Stacked Robust Adaptively Regularized Auto-regressions for Domain Adaptation

Wen Hao Jiang, Hong Chang Gao, Wei Lu, Wei Liu, Fu-lai Chung, and Heng Huang

Abstract—Domain adaptation is the situation for supervised learning in which the training data are sampled from the source domain while the test data are sampled from the target domain that follows a different distribution. The key to solve such a problem is to reduce the effects of the discrepancy between the training data and test data. Recently, deep learning methods that employ stacked denoising auto-encoders (SDAs) to learn new representations for both domains have been successfully applied in domain adaptation. And remarkable performance on multi-domain sentiment analysis datasets has been reported, making deep learning a promising approach to domain adaptation problems. In this paper, a deep learning method called Stacked Robust Adaptively Regularized Auto-regressions (SRARAs) is proposed to learn useful representations for domain adaptation problems. Each layer of SRARAs contains two steps: a linear transformation step, which is based on robust adaptively regularized auto-regression, and a non-linear squashing transformation step. The first step aims at reducing the discrepancy between the training data and test data, and the second step is to introduce non-linearity and control the range of the elements in the outputs. The experimental results on text and image datasets demonstrate that the proposed method is very effective.

Index Terms—Domain adaptation, Robust auto-regression, Classification.

1 INTRODUCTION

Traditionally, the training of a classifier assumes that the training data and the testing data follow the same distribution or they are in the same domain. However, this may not be valid in some situations, thus posing a need to train classifiers without such an assumption. We might have plenty of labeled samples from one domain (source domain) but very few or even no labeled samples from a different domain (target domain), and want to obtain a model that will work well on the target domain. This is the so-called domain adaptation problem [1]. Domain adaptation problem is a fundamental problem in machine learning and has been studied before under different names, including covariate shift [2] and sample selection bias [3], [4]. The term domain adaptation started to attract the interests of researchers recently [5], [6], [7], [8], [9], [10]. It is closely-related to multi-task learning [11] and semi-supervised learning [12], [13]. To extract useful information, multi-task learning and semi-supervised learning employ auxiliary tasks and unlabelled data, respectively, while domain adaptation exploits another domain. They share many overlaps with each other [14], [15], [16], [17]. The need for domain adaptation is prevalent in many areas, e.g., bioinformatics [18], natural language processing [6], [8], [19] and computer vision [20], [21], [22].

The difficulty of domain adaptation comes from the gap between the source distribution and the target distribution, which violates the assumptions of theory for traditional machine learning. Hence, the model trained on the source domain could not be used directly in the target domain. Many methods have been proposed for domain adaptation, e.g., instance re-weighting methods [23], co-training based methods [24], shared feature learning methods [5], [25], [26], feature augmentation methods [8], [27] and deep learning methods [28], [29]. Aiming at providing theoretical foundations for this problem, some formal analyses of the distance between the source domain and the target domain have also been proposed [9], [14], [30], [31].

In domain adaptation, labeled samples from the source domain are always available. According to whether there exist labeled samples or unlabeled samples from the target domain in the training stage, domain adaptation could be categorized into three types: unsupervised domain adaptation, supervised domain adaptation and semi-supervised domain adaptation [27], [32]. In unsupervised domain adaptation, there are no labelled samples from the target domain. While in supervised domain adaptation, a certain amount of labeled samples from the target domain are available for training. For semi-supervised domain adaptation, models have access to both labeled and unlabeled data from the target domain. In this paper, we focus on improving the performance of unsupervised domain adaptation.

Recently, deep learning methods have been applied to unsupervised domain adaptation to learn features that can reduce the gap between the distributions of source domain and target domain. Glorot et al. [28] proposed to learn robust feature representations with stacked denoising auto-
do not perform well in practical applications. Because there is not enough samples and the overlap regions between the source domain and the target domain are relatively small.

2.2 Feature Transformation Methods
Instead of re-weighting the samples, we can also seek a feature transformation so as to decrease the discrepancy between the source distribution and the target distribution. Methods performing feature transformation can be categorized into two types, i.e., shallow learning methods and deep learning methods.

2.2.1 Shallow Learning Methods
Shallow learning methods perform only one transformation step on the features. Most existing shallow methods are subspace methods.

Subspace methods try to find the relationships among features and bridge the gap between the source domain and the target domain by finding some common subspaces with the help of the relationships found. Structural learning [40], an effective semi-supervised method, was proposed to utilize the information contained in the unlabeled data. It aims at finding the common predictive structure shared by the multiple auxiliary problems, which can be used to improve the performance on the target problem. It was extended to domain adaptation setting by structural correspondence learning (SCL) [5]. SCL defines pivot features which are common to both the source domain and the target domain and tries to find the correlation between the pivot features and non-pivot features. It extracts the corresponding subspace and augments the original feature space with it for more effective classification. This method has been proved useful for natural language processing tasks [5]. Transfer component analysis (TCA) [41] seeks a subspace on which the maximum mean discrepancy (MMD) between the domains is decreased. Coupled subspaces [25] is another subspace method for domain adaptation. The key intuition is that linking target-specific features to source features could facilitate training classifiers only on the source domain. In this method, samples are divided into multiple views and cross-correlation matrices are computed, from which projection operators for both the source domain and the target domain could be determined. With these two projection matrices, a linear predictor could be built. In [42] a feature learning method based on discriminative clustering assumption was proposed. It’s goal is to maximize domain similarity while minimize the expected classification error on target domain. According to the experimental results, the performance is comparable to SCL. Geodesic flow kernel (GFK) [21] computes a kernel between source and target points based on geodesic flow along a latent manifold. The kernel obtained is then used to train a classifier which will be applied directly on the target domain. Subspace alignment [43] aligns the subspaces generated by the eigenvectors of the source domain and the target domain with a linear transformation. A similarity function can be obtained with the aligned subspaces, on which a classifier will be trained.

Subspace methods can only capture limited information, hence the full space should be considered in order to include

2 RELATED WORKS

In this paper, we focus on the setting that there is no available labeled samples from the target domain when training models on the source domain. There have been many methods proposed for this setting, and here we categorize them and highlight a few representative ones for each category.

2.1 Instance Weighting Methods
The methods based on instance weighting assign a weight to each sample from the source domain such that the distance between the source distribution and target distribution is decreased [34], [35], [36], [37], [38], [39]. Then the weight will be used to construct the weighted loss function for classifiers. Two conditions are required for the success of such methods. The first one is the class posteriors are identical for both domains, and the second one is the weights are unbiased estimates of the ratio of the target density to the source density. Even though The methods in this category usually
more useful information in the model. FLDA [44] employs a parametric transfer model to capture the relationships between the source domain and the target domain. And a classifier with loss on the source domain under this transfer model will be trained. CORAL [45] is a simple linear feature learning algorithm. It reduces the distance between the source domain and the target domain by just re-coloring whitened source features with the covariance matrix of the target distribution. Then, a classifier is trained on the transformed source features. CORAL has been reported it outperformed many state-of-the-art methods on computer vision and natural language processing tasks.

2.2.2 Deep Learning Methods

Deep learning methods proposed for domain adaptation usually learn new representations for samples from both domains with a decreased domain discrepancy. Glorot et al. [28] proposed to learn robust feature representations with stacked denoising auto-encoders (SDA) [46] for domain adaptation. Auto-encoders are one-layer neural networks that find hidden representations by reconstructing the original input data. Denoising auto-encoders are similar to auto-encoders, except that the input data are randomly corrupted by adding some kinds of noise, e.g., Gaussian noise. Denoising auto-encoders are forced to undo the effect caused by corruptions, hence they could find useful structures of input data. The denoising auto-encoders can be stacked into deep architectures. The outputs of their last layers are then used as input features for Support Vector Machine (SVM) classification. Glorot et al. showed that using SDA-learned features with linear SVM classifiers yields recordable performance for the tasks of sentiment analysis across different product domains on the Amazon review dataset [28]. But the method is slow which limits its use in practice. Marginalized stacked denoising auto-encoder (mSDA) [29], a variant of SDA with slightly different network structure, was proposed to overcome this drawback. Chen et al. [29] noticed that the random feature corruptions for SDA can be marginalized out and this is equivalent to training the models with an infinitely large number of corrupted input data conceptually. Moreover, its linear denoising auto-encoders have a closed form, which helps to speed it up. The denoising step is followed by a non-linear step, which is just a hyperbolic tangent function $\tanh(\cdot)$. In [29], promising performance is reported for multi-domain sentiment analysis tasks. Inspired by the theory of domain adaptation [7], domain-adversarial neural networks (DANN) [47] optimizes a approximated domain distance and empirical training error on the source domain to find the hidden representations for samples from both the source domain and the target domain. They could also be seen as a classifier for domain adaptation problem and can be trained on the outputs of any feature learning algorithms, e.g., mSDA.

2.3 Co-training Methods

Co-training [48] is a semi-supervised learning method for data with two views (feature sets). It trains two separate classifiers based on the two views. The most confident predictions of each classifier on the unlabeled data are then used to train the other classifier on the next round. Chen et. al. [24] proposed co-training for domain adaptation (CODA), which is a variant of the co-training method. CODA does not assume two available views. Instead, in each iteration, CODA formulates an individual optimization problem which simultaneously learns a target predictor, a split of the feature space into views [49], and a subset of source and target features to be included in the predictor. It bridges the gap between the source domain and the target domain by slowly adding both the features and instances that the current algorithm is most confident with to the training set.

3 Stacked Robust Adaptively Regularized Auto-regressions

3.1 Notations

Throughout this paper, the following definitions and notations are used. For a vector $v$, we denote the $\ell_2$-norm of $v$ by $\|v\|_2$. For a matrix $M \in \mathbb{R}^{m \times n}$, we denote the $(i,j)$-th element by $M_{ij}$, the $i$-th row by $M_i$, and the $j$-th column by $M_j$. Also, we denote $\|M\|_F = \sqrt{\text{tr}(M^T M)}$, where $\text{tr}(\cdot)$ is the trace operator for square matrix and $\|M\|_2 = \sqrt{\sum_{i=1}^n \|M_i\|_2}$ as the $\ell_2$-norm of matrix $M$.

We follow the setup of traditional domain adaptation as follows. Let us assume that the data come from two domains, i.e., the source domain $S$ and the target domain $T$. From the source domain, data $D_S = \{x_1, \ldots, x_n\} \in \mathbb{R}^d$ with labels $L_S = \{y_1, \ldots, y_n\}$ are sampled, while from the target domain data $D_T = \{x_{n+1}, \ldots, x_p\} \in \mathbb{R}^d$ without labels are sampled. As the features from the two domains might be different, the data vectors are filled with zeros to make both domains of the same dimensionality $d$. The data samples from these two domains form the data matrix $X = (x_1, \ldots, x_n) \in \mathbb{R}^{d \times n}$ and the label vector of source domain is denoted as $Y_S$. Our goal is to learn new representations, on which a classifier can be trained with samples from the source domain to predict the labels of samples from the target domain.

3.2 Background

The basic structure of feature learning methods for domain adaptation is auto-encoder. There exist many auto-encoders developed in the deep learning literature [46], [50], [51], [52], [53]. Usually, an auto-encoder contains two parts, an encoder and a decoder. The encoder is a parametrized feature extracting function denoted as $enc(\cdot)$, which maps the input data vector $x \in \mathbb{R}^d$ to hidden representations $h = enc(x)$ where $h \in \mathbb{R}^{d_h}$. The decoder $dec(\cdot)$ is also a parametrized function which maps the hidden representations back to the input space forming a reconstruction $rec = dec(h)$. The parameters of the auto-encoders are learned to minimize the reconstruction error, measured by a loss function $l(x, rec)$.

Denoising auto-encoders (DAEs) are slightly different from auto-encoders. The inputs of a denoising auto-encoders are corrupted by adding noises before mapping them into the hidden representations. DAEs are trained to reconstruct the original input $x$ from its corrupted input $\tilde{x}$ by minimizing the loss function $l(x, dec(enc(\tilde{x})))$. Stacked denoising auto-encoders (SDAs) [46] just stacks DAEs together by feeding the output of the previous DA into the...
current DA to learn high-level features. The parameters of SDA are learned layer by layer greedily. mSDA approximates SDAs by combining the encoder and decoder with a linear mapping and outputs the results with a element wise non-linear transformation. Our method adopts similar strategy and models the linear transformation step as a robust adaptively regularized auto-regression problem.

3.3 Robust Adaptively Regularized Auto-regression

The difficulty of domain adaptation lies in the fact that different domain might have different specific features. For example, the words comprehensive and detailed are usually used to describe books but they are seldom used to describe DVDs, electronics and kitchen appliances. Moreover, comprehensive and detailed are usually good indicators for positive reviews. A classifier trained on book domain which has high weights on the book domain specific words will not perform well on other domains. Contrarily, excellent and satisfied can be used more generally. A classifier with higher weights on such domain non-specific features can work well on target domains. To overcome the difficulty of domain adaptation, we should learn new representations such that there are no such domain specific features. Since these domain specific features only occur in the related domains, their variances are often small compared to those of common words. We use the Amazon review dataset, with book as source domain and DVD as target domain, to illustrate that. For 500 domain specific features and 500 non-domain specific features, the average variances on whole dataset are $9.7 \times 10^{-3}$ and $3.3 \times 10^{-3}$, respectively. Thus, domain specific features tend to have lower variances on the whole dataset. Hence, decreasing the weights for features with small variances help to mitigate the difficulty of the domain adaptation problem.

Similarly, if a component is domain specific, the corresponding variances on both domains tend to be small. Hence, it is reasonable to adjust the weights of components. In this paper, we make an attempt to mitigate the domain adaptation problem by proposing a architecture similar to SDA. But the auto-encoder in SDA is replaced with a component weight adjusting model in our method.

Principal component analysis (PCA) is natural choice for adjusting the relative weights of components. One may know that PCA truncates the components with small singular values, which will cause information deficiency for the next layer. Hence, we can increase the larger variance components in a soft way, i.e., shrinking the eigenvalue $r_i$ of a covariance matrix by a factor $r_i = a(r_i) r_i$. If an eigenvalue $r_i$ is large, meaning that the corresponding component is not noise, it should not be deleted and consequently $a(r_i)$ should be close to 1. On the other hand, if $r_i$ is very small, the shrinkage factor $a(r_i)$ should be close to 0 in order to reduce the noise. Since the eigenvalue of covariance matrix is just the square of the corresponding singular value of the centered data matrix, we can carry out such shrinkage on the singular values of centered matrix.

Inspired by ridge regression [54], one obvious option to implement the required shrinkage is

$$\hat{s}_i = \frac{s_i}{s_i^2 + \lambda},$$

(1)

where $\lambda \geq 0$ and $s_i$'s are singular values of a data matrix that is centered. It could be seen as the result of the following regularized auto-regression problem:

$$\min_W \|X^T - X^T W\|_F^2 + \lambda \|W\|_F^2,$$

(2)

where $X \in \mathbb{R}^{d \times n}$ is the centered data matrix.

It is well known that the squared loss is vulnerable to noises and outliers. The occurrences of outliers might largely influence the result of the objective function. To reduce the effects of outliers, we use robust loss to measure the reconstruction errors. $\ell_{2,1}$-norm loss was first proposed in [55] as a rotational invariant $\ell_1$ norm. It has been successfully adopted in robust principal component analyses [55] and robust feature selection [56] because of its outlier-resistant property.

In this paper, we use $\ell_{2,1}$-norm as a measure of reconstruction error for our robust auto-regression. The objective function can be written as:

$$\min_W \|X^T - X^T W\|_{2,1} + \lambda \|W\|_{2,1}^2.$$

(3)

However, in this formulation, the variance of feature might be different, hence each feature does not contribute equally. In order to enforce equal contributions, we introduce the feature variances to obtain a adaptive regularizer. The objective function of our robust adaptively regularized auto-regression (RARA) is written as:

$$\min_W \|X^T - X^T W\|_{2,1} + \lambda \text{tr}(W^T \Lambda W),$$

(4)

where $\Lambda$ is a diagonal matrix and $\Lambda_{ii} = X_i^T X_i$. The above problem is equivalent to reconstructing $X$ with a normalized data matrix and $\|W\|_{2,1}$ as regularizer. The objective of our proposed RARA is a convex problem with a non-smooth loss function and a smooth regularization. By setting the derivative of Equation (4) w.r.t $W$ to zero, we have

$$\sum_i \frac{1}{2 \|x_i - W^T x_i\|^2} (-2x_i x_i^T + 2x_i x_i^T W) + 2\lambda W = 0.$$

(5)

We denote

$$D_{ii} = \frac{1}{2 \|x_i - W^T x_i\|^2},$$

(6)

and the above equation becomes

$$\sum_i D_{ii}(-2x_i x_i^T + 2x_i x_i^T W) + 2\lambda W$$

$$= -2X DX^T + 2X DX^T W + 2\lambda W = 0.$$

(7)

Note that $D$ is dependent on $X$ and this equation is difficult to solve. However, we can solve it by updating $D$ and $W$ alternatively. If $D$ is given, $W$ could be computed by

$$W = (X DX^T + \lambda \Lambda)^{-1} X DX^T.$$

(8)
We can solve the objective function (4) by alternatively updating $W$ and $D$ by (6) and (8) until convergence.

**Theorem 3.1.** The procedure that repeats updating $W$ and $D$ by (6) and (8) alternately will monotonically decrease the objective function (4).

**Proof.** We define the following functions

$$f_i(W) = ||x_i - W^Tx_i||_2^2,$$

$$g(u) = \sqrt{u} \ (u > 0),$$

$$h_i(W) = ||x_i - W^Tx_i||_2^2.$$  

We can see that $g(u)$ is a concave function and $f_i(W) = g(h_i(W)).$

Using concave duality [57], we can rewrite $f_i(W)$ as

$$f_i(W) = \inf_{d_i} \left[ d_i^*h_i(W) - g^*(d_i) \right],$$

where $g^*(d_i)$ is the concave dual of $g(u)$ given by

$$g^*(d_i) = \inf_{u > 0} \left[ d_i u - g(u) \right] = \inf_{u > 0} \left[ d_i u - \sqrt{u} \right].$$

$g^*(d_i)$ should satisfy $g^*(d_i) > -\infty,$ and hence $d_i$ should be positive. Also, we have

$$g^*(d_i) = \inf_{u > 0} \left[ d_i u - g(u) \right] = -\frac{1}{4d_i}.$$  

Hence, we can express $f_i(W)$ as

$$f_i(W) = \inf_{d_i > 0} \left[ d_i ||x_i - W^Tx_i||_2^2 + \frac{1}{4d_i} \right],$$

and the problem (4) could be written as

$$\min_{W,d_i > 0} \sum_i \left( d_i ||x_i - W^Tx_i||_2^2 + \frac{1}{4d_i} \right) + \lambda \tr(W^TAW).$$

We can solve this problem by updating $W$ and $d_i$ alternatively:

**Updating W:** We have to solve

$$\min_{W} \sum_i d_i ||x_i - W^Tx_i||_2^2 + \lambda \tr(W^TAW),$$

which is just the problem (9) and the solution is given by (8).

**Updating d_i:** We have to solve

$$\min_{d_i > 0} d_i ||x_i - W^Tx_i||_2^2 + \frac{1}{4d_i}$$

for $i = 1, \ldots, n.$ The solution is

$$d_i = \frac{1}{2||x_i - W^Tx_i||_2^2},$$

which is the updating rule for matrix $D$ if we define $D_{ii} = d_i.$

The above two steps both decrease function value in (18), therefore the updating rules (6) and (8) will decrease the objective function value of (4).

Our algorithm usually converges in less than 10 iterations with relative error $10^{-5},$ which can be seen from the experimental results section. In practice, we can also use the maximum number of iterations as stopping criteria. In the extreme case that the maximum number of iterations is set as 1, our method is just the regularized least square regression. And it learns useful features as well. In practical implementation, to avoid divide by zero error, we use the following updating rule for $D$:

$$D_{ii} = \frac{1}{2||x_i - W^Tx_i||_2^2 + \varepsilon},$$

where $\varepsilon$ is a very small positive number. Moreover, updating $W$ step can be seen as a weighted regularized regression and each column of $W$ is independent, hence this step can be easily paralleled.

### 3.4 Interpreting RARA

We notice that

$$\hat{X} = W^T X = X D^T X + \Lambda^{-\frac{1}{2}} \Lambda^{-\frac{1}{2}} X = X D^T X + \Lambda^{-\frac{1}{2}} \Lambda^{-\frac{1}{2}} X$$

in which we denote $\hat{X} = \Lambda^{-\frac{1}{2}} X D^\frac{1}{2}.$ The matrix $\hat{X}$ is a normalized version of $X$ by left multiplying $\Lambda^{-\frac{1}{2}}$ and right multiplying $D^\frac{1}{2}.$ Both $\Lambda$ and $D$ are diagonal matrices and the diagonal element are related to the norms of features and the reconstruction errors of samples. Hence, this normalization decreases the weights of the features with large norm and samples with large reconstruction error.

We denote the SVD of $\hat{X}$ as $\hat{X} = USV^T$ and we have the following equations:

$$\hat{X} = X D^T X = X D^T X + \Lambda^{-\frac{1}{2}} \Lambda^{-\frac{1}{2}} X$$

where $\hat{X} = X U S^2 (S^2 + \Lambda I)^{-1} S V^T.$ We denote the $i$-th column of $X$ as $\hat{x}_i$ and the $D$ updating step can be rewritten as:

$$D_{ii} = \frac{1}{2||x_i - \hat{x}_i||_2^2},$$

Hence, the optimization procedure of problem (4) can be seen as a procedure that iteratively executes the following three steps until convergence:

1. Updating $\hat{X}$ by $\hat{X} = \Lambda^{-\frac{1}{2}} X D^\frac{1}{2},$
2. Updating $\hat{X}'$ by adjusting the singular values of $\hat{X},$
3. Updating $\hat{X}$ and $D.$

Therefore, our method essentially seeks a matrix $D$ and performs denoising on the specially normalized data matrix $\hat{X}'$ and then recovers the data matrix with $\hat{X}.$
parameters. The necessity can be seen from the following framework is more common in statistics. Secondly, the in-convenient to analyze since the loss + regularizer instead of marginalized corruptions. Hence our method is are some essential differences. Firstly, we use regularization supervised learning. The whole feature learning procedure is summarized in Algorithm 1.

Algorithm 1 Robust Adaptively Regularized Auto-regression.

Input: $X, \lambda$.

Initialize $D_{ii} = 1$ for $i = 1, \ldots, d$.

repeat

Update $W$ by (8).
Update $D$ by (6).

until convergence

Output: $\tanh(\alpha(W^TX))$

Algorithm 2 Feature learning with SRARAs.

Input: data matrix $X$, $\alpha$, $\lambda$ and number of layers $L$.

Initialize $H^{(0)} = X$.

for $i = 1$ to $L$ do

Compute $H^{(i)}$ by Algorithm 1 with $H^{(i-1)}$ and $\lambda$ as inputs.

end for

Output: Concatenate $H^{(0)}$ and all the learned features to form new representations.

3.5 Feature Learning with RARA

The output of our proposed robust auto-regression is written as

$$\hat{X} = W^TX,$$  (25)

which is essentially a linear transformation of input data matrix. To introduce nonlinearity, the output of linear transformation is then fed into the hyperbolic tangent function $\tanh(\alpha x)$, where $\alpha$ is a parameter to be determined by users. As in mSDA [29], the non-linear step is independent of the previous step which is different from other traditional auto-encoders [53]. Hence, it is necessary for users to control the intensity of the squashing effect. The steps of RARA with non-linear transformation step is summarized in Algorithm 1.

Following the same strategy adopted by other auto-encoder based deep learning methods, we learn the new representations layer by layer greedily. Let us denote the input sample of $l^{th}$ layer by $h^{(l-1)}$ and the original input as $h^{(0)} = x$. The output of $l^{th}$ layer which is also the input of $(l+1)^{th}$ layer could be written as $h^{(l)} = \tanh(\alpha W^{T(l)}h^{(l-1)})$, where $W^{(l)}$ is the transformation matrix computed with all input samples of layer $l$. The original input and output from all layers are concatenated to form the final representations. We can use these new representations directly for classification tasks as in traditional supervised learning. The whole feature learning procedure with stacked robust adaptively regularized auto-regressions (SRARAs) is summarized in Algorithm 2.

Our method uses similar architecture to mSDA, but there are some essential differences. Firstly, we use regularization instead of marginalized corruptions. Hence our method is much convenient to analyze since the loss + regularizer framework is more common in statistics. Secondly, the intensity of non-linear transformation could be controlled by parameters. The necessity can be seen from the following section. Lastly, we use robust loss function to reconstruct inputs. Hence, our method is more robust to noise and the advantages could be seen from the performance comparisons in the next section.

3.6 Extension for High Dimensional Data

For high dimensional data, it is very time consuming to compute the linear transformation matrix in Algorithm 1 due to the high dimensionality. We can apply some dimensionality reduction technique first to obtain the low dimensional data, and then our proposed method can be applied. In this paper, in order to obtain the low-dimensional transformation matrix, we recover the most frequent features with all the features. Denoting the data matrix composed of most frequent features as $Z$ and adopting the $\ell_{2,1}$-norm loss as before, the objective function is written as:

$$\min_W \|Z^T - XT W\|_{2,1} + \lambda \text{tr}(W^T \Lambda W),$$  (26)

where $\Lambda_{ii} = X_i X_i^T$. We set the derivative to zero and obtain:

$$\sum_i 2\|z_i - W^T x_i\|^2 - 2x_i z_i^T + 2x_i x_i^T W + 2\lambda AW = 0.$$  (27)

We denote

$$D_{ii} = \frac{1}{2\|z_i - W^T x_i\|^2},$$  (28)

similarly as before and the above equation becomes

$$\sum_i D_{ii}(-2 x_i z_i^T + 2x_i x_i^T W + 2\lambda W) = -2X^TD^T X^T W + 2\lambda W = 0.$$  (29)

We can also solve it by updating $D$ and $W$ alternatively. If $D$ is given, $W$ could be computed by

$$W = (X^TD^TX + \lambda \Lambda)^{-1}X^TD^T.$$  (30)

We can see that updating $W$ requires computing a inverse of $X^TD^TX + \lambda \Lambda$, which is quite time consuming. Hence, the above updating rule is not suitable for practical application. However, it is easy to see that finding $W$ is equivalent to solving the following problem:

$$\min_W \sum_i D_{ii} \|z_i - W^T x_i\|^2 + \lambda \text{tr}(W^T \Lambda W).$$  (31)

And the gradient of this objective function could be expressed as:

$$\frac{\partial f}{\partial W} = -2X^TD^T X^T W + 2\lambda W.$$  (32)

We can compute $W$ by gradient descent method to a certain precession, which requires less computations. Once $W$ is obtained, $D$ can be updated by (28). Hence, the problem (26) can be solved by updating $W$ and $D$ alternatively. The non-linear transformation is also used in the dimensionality reduction step and the whole algorithm is described in Algorithm 3. After we get the low-dimensional data, the same methods as before can be adopted to learn the features and the complete algorithm is described in Algorithm 4.
Algorithm 3 Dimensionality reduction with $f_{2,1}$-norm loss for high-dimensional data.

**Input:** $X, \lambda$.

Initialize $D_{ii} = 1$ for $i = 1, \ldots, d$.

repeat
  repeat
    Update $W$ by gradient descent.
  until convergence
  Update $D$ by (28).
until convergence

**Output:** tanh$(\alpha(W^T X))$

Algorithm 4 Feature learning with SRARAs for high-dimensional data.

**Input:** data matrix $X$, $\alpha$, $\lambda$ and number of layers $L$.

Initialize $H^{(0)} = X$.

Perform dimensionality reduction by Algorithm 3 and obtain the output as $H^{(1)}$.

for $i = 2$ to $L$ do
  Compute $H^{(i)}$ by Algorithm 1 with $H^{(i-1)}$ and $\lambda$ as inputs.
end for

**Output:** Concatenate $H^{(0)}$ and all the learned features to form new representations.

4 Experimental Results on Textual Datasets

In this section, we compare our method with other state-of-the-art methods on three textual datasets and present the performance analysis of our method.

4.1 Datasets

We test and analyze the proposed method on Amazon review dataset $^2$ [58], ECML/PKDD 2006 spam dataset $^3$ [59] and 20 newsgroups dataset $^4$.

**Amazon review dataset.** The Amazon review dataset contains more than 340,000 reviews from 25 different types of products (domains) from Amazon.com. In this paper, we only consider binary classification problem, i.e., whether a review is positive (higher than 3 stars) or negative (3 stars or lower) as in [28], [29]. Our goal is to train a classifier on one domain and apply it to another domain. In this complete dataset, different domains have totally different number of samples and class distributions. In order to reduce the effects of class size and domain size, we follow the convention in [58], and use a smaller subset which contains reviews of effects of class size and domain size, we follow the convention in [58], and use a smaller subset which contains reviews of

Spam dataset. The second dataset is from the ECML/PKDD 2006 discovery challenge. In this dataset, 4,000 labeled training samples were collected from publicly available sources (source domain), with half of them are spam and the other half are non-spam. The testing samples were collected from 3 different user inboxes (target domains), each of which consists of 2,500 samples. The distributions of samples from source domain and target domain are different since the sources are different. As in Amazon review dataset, we also chose the 5,000 most frequent terms as features. In our experiments, three testing samples were deleted as a result of not containing any of these 5,000 terms. Hence, we have 7,497 testing samples in this dataset.

**20 newsgroups dataset.** The third dataset was generated from the well-known 20 newsgroups dataset. The 20 newsgroups dataset contains 18,774 news documents with 61,188 features. It is organized in a hierarchical structure which consists of 6 main categories and 20 subcategories. The task is to classify the main categories and we have adopted a setting similar to [60]. The four largest main categories (i.e., `comp`, `rec`, `sci`, and `talk`) were chosen for evaluation. For each main category, the largest subcategory was selected as the source domain, while the second largest subcategory was chosen as the target domain. In our experiments, the largest category `comp` is considered as the positive class and one of the three other categories is considered as the negative class for each setting. Since the subcategories are different, the distributions of source domain and target domain are totally different. The settings of this dataset are listed in Table 1.

Since the distributions of source domain and target domain are different, the traditional cross validation can only select parameters that are suitable for source domain. Hence, we simply use a validation set containing a small number of labeled samples selected randomly from target domain to select parameters for feature learning algorithms. After we have the new representations, we treat the classification tasks as traditional ones and find the best classifier on source domain and apply it to the target domain. The validation set was not used to select parameters for classification. The descriptions of the datasets used are summarized in Tables 2, 3 and 4.

4.2 Performance Comparison

As a baseline, we train a linear SVM [61] on the tf-idf representation [62] of the labeled source domain data and test it on the target domain. When learning the new representations, we do not use any samples from other domains. We also present the results of new representation obtained by PCA, for which samples from both domains are used to

---


<table>
<thead>
<tr>
<th>Setting</th>
<th>Source Domain</th>
<th>Target Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comp vs. Rec</td>
<td>comp.windows.x</td>
<td>comp.sys.ibm.pc.hardware</td>
</tr>
<tr>
<td></td>
<td>rec.sport.hockey</td>
<td>rec.motorcycles</td>
</tr>
<tr>
<td>Comp vs. Sci</td>
<td>comp.windows.x</td>
<td>comp.sys.ibm.pc.hardware</td>
</tr>
<tr>
<td></td>
<td>sci.crypt</td>
<td>sci.med</td>
</tr>
<tr>
<td>Comp vs. Talk</td>
<td>talk.politics.mideast</td>
<td>talk.politics.guns</td>
</tr>
</tbody>
</table>

**TABLE 1: Descriptions of domain adaptation tasks generated from 20 Newsgroups.**
TABLE 2: Statistics of adaptation tasks on the Amazon review dataset. (B='Books', D='DVD', E='Electronics', and K='Kitchen').

<table>
<thead>
<tr>
<th>Dataset Source domain</th>
<th>Target domain Validation set</th>
</tr>
</thead>
<tbody>
<tr>
<td>B → D</td>
<td>2,000</td>
</tr>
<tr>
<td>B → E</td>
<td>2,000</td>
</tr>
<tr>
<td>B → K</td>
<td>2,000</td>
</tr>
<tr>
<td>D → B</td>
<td>2,000</td>
</tr>
<tr>
<td>D → E</td>
<td>2,000</td>
</tr>
<tr>
<td>D → K</td>
<td>2,000</td>
</tr>
<tr>
<td>E → B</td>
<td>2,000</td>
</tr>
<tr>
<td>E → D</td>
<td>2,000</td>
</tr>
<tr>
<td>E → K</td>
<td>2,000</td>
</tr>
<tr>
<td>K → B</td>
<td>2,000</td>
</tr>
<tr>
<td>K → D</td>
<td>2,000</td>
</tr>
<tr>
<td>K → E</td>
<td>2,000</td>
</tr>
</tbody>
</table>

TABLE 3: Statistics of adaptation tasks on the spam dataset.

<table>
<thead>
<tr>
<th>Dataset Source domain</th>
<th>Target domain Validation set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public → U0</td>
<td>4,000</td>
</tr>
<tr>
<td>Public → U1</td>
<td>4,000</td>
</tr>
<tr>
<td>Public → U2</td>
<td>4,000</td>
</tr>
</tbody>
</table>

TABLE 4: Statistics of adaptation tasks on the 20 newsgroups dataset.

<table>
<thead>
<tr>
<th>Dataset Source domain</th>
<th>Target domain Validation set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comp vs. Rec</td>
<td>1,979</td>
</tr>
<tr>
<td>Comp vs. Sci</td>
<td>1,971</td>
</tr>
<tr>
<td>Comp vs. Talk</td>
<td>1,922</td>
</tr>
</tbody>
</table>

compute the projection matrix. Besides these two baselines, we also compare with CODA [49], which is a state-of-the-art domain adaptation algorithm based on co-training. We compare our method with deep learning methods. Since mSDA is better than SDA [29], we only provide comparisons with mSDA. At last, the shallow methods GFK [21], CORAL [45], TCA [41] and FLDA [44] are also compared in our experiments. The performance metric is classification accuracy.

For mSDA and SRARAs, we use the raw binary unigram/bigram features as their inputs as in [29]. For all other algorithms, we use sample normalized $\text{tf-idf}$ representations. For representation learning algorithms, we select parameters with validation set. For PCA, we set the reduced dimensionality as 100 on the Amazon dataset and the 20 newsgroups dataset, and 200 on the Spam dataset.

For our method, with respect to the number of layers, 5 is used for Amazon review dataset, and 3 is used for both spam dataset and 20 newsgroups dataset. On Amazon dataset, $\alpha$ and $\lambda$ are selected from $[0.5, 0.9, 1, 2, 3, 4]$ and $[0.4, 0.5, 0.6, 0.7]$, respectively. On the spam dataset and 20 newsgroups dataset, they are both selected from $[2^{1}, 2^{2}, \cdots, 2^{7}]$. For CODA, the default parameters are used for the Amazon dataset and $\gamma$ is set to $10^{-4}$ for the spam dataset and 20 newsgroups dataset. Since the performance of CODA depends on the initializations, we run it 10 times and report the average performance. The corruption probability of mSDA is selected from $[0.0, 0.1, \cdots, 0.9]$. For FLDA, we use the code from the authors’ and the weight of $\ell_2$ regularizer is selected from $[10^{-5}, 10^{-4}, \cdots, 10^{5}]$. For SCL, the number of pivot feature is set to 2000 for the Amazon dataset and 1000 for the others. The low dimensionalities for SCL and GFK are both selected from $[305070]$. We use linear kernel for TCA, $\mu$ is selected from $[10^{-2}, 10^{-1}, \cdots, 10^{3}]$ and the low dimensionality is selected from $[5, 10, 20, 30, 50, 70, \cdots, 150]$ on the Amazon dataset and $[50, 100, \cdots, 300]$ for the other datasets.

The performance on Amazon review dataset, spam dataset, and 20 newsgroups dataset are shown in Tables 5, 6 and 7, respectively. The best results for each adaptation task have been marked in bold. One may observe that the deep learning methods, e.g., SRARAs and mSDA perform better than shallow learning methods, which is quite normal since deep learning methods usually learn better representations than shallow learning methods. Also, SRARAs is better than mSDA in quite a consistent manner, which proves that our feature learning method is more effective. Moreover, we can observe that PCA performs quite well on the Amazon dataset, but it does not provide good results on the other two datasets. The reason might be that the shared common subspace for both domains is easier to obtain on the Amazon dataset. It seems that domain adaptation on the spam dataset and 20 newsgroups dataset is more difficult than that on the Amazon dataset.

4.3 Convergence Analysis

We analyze the convergence property first. The convergence property is important for our method. If it converges very slowly, it will be not practicable. The objective values of Eq. (4) in the first layer during the optimization progress for tasks $B \rightarrow D$, Public $\rightarrow U0$ and Comp vs. Rec are presented in Figure 1. We can see that Algorithm 1 converges in only several iterations. This property makes it applicable for real world datasets.

4.4 Parameter Analysis

There are three parameters in our method: the intensity of non-linear transformation $\alpha$, the regularizer coefficient $\lambda$ and the number of layers. We study the effects of these parameters on the $B \rightarrow D$, Public $\rightarrow U0$ and Comp vs. Rec tasks.

We fix the number of layers and plot the accuracies with different values of $\alpha$ and $\lambda$ in Figure 2. The number of layers are 5, 3 and 3 for the $B \rightarrow D$, Public $\rightarrow U0$ and Comp vs. Rec tasks respectively. $\alpha$ controls the intensity of the squashing function. If $\alpha$ is large, the squashing will look like a Heaviside step function. Small alpha values, say 2 or 3, are good choices for Amazon review dataset. But for spam dataset and 20 newsgroups dataset, large $\alpha$s are needed. In addition, from Figure 2 we can see that the accuracy curves with different $\lambda$ show similar trends.

We also study the effect of the number of layers. We plot the accuracies with different $\alpha$, $\lambda$ and the number of layers on the same datasets as above, which are shown in Figure 3. We can see that our method usually performs the best with $3 \sim 5$ layers. Hence, we use 5, 3 and 3 for Amazon dataset, spam dataset and 20 newsgroups respectively for performance comparisons.

7. https://github.com/wmkouw/flda

TABLE 5: Domain adaptation performance of all methods on Amazon review dataset. The metric is classification accuracy. All values are reported as percentage (%). The best results are marked in bold.

<table>
<thead>
<tr>
<th>Method</th>
<th>mSDA</th>
<th>tf-idf</th>
<th>CORAL</th>
<th>FLDA</th>
<th>SCL</th>
<th>GFK</th>
<th>TCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>B → D</td>
<td>84.09</td>
<td>81.86</td>
<td>82.16</td>
<td>83.57</td>
<td>81.36</td>
<td>81.46</td>
<td>82.24</td>
</tr>
<tr>
<td>B → E</td>
<td>76.79</td>
<td>81.49</td>
<td>75.28</td>
<td>77.09</td>
<td>71.66</td>
<td>74.69</td>
<td>75.37</td>
</tr>
<tr>
<td>B → K</td>
<td>85.27</td>
<td>83.09</td>
<td>77.69</td>
<td>84.92</td>
<td>75.49</td>
<td>76.73</td>
<td>76.51</td>
</tr>
<tr>
<td>D → B</td>
<td>83.88</td>
<td>79.85</td>
<td>78.64</td>
<td>83.76</td>
<td>79.84</td>
<td>79.37</td>
<td>80.43</td>
</tr>
<tr>
<td>D → E</td>
<td>83.59</td>
<td>82.77</td>
<td>76.47</td>
<td>85.06</td>
<td>74.41</td>
<td>76.45</td>
<td>77.55</td>
</tr>
<tr>
<td>D → K</td>
<td>87.67</td>
<td>84.93</td>
<td>82.02</td>
<td>87.49</td>
<td>78.41</td>
<td>79.37</td>
<td>80.81</td>
</tr>
<tr>
<td>E → B</td>
<td>80.55</td>
<td>78.07</td>
<td>75.97</td>
<td>80.10</td>
<td>72.73</td>
<td>73.36</td>
<td>73.8</td>
</tr>
<tr>
<td>E → D</td>
<td>80.36</td>
<td>78.78</td>
<td>77.32</td>
<td>78.81</td>
<td>74.86</td>
<td>74.36</td>
<td>76.64</td>
</tr>
<tr>
<td>E → K</td>
<td>88.85</td>
<td>86.23</td>
<td>86.94</td>
<td>88.19</td>
<td>86.02</td>
<td>86.18</td>
<td>86.32</td>
</tr>
<tr>
<td>K → B</td>
<td>79.02</td>
<td>77.67</td>
<td>75.24</td>
<td>79.14</td>
<td>72.25</td>
<td>72.66</td>
<td>71.75</td>
</tr>
<tr>
<td>K → D</td>
<td>79.88</td>
<td>78.50</td>
<td>77.49</td>
<td>78.52</td>
<td>75.11</td>
<td>74.82</td>
<td>76.77</td>
</tr>
<tr>
<td>K → E</td>
<td>87.31</td>
<td>85.68</td>
<td>85.56</td>
<td>87.45</td>
<td>85.24</td>
<td>85.75</td>
<td>86.39</td>
</tr>
<tr>
<td>AVG</td>
<td>83.11</td>
<td>81.58</td>
<td>79.23</td>
<td>82.84</td>
<td>77.30</td>
<td>77.95</td>
<td>78.71</td>
</tr>
</tbody>
</table>

TABLE 6: Domain adaptation performance of all methods on spam review dataset. The metric is classification accuracy. All values are reported as percentage (%).

<table>
<thead>
<tr>
<th>Method</th>
<th>CODA</th>
<th>PCA</th>
<th>mSDA</th>
<th>tf-idf</th>
<th>CORAL</th>
<th>FLDA</th>
<th>SCL</th>
<th>GFK</th>
<th>TCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>B → D</td>
<td>93.05</td>
<td>90.81</td>
<td>63.88</td>
<td>91.64</td>
<td>61.93</td>
<td>77.78</td>
<td>82.29</td>
<td>64.18</td>
<td>83.99</td>
</tr>
<tr>
<td>B → E</td>
<td>95.45</td>
<td>92.65</td>
<td>73.62</td>
<td>92.79</td>
<td>69.76</td>
<td>80.98</td>
<td>84.33</td>
<td>72.72</td>
<td>84.33</td>
</tr>
<tr>
<td>B → K</td>
<td>94.3</td>
<td>94.4</td>
<td>73</td>
<td>93.65</td>
<td>74.65</td>
<td>87</td>
<td>86.7</td>
<td>77.3</td>
<td>86.65</td>
</tr>
<tr>
<td>AVG</td>
<td>94.26</td>
<td>92.62</td>
<td>70.16</td>
<td>92.69</td>
<td>68.78</td>
<td>81.92</td>
<td>84.44</td>
<td>71.40</td>
<td>84.86</td>
</tr>
</tbody>
</table>

TABLE 7: Domain adaptation performance of all methods on 20 newsgroups review dataset. The metric is classification accuracy. All values are reported as percentage (%).

<table>
<thead>
<tr>
<th>Method</th>
<th>CODA</th>
<th>PCA</th>
<th>mSDA</th>
<th>tf-idf</th>
<th>CORAL</th>
<th>FLDA</th>
<th>SCL</th>
<th>GFK</th>
<th>TCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comp vs. Rec</td>
<td>90.08</td>
<td>92.51</td>
<td>73.43</td>
<td>79.07</td>
<td>72.07</td>
<td>83.01</td>
<td>85.73</td>
<td>73.30</td>
<td>90.28</td>
</tr>
<tr>
<td>Comp vs. Sci</td>
<td>94.68</td>
<td>89.08</td>
<td>75.03</td>
<td>85.61</td>
<td>69.98</td>
<td>71.22</td>
<td>77.15</td>
<td>72.71</td>
<td>83.69</td>
</tr>
<tr>
<td>Comp vs. Talk</td>
<td>97.55</td>
<td>96.46</td>
<td>92.97</td>
<td>96.83</td>
<td>91.85</td>
<td>93.62</td>
<td>95.81</td>
<td>93.58</td>
<td>96.14</td>
</tr>
<tr>
<td>AVG</td>
<td>94.10</td>
<td>92.68</td>
<td>80.42</td>
<td>87.17</td>
<td>77.97</td>
<td>82.62</td>
<td>86.23</td>
<td>79.86</td>
<td>90.03</td>
</tr>
</tbody>
</table>

Fig. 1: Objective values of Eq. (4) during optimizations on different datasets.

4.5 Denoising Effects

The non-linear transformation step does not change the singular values too much. It just squashes the elements to interval $[−1, 1]$. Moreover, we find that even if we delete the non-linear transformation step, our method still improve the performance. But the performance is not as good as that of the method with the non-linear squashing step. Moreover, we plot the eigenvalues of the covariance matrices of output data matrix of each layer in Figure 4. The output of layer 0 is just the original raw data. For each set of eigenvalues that obtained from the same matrix, we normalized them such that they have sum equal to 1. To depict clearly, we only plot the first few eigenvalues. We can see that the big eigenvalues become relatively bigger and bigger through layers. The components with smaller eigenvalues become less important. Hence, SRARAs can be roughly seen as a procedure of denoising process layer by layer. Through the procedure, the components that are not important for both domains are weakened, the components that are important for both domains are strengthened. We believe that this property is important for SRARAs to achieve the best performance.

4.6 Distance Between Domains

The A-distance [63] is a measure of distance between two distributions. Smaller A-distance means the two distributions are similar to each other. In [7], it was suggested as a measure of similarity between the source domain and the target domain. Additionally, it was shown that A-distance is
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2837085, IEEE Transactions on Knowledge and Data Engineering.

Fig. 2: Classification accuracies with different $\alpha$ and $\lambda$ on different datasets.

Fig. 3: Classification accuracies with different number of layers on different datasets.

Fig. 4: Eigenvalues of covariance matrices of output from each layer on different datasets.

Fig. 5: Proxy-A-distance on different datasets.
In this subsection, we evaluate the extension for high dimensional data. The metric is classification accuracy. All values are reported as percentage (%). The best results are marked in bold.

<table>
<thead>
<tr>
<th>SRARAs</th>
<th>CODA</th>
<th>PCA</th>
<th>mSDA</th>
<th>tf-idf</th>
<th>FLDA</th>
<th>SCL</th>
<th>GFK</th>
<th>TCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>B → D</td>
<td>88.45</td>
<td>83.95</td>
<td>82.33</td>
<td>84.73</td>
<td>83.34</td>
<td>83.9</td>
<td>84.38</td>
<td>66.95</td>
</tr>
<tr>
<td>B → E</td>
<td>83.97</td>
<td>83.28</td>
<td>73.13</td>
<td>82.73</td>
<td>72.34</td>
<td>76.11</td>
<td>70.07</td>
<td>67.86</td>
</tr>
<tr>
<td>B → K</td>
<td>87.58</td>
<td>85.16</td>
<td>79.83</td>
<td>87.25</td>
<td>77.33</td>
<td>78.6</td>
<td>80.59</td>
<td>68.94</td>
</tr>
<tr>
<td>D → B</td>
<td>86.22</td>
<td>82.82</td>
<td>81.89</td>
<td>85.04</td>
<td>81.28</td>
<td>80.96</td>
<td>82.42</td>
<td>64.97</td>
</tr>
<tr>
<td>D → E</td>
<td>84.69</td>
<td>84.63</td>
<td>77.26</td>
<td>84.4</td>
<td>75.51</td>
<td>78.34</td>
<td>78.17</td>
<td>66.22</td>
</tr>
<tr>
<td>D → K</td>
<td>89.81</td>
<td>87.65</td>
<td>82.93</td>
<td>88.87</td>
<td>80.66</td>
<td>80.92</td>
<td>83.49</td>
<td>68.52</td>
</tr>
<tr>
<td>E → B</td>
<td>82.21</td>
<td>77.91</td>
<td>78.83</td>
<td>81.03</td>
<td>73.99</td>
<td>74.45</td>
<td>77.83</td>
<td>67.94</td>
</tr>
<tr>
<td>E → D</td>
<td>81.43</td>
<td>79.29</td>
<td>79.19</td>
<td>80.07</td>
<td>75.14</td>
<td>76.22</td>
<td>78.35</td>
<td>63.29</td>
</tr>
<tr>
<td>E → K</td>
<td>89.51</td>
<td>89.09</td>
<td>86.97</td>
<td>87.52</td>
<td>87.35</td>
<td>87.82</td>
<td>88.82</td>
<td>73.54</td>
</tr>
<tr>
<td>K → B</td>
<td>82.92</td>
<td>79.57</td>
<td>77.52</td>
<td>82.29</td>
<td>73.16</td>
<td>74.3</td>
<td>75.99</td>
<td>65.78</td>
</tr>
<tr>
<td>K → D</td>
<td>82.53</td>
<td>77.93</td>
<td>78.09</td>
<td>81.59</td>
<td>76.02</td>
<td>77.35</td>
<td>77.99</td>
<td>64.97</td>
</tr>
<tr>
<td>K → E</td>
<td>88.53</td>
<td>87.89</td>
<td>83.32</td>
<td>88.11</td>
<td>80.54</td>
<td>87.34</td>
<td>87.63</td>
<td>74.27</td>
</tr>
<tr>
<td>AVG</td>
<td>85.41</td>
<td>83.26</td>
<td>80.11</td>
<td>84.08</td>
<td>78.59</td>
<td>79.65</td>
<td>80.89</td>
<td>67.77</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SRARAs</th>
<th>CODA</th>
<th>PCA</th>
<th>mSDA</th>
<th>tf-idf</th>
<th>FLDA</th>
<th>SCL</th>
<th>GFK</th>
<th>TCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public  → U0</td>
<td>92.85</td>
<td>86.69</td>
<td>74.2</td>
<td>89.75</td>
<td>66.5</td>
<td>83.65</td>
<td>69.15</td>
<td>75.85</td>
</tr>
<tr>
<td>Public  → U1</td>
<td>94.9</td>
<td>91.88</td>
<td>63.6</td>
<td>93.6</td>
<td>74.65</td>
<td>83.85</td>
<td>74.35</td>
<td>79.85</td>
</tr>
<tr>
<td>Public  → U2</td>
<td>95.4</td>
<td>94.39</td>
<td>80.7</td>
<td>93.95</td>
<td>80.35</td>
<td>87.7</td>
<td>80.3</td>
<td>88.8</td>
</tr>
<tr>
<td>AVG</td>
<td>94.38</td>
<td>90.98</td>
<td>72.83</td>
<td>92.43</td>
<td>73.83</td>
<td>85.07</td>
<td>74.6</td>
<td>81.5</td>
</tr>
</tbody>
</table>

For our method, the parameter $\alpha$ are selected from $[20, 22, \cdots, 30]$ and $[1, 3, \cdots, 20]$ on the spam dataset and the 20 newsgroups dataset, respectively. The parameter $\lambda$ are selected from $[1, 3, \cdots, 15]$ on those two datasets. The other settings are the same as in the experiments for low dimensional data. CORAL is not compared here, since it involves eigenvalue decomposition and is not suitable for high dimensional data. The reduced dimensionality for the first layer is set as 5,000. The results of all competing methods are presented in Tables 8, 9 and 10. We can see that our method outperforms the competing methods with a large margin. Hence, our method is better at extracting a crucial part of upper bound of generalization error for domain adaptation [7]. Hence, the A-distance should be small in order to have good generalization from source domain to target domain. In practice, the exact A-distance is impossible to compute. A proxy-A-distance is used instead and it is defined as $d_A = \frac{1}{2}(d_A - 2\epsilon)$, where $\epsilon$ is the generalization error of a classifier trained to discriminate the source domain and the target domain. In this paper, $\epsilon$ is computed with a linear SVM.

References [28] and [29] showed that the proxy-A-distance actually increases after the representation learning on the Amazon dataset, meaning that the new representations are suitable for both sentiment analysis tasks and domain classification tasks. This phenomenon is also observed on features learned by our method on the Amazon review dataset, which can be seen from Figure 5(a). But Figure 5(b) and Figure 5(c) show that the proxy-A-distance decreased on the spam dataset and the 20 newsgroups dataset. However, our method performed well on all the three datasets. Therefore, proxy-A-distance is not necessarily a good indicator for good domain adaptation performance. The features that are useful for discriminating domain might be either weakened or strengthened in the feature learning process. Hence, the proxy-A-distance might get either bigger or smaller after feature learning.

### 4.7 High Dimensional Data

In this subsection, we evaluate the extension for high dimensional data on the three datasets used. For spam dataset, tokens occurred more than 4 times are considered. For the Amazon dataset and the 20 newsgroups dataset, all features are used in our experiments. The numbers of features are listed in Table 11. We can see that the number of features is 6 → 10 times larger than that in the low dimensional dataset. Hence, much more low frequency words are contained. The adaptation model is required to extract helpful information from these low frequency features. Otherwise, those words might have negative effects on the adaptation performance.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon</td>
<td>30,000</td>
</tr>
<tr>
<td>Spam</td>
<td>46,256</td>
</tr>
<tr>
<td>20 newsgroups</td>
<td>61,188</td>
</tr>
</tbody>
</table>

For our method, the parameter $\alpha$ are selected from $[20, 22, \cdots, 30]$ and $[1, 3, \cdots, 20]$ on the spam dataset and the 20 newsgroups dataset, respectively. The parameter $\lambda$ are selected from $[1, 3, \cdots, 15]$ on those two datasets. The other settings are the same as in the experiments for low dimensional data. CORAL is not compared here, since it involves eigenvalue decomposition and is not suitable for high dimensional data. The reduced dimensionality for the first layer is set as 5,000. The results of all competing methods are presented in Tables 8, 9 and 10. We can see that our method outperforms the competing methods with a large margin. Hence, our method is better at extracting...
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2837085, IEEE Transactions on Knowledge and Data Engineering

5 Experimental Results on Image Dataset

In this section, we will illustrate the effectiveness of our method on image dataset. We adopt the same datasets used in [21]. In this dataset, 10 common classes were extracted from Amazon, Webcam (low-resolution images), DSLR (high-resolution images) [20] and Caltech-256 datasets [64]. There are 2,533 images in total, and 8 to 151 samples per category per domain. Some examples are presented in the Figure 6. The numbers of samples in each domain are shown in Table 12.

![Example images](image)

**TABLE 12: Statistics of image domain adaptation dataset.**

<table>
<thead>
<tr>
<th>Domain</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon</td>
<td>958</td>
</tr>
<tr>
<td>DSLR</td>
<td>157</td>
</tr>
<tr>
<td>Webcam</td>
<td>295</td>
</tr>
<tr>
<td>Caltech256</td>
<td>1,123</td>
</tr>
<tr>
<td>Total</td>
<td>2,533</td>
</tr>
</tbody>
</table>

The SURF features were encoded with 800-bin bag-of-words histograms. There are four domains, hence there are 12 adaptation tasks, namely, A → C, A → D, A → W, and so on. Since the number of samples is small, we do not use validation set. For our method, the normalized tf-idf is used and the parameters α and λ are set to 2.5 and 1.5. For original data, we do not perform any preprocessing. For CORAL, each sample is normalized to have unit length and features are standardized to have zero mean and unit standard deviation for each domain as in [45]. For PCA, we use tf-idf representation and the dimensionality was reduce to 50. CODA [24] is not designed for multi-class classification, we do not compare our method with CODA. The comparison results are presented in Table 13. We can see that our method achieved the best performance. But the superiority is not that obvious. The reason might be that the number of samples is small and the “words” used in SURF features are not as semantically meaningful as those in the text datasets.

6 Conclusion

The study of deep learning for domain adaptation is still in its preliminary stage. Motivated by the remarkable performance of mSDA [29], which is a new deep learning method for domain adaptation, we proposed the stacked robust adaptively regularized auto-regressions (SRARAs) to learn better representations for domain adaptation tasks. Our method is a simple combination of statistics tool and deep architecture. The intuition behind SRARAs is that we should adjust the weights of components of data matrix consisting of all samples from both source domain and target domain. The proposed method is based on a loss + regularizer framework, which makes our method easier to implement and analyze compared to neural network based methods. We made an attempt to provide understandings by analyzing the eigenvalues of covariance matrices of outputs and showed that the noisy components are weakened relatively through layers. The experimental results on text and image datasets showed that the proposed method is very promising for domain adaptation tasks.

References

TABLE 13: Domain adaptation performance on image domain adaptation dataset. The metric is classification accuracy. All values are reported as percentage (%). (A='Amazon', C='Caltech-256', D='DSLR' and W='Webcam').

<table>
<thead>
<tr>
<th></th>
<th>SRARAs</th>
<th>CORAL</th>
<th>PCA</th>
<th>mSDA</th>
<th>original</th>
<th>FLDA</th>
<th>SCL</th>
<th>GFK</th>
<th>TCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>A → C</td>
<td>47.55</td>
<td>45.08</td>
<td>46.57</td>
<td>47.01</td>
<td>44.87</td>
<td>45.77</td>
<td>45.59</td>
<td>40.25</td>
<td>46.92</td>
</tr>
<tr>
<td>A → D</td>
<td>38.21</td>
<td>39.49</td>
<td>38.21</td>
<td>39.49</td>
<td>40.76</td>
<td>38.85</td>
<td>38.85</td>
<td>36.31</td>
<td>37.58</td>
</tr>
<tr>
<td>A → W</td>
<td>41.35</td>
<td>44.41</td>
<td>42.03</td>
<td>38.98</td>
<td>31.86</td>
<td>37.63</td>
<td>35.59</td>
<td>38.98</td>
<td>39.66</td>
</tr>
<tr>
<td>C → A</td>
<td>59.29</td>
<td>52.08</td>
<td>58.03</td>
<td>58.35</td>
<td>48.32</td>
<td>52.19</td>
<td>57.41</td>
<td>41.02</td>
<td>59.08</td>
</tr>
<tr>
<td>C → D</td>
<td>52.86</td>
<td>56.31</td>
<td>49.04</td>
<td>55.41</td>
<td>50.30</td>
<td>45.93</td>
<td>48.15</td>
<td>38.85</td>
<td>50.14</td>
</tr>
<tr>
<td>C → W</td>
<td>43.72</td>
<td>46.44</td>
<td>46.1</td>
<td>43.05</td>
<td>26.10</td>
<td>43.73</td>
<td>43.73</td>
<td>40.68</td>
<td>41.02</td>
</tr>
<tr>
<td>D → A</td>
<td>40.81</td>
<td>37.68</td>
<td>34.34</td>
<td>35.20</td>
<td>28.28</td>
<td>36.33</td>
<td>32.77</td>
<td>30.28</td>
<td>36.42</td>
</tr>
<tr>
<td>D → C</td>
<td>37.31</td>
<td>33.85</td>
<td>30.66</td>
<td>37.67</td>
<td>30.81</td>
<td>34.11</td>
<td>32.77</td>
<td>30.28</td>
<td>36.42</td>
</tr>
<tr>
<td>D → W</td>
<td>84.74</td>
<td>84.74</td>
<td>84.06</td>
<td>81.69</td>
<td>67.79</td>
<td>81.69</td>
<td>75.25</td>
<td>75.59</td>
<td>83.39</td>
</tr>
<tr>
<td>W → A</td>
<td>39.45</td>
<td>35.91</td>
<td>38.51</td>
<td>39.14</td>
<td>32.46</td>
<td>38.31</td>
<td>33.92</td>
<td>29.75</td>
<td>34.66</td>
</tr>
<tr>
<td>W → C</td>
<td>38.46</td>
<td>33.74</td>
<td>35.44</td>
<td>37.04</td>
<td>31.25</td>
<td>35.44</td>
<td>33.84</td>
<td>30.72</td>
<td>36.51</td>
</tr>
<tr>
<td>W → D</td>
<td>88.53</td>
<td>86.62</td>
<td>88.53</td>
<td>85.98</td>
<td>80.25</td>
<td>84.08</td>
<td>85.35</td>
<td>80.89</td>
<td>87.26</td>
</tr>
<tr>
<td>AVG</td>
<td>51.02</td>
<td>48.02</td>
<td>49.74</td>
<td>50.17</td>
<td>41.48</td>
<td>47.67</td>
<td>46.96</td>
<td>42.94</td>
<td>49.64</td>
</tr>
</tbody>
</table>


Wenhao Jiang received his B.E. and M.E. in computer science from Shandong University and Harbin Institute of Technology Shenzhen Graduate School, China, in 2006 and 2009, respectively. He received the Ph.D. degree in the Department of Computing from the Hong Kong Polytechnic University in 2014 and has worked at Tencent AI Lab since 2016. His research interests include machine learning and computer vision.

Hongchang Gao received the BS degree from Ocean University of China in 2011 and the MS degree from Beihang University, China, in 2014. He is currently working toward the Ph.D degree at University of Pittsburgh under the supervision of Dr. Heng Huang. His research interests include machine learning, data mining and computer vision.

Wei Lu received her B.E. in Electronic Engineering from University of Electronic Science and Technology of China, and M.S. in Electrical and Computer Engineering from Duke University in 2009 and 2012, respectively. She received the Ph.D. degree in Computer Science from the Hong Kong Polytechnic University in 2017. Since 2017, she has been an assistant professor in the School of Management and Economics, University of Electronic Science and Technology of China. Her research interests include machine learning, data mining and information systems.

Wei Liu received the MPhil and Ph.D degrees in electrical engineering from Columbia University, New York, in 2012. He is currently a computer vision director of the Tencent AI Lab, Shenzhen, China. He has been a research scientist with the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, since 2012. He received the 2011-2012 Facebook Fellowship and 2013 Jury Award for best thesis of the Department of Electrical Engineering, Columbia University. He holds adjunct faculty positions at the Rensselaer Polytechnic Institute (RPI) and Stevens Institute of Technology. His research interests include machine learning, computer vision, artificial intelligence, data mining, and information retrieval. He is a member of the IEEE.

Fu-lai Chung Upon receiving my M.Phil. and Ph.D. degrees from the Chinese University of Hong Kong and a B.Sc. degree from the University of Manitoba, I joined the Department of Computing, Hong Kong Polytechnic University and am now an associate professor. My primary research interests are in the study of intelligence, learning and recognition and their applications to a wide spectrum of tasks which involves text data (social media), time series data, multimedia data, social network data, bioinformatics data, etc. I published over 90 journal papers in major international journals, including IEEE TIFS, IEEE TNNLS, IEEE TEC, IEEE TSMC-B, Pattern Recognition and Neural Networks. I also serve on program committees of top international conferences, including IEEE ICDM and ICPR.

Heng Huang received the BS and MS degrees from Shanghai Jiao Tong University, Shanghai, China, in 1997 and 2001, respectively. He received the Ph.D degree in computer science from Dartmouth College in 2006. Since 2017, he has been a professor in the Department of Electrical and Computer Engineering, University of Pittsburgh. His research interests include machine learning, data mining, bioinformatics, neuroinformatics, and health informatics.