Feature Selection for Unsupervised Domain Adaptation using Optimal Transport

Léo Gautheron[12], Ievgen Redko[12], Carole Lartizien[2]

{leo.gautheron,ievgen.redko}@univ-st-etienne.fr carole.lartizien@creatis.insa-lyon.fr

[1] Univ Lyon, UJM-Saint-Etienne, CNRS, Institut d Optique Graduate School, Laboratoire Hubert Curien UMR 5516, F-42023, SAINT-ETIENNE, France

[2] Univ Lyon, INSA-Lyon, Université Claude Bernard Lyon 1, UJM-Saint Etienne, CNRS, Inserm, CREATIS UMR 5220, U1206, F-69621, LYON, France

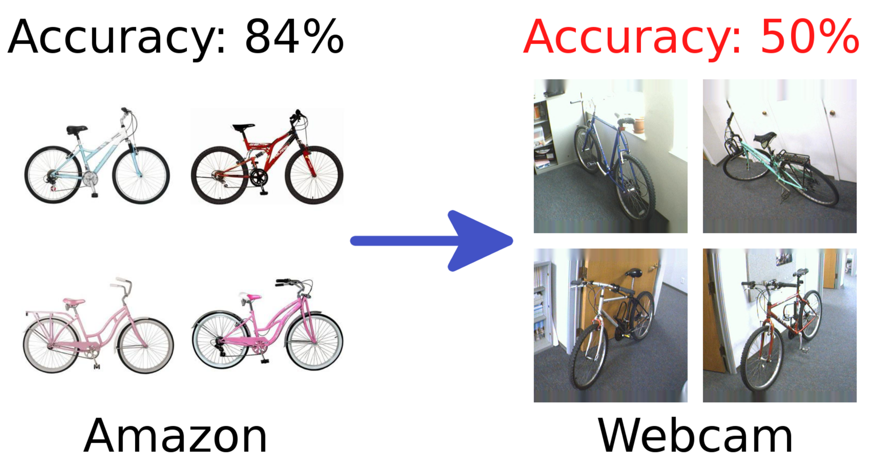
Introduction

Issues of Traditional ML:

- near-human performance is achieved using lots of labeled data
- Some tasks do not have that much labeled data (biology, physics etc)
- There exists too many tasks!

Solution: Domain adaptation

+ **Learn** when *labeled training set* $\boldsymbol{S}$ and unlabeled *test set* $\boldsymbol{T}$ do not follow the same probability distribution.



A **significant drop** in performance due to the discrepancy between training and test distributions!

Discrete optimal transport

Consider two empirical measures defined on $\boldsymbol{S} \in \mathbb{R}^{N_S \times d}$ and $\boldsymbol{T} \in \mathbb{R}^{N_T \times d}$ by

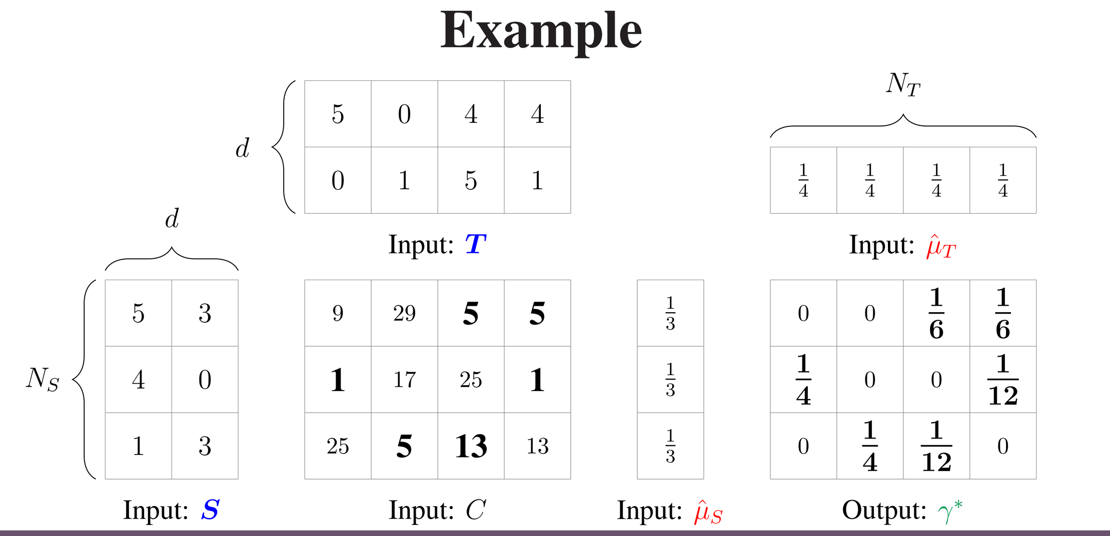
$$\hat{\mu}_S = \frac{1}{N_S}\sum_{i=1}^{N_S} \delta_{x_i^S} \text{ and } \hat{\mu}_T = \frac{1}{N_T}\sum_{i=1}^{N_T} \delta_{x_i^T}.$$

The goal of optimal transport (OT) is to find a coupling matrix $\gamma^* \in \mathbb{R}_+^{N_S \times N_T}$ such that

$$\gamma^* = \underset{\gamma \in \Pi(\hat{\mu}_S, \hat{\mu}_T)}{\arg\min} \ \langle \gamma, C \rangle_F,$$

where $C \in \mathbb{R}^{N_S \times N_T}$ is a transport cost with C_{ij} given by $c : \boldsymbol{S} \times \boldsymbol{T} \to \mathbb{R}_+$ and $\Pi(\hat{\mu}_S, \hat{\mu}_T) = \{\gamma \in \mathbb{R}_+^{N_S \times N_T} | \gamma \mathbf{1} = \hat{\mu}_S, \gamma^T \mathbf{1} = \hat{\mu}_T\}$.

Example



Optimal transport variations

Entropy regularized OT [Cuturi 2013]

$$\gamma^* = \underset{\gamma \in \Pi(\hat{\mu}_S, \hat{\mu}_T)}{\arg\min} \ \langle \gamma, C \rangle_F - \frac{1}{\lambda} E(\gamma)$$

where $E(\gamma) = -\sum_{ij} \gamma_{ij} \log \gamma_{ij}$ is the entropy of γ.

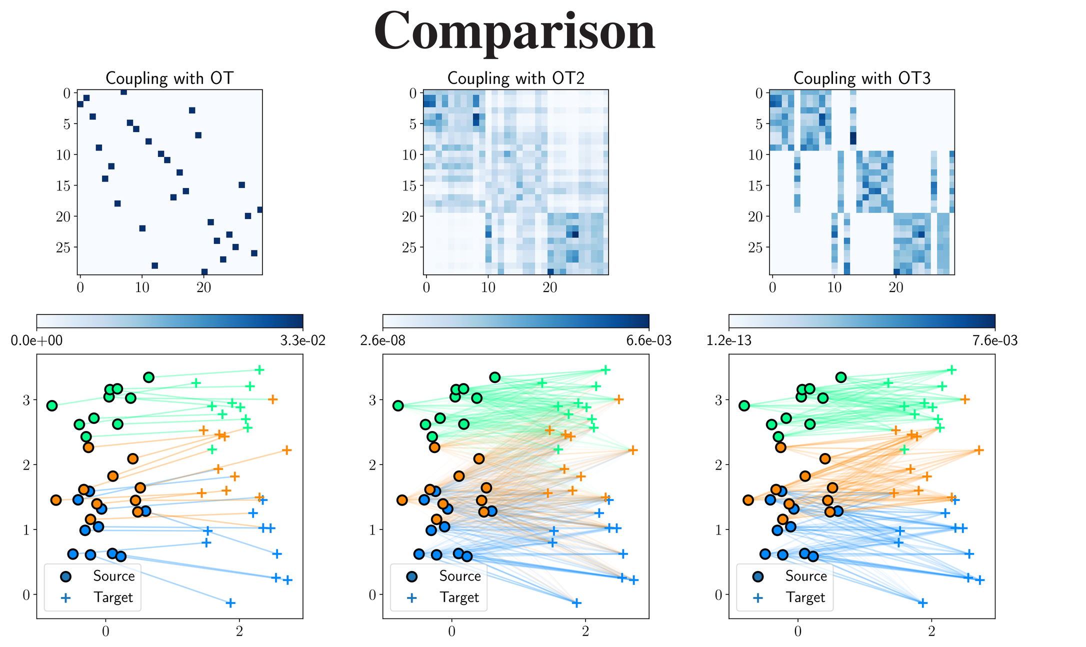
Class regularized OT [Courty et al., 2014]

$$\gamma^* = \underset{\gamma \in \Pi(\hat{\mu}_S, \hat{\mu}_T)}{\arg\min} \ \langle \gamma, C \rangle_F - \frac{1}{\lambda} E(\gamma) + \eta \Omega(\gamma)$$

where $\Omega(\gamma) = \sum_j \sum_{\mathcal{L}} \|\gamma(I_{\mathcal{L}}, j)\|_1^{0.5}$.

In general, the **coupling** can be used to align $\boldsymbol{S}$ and $\boldsymbol{T}$ by using this reweighting: $\boldsymbol{S} \leftarrow N_S \gamma^* \boldsymbol{T}$

Comparison



Our contribution and its algorithmic implementation

Motivation

When $\boldsymbol{S}$ and $\boldsymbol{T}$ are described by the same features, the **discrepancy** between them can be **reduced** by **finding and eliminating the most shifted features**.

Proposed idea

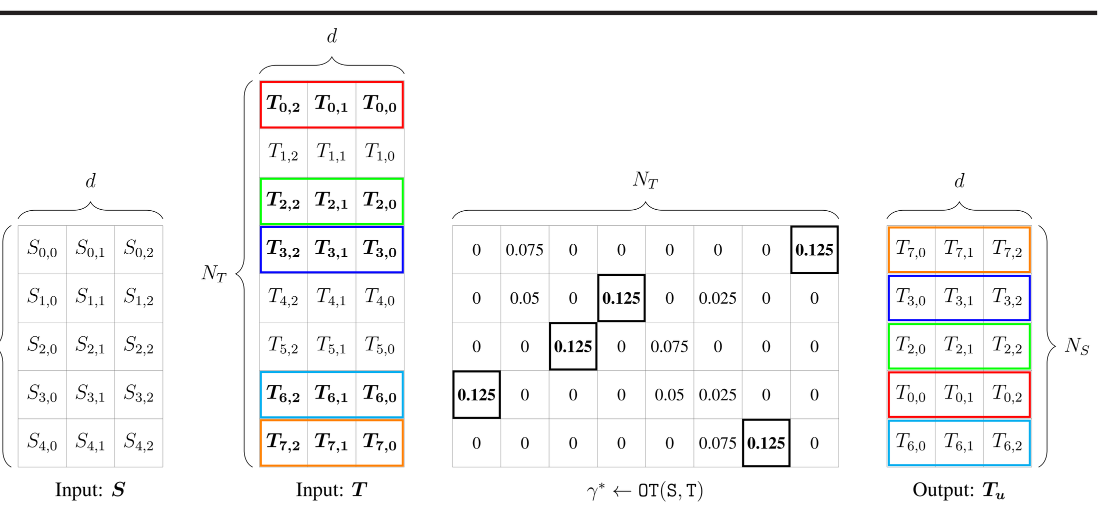
Step 1. Find a **coupling** $\gamma^* \in \mathbb{R}_+^{d \times d}$ **between the features** of $\boldsymbol{S}$ and $\boldsymbol{T}$. The larger γ_{ii}^*, the most similar the feature number i between $\boldsymbol{S}$ and $\boldsymbol{T}$.

Step 2. Sort the features by decreasing similarity between source $\boldsymbol{S}$ and target $\boldsymbol{T}$ domains given in the diagonal of γ^*. Keep the most similar.

Step 1: Sample selection in target domain

Input : $\boldsymbol{S} \in \mathbb{R}^{N_S \times d}, \boldsymbol{T} \in \mathbb{R}^{N_T \times d}$
Output : $\boldsymbol{T_u} \in \mathbb{R}^{N_S \times d}$

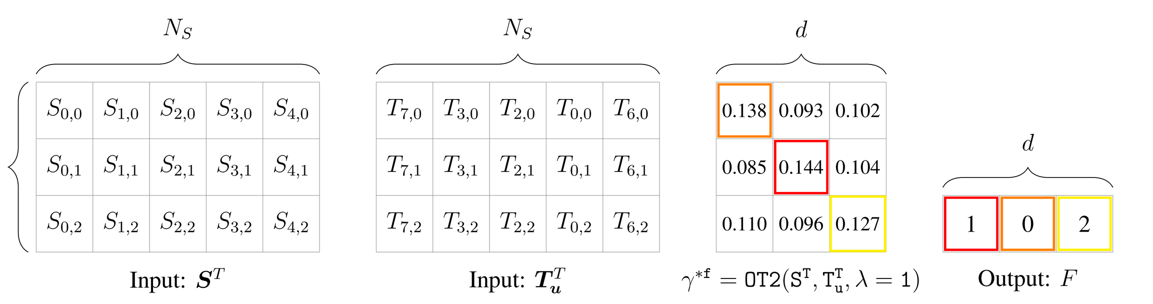
$\mathtt{S} = \mathtt{zscore(S)};\ \mathtt{T} = \mathtt{zscore(T)}$
$\gamma^* \leftarrow \mathtt{OT(S, T)}$
$\mathtt{T_u} \leftarrow \{\mathtt{x_j} \in \mathtt{T} | \mathtt{j} = \underset{\mathtt{i=1,\ldots,N_S}}{\mathrm{argmax}}\ \gamma^*_{\mathtt{ij}}\}$



Step 2: Feature ranking for domain adaptation

Input : $\boldsymbol{S} \in \mathbb{R}^{N_S \times d}, \boldsymbol{T} \in \mathbb{R}^{N_T \times d}$
Output : List F of d most similar features from $\boldsymbol{S}$ and $\boldsymbol{T}$

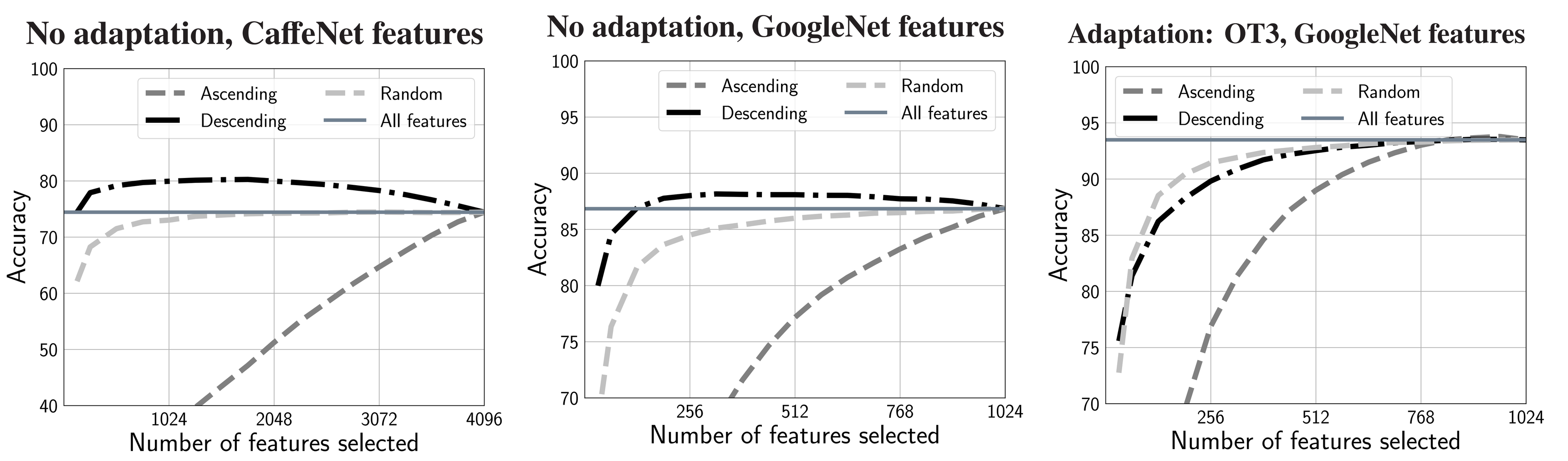
$\mathtt{T_u} \leftarrow \mathtt{Algorithm1}(\boldsymbol{S}, \boldsymbol{T})$
$\mathtt{S^T} = \mathtt{zscore(S^T)};\ \mathtt{T_u^T} = \mathtt{zscore(T_u^T)}$
$\gamma^{*f} = \mathtt{OT2(S^T, T_u^T, \lambda = 1)}$
$\mathtt{F} = \mathtt{argSortDesc}(\{\{\gamma^{*f}\}_{\mathtt{ii}} | \mathtt{i} \in [1, \mathtt{d}]\})$



Office-Caltech benchmark

Data: Images from Amazon(A), Caltech(C), Webcam(W) and DSLR(D) datasets.

Accuracy results



A→W	**77.6±1.9**	20.2±3.5	66.0±4.6	C→A	**83.7±1.8**	38.7±4.5	82.1±2.2	D→A	**75.4±2.1**	20.8±3.8	68.7±2.9	W→A	**81.5±1.2**	18.8±2.4	68.3±3.0
A→C	**74.9±2.0**	29.8±2.4	71.7±3.5	C→D	**76.2±3.6**	24.1±3.4	74.2±4.9	D→C	65.0±2.6	21.5±2.5	**66.6±1.8**	W→C	**72.2±1.1**	23.4±2.1	61.2±2.1
A→D	**78.8±3.5**	20.4±2.8	76.0±3.5	C→W	**75.4±3.5**	20.3±3.2	70.3±5.3	D→W	**92.6±2.0**	32.8±5.1	91.9±1.9	W→D	**96.5±1.5**	49.7±3.2	96.3±1.0
Pairs	↘ 512	↗ 512	4096									Mean	**79.2±2.2**	26.7±3.3	74.4±3.0

Speed-up comparison

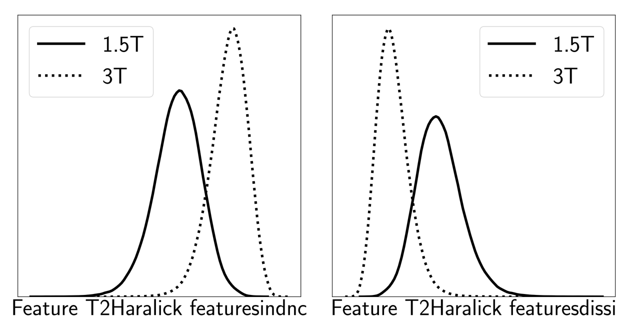
Method	↘512		↘1024		↘2048		4096	
No adapt.	79.2±2.2	0.00s	79.9±2.3	0.00s	80.0±2.2	0.00s	74.4±3.0	0.00s
CORAL	80.5±1.8	110.43s	80.8±1.9	587.69s	80.4±1.7	3996.20s	80.1±1.7	29930.39s
SA	81.8±2.0	13.25s	82.5±1.8	32.09s	82.9±1.7	66.71s	83.0±1.7	169.71s
TCA	83.5±2.2	221.08s	85.0±1.9	223.62s	85.8±1.8	229.48s	85.9±1.7	242.71s
OT3	84.2±2.4	19.50s	86.7±1.9	31.76s	88.8±1.5	54.07s	88.8±1.4	97.47s

Medical application: prostate cancer mapping

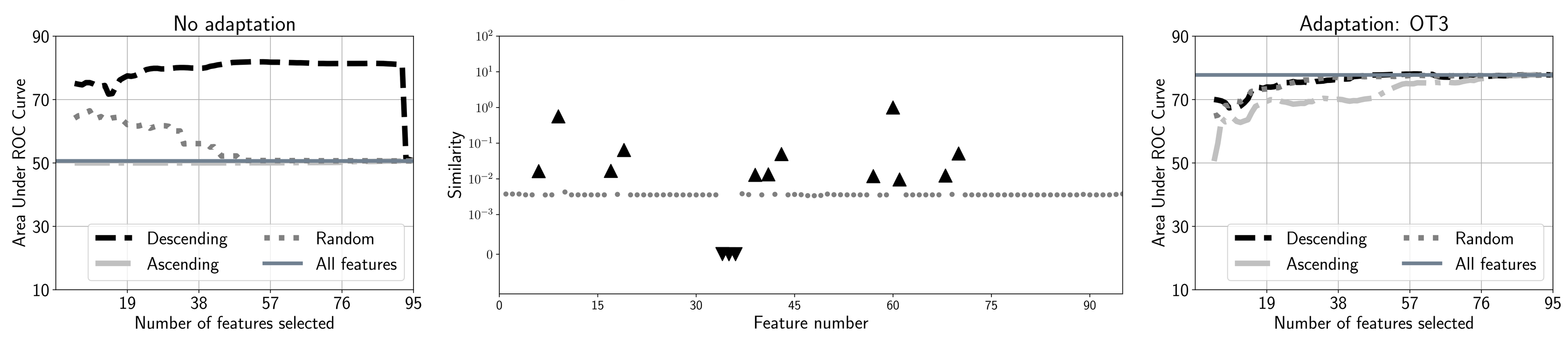
Data: Images from 1.5T and 3T MRI scanners of different resolution with 95 handcrafted features per voxel (3D pixels).

Goal: Learn on 1.5T voxels to predict cancer voxels in 3T images.

Class	#voxels 1.5T	#voxels 3T
Non cancer	363,222	846,556
Cancer	56,126	140,840
Total	419,348	987,396



Obtained results



Conclusion

+ Learning from the selected features gives **improved** performances in less time without adaptation, and **similar** performances in less time when adapting.
+ **Interpretable results** by identifying the most shifted original features.

Try it!

Our source code is available at `https://leogautheron.github.io`

(requires Python Optimal Transport Library `https://github.com/rflamary/POT`)