Heterogeneous Domain Adaptation Based on Class Decomposition Schemes

Heterogeneous Domain Adaptation Based on Class Decomposition Schemes

Firat Ismailoglu[1], Evgueni Smirnov[2], Ralf Peeters[2], Shuang Zhou[3], and Pieter Collins[2]

[1] Department of Computer Engineering, Cumhuriyet Univertsity, Sivas, Turkey
fismailoglu@cumhuriyet.edu.tr
[2] Department of Data Science and Knowledge Engineering, Maastricht University, Maastricht, The Netherlands
{smirnov,ralf.peeters,pieter.collins}@maastrichtuniversity.nl
[3] College of Control Engineering, Chengdu University of Information Technology, Chengdu, People's Republic of China
s.zhou@cuit.edu.cn

Abstract This paper introduces a novel classification algorithm for heterogeneous domain adaptation. The algorithm projects both the target and source data into a common feature space of the class decomposition scheme used. The distinctive features of the algorithm are: (1) it does not impose any assumptions on the data other than sharing the same class labels; (2) it allows adaptation of multiple source domains at once; and (3) it can help improving the topology of the projected data for class separability. The algorithm provides two built-in classification rules and allows applying any other classification model.

1 Introduction

Heterogeneous domain adaptation (HDA) for classification has recently received a significant attention [13]. It assumes a given target domain (the primary domain of interest) and at least one (auxiliary) source domain. These domains are represented by different input features and usually share the same class labels. The goal is to improve classification models in the target domain by utilizing data from the source domain(s).

The approaches to HDA can be either symmetric or asymmetric [13]. The symmetric approaches project the target and source data into a common feature space and train prediction models on the projected data. The asymmetric approaches project the source data into the target domain and train prediction models on the target data and projected source data. Below we describe the main approaches within each group.

Domain Adaptation Manifold Alignment (DAMA) [12] is one of the main symmetric approach to HDA. DAMA is applicable to problems with a single target domain and $P-1$ source domains that share the same class-label set. Assuming that each domain is a manifold, a manifold alignment algorithm learns P projection functions, one for each domain. It maintains a combinatorial graph Laplacian matrix to reflect inter-domain class similarity, a combinatorial graph Laplacian matrix to reflect inter-domain class dissimilarity, and a diagonal block-matrix of graph Laplacian matrices each reflecting instance similarities within a domain. The algorithm minimizes an objective function

defined over the matrices so that the resulting P projection functions match same class instances and separate different class instances when projected. DAMA has its own prediction algorithm. It first trains regression models for the source data projected, and then adapts these models to the target domain using manifold regularization based on the projected target data.

Heterogeneous Feature Augmentation (HFA) [3] is another well-known symmetric approach to HDA. It is applied to problems with a target domain and a source domain that share the same class-label set. HFA augments the common latent feature space with the target-domain features and source-domain features. The projection functions are jointly represented by a matrix. To find the joint projection matrix, a problem to minimize the structural risk functional of SVMs is defined. The problem is tackled by an alternating optimization algorithm that solves the dual problem of SVMs and finds the corresponding optimal joint transformation matrix. The final classifier for HFA is the SVM classifier derived together with the transformation matrix.

Asymmetric Regularized Cross-Domain Transformation (ARC-t) [7] is one of the main asymmetric approach to HDA. ARC-t is applied to problems with a target domain and a single source domain that share the same class-label set. The projection function is represented by a matrix that maps an instance from the source domain to an instance in the target domain. This matrix is learned in a non-linear Gaussian RBF kernel space. This is done by minimizing a matrix regularizer and a set of constraints imposed on any pair of a target instance and a projected source instance.

Sparse Heterogeneous Feature Representation (SHFR) [6] is another well-known asymmetric approach to HDA. It is applied to problems with a target domain and a single source domain that share the same class-label set. SHFR is similar ARC-t with a difference that the target data and source data are first represented in a code space based on a class decomposition scheme [2]. This allows a projection matrix to be learned by a nonnegative LASSO optimization.

We summarize the main HDA approaches in Table 1. We observe that an HDA algorithm is lacking that can adapt multiple source domains without imposing any domain assumptions. Such an algorithm has to preserve the data topology (at least locally) using class correspondence. The adapted datasets have to applicable for any classification model. Still, it is desirable that the algorithm can provide a specific classifier tailored to the projections employed.

In the rest of this paper we introduce a new HDA algorithm called class code alignment (CCA) algorithm that has all the characteristics listed above. In Sections 2 and 3 we provide a problem formulation and background information. The new algorithm is given in Section 4. In Sections 5 and 6 we present experiments and conclusions.

Table 1: HDA Approaches

HDA Method	# Source Domains	Domain Assumption	Class Correspondence	Topology Preservation	Classifier Independence	Specific Classifiers
DAMA	≥ 1	yes	yes	yes	yes	yes
HFA	1	no	yes	no	yes	yes
ARC-t	1	no	yes	yes	yes	no
SHFR	1	no	yes	yes	yes	no

2 Problem Formulation

We consider the problem of HDA in the context of classification. We define a classification domain as a triple that consists of an instance space X with d continuous features $X(j)$ ($j \in \{1, \ldots, d\}$), a finite set Y of $K = |Y|$ class labels, and an unknown probability distribution p over $X \times Y$. We assume the presence of a target domain and at least one source domain. The target domain is the domain of interest. It is given by a target instance space X^T with d_T features, a class-label set Y^T, and a target probability distribution p^T over $X^T \times Y^T$. The target training data D_T is a set of N_T instances $(x_i^T, y_i^T) \in X^T \times Y^T$ generated from p^T. Any source domain is an auxiliary domain. It is given by a source instance space X^S with d_S features, a class-label set Y^S, and a source probability distribution p^S over $X^S \times Y^S$. The source training data D_S is a set of N_S instances $(x_i^S, y_i^S) \in X^S \times Y^S$ generated from p^S.

In HDA, the instance spaces X^T and X^S are different. The classification problem in HDA is to provide a good estimate $\hat{y} \in Y^T$ of the true class of a target query instance $x_q^T \in X^T$ according to the target probability distribution p^T given the target and source training data D_T and D_S. In this paper we consider this problem under the assumptions that: (1) the target and source domains share the same label set Y, i.e.: $Y^T = Y^S = Y$, and (2) the number K of class labels in Y is larger than 2.

3 Class Decomposition Schemes and Coding Matrices

The HDA algorithm we propose is based on class decomposition schemes [2]. Such schemes consider any multi-class classification problem ($K > 2$) as a set of B binary classification problems [2] for some positive integer B. Any b-th binary classification problem is given by a binary class partition $P_b(Y)$ of the class-label set Y for $b \in \{1, ..., B\}$. The first (second) element $Y_b^+ \subseteq Y$ ($Y_b^- \subseteq Y$) of the partition $P_b(Y)$ stands for a positive (negative) binary super class. In this context *a class decomposition scheme* is a set $S(Y)$ of B binary class partitions $P_b(Y)$ such that for any two distinct classes $y_1, y_2 \in Y$ there exists a binary class partition $P_b(Y) \in S(Y)$ with super classes Y_b^- and Y_b^+ that separate y_1 and y_2; i.e. $\neg(y_1, y_2 \in Y_b^-) \wedge \neg(y_1, y_2 \in Y_b^+)$.

Any class decomposition scheme $S(Y)$ is represented by a binary coding matrix $C \in \{0, 1\}^{B \times K}$ [2]. For any index $b \in \{1, ..., B\}$ and class label $y \in Y$ the element $C(b, y)$ equals 1 if y belongs to the positive super class Y_b^+ of the class partition $P_b(Y)$. If y belongs to the negative super class Y_b^- of $P_b(Y)$, the element $C(b, y)$ equals 0.

Any binary class partition $P_b(Y)$ is represented by exactly one row $C(b, \cdot)$ in C that we call *the partition code word for* $P_b(Y)$. Any class label $y \in Y$ is represented by exactly one column $C(\cdot, y)$ in C that we call *the class code word for* y. Class code words $C(\cdot, y)$ are viewed as images of the class labels $y \in Y$ in a *code space* $\mathcal{C}$.

Definition 1. (Code Space) *Given a class decomposition scheme $S(Y)$ with B binary partitions $P_b(Y)$, the code space $\mathcal{C}$ is equal to* $[0, 1]^B$.

To solve a multi-class classification problem using a class decomposition scheme $S(Y)$ we take three steps. First, we train an encoding mapping h that can project any instance to the code space $\mathcal{C}$. The mapping h consists of binary classifiers h_b, one for

each binary class partition $P_b(Y) \in S(Y)$. Second, we encode a query instance x using the mapping h; i.e., we create an instance code word consisting of bits $h_b(x)$ assigned by the binary classifiers h_b for that instance. Third, we decode the class of the instance x using the coding matrix C of $S(Y)$. We assign a class label $y \in Y$ of which the code word matches best with the instance code word of the instance x.

4 Class Code Alignment Algorithm

This section introduces our class code alignment (CCA) algorithm for the HDA classification problem from Section 2. The algorithm follows the symmetric approach to HDA. It projects the target and source data into a common feature space using any class decomposition scheme $S(Y)$. Given that the target and source domains share the same class-label set Y, the common feature space is the code space $\mathcal{C}$ associated with $S(Y)$. To project the data into $\mathcal{C}$ the algorithm builds the encoding mappings for the target data and source data as well as an additional encoding mapping for the class labels. This is done so that the target and source instances are projected close to the code words of their class labels in $\mathcal{C}$; i.e., they become class-code aligned.

4.1 Detailed Description

The CCA algorithm assumes that the class-label set Y consists of K standard unit vectors in $\mathbb{R}^K$ (i.e. $Y \subseteq \mathbb{R}^K$) so that the label of the k-th class ($k \in \{1, \ldots, K\}$) is given by a standard unit vector whose k-th bit equals 1. Given a class decomposition scheme $S(Y)$ with B class partitions, the algorithm builds three encoding mappings:

(1) $\sigma \circ T^T : \mathbb{R}^{d_T} \rightarrow [0,1]^B$ from the target feature space X^T to the code space $\mathcal{C}$, and
(2) $\sigma \circ T^S : \mathbb{R}^{d_S} \rightarrow [0,1]^B$ from the source feature space X^S to the code space $\mathcal{C}$, and
(3) $\sigma \circ C : \mathbb{R}^K \rightarrow [0,1]^B$ from the set Y of K class labels to the code space $\mathcal{C}$,

where

- $\sigma : \mathbb{R}^B \rightarrow [0,1]^B$ is a multivariate logistic function defined for $b \in \{1, ..., B\}$ as $\sigma_b(w) = (1 + e^{-w(b)})^{-1}$, and
- $T^T : \mathbb{R}^{d_T} \rightarrow \mathbb{R}^B$ is a linear mapping given as a matrix in $\mathbb{R}^{B \times d_T}$, and
- $T^S : \mathbb{R}^{d_S} \rightarrow \mathbb{R}^B$ is a linear mapping given as a matrix in $\mathbb{R}^{B \times d_S}$, and
- $C : \mathbb{R}^K \rightarrow \mathbb{R}^B$ is a linear mapping given as a matrix in $\mathbb{R}^{B \times K}$.

The mappings $\sigma \circ T^T$ and $\sigma \circ T^S$ are encoding mappings of the class decomposition scheme used for the target domain and the source domain, respectively. More precisely, the rows of the target matrix T^T (the source matrix T^S) represent logistic regression models of the binary classifiers trained on the target data (the source data). The mapping $\sigma \circ C$ is a class-label encoding mapping: it determines the code word for each class label and is common for the target domain and source domain. We note that the matrix C is a real-value coding matrix of $S(Y)$ in contrast to the standard binary coding matrices. The CCA algorithm adjusts C to better fit the target and source domains.

Once the mappings T^T, T^S, and C are available, any target instance $x^T \in X^T$ is projected into $\sigma(T^T x^T) \in \mathcal{C}$, any source instance $x^S \in X^S$ is projected into $\sigma(T^S x^S) \in \mathcal{C}$, and any class label $y \in Y$ is projected into $\sigma(Cy) \in \mathcal{C}$ (see Figure 1). Below we describe how to build the mappings so that the projected instances are grouped around the code words of their classes.

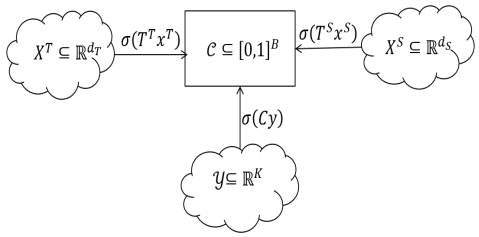

Figure 1: Projection Scheme

Any element p of the code space $\mathcal{C}$ is viewed as a parameter vector for a multivariate Bernoulli variable. If p is a true parameter vector and q is an approximate parameter vector in $\mathcal{C}$, to quantify the information gain from q to p we use *KL* divergence:

$$KL\left[p \| q\right] = \sum_{b=1}^{B} \left[p(b) \log \left(\frac{p(b)}{q(b)} \right) + (1 - p(b)) \log \left(\frac{1 - p(b)}{1 - q(b)} \right) \right]. \tag{1}$$

As we aim at mapping instances and their class labels into similar locations in the code space $\mathcal{C}$ we define a loss l for each instance label pair $(x, y) \in D_T \cup D_S$. Since our measure in $\mathcal{C}$ is the KL divergence, a loss l of the instance x for class label y for the mappings T and C can be defined as:

$$l(x, y \mid C, T) = KL\left[\sigma\left(Cy\right) \| \sigma\left(Tx\right)\right], \text{where } T = \begin{cases} T^T \text{ if } (x, y) \in D_T, \\ T^S \text{ if } (x, y) \in D_S. \end{cases}$$

Thus, the loss $\mathcal{L}$ for all N_T target instances $(x_i^T, y_i^T) \in D_T$ is defined as:

$$\mathcal{L}(D_T \mid C, T^T) = \sum_{i=1}^{N_T} l(x_i^T, y_i^T \mid C, T^T),$$

and the loss $\mathcal{L}$ for all N_S source instances $(x_i^S, y_i^S) \in D_S$ is defined as:

$$\mathcal{L}(D_S \mid C, T^S) = \sum_{i=1}^{N_S} l(x_i^S, y_i^S \mid C, T^S).$$

Minimizing $\mathcal{L}(D_T \mid C, T^T)$ for T^T and C forces the projected target instances $\sigma(T^T x_i^T)$ to be close to the code words $\sigma(Cy_i^T)$ of their class labels y_i^T in the code space $\mathcal{C}$. The same is the effect of minimizing $\mathcal{L}(D_S \mid C, T^S)$ for T^S and C for the projected source instances. Taken together, minimizing:

$$\mathcal{L}(D_T \mid C, T^T) + \mathcal{L}(D_S \mid C, T^S) \tag{2}$$

causes the same-class target and source instances to be close in the code space $\mathcal{C}$ independently on their domain. For this reason we analyze the mapping (matrix) C.

The matrix C has to have large separation properties [2]. The large column separation property means that for any class label $y_1 \in Y$ the code word Cy_1 has to be distant from the code Cy_2 of any other class label $y_2 \in Y$. If this property holds, then the code words $\sigma(Cy_1)$ and $\sigma(Cy_2)$ are distant in the code space $\mathcal{C}$; i.e. $KL\left[\sigma\left(Cy_1\right) \| \sigma\left(Cy_2\right)\right]$ and $KL\left[\sigma\left(Cy_2\right) \| \sigma\left(Cy_1\right)\right]$ are both large. Since the projected instances arrive close to the code words of their class labels, this causes the projected instances of different class labels to be distant in the code space $\mathcal{C}$.

The large row separation property means that for any $b_1 \in \{1, \ldots, B\}$ the partition code word $C(b_1, \cdot)$ has to be distant from the partition code word $C(b_2, \cdot)$ and its complement for any other $b_2 \in \{1, \ldots, B\} \setminus \{b_1\}$. If this property holds, the logistic regression models represented by rows $C(b_1, \cdot)$ and $C(b_2, \cdot)$ commit less errors simultaneously. This causes the loss $l(x, y \mid C, T)$ (defined in Equation (1)) for any labeled instance (x, y) to decrease which in turn decreases the objective function (2) for all the projected target and source instances. The latter means that the projected instances are grouped more around the code words of their classes in the code space $\mathcal{C}$.

From the above it is clear that the large separation properties of the matrix C improves the topology of the data for class separability in the code space $\mathcal{C}$. Thus, we initialize C using a reference coding matrix C_{ref} with known large separation properties. Since we learn C, we make sure that it is close to C_{ref}. Hence, we add the following *loss* term to the objective function (2) to penalize the KL divergence between the projections of the classes obtained by C and C_{ref} for the target and the source instances:

$$\mathcal{L}(D_T \mid C_{ref}, C) + \mathcal{L}(D_S \mid C_{ref}, C), \tag{3}$$

where

- $\mathcal{L}(D_T \mid C_{ref}, C) = \sum_{(x_i^T, y_i^T) \in D_T} KL\left[\sigma\left(Cy_i^T\right) \| \sigma\left(C_{ref}\, y_i^T\right)\right]$, and
- $\mathcal{L}(D_S \mid C_{ref}, C) = \sum_{(x_i^S, y_i^S) \in D_S} KL\left[\sigma\left(Cy_i^S\right) \| \sigma\left(C_{ref}\, y_i^S\right)\right]$.

The final objective function to minimize for T^T, T^S and C is defined as follows:

$$\begin{aligned} \mathcal{O}(D_T, D_S \mid T^T, T^S, C) =& \mathcal{L}(D_T \mid C, T^T) + \mathcal{L}(D_S \mid C, T^S) \\ &+ \alpha\left(\mathcal{L}(D_T \mid C_{ref}, C) + \mathcal{L}(D_S \mid C_{ref}, C)\right), \end{aligned} \tag{4}$$

where $\alpha > 0$ is a regularization parameter.

Once the objective function (4) is defined, we introduce the CCA algorithm. The algorithm learns the mappings T^T, T^S, and C from the target and source data by minimizing this function. It is given in Algorithm 1. The CCA algorithm is of *Alternating Minimization* type. It first improves the matrix T^T for the matrix C, then improves the matrix T^S for the matrix C, and, finally, improves the matrix C for the matrices T^T and T^S. The process is repeated to minimize the objective function (4).

Algorithm 1 Class Code Alignment Algorithm

Input: target data D_T, source data D_S, reference projection matrix C_{ref}, initial projection matrices T_0^T and T_0^S, step parameter $\beta \in (0,1)$, regularization parameter $\alpha > 0$, outer iteration number $M > 1$, and inner iteration number $N > 1$.

Output: projection matrices T^T, T^S, and C.

1: $C_0 := C_{ref}$;
2: **for** $t := 1$ **to** M **do**
3: $\quad T_t^T :=$ IMPROVE-T$(D_T, C_{t-1}, T_{t-1}^T, \beta, N)$;
4: $\quad T_t^S :=$ IMPROVE-T$(D_S, C_{t-1}, T_{t-1}^S, \beta, N)$;
5: $\quad C_t :=$ IMPROVE-C$(D_T, D_S, \alpha, C_{ref}, T_{t-1}^T, T_{t-1}^S)$;
6: **return** T^T, T^S, and C.

The IMPROVE-T function improves the mapping T^T (resp. T^S) for C (see Algorithm 2). It minimizes the loss term $\mathcal{L}(D_T|C,T^T)$ (resp. $\mathcal{L}(D_S|C,T^S)$) of the objective function (4). The IMPROVE-T function is a gradient descent algorithm with the backtracking line-search method. In each iteration it first computes the gradient matrix G and then executes the backtracking method using a parameter β to determine a step size η that reduces the objective function (4) (see lines 6-7 in IMPROVE-T). The function stops after N iterations and outputs the improved matrix T^T (resp. T^S).

Algorithm 2 IMPROVE-T Function

Input: data D, projection matrices C and T, step parameter $\beta \in (0,1)$, and iteration number $N > 1$.

Output: projection matrix T.

1: let B and d be the sizes of T;
2: **for** $t := 1$ **to** N **do**
3: $\quad \eta := 1$;
4: $\quad$ **for** $i := 1$ **to** B and $j := 1$ **to** d **do**
5: $\quad\quad G(i,j) = \sum_{(x,y)\in D} x(j)\Big(\sigma\big(-C(i,\cdot)y\big)\sigma\big(T(i,\cdot)x\big) - \sigma\big(C(i,\cdot)y\big)\sigma\big(-T(i,\cdot)x\big)\Big)$;
6: $\quad\quad$ **while** $\mathcal{L}(D|C, T-\eta G) > \mathcal{L}(D|C,T) - \frac{\eta}{4}\|G\|^2$ **do**
7: $\quad\quad\quad \eta := \beta\eta$;
8: $\quad\quad T := T - \eta \times G$;
9: **return** T.

Algorithm 3 IMPROVE-C Function

Input: target data D_T, source data D_S, regularization parameter $\alpha > 0$, reference projection matrix C_{ref}, and projection matrices T^T and T^S.
Output: projection matrix C.

1: let B and K be the sizes of C_{ref};
2: **for** $i := 1$ **to** B and $j := 1$ **to** K **do**
3: $\quad D_{Tj} := \left\{ \left(x^T, y\right) \in D_T \mid y = y_j \wedge y_j \in Y \right\}$;
4: $\quad D_{Sj} := \left\{ \left(x^S, y\right) \in D_S \mid y = y_j \wedge y_j \in Y \right\}$;
5: $\quad C(i,j) := \frac{1}{1+\alpha} \left(\alpha C_{ref}(i,j) + \frac{\sum\limits_{(x^T,y)\in D_{Tj}} T^T x^T + \sum\limits_{(x^S,y)\in D_{Sj}} T^S x^S}{|D_{Tj}|+|D_{Sj}|} \right)$;
6: **return** C.

The IMPROVE-C function improves the mapping C for T^T and T^S by minimizing the term (3) in the objective function (4) (see Algorithm 3). Minimizing is analytical, since the function (4) has a unique stationary point for a fixed choice of T^T and T^S.

The objective function (4) can be extended for HDA classification problems with several source domains. In this case the IMPROVE-T function is called for each source domain and the IMPROVE-C function reflects the data from all the source domains.

4.2 CCA Classification of Target Instances

Once T^T, T^S, and C are available, any target instance $x^T \in X^T$ is projected into $\sigma(T^T x^T) \in \mathcal{C}$, any source instance $x^S \in X^S$ is projected into $\sigma(T^S x^S) \in \mathcal{C}$, and any label $y \in Y$ is projected into $\sigma(Cy) \in \mathcal{C}$. This allows three CCA classification rules.

The first classification rule CCA.CDS is based on the **c**lass **d**ecomposition **s**chemes [2]. It first projects a target query instance $x_q^T \in X^T$ into $\sigma(T^T x_q^T) \in \mathcal{C}$. Then CCA.CDS assigns to x_q^T a class label $\hat{y}$ whose code word $\sigma(Cy)$ in $\mathcal{C}$ is closest to the projection $\sigma(T^T x_q^T)$; i.e. $\hat{y} = \operatorname*{argmin}_{y \in Y} KL[\sigma(Cy) \parallel \sigma(T^T x_q^T)]$.

The second classification rule CCA.IDS is based on the **i**nstance **d**ecomposition **s**chemes [5]. It first projects a target query instance $x_q^T \in X^T$ into $\sigma(T^T x_q^T) \in \mathcal{C}$. Then CCA.IDS determines set NN of the nearest neighbors of $\sigma(T^T x_q^T)$ from the set of the projected target and source instances in $\mathcal{C}$. It assigns to x_q^T a class label $\hat{y}$ that has a majority among the instances in NN; i.e. $\hat{y} = \operatorname*{argmax}_{y \in Y} \#\{x_i \in NN | y_i = y\}$.

In addition any other classification rule is possible. Once the target and source data are in the code space $\mathcal{C}$ we can train any classifier on these data.

5 Experiments

This section provides the experiments of the CCA algorithm applied on three HDA datasets. The CCA generalization performance is compared with that of baseline classifiers trained on the target data only and two domain adaptation approaches.

5.1 Settings of the CCA Algorithm

The CCA algorithm was set up as follows. The reference coding matrix C_{ref} was set to be the One-vs-All coding matrix [10]. The regularization parameter α took values from the set $\{0.04, 0.02, 1, 5, 25\}$ and the results are reported for the α value that maximizes the accuracy. The iteration numbers M and N were set to 50 and 10, respectively. The parameter of the backtracking method β of the function IMPROVE-T was set to 0.5. The CCA.CDS and CCA.IDS classification rules were applied.

We note that the reference coding matrix C_{ref} was set equal to the One-vs-All coding matrix to make the setup of the CCA algorithm comparable with that of the baseline classifiers. The C_{ref} setup ensures a kind of worst-case generalization performance of the CCA.CDS and CCA.IDS rules due to small column and row separation of the One-vs-All matrices. This means that the comparison with the baseline classifiers is rather fair: the setup is unfavorable for the CCA algorithm.

5.2 Baseline Classifiers

We compare the CCA algorithm against two groups of baseline classifiers. The first group is the group of classifiers trained on the target data only[4]. It consists of:

- kNN.T, a first nearest neighbor classifier [4].
- SVM.T, a SVM classifier with a linear kernel and default setting from LIBSVM. It employs the One-vs-All class decomposition scheme for multi-class classification.
- Bunching.T, the Bunching algorithm [1] with One-vs-All reference coding matrix C_{ref}, regularization parameter α equal to 1.0, and iteration number equal to 100.

The second group is the group of domain adaptation approaches trained on the target data and source data. It consists of:

- HFA, the Heterogeneous Feature Augmentation approach with default setting [3]: the regularization parameter was set to 1 and the parameter that controls the complexities of the transformation matrices was set to 100.
- SHFR, the Sparse Heterogeneous Feature Representation approach with default setting (One-vs-All coding matrix was used for asymmetric transformation) [7].

HFA and SHFR used one-vs-all SVM ensembles with linear kernel in common spaces.

5.3 Experiments on the Office Dataset

The Office dataset was introduced in [11]. It contains 4652 images from three domains: Amazon, dSLR, and webcam, that share the same 31 classes. The images in each domain were captured under different lighting conditions. We defined two classification problems. The first problem (resp. second problem) considers the dSLR domain as target domain and the amazon domain (resp. the webcam domain) as source domain. To estimate the accuracy of the classifiers we followed the hold-out protocol from [3,7]. 20

[4] This is indicated with postfix T.

(resp. 8) training images were randomly selected for the source domain amazon (resp. webcam) and 3 training images were randomly selected from the target domain dSLR from each class. The remaining target dSLR images were served as target test instances. The hold-out evaluation was repeated 10 times. The results are given in Figure 2.

Table 2: Classification accuracies (in percent) for the Office Data. Bold numbers indicate accuracies that are statistically greater than others based on a t-test on significance level of 0.05.

Target Domain	Source Domain	kNN.T	SVM.T	Bunching.T	HFA	SHFR	CCA.IDS	CCA.CDS
dSLR	amazon	46.12	53.13	49.17	54.7	52.66	54.66	**57.7**
	webcam				53.77	55.1	54.13	**58.65**

5.4 Experiments on the Wikipedia Dataset

The Wikipedia dataset was introduced in [9]. It contains 5732 instances from a text domain and an image domain. Both domains have 2866 instances and share the same 10 classes. One classification problem was considered. It views the text domain as target domain and the image domain as source domain. To estimate the accuracy of the classifiers we followed a hold-out protocol. For each class 10 training text instances were randomly selected as target instances and the remaining 2856 text instances were used as target test instances. All the image instances served as source instances. The hold-out evaluation was repeated 10 times. The results are given in Figure 2.

Table 3: Classification accuracies (in percent) for the Wikipedia Data.

Target Domain	Source Domain	kNN.T	SVM.T	Bunching.T	HFA	SHFR	CCA.IDS	CCA.CDS
Text	Image	59.95	60.02	61.66	62.35	63.59	56.88	64.53

5.5 Experiments on the Multiple Feature Dataset

The Multiple Feature (Mfeat) dataset [8] contains 2000 hand-written images of ten numbers (classes) from 0 to 9. The images were preprocessed using different techniques which resulted in six domains: mfeat-fou (given with 76 Fourier coefficients), mfeat-fac (given with 216 profile correlation features), mfeat-kar (given with 64 Karhunen-Love coefficients), mfeat-pix (given with 240 pixel average features), mfeat-zer (given with 47 Zernike moment features), and mfeat-mor (given with 6 morphological features). The mfeat-mor domain is considered as target domain. In this context, we defined five classification problems so that each remaining domain is considered as source domain once. To estimate the accuracy of the classifiers we followed a hold-out protocol: for each number 10 training mfeat-mor instances were randomly selected as target instances and the remaining 1990 mfeat-mor instances were used as target test instances.

All the instances of the corresponding source domain served as source instances. The hold-out evaluation was repeated 10 times. The results are given in Figure 4.

Table 4: Classification accuracies (in percent) for the Mfeat Data. Bold numbers indicate accuracies that are statistically greater than others based on a t-test on significance level of 0.05.

Target Domain	Source Domain	kNN.T	SVM.T	Bunching.T	HFA	SHFR	CCA.IDS	CCA.CDS
mfeat-mor	mfeat-fou	44.71	43.7	50.14	63.03	64.22	**70.21**	62.92
	mfeat-fac				64.4	65.28	**70.91**	62.47
	mfeat-pix				63.07	64.45	**69.41**	58.12
	mfeat-kar				63.5	66.16	**71.1**	60.53
	mfeat-zer				63.51.	65.12	**70.01**	65.01

5.6 Results and Discussions

Tables 2, 3 and 4 provide the accuracies of the CCA classification rules and the baseline classifiers. They show that the HDA methods outperform the classifiers trained on the target data. For example, the CCA.IDS rule improves the accuracy with 25% on the Mfeat dataset compared with the kNN.T classifier (a target-domain classifier with a similar classification rule) and the CCA.CDS rule improves the accuracy in the range of $[2.87\%, 14.87\%]$ compared with the Bunching.T classifier (a target-domain classifier similar to CCA). Thus, source data from different instances spaces can improve the classification accuracy on unseen target instances.

When comparing the CCA classification rules with the HDA methods, HFA and SHFR, we observe that they have a similar accuracy on the Wikipedia dataset while on the Office dataset and the Mfeat dataset one of the CCA classification rules is a winner (statistically significant improvement in accuracy is in the range of $1\% - 4\%$). Thus, the CCA classification rules are capable of outperforming the HFA and SHFR methods.

When comparing the CCA classification rules themselves, we observe that they are rather different. The CCA.CDS rule outperforms the CCA.IDS rule on the Office dataset and Wikipedia dataset. This is due to the fact that the projected instances are spread around the code words of their classes in the code space $\mathcal{C}$ for these datasets. For the Mfeat dataset, however, this does happen and in this case the CCA.IDS rule outperforms the CCA.CDS rule (the accuracy improvement is around 10%). Thus, the CCA classification rules correspond to very different states of HDA. The CCA.CDS rule is preferred when the minimized sub-loss (2) has relatively *low* values; i.e. the projected instances *are spread* around the code words of their classes. The CCA.IDS rule is preferred when the minimized sub-loss (2) has relatively *high* values; i.e. the projected instances *are not spread* around the code words of their classes.

6 Conclusion

This paper proposed the CCA algorithm for HDA classification that can use several source domains. The algorithm builds the encoding mappings for the target data, source

data, and class labels by minimizing the total loss function (4). This aligns the projected target and source instances with the code words of their classes in the code space $\mathcal{C}$. To make different-class instances more separable in $\mathcal{C}$ the class encoding mapping is initialized using a coding matrix with well-presented separation properties [2]. This allows improving the topology of the projected data in the code space $\mathcal{C}$ for class separability. Once the encoding mappings have been learned, the CCA algorithm projects all the data and class labels in the common code space $\mathcal{C}$. In this context, we note that the CCA algorithm does not make any assumption on the underlying structure of the target and source domains. It only necessities common class labels for domain correspondence.

The CCA algorithm offers two built-in classification rules: the CCA.CDS rule and the CCA.IDS rule. CCA.CDS (resp. CCA.IDS) is preferable when the spread of the same class instances is relatively low (high) around the code words of their classes. In addition, any other classification rule can be trained on the projected data.

The CCA algorithm was experimentally tested. The experiments showed that the CCA algorithm is capable of outperforming standard HDA methods.

References

1. O. Dekel and Y. Singer. Multiclass learning by probabilistic embeddings. In *Advances in Neural Information Processing Systems 15*, pages 945–952, 2002.
2. T. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.
3. L. Duan, D. Xu, and I. Tsang. Learning with augmented features for HDA. In *Proceedings of the 29th Internatiobal Conference on Machine Learning (ICML 2012)*, 2012.
4. R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley Press, 2000.
5. F. Ismailoglu, E. Smirnov, N. Nikolaev, and R. Peeters. Instance-based decompositions of error correcting output codes. In *Proceedings of the 12th International Workshop on Multiple Classifier Systems (MCS 2015)*, pages 51–63, 2015.
6. Zhou J., Tsang I., Pan S., and Tan M. Heterogeneous domain adaptation for multiple classes. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS 2014)*, pages 1095–1103, 2014.
7. B. Kulis, K. Saenko, and T. Darrell. What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. In *Proceedings of the 24th IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2011)*, pages 1785–1792. IEEE, 2011.
8. M. Lichman. UCI machine learning repository, 2013.
9. N. Rasiwasia, J. Pereira, E. Coviello, G. Doyle, G. Lanckriet, R. Levy, and N. Vasconcelos. A new approach to cross-modal multimedia retrieval. In *Proceedings of the 18th International Conference on Multimedia (MM 2010)*, pages 251–260. ACM, 2010.
10. R. Rifkin and A. Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004.
11. K. Saenko, B. Kulis, M. Fritz, and T. Darrell. Adapting visual category models to new domains. In *Proceedings Computer Vision of the 11th European Conference on Computer Vision (ECCV 2010)*, pages 213–226, 2010.
12. C. Wang and S. Mahadevan. Heterogeneous domain adaptation using manifold alignment. In *Proceedings of the 22th International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 1541–1546. AAAI Press, 2011.
13. K. Weiss, T. Khoshgoftaar, and D.D. Wang. A survey of transfer learning. *Journal of Big Data*, 3(9):45–85, 2016.