Minimum Description Length Principle applied to Structure Adaptation for Classification under Concept Drift

Pierre-Alexandre Murena
LTCI CNRS, Télécom ParisTech, Université Paris-Saclay
46 rue Barrault, 75013 Paris, France
murena@telecom-paristech.fr

Antoine Cornuéjols
AgroParisTech
INRA UMR MIA 518 Paris
16 rue Claude Bernard, 75231 Paris, France
antoine.cornuejols@agroparistech.fr

***Abstract*—Traditional supervised machine learning tests the learned classifiers on data which are drawn from the same distribution as the data used for the learning. In practice, this hypothesis does not always hold and the learned classifier has to be transferred from the space of learning data (also called source data) to the space of test data (also called target data) where it is not directly applicable. To operate this transfer, several methods aim at extracting common structural features in the source and target.**

Our approach employs a neural model to encode the structure of data: such a model is shown to compress the information in the sense of Kolmogorov theory of information. To perform transfer from source to target, we adapt a result shown for analogy reasoning: the structure of the source and target models are learned by applying the Minimum Description Length Principle which assumes that the chosen transformation has the shortest symbolic description on a universal Turing machine. We encounter a minimization problem over the source and target models. To describe the transfer, we develop a multi-level description of the model transformation which is used directly in the minimization of the description length. Our approach has been tested on toy examples, the difficulty of which can be controlled easily by a one-dimensional parameter and is shown to work efficiently on a wide range of problems.

I. Introduction

Current machine learning approaches to classification rely primarily on the assumption that the labeled data used for learning (*training* or *learning data*) and the unlabeled data used to evaluate the learning performance (*test data*) are drawn from the same space, using the same distribution. The classification problem then consists of learning a function called *classifier* which takes a point as input and predicts its label. According to this assumption, defining a classifier as a function from the input space $\mathcal{X}$ to the label space $\mathcal{Y}$ and evaluating it on the *test data* makes sense.

In practice, this hypothesis does not always hold: either the data space or the distribution has changed between learning and testing. For instance, the training data (used to learn the classifier) may be selected by experts and therefore be restricted to a smaller region of the input space $\mathcal{X}$ than the test data. This difference in the distribution corresponds with a *concept drift*: the concept, in this case the distributions, drifted from a *source distribution* to a *target distribution*. Using this vocabulary, we will refer to the learning data as *source data* and to the test data as *target data*.

In scenarios of space change or concept drift, classifiers learned from the training data cannot be directly used on the test data: they have to be *transferred* to the test data. In this paper, we will study the case of classification under concept drift, often called *Domain Adaptation*: the system can learn from labeled source data and has to classify unlabeled target data.

Two main families of methods are commonly used for such problems. The first family is based on the estimation of the ratio $p_T(x)/p_S(x)$ for sample selection bias [1], where p_S (resp. p_T) designates the source (resp. target) probability distribution. This ratio is used as a weight in a weighted version of the empirical risk. Various methods have been found to estimate this quantity (see in particular [2], [3], [4]). The second family [5], [6] aims at identifying common features in source and target data.

Our method belongs to the second family. We represent the internal structure of the classes by several neural prototypes. The prototypes are used as a model of the data distribution. To operate the model transfer from source to target, we use the Minimum Description Length Principle (MDLP). This principle assumes that the best theory to describe a system is the one the programming of which has the shortest length on a Turing machine.

The Minimum Description Length Principle has previously been used by Cornuéjols [7] to manage analogy reasoning, a highly similar problem to Domain Adaptation. He demonstrates that using an intermediate model to encode both the description of the inputs and the decision rules is the correct way to apply the symbolic approach of algorithmic complexity required by MDLP to the high-level reasoning of analogy.

We apply this theory to Domain Adaptation seen as an analogy reasoning problem. The model we use to encode the inputs and decision rules is given by a labeled neural network and the decision rules are chosen to be 1-Nearest Neighbor. We demonstrate how such a model corresponds to information compression and apply MDLP to it the same way it is done in analogy reasoning.

II. Application of the Minimum Description Length Principle to Domain Adaptation

A. Domain Adaptation as analogy reasoning

Analogy reasoning is defined by Davies et al. [8] as infering a conclusion $Q(T)$ of a property $P(T)$ knowing that a conclusion $Q(S)$ has been deduced from a source property $P(S)$. The main purpose of analogy reasoning consists of estimating the decision function Q.

The domain adaptation framework corresponds strictly to a problem of this kind: the property $P(S)$ corresponds to the source data X_S and implies the labels Y_S which are the conclusion $Q(S)$. The aim of Domain Adaptation is to deduce the labels Y_T (ie. the conclusion $Q(T)$) from the target data X_T (ie. the property $P(T)$). Knowing the source data and their corresponding labels, domain adaptation consists of finding a decision rule on the source domain and interpolating it to the target domain.

Thus, the results for analogy reasoning can be applied directly to domain adaptation.

B. Minimum Description Length Principle in analogy reasoning

The Minimum Description Length Principle (MDLP) is a principle derived from algorithmic theory of information. Originally expressed in Solomonov's theory of induction [9], the MDLP is formalized by Wallace and Boulton [10] as follows:

The best theory to describe observed data is the one which minimizes the sum of the description length (in bits) of:

- *the theory description*
- *the data encoded from the theory*

The tool to express this principle formally is given by Kolmogorov's theory of information [11]: the information encoded in a string x is defined by this theory as the length (in bits) of the shortest program producing x in a universal Turing machine. This quantity, defined up to a constant, is called *Kolmogorov complexity* and is denoted by $K(x)$. A similar definition is provided for the conditional Kolmogorov complexity of the string x given a string y: $K(x|y)$ is defined as the minimal length of a program taking y as input and producing x.

In Domain Adaptation, we have inputs X_S and X_T for both source and target. Knowing the output Y_S for the source only, the objective is to find the optimal rules β_S and β_T which deduce the outputs Y_S and Y_T (which is unknown). As explained by Cornuéjols [7], applying MDLP directly in analogy reasoning would consist of finding the shortest program describing the target rule β_T from the source rule β_S. This does not make much sense as it strongly relies on the programming on a Turing machine, ie. on the symbol level, and cannot express the task of interpretation as described by Hofstadter [12]. He introduces a slight modification of this direct approach by adding a so-called *model* describing both inputs and decision rules (figure 1). The transfer from source to target is then completely described by the transfer from the source model M_S to the target model M_T. In this framework, MDLP is reduced to the minimization over the models M_S and M_T and the rules β_S and β_T of the objective:

$$K(M_S) + K(X_S|M_S) + K(\beta_S|M_S) + K(M_T|M_S) + K(X_T|M_T) + K(\beta_T|M_T) \quad (1)$$

where X designates the inputs, M the models and β the rules. We use here, and in the rest of the article, the notation S and T to refer respectively to the source and the target.

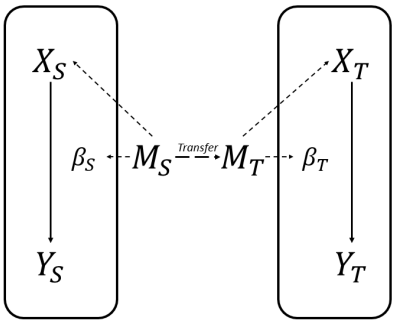


Fig. 1. Summary of Domain Adaptation problem with an underlying model. The transfer task consists of finding the optimal model transition from the source to target in terms of information compression.

C. Implications for Domain Adaptation

As previously explained, this minimization problem can be applied directly to domain adaptation. We can give simple interpretation of the terms in this context. The models M describe the way the distributions are encoded: knowing M, it is possible to describe how the inputs X are generated and how the labels $Y = \beta(X)$ are calculated. The rules β correspond to the classifiers.

Equation 1 can be divided into three terms:

- The first term corresponds to the complexity of the source description: $K(M_S)$ is the complexity of the source model, $K(X_S|M_S)$ is the complexity of the source inputs according to source model and $K(\beta_S|M_S)$ is the complexity of the source classifier according to the source model.
- The second term corresponds to the complexity of the transfer: $K(M_T|M_S)$ is the complexity of the target model according to the source model.
- The third term corresponds to the complexity of the target description: $K(X_T|M_T)$ is the complexity of target inputs according to the target model and $K(\beta_T|M_T)$ is the complexity of the target classifier according to the target model.

It is important to note that calculating Kolmogorov complexity is a NP-hard problem. In the following, we will only consider upper-bounds of Kolmogorov complexity: such bounds are obtained by selecting the program of minimal length not in the whole set of programs of a universal Turing

machine but on a restricted set of programs on which the calculation is possible. In the following, we will abusively assimilate Kolmogorov complexity to this quantity.

In the following sections, we will give an example of data description and modeling based on neural prototypes, and will show how MDLP can be applied to it in practice.

III. Description of the neural model

A. General description

Such as previously described for analogy reasoning, domain adaptation involves two domains: the source and target. The source domain is provided with a labeled data set. The source input vectors are summarized in the design matrix X_S and the labels in the output vector Y_S. The classifier β_S is a function from $\mathcal{X} = \mathbb{R}^d$ to the space of labels $\mathcal{Y}$.

We assume that the input spaces for source and target are the same and that the labels are the same ($\mathcal{X}_S = \mathcal{X}_T$ and $\mathcal{Y}_S = \mathcal{Y}_T$). This implies in particular that no class can be created in the process.

We propose a model similar to the neural networks used for vector quantization. The model is made up of C vectors called *prototypes* (or *centroids*). Each prototype is associated to a class $y \in \mathcal{Y}$. Such a model is frequently used in techniques like Self-Organizing Maps [13].

We give a basic example of domain adaptation in figure 2. Instead of directly transfering the classifier, we transfer the neural model using MDLP. The target model is then used to build the target classifier β_T.

B. Information Compression in the Neural Model

Using a neural model can be interpreted as information compression: rather than describing the whole data set by giving a complete description of each point, the system describes the prototypes fully and gives only a relative description of the input points (figure 3).

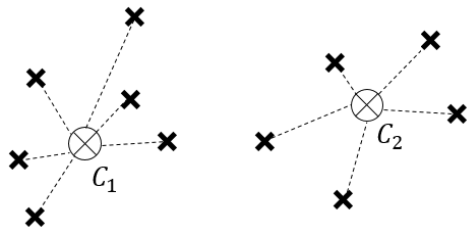


Fig. 3. The prototypes C_1 and C_2 (circled crosses) bring a more compact description of the points $X_1, \ldots, X_n$ (bold crosses). Instead of describing all points by their absolute coordinates, they can be described by their relative position to a prototype.

In the following, we consider that we have a function *size* which returns the size (in bits) of a real number x. We will discuss a simple construction of this function later. Intuitively, this function depends on the module of x. Using this function, a vector $X \in \mathbb{R}^d$ is encoded on $\sum_{i=1}^{d} size(X^i)$ bits, where X^i denotes the i-th component of the vector X.

The direct way of describing the input data $\mathbf{X} = \{X_1, \ldots, X_n\}$ consists of giving the d components of each vector one by one. This corresponds to a Kolmogorov complexity:

$$K(\mathbf{X}) = \sum_{i=1}^{n} \sum_{j=1}^{d} size(X_i^j) \tag{2}$$

On the other hand, the model of Vector Quantization encodes the full description of the C prototypes and the description of the input relatively to the prototypes. Each prototype is encoded by the specification of its key (encoded on $\log_2 C$ bits) and by the complete description of its components one by one:

$$K(\mathbf{M}) = \sum_{c=1}^{C} \left(\log_2 C + \sum_{j=1}^{d} size(M_c^j) \right) \tag{3}$$

To describe a vector, a more subtle approach consists of specifying its relative position to the closest prototype. To be able to return the exact position of the vector, a program also needs the key of the chosen prototype:

$$K(\mathbf{X}|\mathbf{M}) = \sum_{i=1}^{n} \left(\log_2 C + \min_{c=1,\ldots,C} \sum_{j=1}^{d} size(X_i^j - M_c^j) \right) \tag{4}$$

The idea of the compression is that groups of points share a mean position and can be described by their relative position compared to this mean value.

A model is effective when the second description is more compressed than the basic description: $K(\mathbf{X}) \geq K(\mathbf{M}) + K(\mathbf{X}|\mathbf{M})$. Basically, it is obvious that such a compression will be particularly efficient in situations in which many points are close to each other.

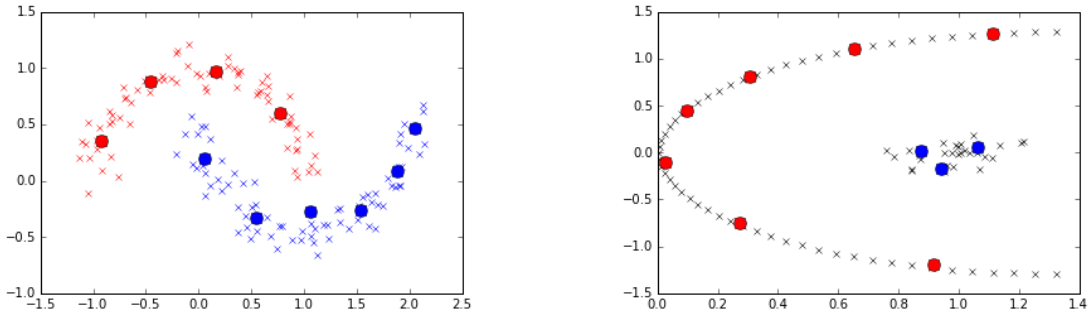
C. Decision rule

A trivial decision rule $\beta_{trivial}$ consists of returning the observed label for each of the input point and any value for other points of the domain. In statistical learning, such a decision rule is said to correspond to *overfitting* as it gives correct results only for input points and is not able to generalize the learning to other points. Furthermore, it does not use the information offered by the neural model.

The most direct non-trivial decision rule consists of associating a point with the class of the closest prototype. The method to extract this information is straightforward: it only consists of computing the distance of a point to all prototypes. Thus, the Kolmogorov complexity of the rule β given a model $\mathbf{M}$ up to a constant is given by:

$$K(\beta|\mathbf{M}) = \log_2 C \tag{5}$$

The problem with this decision rule is that in general it does not describe the source correctly. Actually, it is necessary in our framework that the model M_S explains completely the source data: thus, we may adapt the 1-Nearest Neighbor (1-NN) decision rule by correcting the done errors. If the point



(a) Source data and model

(b) Target data and model

Fig. 2. An example of domain adaptation problem and the corresponding ideal model transfer. The data points are represented by crosses and the prototypes by circles. The color corresponds to he class: blue corresponds to class 0, red to class 1, and black to unlabeled data.

belongs to the input set and the class of the nearest prototype is different from the actual label of the point, the program has to display the result explicitly. If the function y_{NN} denotes the label of the nearest neighbor and $\mathbb{I}$ corresponds to the indicator function, the Kolmogorov complexity of the rule relative to the model and the data is:

$$K(\beta|\mathbf{M},\mathbf{X},\mathbf{Y}) = \log_2 C + \sum_{i=1}^{n}(\log_2 n + size(Y_i))\mathbb{I}(Y_i \neq y_{NN}(X_i)) \quad (6)$$

The better the model predicts the right labels, the lower the complexity of the rule. In the learning step, the minimization of this term will lead to a model which minimizes the prediction error over the data.

D. Algorithmic size of real numbers and vectors

As explained previously, calculating the Kolmogorov complexity is a NP-hard problem. This property is due to the size of the research space: finding the program of minimum length on a Turing machine M requires exploring the entire set of programs encoded on the considered machine. The common approximation consists of considering a restricted set of programs. The complexity obtained on this restricted set is by construction an upper-bound of the Kolmogorov complexity. In particular, the choice of the restricted set of programs defines the function $size$ used previously by the choice of the encoding of real numbers.

A very straightforward encoding of the real numbers is based on a subdivision of the real line in ordered sections of fixed length Δx. A real number $X \in \mathbb{R}$ is described (with a precision Δx) by the index of the section it belongs to. This encoding leads to the following definition of the $size$ function:

$$size(X) = \log \left\lfloor 1 + \frac{|X|}{\Delta x} \right\rfloor \quad (7)$$

The parameter Δx controls the precision of the encoding: two numbers can be distinguished only if their distance is greater than Δx. In practice, the $size$ can be bounded by simpler measures:

$$size(X) \leq \log\left(1 + \frac{|X|}{\Delta x}\right) \leq \frac{|X|}{\Delta x} \quad (8)$$

This upper-bound is particularly remarkable when generalized to vectors $X \in \mathbb{R}^d$: with the previous approximation $K(X) = \sum_{i=1}^{d} size(X_i)$, the Kolmogorov complexity of a vector X is upper-bounded by the L^1-norm, up to the precision parameter Δx. This upper-bound will be used especially in the algorithms derived from the MDLP.

In the following, we will consider only machines which use this encoding of the real numbers and vectors.

E. Learning the Neural Model in the Source only

Knowing inputs, we have to learn the optimal model in the sense of information compression:

$$\underset{\mathbf{M},C,\beta}{\text{minimize}} \quad K(\mathbf{M}) + K(\mathbf{X}|\mathbf{M}) + K(\beta|\mathbf{M})$$

Using the expressions found above, we obtain a well-defined optimization problem, depending on the definition of the function *size*.

From now on, we will consider only an alternative description of the model so that the term $K(\mathbf{M})$ is a constant. In this description, the prototypes are all described in a fixed number of bits. In terms of *algorithmic probability*, for which the probability of a number depends on its Kolmogorov complexity, this stronger hypothesis on the encoding of $\mathbf{M}$ corresponds to considering a uniform prior for the prototype positions, whereas the previous approach provided a higher prior to the prototypes with low-valued coordinates. Using a more subtle description of the neural model could be interesting when the model is based on well-defined structures, eg. when the prototypes are aligned, or belong to a circle. The encoding of such structures and its impact over domain adaptation has to be studied in future works.

According to the assumption that $K(\mathbf{M})$ is constant equal to CL (where L is the fixed number of bits used to describe a single prototype), this first term only depends on the number

of prototypes C. The remaining terms in the MDL equation correspond to the sum of the description length of the data and of the decision rule relative to the description of the model.

For a fixed number of prototypes C, the term we aim to minimize can be developed as a sum over the input points:

$$K(\mathbf{X}|\mathbf{M}) + K(\beta|\mathbf{M}) = \sum_{i=1}^{n} \left(\min_{c} \sum_{j=1}^{d} size(X_i^j - M_c^j) + \alpha \mathbb{I}(Y_i \neq y(X_i)) \right) \quad (9)$$

where $\alpha = \log_2 n + \log_2 n_{classes}$ is the misclassification penalty and $y(X_i)$ designates the class of the nearest prototype of input point X_i. We observe that taking the function $size$ defined as a constant is equivalent to a classical risk minimization. On contrary, when the penalization term is null, the problem is similar to the K-Means problem.

This minimization problem can be solved by EM algorithm: in the E step, the closest prototype is chosen for each data point; in the M step, the positions of the prototypes and their labels are updated in order to minimize the objective with the chosen point-prototype association.

F. Learning the Source and Target Neural Models

In the Domain Adaptation problem, we must minimize the objective function (1) in order to learn the models M_S and M_T and the rules β_S and β_T.

As explained, this objective can be split in three parts. In the source part, the term $K(\mathbf{M}_S)$ is given by equation (3), the term $K(\mathbf{X}_S|\mathbf{M}_S)$ by equation (4) and $K(\beta_S|\mathbf{M}_S)$ by equation (6). In the target part, the term $K(\mathbf{X}_T|\mathbf{M}_T)$ is given by equation (4) and $K(\beta_T|\mathbf{M}_T)$ by equation (5).

The transfer part consists of only one term: $K(\mathbf{M}_T|\mathbf{M}_S)$. This term corresponds to the complexity of description of the target model given the source model. It is essential to bind the source and target situations together. This term is used to make the source and target independent on each other. We discuss in the next section the construction of the neural model transfer.

IV. Transfer of the neural model

To transfer the neural model from the source domain to the target domain, we aim to characterize the transformation of the source model returning the target model. To achieve this, we use a multi-level approach to describe the changes:

- Global transformation: a transformation which affects all prototypes
- Class transformation: a transformation which affects all prototypes belonging to a given class
- Local transformation: a transformation which affects one single prototype

The use of such a multi-level approach enables one to take global effects into account, such as a global translation of the data, or effects shared by points of the same class.

In this framework, the resulting complexity $K(\mathbf{M}_T|\mathbf{M}_S)$ can be split into a sum of three terms corresponding to each level of the total transformation:

$$K(\mathbf{M}_T|\mathbf{M}_S) = K_g(\mathbf{M}_T|\mathbf{M}_S) + K_c(\mathbf{M}_T|\mathbf{M}_S) + K_l(\mathbf{M}_T|\mathbf{M}_S)$$

We will give an upper-bound of each term by studying a class of transformation of each type.

A. Global transformation

The global transformation affects the whole neural structure and corresponds to a movement of all prototypes. The main transformation consists of a global translation of the prototypes: it is described by the coordinates of the translation vector $\Delta\mu$.

In practice, the vector $\Delta\mu$ can be initialized as the difference of means between target data points and source data points: $\mu^{(T)} - \mu^{(S)}$. However, this equality does not necessarily hold. Consider for example the case in which only one class is translated from the source to target. It costs more in terms of Kolmogorov complexity to translate all points by a vector $\mu^{(T)} - \mu^{(S)}$ and then to translate each class individually to its position, than to only translate the right class to its new position.

With this approach, the global transfer complexity K_g is defined by:

$$K_g(\mathbf{M}_T|\mathbf{M}_S) = \sum_{i=1}^{d} size(\Delta\mu^i) \quad (10)$$

For two-dimensional problems ($d = 2$), an intuitive transformation is the rotation. Such a transformation is fully described by the center vector Ω and the angle θ. In higher dimensions, we can generalize by considering the set of affine transformations characterized by a translation vector $\mathbf{u}$ and a linear transformation of matrix $\mathbf{A}$. We won't discuss such a model in this paper.

B. Class transformation

Once the global transformation has been applied to the model, we may wish to characterize the common changes shared by a whole class of points. Such a class transformation is defined relatively to the global transformation.

If l is a class label, we designate by $\Delta\mu_l$ the class translation vector. The complete class transformation is given by the set of vectors $\{\Delta\mu_1, \ldots, \Delta\mu_L\}$ where L is the total number of classes. If $\Delta\mu_{l,i}$ designates the i-the component of the translation vector of class l, the complete class transformation complexity is given by:

$$K_c(\mathbf{M}_T|\mathbf{M}_S) = \sum_{l=1}^{L} \sum_{i=1}^{d} size(\Delta\mu_l^i) \quad (11)$$

Such as for the global transformation, the class transformation can be refined in order to include a general linear transformation (such as a class rotation), but we won't consider this generalization here.

C. Local transformation

The local transformation is the residual transformation to describe completely the position of a target prototype given the source neural model $\mathbf{M}_S$. This transformation is applied to each prototype after the first two transformations.

A local transformation can consist of three actions:

- Move a point: this action is encoded by the relative position vector.
- Create a point: this action is encoded by the class index and the relative position vector in the class.
- Delete a point: this action is encoded by the index of the point to suppress.

In terms of complexity, the creation of a prototype costs more than the suppression: this observation is consistent with the intuition that it is easier to simplify the model than to complicate it.

The relative position vectors are put together in the local transformation matrix $\Delta\mathbf{M}$. For each prototype i of class l, the local transformation $\Delta\mathbf{M}_i$ of prototype i is defined as:

$$\Delta\mathbf{M}_i = \mathbf{M}_i^{(T)} - \mathbf{M}_i^{(S)} - \Delta\mu_l - \Delta\mu \tag{12}$$

We will denote by n_{del} the number of prototypes to delete and by $\mathbf{N}$ the matrix of positions for the n_{add} points to add. Deleting the points in the source model (made up of C_S prototypes) has a complexity $K_{del} = n_{del}\log_2 C_S$. Once the points have been deleted, adding new points has a complexity:

$$K_{add} = n_{add}\log_2 n_{add} + \sum_{i=1}^{n_{add}}\sum_{j=1}^{d} size(N_i^j) \tag{13}$$

By combining all these complexities, we obtain the complexity of the local transformation:

$$K_l(\mathbf{M}_T|\mathbf{M}_S) = \sum_{i=1}^{C_S - n_{del}}\sum_{j=1}^{d} size(\Delta M_i^j) + K_{del} + K_{add} \tag{14}$$

D. Total description of the target model

Given the target model $\mathbf{M}_S$ and with the previously described transformations, the transfer to $\mathbf{M}_T$ is described by a vector $\Delta\mu$, a set of vectors $\{\Delta\mu_1, \ldots, \Delta\mu_L\}$ (where L denotes the total number of classes), a matrix ΔM corresponding to the individual local transformations, the coordinates $\mathbf{N}$ of the created points and the index of the n_{del} points to delete.

A practical construction of the target model consists of:

1) Deleting the specified points
2) Applying a translation of vector $\Delta\mu$ to each point
3) For each class l, applying a translation of vector $\Delta\mu_l$ to all points in the class
4) Applying the translation of vector $\mathbf{\Delta M}_i$ to each point i.
5) Concatenating the new points $\mathbf{N}$.

The corresponding Kolmogorov complexity is given by:

$$\begin{aligned} K(\mathbf{M}_T|\mathbf{M}_S) =& K(\Delta\mu) + \sum_{l=1}^{L} K(\Delta\mu_l) + K(\mathbf{\Delta M}) \\ &+ K(\mathbf{N}) + n_{del}\log_2 C_S \end{aligned} \tag{15}$$

We can note that in our current elementary model, it is pointless to add a prototype in the target model: as the prototypes are implicitly supposed to be independent, we have no way to determine the class of the added prototype. Nevertheless, this function could be useful in a more advanced model. In the following, we will consider only prototype deletions.

E. Interpretation of the multi-level approach

The multi-level approach to describe the transfer of the neural model from the source to the target is a reduction of the set of programs used to calculate an upper-bound of the actual Kolmogorov complexity $K(\mathbf{M}_T|\mathbf{M}_S)$.

It works in precisely the same way as the neural model with the point description: the three levels are used to refine the representation of the movement. The global movement is encoded only once at the top level and not repeated for each point; and the singular movements which can only be described individually are described at the bottom level.

The choice of three levels here is arbitrary: it could be possible to work with any other number of levels. Our choice is mainly motivated by the simple interpretation which can be done of the results with this point of view.

In the general case, determining the number of levels to use is a central problem: it can be done by applying the MDLP. A large number of levels will make the movement descriptions more compact but will require a larger description, and thus are not necessarily optimal. On contrary, when the neural model consists of a low number of prototypes, the shortest description can be provided by only one or two levels. A work has to be done to investigate this problem.

F. Transfer learning

Transfer learning consists of evaluating the models M_S and M_T. As explained, this transfer is done by minimization of the objective function (1) over the models with the expressions described above (equations 4, 5, 6, 15). In practice, it is more convenient to minimize over the transformation parameters $\Delta\mu, \Delta\mu_c$ and ΔM than over the target model M_T.

When no prototype deletion or adding is considered, the optimization problem is solved directly by EM algorithm. In the E step, the points of both source and target domains are attached to the closest prototype in the corresponding model; in the M step, the source prototypes and the transformation parameters are learned by a subgradient method with the fixed point-prototype association. The optimal value of C is the one which leads to the lowest complexity.

Deleting points can be done after a first model optimization. The points to delete are chosen in order to compensate the deletion cost by a compression gain in the description of the

reduced model. In practice, we run the algorithm with the same number of points in the source and in the target, and delete the points one by one. The deletion is performed in two cases: either the prototype in the target model is useless (ie. no data point is attached to it) or two prototypes in the target have the same class and are so close to each other that deleting one doesn't affect the global result of the classification.

V. Experimental results

A. Measuring the quality of transfer

Domain adaptation is not a well-posed problem; consequently, even if it is possible to define a *classification error rate* over a labeled target set, this quantity does not measure exactly the efficiency of a transfer method. A transfer learning problem has multiple solutions, our approach consisting of selecting the most simple solution in terms of algorithmic complexity. In some problems, even human experts cannot make the distinction between two solutions and, in this sense, penalizing an inversion of two classes in the result of a method would seem to be arbitrary.

The misclassification rate (or error rate) expresses how far the classification results are from the actual labels. This rate can be calculated for source and target data (as the source model and the drift are learned simultaneously). Given a set of points $\{X_1, \ldots, X_n\}$ and their respective labels $\{Y_1, \ldots, Y_n\}$, the misclassification rate of a classifier y is defined as:

$$R = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(Y_i \neq y(X_i)) \tag{16}$$

Because of the previous observation, this quantity has to be considered carefully as a high misclassification rate does not necessarily imply a bad transfer.

Actually, the misclassification rate is a normalized complexity of a basic program designed to correct the classification errors: this program modifies only the points which are misclassified. When the value of R is close to 1, the correction can be done by permuting all the classes, which would have a lower complexity.

B. Toy examples

We test our method on two-dimensional toy examples built artificially. We consider two parameterized problems:

- Class translations: the input points are generated by two normal distributions. The drift consists of a translation of the means of each of the distributions.
- Class deformation (figure 4): one of the class is continuously deformed from a vertical line to a circle surrounding the second class. The deformation is parameterized by a real number $\theta \in [0, 1]$.

The class deformation problem provides a good way to parameterize the difficulty of the transfer. When $\theta = 0$, the points in the deformable class are aligned on a vertical line at the left of the points in the fix class. When θ increases, the points of the deformable class surround progressively the fix class. At $\theta = 1$, they are aligned on a circle centered on fix class.

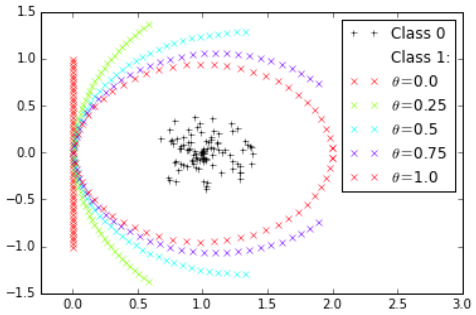


Fig. 4. Toy problem with various difficulty levels. The distribution of class 0 (plotted as black +) doesn't change. The distribution of class 1 (plotted as colored crosses) is parameterized by a real number θ. When $\theta = 0$, the points are aligned on a vertical line; when $\theta = 1$, the points are distributed on a circle surrounding class 0.

C. Results and discussion

The **class translation** problem has been tested on automatically generated sets of 200 points in $\mathbb{R}^2$. In the source, the first class is generated by a normal distribution centered on $(0, 0)$ and the second class by a normal distribution centered on $(2, 0)$. Both distributions have identity covariance matrix. In the target, the same distribution is used for the first class, but the second class is derived from a normal distribution centered on $(t, 0)$.

The results obtained for the transfer from source to target highly depend on the parameter $t \in \mathbb{R}$ (figure 5).

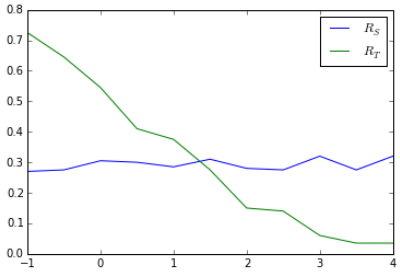


Fig. 5. Evolution of the classification error over the source (R_S) and target (R_T) with the translation parameter t (x-axis).

The source error remains approximately constant for all values of the parameter: the value of the error is due to the noise. When $t < 0$, the situation consists basically in an inversion of the order of the centers along the x-axis. Such an inversion cannot be deduced by any method without further instruction; our method relying on a simplicity principle, it avoids the class inversion (which would be far too complex), and thus leads to a high target error. When t is close to 0, the two classes are not separable and the high target error rate is due primarily to this non-separability. For values of t larger than 2, the transfer is done as expected and the target error rate is quite low. We can note that this value keeps decreasing: the target problem becomes more and more separable.

For **class deformation**, the misclassification error has been calculated for various transfer situations: the source data are

TABLE I
MISCLASSIFICATION RATE FOR TRANSFER (LEFT: IN SOURCE; RIGHT: IN TARGET) WITH SOURCE DATA GENERATED WITH A PARAMETER θ_S AND TARGET DATA GENERATED WITH A PARAMETER θ_T.

	$\theta_T = 0$	$\theta_T = 0.2$	$\theta_T = 0.4$	$\theta_T = 0.6$	$\theta_T = 0.8$	$\theta_T = 1$
$\theta_S = 0$	0%, 0%	0%, 0%	0%, 11.2%	33.7%, 33.7%	0%, 42.7%	0%, 60.7%
$\theta_S = 0.2$	0%, 0%	0%, 0%	0%, 13.5%	33.7%, 34.8%	0%, 5.62%	0%, 52.8%
$\theta_S = 0.4$	10.1%, 20.2%	4.49%, 0%	16.9%, 14.6%	12.4%, 32.6%	11.2%, 3.4%	11.2%, 53.9%
$\theta_S = 0.6$	20.2%, 41.6%	23.6%, 52.8%	19.1%, 47.2%	20.2%, 19.1%	0%, 2.25%	13.5%, 20.2%
$\theta_S = 0.8$	15.7%, 6.74%	24.7%, 0%	10.1%, 12.4%	7.87%, 11.2%	23.6%, 21.3%	22.5%, 52.8%
$\theta_S = 1$	8.99%, 55.1%	27.0%, 0%	20.2%, 12.4%	33.7%, 33.7%	22.5%, 34.8%	19.1%, 13.5%

generated by our predefined process with parameter θ_S and the target data are generated with parameter θ_T. After the learning step, we calculate the misclassification rate on the source and on the target. The obtained results are summed up in table I.

As in the class translation problem, the misclassification rates are lower in the source: this is because misclassification is penalized directly in the learning for source data. The results show that the method has difficulties adapting to topologically different situations: when θ_S is low and θ_T is high, many errors occur by the method which essentially preserves the position of the prototypes from source to target. The cost of the structure adaptation for the model in difficult transfers is too high.

VI. CONCLUSION

In this study, we have proposed a new method for Domain Adaptation based on the introduction of a neural model. Instead of considering the data points directly, we used the neural network as an intermediate encoding of the very structure of the data. This intermediate model allows the use of the Minimum Description Length Principle in a similar way as already done for analogy reasoning in [7]. In the context of this introductory article, we considered a very basic model and basic description rules: the data points are described by their relative positions towards the prototypes in the neural network.

We tested our method on two artificially generated toy datasets, the difficulty of which is parameterized. The results confirm that the method favors the most simple transfer over all other possible transfers. Besides, when a transfer is feasible, the obtained results over both source and target domains are similar to classification results obtained with traditional methods. However, the method is not always efficient when the change of structure is too difficult.

The limitations of our method are a direct consequence of its simplicity. In particular, two direct improvements have to be made to make the data description subtler. First, the prototypes in the model may be given in the form of a graph which has to be learned; this graph would help inducing a fix structures to the prototypes, which would lead to better results in more difficult transfers. Then, the description of the data used for the calculation of $K(\mathbf{M}_S)$ and $K(\mathbf{M}_T|\mathbf{M}_S)$ has to be improved in order to include regularity terms over the data: for example, if the prototypes are aligned, the description can be drastically compressed.

A testing of the method on real data remains to be done. However, such a study would make sense only if the transfer can be interpreted and is feasible. A strict formalization of such questions in *Domain Adaptation* is still non-existent: we think that Kolmogorov complexity (for the analogical point of view) and topological data analysis (for the preservation of structures) could be useful tools to provide a strict theory of learning under concept drift.

ACKNOWLEDGMENT

The authors would like to thank Jean-Louis Dessalles for his insights on Kolmogorov Complexity and Minimum Description Length Principle. They are also grateful to Cristina Manfredotti and Jérémie Sublime for their wise comments on the methodology.

REFERENCES

[1] B. Z. Zadrozny, "Learning and evaluating classifiers under sample selection bias," in *In International Conference on Machine Learning ICML04*, pp. 903–910, 2004.

[2] M. Dudk, R. E. Schapire, and S. J. Phillips, "Correcting sample selection bias in maximum entropy density estimation," in *In Advances in Neural Information Processing Systems*, 2005.

[3] J. Huang, A. Smola, A. Gretton, K. Borgwardt, and B. Schölkopf, "Correcting sample selection bias by unlabeled data," in *Advances in Neural Information Processing Systems*, vol. 19, The MIT Press, Cambridge, MA, 2007. Pre-proceedings version.

[4] J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, *Dataset Shift in Machine Learning*. The MIT Press, 2009.

[5] J. Blitzer, R. McDonald, and F. Pereira, "Domain adaptation with structural correspondence learning," in *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, EMNLP '06, (Stroudsburg, PA, USA), pp. 120–128, Association for Computational Linguistics, 2006.

[6] S. Ben-david, J. Blitzer, K. Crammer, and O. Pereira, "Analysis of representations for domain adaptation," in *In NIPS*, MIT Press, 2007.

[7] A. Cornuéjols, "Analogie, principe déconomie et complexitálgorithmique.," in *Actes des 11èmes Journées Françaises de lApprentissage*, 1996.

[8] T. R. Davies and S. J. Russell, "A logical approach to reasoning by analogy," in *Proc. of the 10th IJCAI*, (Milan, Italy), pp. 264–270, 1987.

[9] R. J. Solomonoff, "A Formal Theory of Inductive Inference: Parts 1 & 2," *Inform. Control*, vol. 7, 1964.

[10] C. S. Wallace and D. M. Boulton, "An information measure for classification," *The Computer Journal*, vol. 11, no. 2, pp. 185–194, 1968.

[11] M. Li and P. M. Vitanyi, *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Publishing Company, Incorporated, 3 ed., 2008.

[12] D. Hofstadter, *Methamagical Themas: Questing for the Essence of Mind and Pattern*. basic books ed., 1985.

[13] T. Kohonen, M. R. Schroeder, and T. S. Huang, eds., *Self-Organizing Maps*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 3rd ed., 2001.