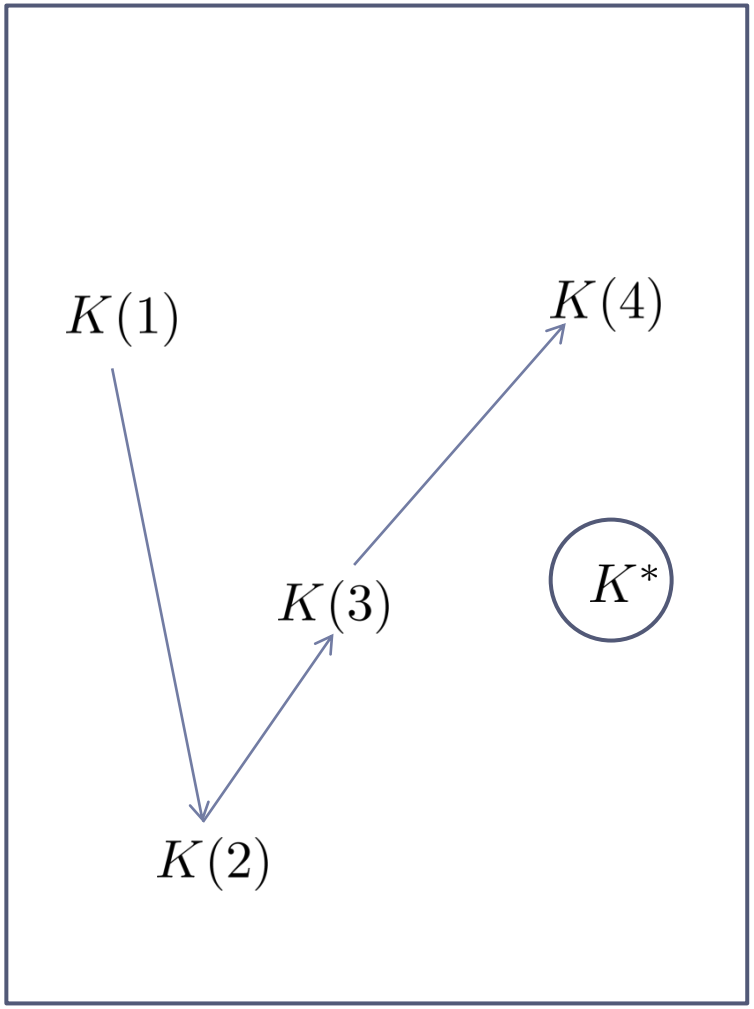
such that

$$a = \arg\max ALC(K'_X(t+1))$$

SOURCE DATASET

TARGET DATASET





Kernel Transforms

- We are interested in defining kernel transformations which do not necessitate of direct access to feature vectors (implicit transformations, kernel trick)

- 4 classes of transforms have been used in the challenge
  - Affine transforms: centering data
  - Spectrum transforms (linear hortogonalization of features)
  - Polynomial transforms (non linear)
  - Algorithmic transforms (HAC kernel)

Affine Transforms: centering

- Centering of examples:

$$\Phi'(x) = \Phi(x) - \frac{1}{m}\sum_i^m \Phi(x_i)$$

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

- Then, this operation in feature space can be seen as a kernel transformation

Spectrum Transforms

- A kernel matrix can always be written as

$$K = \sum_{s=1}^{m} \lambda_s \mathbf{u}_s \mathbf{u}_s^\top$$

where $\{\lambda_1, \ldots, \lambda_m | \lambda_s \geq 0, \lambda_s \geq \lambda_{s+1}\}$ are the eigenvalues and $\mathbf{u}_s$ the eigenvectors of K

- Then any transformation of the form

$$K_\sigma = \sum_{s=1}^{m} \sigma(\lambda_s) \mathbf{u_s} \mathbf{u_s}^\top \qquad \sigma(\lambda) \geq 0$$

produces a valid transformed kernel K_σ

- STEP: (principal directions)

$$\sigma(\lambda) = \begin{cases} 1 & \text{if } \lambda \geq \epsilon\lambda_1, 0 \leq \epsilon \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

- POWER: (soft KPCA)

$$\sigma(\lambda) = (\frac{\lambda}{\lambda_1})^q, q \in \mathbb{R}^+$$

  - After performing an hortogonalization of the features, the weight of components having small eigenvalues are relatively reduced (q>1) or increased (q<1). Lower q's result in more 'complex' kernel matrices.

Polynomial-Cosine Transforms

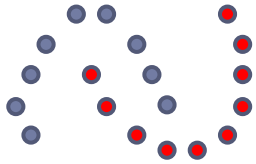
- The transformations above do not change the feature space but they only transform the feature vectors.
- Sometimes this is not sufficient to meet the complexity of a classification problem
- So, we propose a non-linear poly-based transform:

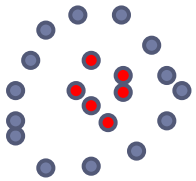
$$
\begin{aligned}
K_\pi(\mathbf{x}_1, \mathbf{x}_2) &= \frac{1}{(1+u)^p}\left(\cos(\Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2)) + u\right)^p \\
&= \frac{1}{(1+u)^p}\left(\frac{K(\mathbf{x}_1,\mathbf{x}_2)}{\sqrt{K(\mathbf{x}_1,\mathbf{x}_1)}\sqrt{K(\mathbf{x}_2,\mathbf{x}_2)}} + u\right)^p
\end{aligned}
$$

- This non linear transformation has the effect of deemphasizing further similarities of dissimilar patterns.

HAC Transforms

- The above kernel is local. It does not consider the global structure (topology) of examples in a feature space (patterns can be considered more similar if they are similar to similar patterns)



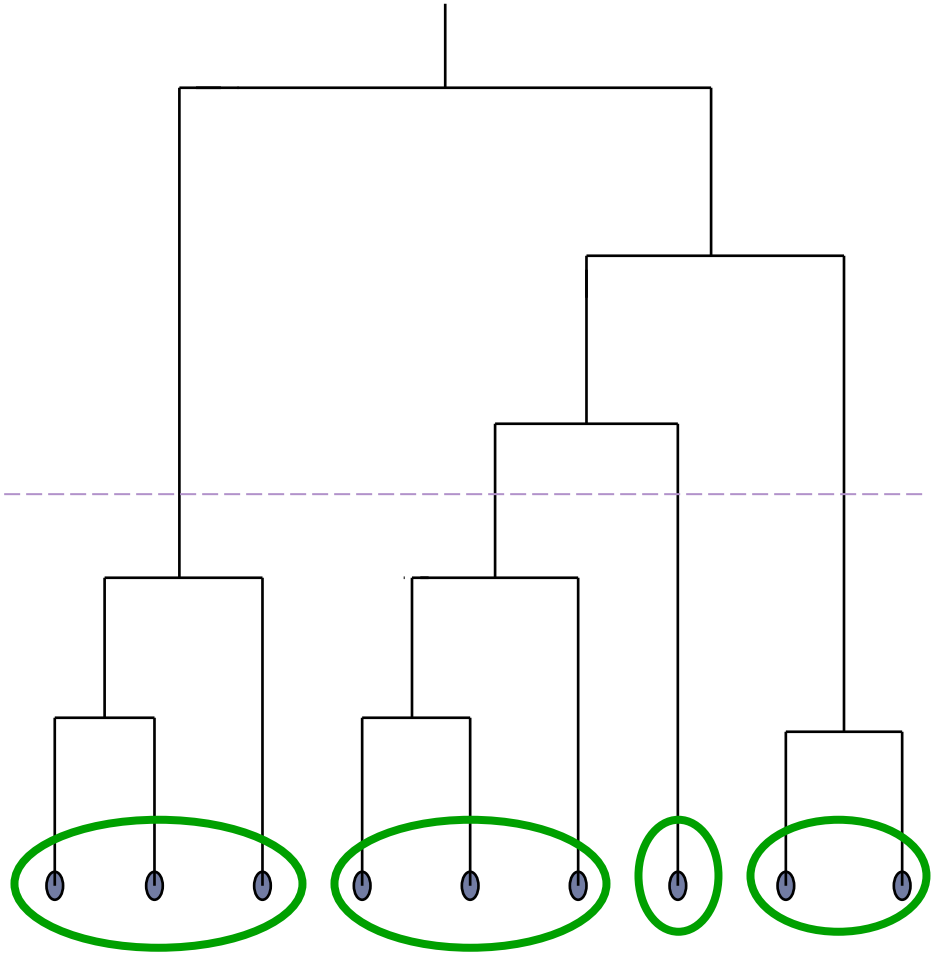


- Hierarchical Agglomerative Clustering (HAC) is a very popular clustering algorithm
- It starts by treating each pattern as a (singleton) cluster and then it merges pairs of clusters until a single cluster is obtained containing all the examples
- This process produces a dendrogram, i.e. a graphical tree-based representation of the cluster produced

The HAC dedrogram

Clustering obtained by cutting the dendrogram at a desired level

- To merge clusters we need of:
  - A Cluster-Cluster similarity matrix
  - A Pattern-Pattern similarity matrix (kernel)



Cluster-Cluster Similarity

- Single Linkage (SL): similarity between the closest patterns in the two clusters
- Complete Linkage (CL): similarity between the furthest patterns in the two clusters
- Average Linkage (AL): average similarity between patterns in the two clusters

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

$\eta = -\infty$ (CL) $\qquad$ $\eta = 0$ (AL) $\qquad$ $\eta = +\infty$ (SL)

HAC Kernel

- Let $C_t \in \{0,1\}^{m \times m}$ be the binary matrix with entries $C_t(i,j)$ equals to 1 whenever x_i and x_j are in the same cluster at the *t*-th agglomerative step.
- So $C_1 = I$ and $C_m = \mathbf{1}\mathbf{1}^\top$

- HAC Kernel is defined by

$$K_h = \frac{1}{m} \sum_{t=1}^{m} C_t$$

and it is proportional to the depth of the LCA of the two examples in the dendrogram

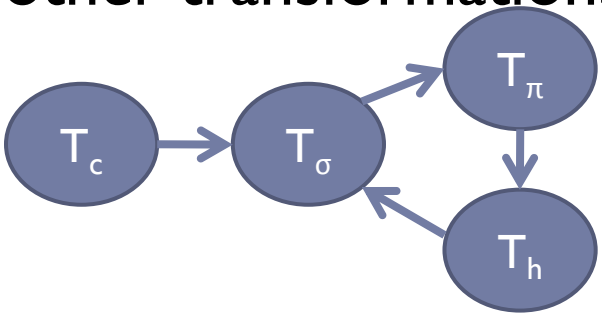
Possible problems in the challenge

- Overfitting the particular dataset
  - If (a too) fine-tuning of the parameters of a kernel is performed there is the risk the kernel overfits data in the particular dataset
  - However, since the validation is performed on different tasks for each dataset (averaging the ALC over the tasks defined on it) the risk to overfit the particular tasks in the dataset is reduced

- Proposed solutions
  - Fix a priori the order of application of transforms (from low complexity ones to higher complexity ones)
  - Limit the set of parameters to try on each transform
  - Transforms are accepted only if the obtained ALC on the source tasks increases significantly

Tricks to avoid over-fitting

- T_c only at the very beginning and only if its application improved upon the raw linear kernel
- After that the other transformations are applied cyclically



- T_σ was validated by a binary search on parameters

$$q = 0.2 \cdot q_0, q_0 \in \{1, \ldots, 10\} \quad a \in \{0, 1\}$$

- T_π was validated with parameters

$$p \in \{1, \ldots, 6\} \qquad a \in \{0, 1\}$$

- T_h was validated with parameters

$$\eta \in \{-10, 0, 10\} \qquad a = 0.05 \cdot a_0, a_0 \in \{0, \ldots, 20\}$$

AVICENNA Results
(Arabic Manuscript, Sparsity: 0%)

RawVal: 0.1034, BestFin: 0.2174(2)

AVICENNA	ValALC	FinALC
Tc	0.124559	0.172279
$T_\sigma(q=0.4)$, a=1	0.155804	0.214540
$T_\pi(p=6, u=1)$, a=1	0.165728	0.216307
$T_h(\eta=0)$, a=0.2	0.167324	0.216075
$T_\sigma(q=1.4)$, a=1	0.173641	**0.223646**(1)

Results on the UTL challenge datasets (Phase I). *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 (its final rank in parentheses). For each dataset, the ALC on the validation and the ALC on the final datasets are reported. Note that only those transforms accepted by the algorithm (a>0) are reported with their optimal parameters.

HARRY Results
(Human Action Recognition, Sparsity: 98.12%)

RawVal: 0.6264, BestFin: 0.806196 (1)

HARRY	ValALC	FinALC
Tc	0.627298	0.609275
$T_{\pi}(p=1, u=0)$, a=1	0.634191	0.678578
$T_{h}(\eta=10)$, a=1	0.861293	0.716070
$T_{\sigma}(q=2)$, a=1	0.863983	0.704331(6)

Results on the UTL challenge datasets (Phase I). *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 (its final rank in parentheses). For each dataset, the ALC on the validation and the ALC on the final datasets are reported. Note that only those transforms accepted by the algorithm (a>0) are reported with their optimal parameters.

RITA results
(Object Recognition, Sparsity: 1.19%)

RawVal: 0.2504, BestFin: 0.489439 (2)

HARRY	ValALC	FinALC
Tc	0.281439	0.462083
$T_\pi(p=5, u=1)$, a=1	0.293303	0.478940
$T_h(\eta=0)$, a=0.4	0.309428	**0.495082**(1)

Results on the UTL challenge datasets (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 (its final rank in parentheses). For each dataset, the ALC on the validation and the ALC on the final datasets are reported. Note that only those transforms accepted by the algorithm (a>0) are reported with their optimal parameters.

SYLVESTER Results
(Ecology, , Sparsity 0%))

RawVal: 0.2167, BestFin: 0.582790 (1)

HARRY	ValALC	FinALC
$T_\sigma(\varepsilon=0.00003),a=1$	0.643296	0.456948(6)

Results on the UTL challenge datasets (Phase I). *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 (its final rank in parentheses). For each dataset, the ALC on the validation and the ALC on the final datasets are reported. Note that only those transforms accepted by the algorithm (a>0) are reported with their optimal parameters.

TERRY Results
(Text Recognition, Sparsity 98.84%)

RawVal: 0.6969, BestFin: 0.843724 (2)

HARRY	ValALC	FinALC
Tc	0.712477	0.769590
$T_\sigma(q=2), a=1$	0.795218	0.826365
$T_h(\eta=0), a=1$	0.821622	**0.846407**(1)

Results on the UTL challenge datasets (Phase I). *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 (its final rank in parentheses). For each dataset, the ALC on the validation and the ALC on the final datasets are reported. Note that only those transforms accepted by the algorithm (a>0) are reported with their optimal parameters.

Remarks (1)

- Expressivity: the method is able to combine different kernels, which individually can perform well on different datasets, in a principled way
  - Spectrum Transform: like a soft-KPCA, very useful when data are sparse (recalls LSI for text)
  - Poly Transform: when data is hardly separable, typically for non-sparse data
  - HAC Transform: for data with some structure. E.g. when they are in a manifold

Remarks (2)

- Few labeled data are needed as they are used for validating the models only
- The method seems quite flexible and additional kernel transforms can be easily plugged in
- Quite low computational burden (one SVD computation for the spectrum transform)
- Learning a sequence of data transformations should make the obtained solution to depend mainly on the domain and far less on particular tasks defined on it

Implementation

- SCILAB for linear algebra routines
  - SVD computations
  - Matrix manipulation
- C++ for HAC computation and for the combination of the produced kernels

Future Work

- I have not been able to participate to the 2nd phase of the challenge ☹
- Labeled data provided in the 2nd phase could have helped to
  - Improve (w.r.t. reliability) the validation procedure
  - Define new kernels based on the similarity with those labeled data
  - Averaging multiple KML kernels (e.g. one for each task)
- In the future we also want to investigate on the connections between this method and other related methods (e.g. the deep learning paradigm).