ADAPT : *Awesome Domain Adaptation Python Toolbox*

A. De Mathelin, F. Deheeger, M. Mougeot, N. Vayatis

Centre Borelli, ENS Paris-Saclay, Michelin

ADAPT

ADAPT is a **Python** package providing some well known **domain adaptation** methods.

The purpose of **domain adaptation (DA)** methods is to handle the common issue encounter in **machine learning** where training and testing data are drawn according to different distributions.

In **domain adaptation** setting, one is aiming to learn a **task** with an estimator f mapping input data X into output data y called also **labels**. y is either a finite set of integer value (for **classification** tasks) or an interval of real values (for **regression** tasks).

Besides, in this setting, one consider, on one hand, a **source** domain from which a large sample of **labeled data** (X_S, y_S) are available. And in the other hand, a **target** domain from which **no (or only a few) labeled data** (X_T, y_T) are available. If no labeled target data are available, one refers to **unsupervised domain adaptation**. If a few labeled target data are available one refers to **supervised domain adaptation** also called **few-shot learning**.

The goal of **domain adaptation** is to build a good estimator f_T on the **target** domain by leaveraging information from the **source** domain. **DA** methods follow one of these three strategies:

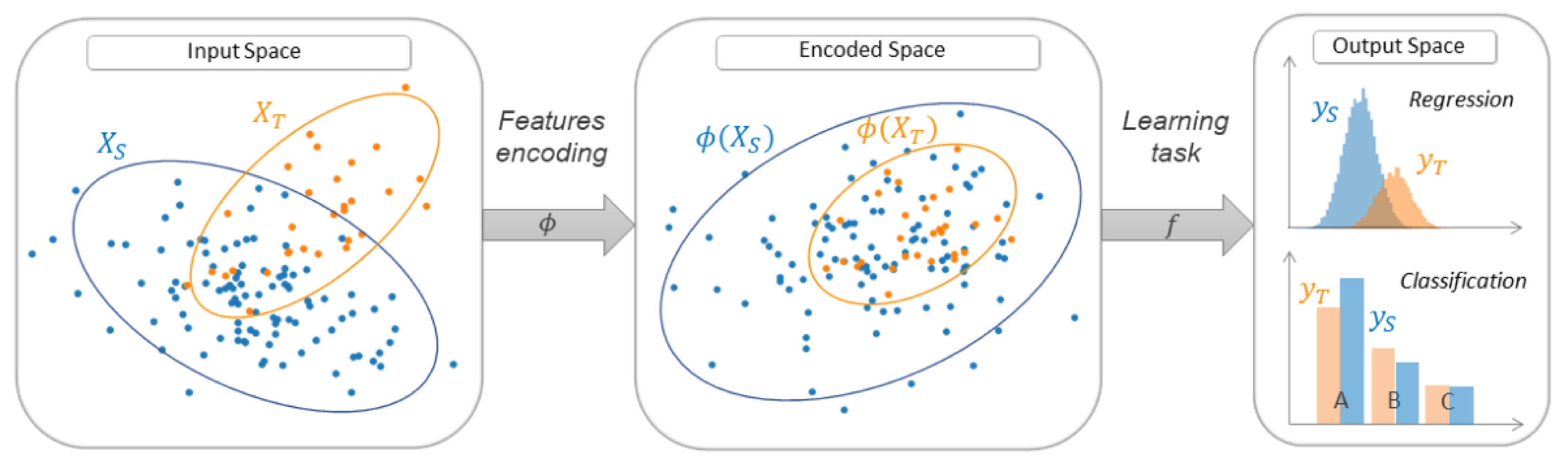
- Feature-Based
- Instance-Based
- Parameter-Based

The following part explains each strategy and gives lists of the implemented methods in the ADAPT package.

adapt.feature_based: Feature-Based Methods

Feature-based methods are based on the research of common features which have similar behaviour with respect to the **task** on **source** and **target** domain.

A new feature representation (often called **encoded feature space**) is built with a projecting application ϕ which aims to correct the difference between **source** and **target** distributions. The **task** is then learned in this **encoded feature space**.



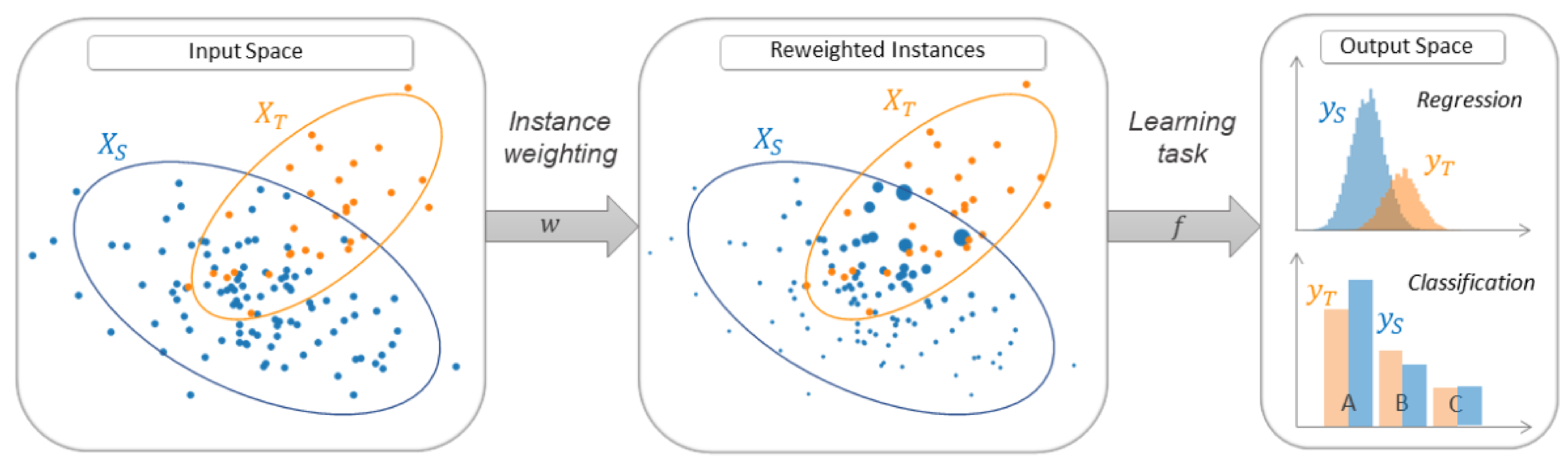
Methods

`feature_based.FE`([get_estimator])	FE: Frustratingly Easy Domain Adaptation.
`feature_based.CORAL`([get_estimator, lambdap])	CORAL: CORrelation ALignment
`feature_based.DeepCORAL`([get_encoder, ...])	DeepCORAL: Deep CORrelation ALignment
`feature_based.DANN`([get_encoder, get_task, ...])	DANN: Discriminative Adversarial Neural Network
`feature_based.ADDA`([get_src_encoder, ...])	ADDA: Adversarial Discriminative Domain Adaptation
`feature_based.mSDA`([get_encoder, ...])	mSDA: marginalized Stacked Denoising Autoencoder.

adapt.instance_based: Instance-Based Methods

The general principle of these methods is to **reweight** labeled training data in order to correct the difference between **source** and **target** distributions. This **reweighting** consists in multiplying, during the training process, the individual loss of each training instance by a positive **weight**.

The **reweighted** training instances are then directly used to learn the task.

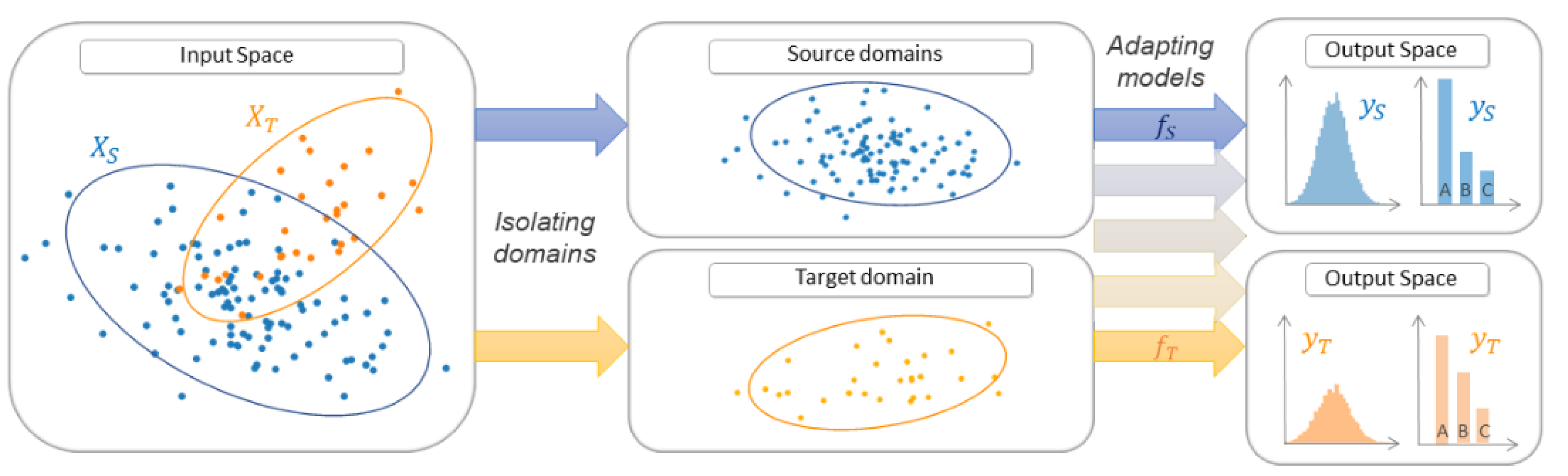


Methods

`instance_based.KLIEP`([get_estimator, ...])	KLIEP: Kullback–Leibler Importance Estimation Procedure
`instance_based.KMM`([get_estimator, B, ...])	KMM: Kernel Mean Matching
`instance_based.TrAdaBoost`([get_estimator, ...])	Transfer AdaBoost for Classification
`instance_based.TrAdaBoostR2`([get_estimator, ...])	Transfer AdaBoost for Regression
`instance_based.TwoStageTrAdaBoostR2`([...])	Two Stage Transfer AdaBoost for Regression

adapt.parameter_based: Parameter-Based Methods

In parameter-based methods, the **parameters** of one or few pre-trained models built with the **source** data are adapted to build a suited model for the **task** on the **target** domain.

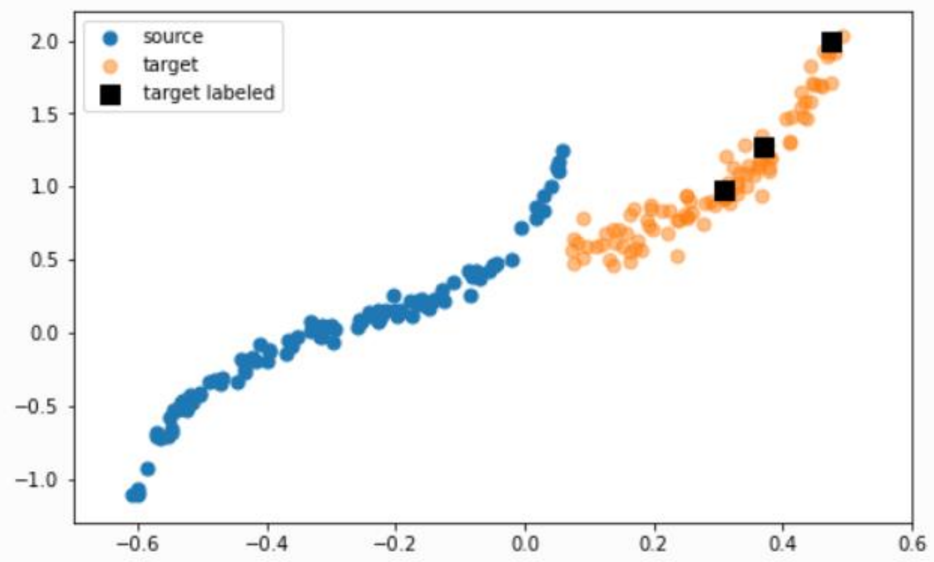


Methods

`parameter_based.RegularTransferLR`([...])	Regular Transfer with Linear Regression
`parameter_based.RegularTransferLC`([...])	Regular Transfer with Linear Classification
`parameter_based.RegularTransferNN`([...])	Regular Transfer with Neural Network

Regression Examples

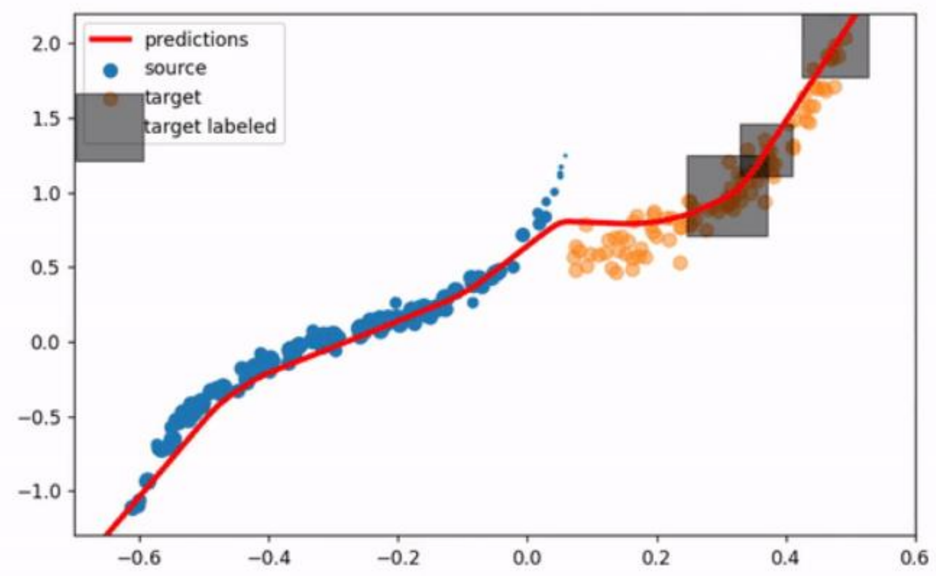
Experimental Setup



As we can see in the figure above (plotting the output data y with respect to the inputs x), source and target data define two distinct domains. We have modeled here a classical supervised DA issue where the goal is to build a good model on orange data knowing only the labels (y) of the blue and black points.

TrAdaBoostR2

We now consider an instance-based method: TrAdaBoostR2. This method consists in a reverse boosting algorithm decreasing the weights of source data poorly predicted at each boosting iteraton.



As we can see on the figure above, TrAdaBoostR2 perfroms very well on this toy DA issue! The importance weights are described by the size of data points. We observe that the weights of source instances close to 0 are decreased as the weights of target instances increase. This source instances indeed misleaded the fitting of the network on the target domain. Decreasing their weights helps then a lot to obtain a good target model.

Pypi Installation

This package is available on Pypi. It has been tested on Linux, MacOSX and Windows for Python versions: 3.5, 3.6 and 3.7. It can be installed with the following command line:

```
pip install adaptation
```

The following dependencies are required and will be installed with the library:

- numpy
- scipy
- tensorflow (>= 2.0)
- scikit-learn

Github Link :

https://github.com/antoinedemathelin/adapt