Population-based Ensemble Classifier Induction for Domain Adaptation

Bach Hoai Nguyen, Bing Xue, Mengjie Zhang, and Peter Andreae
School of Engineering and Computer Science
Victoria University of Wellington, PO Box 600, Wellington 6140, New Zealand
{Hoai.Bach.Nguyen,Bing.Xue,Mengjie.Zhang,Peter.Andreae}@ecs.vuw.ac.nz

ABSTRACT

In classification, the task of domain adaptation is to learn a classifier to classify target data using unlabeled data from the target domain and labeled data from a related, but not identical, source domain. Transfer classifier induction is a common domain adaptation approach that learns an adaptive classifier directly rather than first adapting the source data. However, most existing transfer classifier induction algorithms are gradient-based, so they can easily get stuck at local optima. Moreover, they usually generate only a single classifier which might fit the source data too well, which results in poor target accuracy. In this paper, we propose a population-based algorithm that can address the above two limitations. The proposed algorithm can re-initialize a population member to a promising region when the member is trapped at local optima. The population-based mechanism allows the proposed algorithm to output a set of classifiers which is more reliable than a single classifier. The experimental results show that the proposed algorithm achieves significantly better target accuracy than four state-of-the-art and well-known domain adaptation algorithms on three real-world domain adaptation problems.

CCS CONCEPTS

• **Computing methodologies** → **Transfer learning**; **Bio-inspired approaches**; *Classification and regression trees*;

KEYWORDS

Transfer Learning, Domain adaptation, Classification, Evolutionary Computation

ACM Reference Format:
Bach Hoai Nguyen, Bing Xue, Mengjie Zhang, and Peter Andreae. 2019. Population-based Ensemble Classifier Induction for Domain Adaptation. In *Genetic and Evolutionary Computation Conference (GECCO '19), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3321707.3321716

1 INTRODUCTION

In standard supervised learning, a classifier trained on a training dataset will be applied to a target dataset in the same domain, i.e., same feature space and domain distribution. However, in many real-world applications, the above assumption does not hold — the target data may have a different feature distribution, or even a different feature space [12, 17]. Most machine learning models need to be re-trained if the target data has a different distribution, which is not only time-consuming but also expensive when labeled data is difficult to obtain. For example, in a Wi-Fi localization problem, the Wi-Fi signal distributions change frequently since they depend on many dynamic factors such as usage and location. Furthermore, labeled training Wi-Fi data under different usage or in a new location is difficult to obtain [33]. Thus, it would be desirable if the labeled data from an existing domain (*source domain*) can be reused to improve the learning performance on another similar domain (*target domain*). Such techniques are called transfer learning [24].

If the two domains have the same feature space, the transfer learning task is called domain adaptation [24], which is the focus of this work. The main task of domain adaptation is to reduce the divergence between data from the source and target domains. According to a literature survey [24], domain adaptation can be classified into two main groups: *feature extraction*, which extracts a common feature set on which the source and target data becomes more similar, and *instance reweighting*, which assigns weights to each instance to reduce the differences between the two domains. Feature extraction can be further divided into subcategories: *transfer subspace learning* and *transfer classifier induction* [19]. Transfer subspace learning firstly learns a new common feature space, and then performs classification using the learned feature space. In contrast, transfer classifier induction merges the two steps and directly learns an adaptive classifier, which considers the interaction between the learned features and the learned classifier. Transfer classifier induction usually results in better performance than transfer subspace learning [31], and is the focus of this paper.

Although existing transfer classifier induction algorithms achieve promising results, they have two main limitations. Firstly, most, if not all, algorithms in this category use gradient-based optimization, which makes them prone to getting trapped at local optima. Some other optimization techniques are more able to avoid local optima. Secondly, they usually obtain only a single classifier which is likely to be too specific — overly fitted — to the source domain. Because the target domain is not identical to the source domain in domain adaption, it is critical to have appropriately general classifiers. Building multiple classifiers usually results in stronger generalization ability than a single classifier, which is the primary motivation of ensemble learning [10]. Therefore, it would be promising if multiple classifiers can be obtained instead of a single classifier.

Evolutionary Computation (EC) [1] is a population-based optimization family that is well-known because of their potential global

GECCO '19, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6111-8/19/07…$15.00
https://doi.org/10.1145/3321707.3321716

search ability. In EC, there is a set of candidate solutions that are initialized and iteratively updated using ideas from biological evolution such as natural selection and mutation. As a result, the quality of the candidate solutions can be gradually improved. EC naturally addresses the above two limitations of transfer classifier induction algorithms. Firstly, the population-based searching mechanism is useful to avoid being trapped at local optima. Secondly, using a set of candidate solutions results in a set of classifiers that can be more reliable than the single classifier evolved by existing transfer classifier induction algorithms. However, applying a standard EC technique to solve the transfer classifier induction problem is not straightforward or trivial since the problem is an ultra-high-dimensional optimization problem (easily up to tens of thousands of decision variables) as will be explained later. Since the number of possible solutions increases exponentially with the number of decision variables, standard EC needs a huge population size to explore the promising regions of such ultra-high-dimensional problems well [8]. Given a limited time/resource budget, standard EC does not work as well as the gradient-based algorithms.

Goal: The overall goal of this paper is to develop a population-based algorithm that can take advantages of both EC and gradient-based techniques to effectively and efficiently evolve an ensemble of classifiers for addressing domain adaptation problems. The design of the proposed algorithm addresses four main questions. The first one is how to represent the optimization problem. The second one is how to initialize the population. A poor initialization usually leads to poor results. The third one is how to evaluate the candidate solutions, which is problem-dependent. The last, and most important, question is how to evolve the candidate solutions during the evolutionary process. The evolutionary mechanism of the proposed algorithm exploits the characteristics of domain adaptation problems to generate a good set of classifiers. The proposed algorithms is evaluated on three real-world domain adaptation problems. Specifically, we will investigate:

- whether the proposed algorithm can evolve a set of classifiers that can classify the target data more accurately than directly training standard classification algorithms such as KNN, SVM, and Random Forest on the source data, and
- whether the proposed algorithm improves the classification performance over using existing state-of-the-art domain adaptation algorithms, such as Transfer Component Analysis (TCA) [24], Joint Domain Adaptation (JDA) [20], Geodesic Flow Kernel (GFK) [14], and Manifold Embedded Distribution Alignment (MEDA) [31].

2 BACKGROUND

2.1 Transfer Learning and Domain Adaptation

In transfer learning, there are two essential concepts: *domain* and *task*. A domain consists of two components: a feature space $\mathcal{X}$ and a marginal distribution $P(\mathcal{X})$. Two domains are different if they have different feature spaces or different marginal distributions. On each domain, a learning task is defined by a label space $\mathcal{Y}$ and a prediction function $f(.)$ that maps from the feature space $\mathcal{X}$ to the label space $\mathcal{Y}$. In a classification task, $f(.)$ can be expressed as a conditional distribution $Q(\mathcal{Y}|\mathcal{X})$. The two tasks can be different in terms of their label spaces or their conditional distributions. Transfer learning techniques aim to utilize information from an existing (source) domain to improve the learning performance in the target domain. In this work, we assume that the source domain D_s and the target domain D_t share the same feature space ($\mathcal{X}_s = \mathcal{X}_t$), the two label spaces are also the same ($\mathcal{Y}_s = \mathcal{Y}_t$). The goal is to learn a classifier $f : \mathcal{X}_t \rightarrow \mathcal{Y}_t$ that can classify D_t accurately under the conditions $P(\mathcal{X}_s) \neq P(\mathcal{X}_t)$ and $Q(\mathcal{Y}_s|\mathcal{X}_s) \neq Q(\mathcal{Y}_t|\mathcal{X}_t)$, which is usually known as domain adaptation. Domain adaptation can be further classified as supervised techniques and unsupervised techniques, depending on whether the labeled instances are available in the target domain or not. In this paper, we focus on handling the unsupervised case since it is more challenging. The main task of this work can be expressed as following:

Unsupervised domain adaptation: *Given a labeled source data with n instances, $D_s = \{x_{s_i}, y_{s_i}\}_{i=1}^{n}$ and an unlabeled target data with m instances, $D_t = \{x_{t_j}\}_{j=1}^{m}$, the goal of unsupervised domain adaptation is to learn a classifier $f : \mathcal{X}_t \rightarrow \mathcal{Y}_t$ with a low classification error on D_t, where $\mathcal{X}_s = \mathcal{X}_t, \mathcal{Y}_s = \mathcal{Y}_t, P(\mathcal{X}_s) \neq P(\mathcal{X}_t)$, and $Q(\mathcal{Y}_s|\mathcal{X}_s) \neq Q(\mathcal{Y}_t|\mathcal{X}_t)$.*

In domain adaptation, two main approaches are based on instances and features, respectively [24]. The instance-based approaches assume that some parts of the source data can be reused to improve the learning performance on the target domain. The task is to minimize the distribution differences by re-weighting the instances [13, 27, 32]. The re-weighted instances can be used as additional training data to train a classification algorithm on the target domain. Maximum Mean Discrepancy (MMD) [6] is widely used to measure the distances between different distributions.

Feature-based approaches find a shared latent subspace on which the differences between the two domains are minimized. Transfer Component Analysis (TCA) [23] finds a low-dimensional space where the data variance is preserved as much as possible while the distribution difference measured by MMD is reduced. Transfer Subspace Learning (TSL) [25] uses a Bregman divergence-based measure to calculate the distance between the two data distributions. Joint Domain Adaptation (JDA) [20] is the first approach that can jointly adapt both marginal and conditional distributions while there are no labeled instances on the target domain. Multiple intermediate feature spaces can also be considered as a bridge between the source and the target domains. Gopalan et al. [15] use potential intermediate subspaces to form a geodesic curve (path) over a Grassmann manifold. The path is expected to reveal information about domain changes from which a discriminative classifier can be learned to classify the target data. The idea is further extended in Geodesic Flow Kernel (GFK) [14] that considers all the intermediate subspaces. Instead of finding an intermediate latent feature space, Subspace Alignment (SA) [11] directly maps from the source feature space to the target feature space. However, SA does not adapt feature distributions, which is handled later by Subspace Distribution Alignment (SDA) [26].

Most feature-based approaches firstly find a common feature representation. The classifier is only learned as a second step after the common features are built [22]. It is undeniable that finding a good feature space is essential in adapting the two domains [5, 21], but learning a classifier as a side step may miss the interaction

between the common feature space and the classifier. Some works are recently proposed to directly learn a domain-invariant classifier by incorporating the adaptation of different distributions through model regularization [2, 3, 7, 19]. These methods are usually categorized as transfer classifier induction. However, most, if not all, methods in this category are gradient-based, which makes them easier to be trapped in local optima. Besides, since only one classifier is built during the learning process, the generalization of the learned classifier is low, which may result in a low classification performance on the target domain. In this work, we propose a population-based algorithm to address the two limitations. Note that the proposed algorithm can be viewed as a framework which can be applied to any transfer classifier induction algorithms. In this work, we select Manifold Embedded Distribution Alignment (MEDA) [31] since it is one of the most state-of-the-art transfer classifier induction algorithms.

2.2 Manifold Embedded Distribution Alignment (MEDA)

While existing transfer classifier induction algorithms work directly on the original feature space, MEDA firstly transfers the original space into a manifold space to avoid feature distortion. Particularly, MEDA applies GFK to learn a transformation matrix $\boldsymbol{G}$ that transforms the original space into Grassmann manifold. Denoting $\boldsymbol{X}$ as the data matrix ($\boldsymbol{X} = [\boldsymbol{X}_s, \boldsymbol{X}_t]$), $\boldsymbol{Z}$ is the newly generated data by the transformation matrix $\boldsymbol{G}$ ($\boldsymbol{Z} = \sqrt{\boldsymbol{G}}\boldsymbol{X}$). Thus, $\boldsymbol{Z} = \{z_i\}_{i=1}^{m+n}$ where $\{z_i\}_{i=1}^{n}$ is the transformed data of the n source instances, $\{z_j\}_{j=n+1}^{m+n}$ is the transformed data of the m target instances. MEDA aims to learn a classifier f minimizing the following objective function:

$$F = \arg\min_{f \in H_K} \sum_{i=1}^{n} (y_i - f(z_i))^2 + \mu ||f||_K^2 + \lambda D_f(D_s, D_t) + \rho R_f(D_s, D_t) \quad (1)$$

where H_K is the Hilbert space induced by kernel function $K(.,.)$, μ, λ, and ρ are used to control the contributions of three components which are explained in the rest of this section.

The first component is the loss when f is applied to the source instances (note that n is the number of source instances), and the second component is the regularization term. Following structural risk minimization (SRM) [29], the first two components are to ensure that the classifier f should have a good classification performance on the source data. According to the representer theorem [4], f can be represented by the expansion:

$$f(u) = \sum_{i=1}^{n+m} \beta_i * K(z_i, u) \quad (2)$$

where u is an unlabeled instance, $\beta_i \in \mathbb{R}^d$ (d is the number of original features). Denote $\boldsymbol{\beta} = (\beta_1, \beta_2, ..., \beta_{n+m})^T$, $\boldsymbol{K} \in \mathbb{R}^{(n+m)\times(n+m)}$ is the kernel matrix, $\boldsymbol{A}^{(n+m)\times(n+m)}$ is a diagonal domain indicator matrix with $\boldsymbol{A}_{ii} = 1$ iff $1 \le i \le n$, the SRM part can be rewritten as:

$$\sum_{i=1}^{n} (y_i - f(z_i))^2 + \mu ||f||_K^2 = ||(\boldsymbol{Y} - \boldsymbol{\beta}^T \boldsymbol{K})\boldsymbol{A}||_F^2 + \mu \times tr(\boldsymbol{\beta}^T K \boldsymbol{\beta}) \quad (3)$$

where $||.||_F$ is the Frobenious norm, and $tr(.)$ is the trace of a matrix.

The third component, $D_f(D_s, D_t)$, represents the distribution difference to be minimized. MEDA accounts for the relative importance of marginal and conditional distributions. For example, if the two domains are more different in terms of the marginal distribution, a larger weight is assigned to the marginal distribution difference, and vice versa. The relative importance is controlled by α as in the following equation:

$$D_f(D_s, D_t) = (1 - \alpha) D_f(P_s, P_t) + \alpha \sum_{c=1}^{C} D_f^{(c)}(Q_s, Q_t) \quad (4)$$

where $\alpha \in [0, 1]$, $c \in \{1, ..., C\}$ is the class value, $D_f(P_s, P_t)$ is the marginal distribution difference, and $D_f^{(c)}(Q_s, Q_t)$ is the conditional distribution difference for class c. Denote $\boldsymbol{M}$ as the MMD matrix, the third component can be rewritten as:

$$D_f(D_s, D_t) = tr(\boldsymbol{\beta}^T \boldsymbol{K} \boldsymbol{M} \boldsymbol{K} \boldsymbol{\beta}) \quad (5)$$

The last component $R_f(D_s, D_t)$ is to preserve the similar geometrical property. In particular, if the j^{th} instance is one of the nearest neighbors of the i^{th} instance, it is expected that the two instances should belong to the same class. $R_f(D_s, D_t)$ can be calculated as:

$$R_f(D_s, D_t) = tr(\boldsymbol{\beta}^T \boldsymbol{K} \boldsymbol{L} \boldsymbol{K} \boldsymbol{\beta}) \quad (6)$$

where $\boldsymbol{L}$ is the Laplacian matrix. Interested readers can refer to the original paper of MEDA [31] for more details about how to calculate $\boldsymbol{K}$, $\boldsymbol{M}$, and $\boldsymbol{L}$.

In general, MEDA represents the classifier f as the matrix $\boldsymbol{\beta} \in \mathbb{R}^{(n+m)\times d}$, and its optimization task is to find the optimal matrix $\boldsymbol{\beta}^\star$ that can minimize the following objective function:

$$F = ||(\boldsymbol{Y} - \boldsymbol{\beta}^T \boldsymbol{K})\boldsymbol{A}||_F^2 + \mu \times tr(\boldsymbol{\beta}^T \boldsymbol{K} \boldsymbol{\beta}) + tr(\boldsymbol{\beta}^T \boldsymbol{K} (\lambda \boldsymbol{M} + \rho \boldsymbol{L}) \boldsymbol{K} \boldsymbol{\beta}) \quad (7)$$

Setting the derivative $\partial F / \partial \boldsymbol{\beta} = 0$, MEDA obtains the solution:

$$\boldsymbol{\beta}^\star = ((\boldsymbol{A} + \lambda \boldsymbol{M} + \rho \boldsymbol{K}) + \mu \boldsymbol{I})^{-1} \boldsymbol{A} \boldsymbol{Y}^T \quad (8)$$

where $\boldsymbol{I}$ is the identity matrix.

Since $\boldsymbol{A}, \boldsymbol{K}, \boldsymbol{I}$ are fixed matrices, $\boldsymbol{M}$ depends on $\boldsymbol{Y}$, $\boldsymbol{\beta}^\star$ actually depends only on $\boldsymbol{Y}$ that contains labels of both source and target instances. However, in the beginning, the target instances are unlabeled, so MEDA trains a KNN classifier (K=1) on the source instances and applies the learned classifier to obtain pseudo labels for the target instances. The optimization process starts with the pseudo labels to obtain $\boldsymbol{\beta}$ that in turns generates new pseudo labels for the target instances. This process repeats until convergence, or a maximum number of iterations is reached. MEDA does not calculate the objective values of its candidate solutions. Therefore, in MEDA, sometimes solutions generated in the later generation generated is even worse than the previous one.

MEDA is also a gradient-based algorithm which generates only one classifier and is easily stuck at local optima. It is expected that a population-based search mechanism such as EC can address the two limitations by evolving an ensemble of classifiers. However, the task is to find an optimal matrix $\boldsymbol{\beta}$ that contains a large number of values. Specifically, $\boldsymbol{\beta} \in \mathbb{R}^{(n+m)\times d}$ which means $\boldsymbol{\beta}$ contains $(n + m) \times d$ values to be optimized. For a simple domain adaptation case, GasSensor1-4 which has 128 features ($d = 128$), 178 source instances

(n = 178), 161 target instances (m = 161), the total number of values is (178+161)*128 = 43,392. For such a high-dimensional problem, directly applying EC algorithms may not work well. Therefore, this work aims to develop a population-based algorithm that can integrate well with MEDA to obtain a set of classifiers which has better performance than the classifier evolved by MEDA.

3 THE PROPOSED METHOD

This section presents our proposed algorithm , i.e. population-based manifold embedded distribution alignment (P-MEDA). The first subsection shows how to represent the problem and how to evaluate the candidate solutions. The second one explains how the candidate solutions are evolved. The last subsection presents the overall algorithm including how to initialize the population.

3.1 Representation and Fitness Function

The task of MEDA is to find an optimal matrix $\boldsymbol{\beta} \in \mathbb{R}^{(n+m)\times d}$ which minimizes the fitness function as shown in Eq. (7). $\boldsymbol{\beta}$ actually consists of the parameters for a classifier described in Eq. (2). P-MEDA builds on MEDA, and we represent $\boldsymbol{\beta}$ by a vector containing $(n + m) \times d$ real values. This is a straightforward representation which is very similar to other vector-based EC algorithms such as particle swarm optimization [18] or genetic algorithms [9].

Unlike MEDA, P-MEDA explicitly evaluates the candidate solutions. The first step is to convert each candidate solution to the corresponding matrix $\boldsymbol{\beta}$. The obtained $\boldsymbol{\beta}$ is then used to classify the target instances, which results in pseudo target labels. Based on the pseudo target labels, the MMD matrix $\boldsymbol{M}$ can be calculated. Finally, all the matrices are substituted in Eq. (7) to obtain the fitness value of the candidate solution. Once calculated, the pseudo target labels and the fitness value are recorded in the candidate solution.

3.2 Evolving Solutions

This subsection presents how P-MEDA generates new candidate solutions from previous ones. Similar to other EC-algorithms, each candidate solution (sol_c) plays the role of a parent in generating a new candidate solution. Based on the pseudo target labels of sol_c, a new $\boldsymbol{\beta}$ matrix is generated by Eq. (8), called sol_n which will be the best candidate in the immediate neighborhood of sol_c.

While MEDA replaces the current candidate solution by the newly generated solution regardless of the goodness of the new solution, P-MEDA does not follow that strategy. P-MEDA firstly evaluates a new solution, sol_n, using Eq. (7). P-MEDA then compares the fitness value of sol_n and its parent sol_c. If the fitness value of sol_n is better (smaller) than that of sol_c, sol_c is replaced by sol_n. If the fitness value of sol_n is worse than sol_c then sol_c must be a local optimal solution which cannot be further improved by the application of Eq. (8). P-MEDA will add sol_c to its archive set $\mathcal{A}$ which stores all such local optimal solutions, and then replaces sol_c by a newly initialized candidate solution. Each member of the archive set is considered a good classifier that can be used in the final ensemble of classifiers to classify the target instances.

After adding sol_c to the archive set, P-MEDA re-initializes sol_c to a new location. A simple strategy is to generate a new position randomly, but this does not guarantee that the new position is in a promising region to explore. Another strategy is to inherit useful information obtained from the previous generations to get a promising point, which follows the main idea of genetic operators in EC techniques. Particularly, the archive set $\mathcal{A}$ contains all the good candidate solutions, so far. Each member in $\mathcal{A}$ is a single classifier along with its pseudo target labels. These pseudo labels are likely to be more reliable than random or even than labels generated by a standalone classifier. They can be used to generate a good starting point. Specifically, for each unlabeled target instance, the instance's label is selected based on roulette wheel selection on a list of pseudo labels obtained by using all the members in $\mathcal{A}$. After obtaining pseudo labels for all target instances, Eq. (8) can be used to generate a new $\boldsymbol{\beta}$ to be the new candidate solution. It is expected that using good pseudo labels induces a good staring point for the new solution. Note that the new $\boldsymbol{\beta}$ is the optimal solution for minimizing F given the pseudo labels. Nevertheless, when applying $\boldsymbol{\beta}$ to classify the target instances, the obtained labels can be different from the pseudo labels.

Algorithm 1 : Generating a new candidate solution

Input: current solution sol_c, archive set $\mathcal{A}$, $MinSize$, $RandRate$
Output: new solution sol_n to replace sol_c

```
generate a new solution sol_n using Eq. (8)
calculate the fitness value of sol_n using Eq. (7)
if Fitness(sol_n) ≥ Fitness(sol_c) then
    add sol_c to the archive set A
    if |A| < MinSize ∨ random() < RandRate then
        randomly generate a new solution sol_n
    else
        generate a pseudo label for target instances using A
        based on the generated pseudo target labels, obtain sol_n
        using Eq. (8)
    end if
end if
replace sol_c by sol_n
```

However, it is not good to always generate the new candidate solution based on $\mathcal{A}$. There are two possible risks here. Firstly, this mechanism tends to make the newly generated solutions similar, which reduces the diversity of the ensemble classifier at the end. The less diverse ensemble may result in a poor classification performance. Secondly, when the size of $\mathcal{A}$ is too small, it is likely that the newly generated solution is very similar to one of the solutions in $\mathcal{A}$, which also reduces the diversity of the ensemble classifier. In order to avoid the above two risks, it is necessary to randomly generate the new candidate solution when the archive set is small, and with some probabilities even when the archive set is larger.

The pseudo code to generate new candidate solutions is shown in Algorithm 1. $MinSize$ is used as a lower bound that indicates whether $\mathcal{A}$ is large enough to generate new candidate solutions. $RandRate$ is the probability that the new solution is randomly generated, $random()$ is a function returning a value in the range [0,1].

3.3 Overall Algorithm

The overall algorithm P-MEDA is presented in Algorithm 2. Similar to MEDA, P-MEDA firstly transforms the original space into a

Algorithm 2 : Pseudo code for P-MEDA

Input: Data matrix $\boldsymbol{X} = [\boldsymbol{X}_s, \boldsymbol{X}_t]$, source domain labels y_s, population size P, maximum number of iterations I
Output: An ensemble classifier f

transform data to get manifold feature $\boldsymbol{Z} = \sqrt{\boldsymbol{G}}\boldsymbol{X}$
initialize N candidate solutions
initialize the archive set $\mathcal{A} = \emptyset$
while the maximum number of iterations is not reached **do**
  for each candidate solution sol_c **do**
    replace sol_c by the new solution generated by Algo. 1
  end for
end while
output $\mathcal{A}$ as an ensemble of classifiers to classify the target instances using a voting mechanism

Grassmann manifold. As a population-based algorithm, initialization is an essential process. A poor initialization usually results in a bad performance. In the initialization process, P-MEDA trains a set of N different standard classification algorithms on the source instances. The N classifiers are used to classify the target data, which results in N sets of pseudo labels for the target data. Each set of pseudo labels is used to initialize one candidate solution using Eq. (8). It is expected that using N standard classification algorithms provides a better initialization than a random initialization.

After initialization, the evolutionary process starts by evolving the candidate solutions using Algorithm 1. During the evolutionary process, all the promising solutions are added to the archive set $\mathcal{A}$ which has two essential functions. Firstly, $\mathcal{A}$ is used to re-initialize candidate solutions once a candidate solution cannot be improved by Eq. (8). This usage of $\mathcal{A}$ is to ensure that the new position is in a promising region. Secondly, $\mathcal{A}$ is output as an ensemble classifier that contains all the promising classifiers (candidate solutions) obtained during the evolutionary process. The main differences between P-MEDA and MEDA are highlighted as follows:

- P-MEDA has N candidate solutions to simultaneously explore the search space, while MEDA has only one candidate solution. The population-based searching mechanism is expected to make P-MEDA has a better global search ability.
- P-MEDA outputs a set of promising classifiers as an ensemble classifier, while MEDA outputs only a single classifier. The ensemble classifier is expected to make P-MEDA achieve higher classification performance than MEDA.
- P-MEDA considers the fitness value of the newly generated solution, while MEDA blindly replaces the current solution by the newly generated solution. The consideration allows P-MEDA to detect whether it already reaches a local optimal solution around the current candidate solution. Thereby, P-MEDA can jump to other promising regions without wasting resources to keep exploring the current region.

4 EXPERIMENTAL DESIGNS

4.1 Benchmark Datasets

P-MEDA is examined on three well-known real-world problems, Gas Sensor [30], Handwritten Digits [28] and Object Recognition [15, 16]. Details of the three problems are given in Table 1, where #C denotes the number of class labels, and #F stands for the number of original features, X_s is the source instances, X_t is the target instances. Note that in this work, there are no labeled instances in the target domain. The numbers of source and target instances are $|X_s|$ and $|X_t|$, respectively. Each problem has many domain adaptation cases that can have different source and target data.

Table 1: Domain adaptation problems.

Problem	Cases	#C	#F	$\lvert X_s\rvert$	$\lvert X_t\rvert$
Gas Sensor	1-2	6	129	178	1244
	1-3	6	129	178	1586
	1-4	6	129	178	161
	1-5	6	129	178	197
	1-6	6	129	178	2300
	1-7	6	129	178	3613
	1-8	6	129	178	294
	1-9	6	129	178	470
	1-10	6	129	178	3600
Object Recognition	A-C	10	800	958	1123
	A-D	10	801	958	157
	A-W	10	801	958	295
	C-A	10	801	1123	958
	C-D	10	801	1123	157
	C-W	10	801	1123	295
	D-A	10	801	157	958
	D-C	10	801	157	1123
	D-W	10	801	157	295
	W-A	10	801	295	958
	W-C	10	801	295	1123
	W-D	10	801	295	157
Handwritten Digits	M-U	10	257	2000	1800
	U-M	10	257	1800	2000

The gas sensor array drifts data is collected by Vergara et al. [30] using 16 gas sensors over 36 months. There are six different gas labels. The data is divided into ten batches according to the collection time. Nine domain adaptation cases are formed by using the first batch as a source data, and each of the other nine batches as a target data. The nine cases are named 1-2, 1-3,..., 1-9 as shown in Table 1.

The second real-world problem contains four different image domains: Amazon (A), Webcam (W), DSLR (D) from Office-31 [15], and Caltech-256 (C) [16]. The task is to recognize ten different objects corresponding to ten class labels. From the four domains, 12 domain adaptation cases can be formed by selecting one domain as the source domain and one of the other three domains as the target domain. Each domain case is noted in the form "A-B" where "A" is the source domain and "B" is the target domain.

The last one is a handwritten digit recognition problem that contains two datasets: USPS and MNIST [28]. The former is collected by scanning envelops from US Postal Service. The latter is taken from mixed American Census Bureau employees and American high school students. The task is to classify an instance as one of 10 digits (from 0 to 9). Since the two datasets are collected from different sources, it is expected that they have different data distributions. From the two datasets, two domain adaptation cases can be

formed by using one of them as the source data and the other one as the target data. In the rest of the paper, we denote MNIST-USPS and USPS-MNIST as M-U and U-M, respectively.

In total, there are 23 domain adaptation cases with different numbers of features, classes, source instances, and target instances. The 23 cases are expected to be a good representative for real-world domain adaptation problems.

4.2 Benchmark Techniques

In this work, we firstly compare P-MEDA with three widely used standard classification algorithm KNN (K=1), Random Forest (RF), and SVM. The number of trees in RF is set to 200. SVM uses radial basis function kernel (RBF). All other settings are set to default as in the scikit-learn package. The three classification algorithms are trained on the source data one time. The classifiers are then tested on the target data to obtain their target classification accuracies.

We also compare P-MEDA with four well-known and state-of-the-art domain adaptation algorithms:

- Transfer Component Analysis (TCA) [23] that reduces the marginal distribution difference,
- Joint Distribution Alignment (JDA) [20] that reduces both marginal and conditional distribution difference,
- Geodesic Flow Kernel (GFK) [14] that performs manifold feature learning, and
- Manifold Embedded Distribution Alignment (MEDA) [31] that induces an adaptive classifier.

4.3 Parameter Settings

The optimal parameters of all the benchmark algorithms are set according to their original papers. To have a fair comparison, the weights in Eq. (7) are set according to the original paper of MEDA: $\mu = 0.1$, $\lambda = 10$, and $\rho = 1$. RBF kernel is used with its bandwidth set to be the variance of inputs.

The population size N of P-MEDA is set to 10, the maximum number of iterations I is set to 10. THe archive size is not limited. Since the population is initialized by classification algorithms, the selected classification algorithms should be diverse to ensure the population diversity. Due to the limited number of classification algorithms, we currently use a small population size. The ten selected classification algorithms are KNN (K = 1, 3, 5), SVM ("linear" or "RBF" kernels), Gaussian Process Classifier, Naive Bayes, Decision Tree, Random Forest, and AdaBoost. *RandRate* and *MinSize* are set to 0.5 and 10 (equal to N), respectively. Further analysis of I, *RandRate* and *MinSize* will be discussed later.

5 EXPERIMENTAL RESULTS

5.1 Comparison with Standard Classification Algorithms

P-MEDA is compared with three widely used classifications algorithms: 1NN, RF, and SVM. For each domain adaptation case, P-MEDA is run 30 independent times. The classification algorithms are trained using the source data and then tested on the target data. The target accuracies of the four algorithms are shown in Table 2, where the best (highest) accuracy is marked in bold. The Wilcoxon test with a significant level of 0.05 is used to compare P-MEDA with the other three algorithms. ↑, ∘, or ↓ show that P-MEDA is significantly better, similar, or worse than the other algorithms.

Table 2: Comparison with standard classifiers.

Dataset	1NN	RF	SVM	P-MEDA
1-2	68.33 (↓)	**71.62** (↓)	13.18 (↑)	65.91
1-3	71.06 (↑)	60.53 (↑)	23.01 (↑)	**82.50**
1-4	**63.35** (↓)	50.31 (↑)	39.75 (↑)	60.60
1-5	69.04 (↑)	58.38 (↑)	14.21 (↑)	**75.80**
1-6	89.22 (↑)	77.61 (↑)	22.35 (↑)	**90.96**
1-7	53.36 (↑)	44.12 (↑)	17.96 (↑)	**71.30**
1-8	27.89 (↑)	23.81 (↑)	10.20 (↑)	**39.33**
1-9	49.36 (↑)	38.72 (↑)	12.98 (↑)	**66.34**
1-10	48.92 (↑)	32.06 (↑)	16.67 (↑)	**59.37**
A-C	26.00 (↑)	27.69 (↑)	7.57 (↑)	**45.38**
A-D	25.48 (↑)	22.93 (↑)	6.37 (↑)	**44.44**
A-W	29.83 (↑)	27.80 (↑)	9.15 (↑)	**44.60**
C-A	23.70 (↑)	30.06 (↑)	9.60 (↑)	**57.01**
C-D	25.48 (↑)	28.03 (↑)	7.64 (↑)	**57.96**
C-W	25.76 (↑)	33.22 (↑)	9.83 (↑)	**54.17**
D-A	28.50 (↑)	22.13 (↑)	10.44 (↑)	**43.89**
D-C	26.27 (↑)	23.78 (↑)	11.40 (↑)	**34.11**
D-W	63.39 (↑)	36.95 (↑)	10.17 (↑)	**88.01**
W-A	22.96 (↑)	25.05 (↑)	10.33 (↑)	**42.97**
W-C	19.86 (↑)	22.53 (↑)	11.84 (↑)	**31.74**
W-D	59.24 (↑)	45.22 (↑)	14.01 (↑)	**89.60**
M-U	64.44 (↑)	42.78 (↑)	9.17 (↑)	**80.21**
U-M	35.85 (↑)	13.55 (↑)	9.80 (↑)	**65.62**

As can be seen in Table 2, P-MEDA is significantly better than the three classification algorithms on most datasets. Among the three classification algorithms, 1NN achieves the best performance, which is followed by RF. Meanwhile, SVM seldom achieves more than 20% accuracy which is also the worst classification performance. Although SVM is usually claimed to be better than 1NN in many standard learning problems, it is not the case in domain adaptation. The possible reason is that 1NN is simpler than SVM and RF, which makes it more generalizable. Thereby, generalization is crucial in domain adaptation, especially when target labels are not available.

Compared with 1NN, P-MEDA achieves significantly better accuracies on 21 out of the 23 cases. The largest difference is on W-D where P-MEDA is 30% better than 1NN. On high-dimensional problems such as handwritten digits or object recognition, the accuracy of P-MEDA is usually 1.5 times higher than that of 1NN. The experimental results show that P-MEDA can construct a more accurate ensemble classifier than the standard classification algorithms.

5.2 Comparison with State-of-the-art Domain Adaptation Methods

In this subsection, P-MEDA is compared with four well-known and state-of-the-art domain adaptation methods: TCA, JDA, GFK, and MEDA. The comparison is shown in Table 3.

Firstly, P-MEDA significantly outperforms all other benchmark domain adaptations on most cases (16/23 cases). In particular, both TCA and JDA are worse than P-MEDA on 21 out of the 23 cases. On some cases such as 1-2 or M-U, P-MEDA achieves at least 25% more accurate than both TCA and JDA. A possible reason is that the two

Table 3: Comparison with state-of-the-art domain adaptation methods

Dataset	TCA	JDA	GFK	MEDA	P-MEDA
1-2	60.05 (↑)	**75.72** (↓)	70.10 (↓)	62.14 (↑)	65.91
1-3	62.23 (↑)	43.88 (↑)	72.70 (↑)	**83.48** (↓)	82.50
1-4	34.16 (↑)	44.72 (↑)	**62.73** (↓)	55.28 (↑)	60.60
1-5	50.76 (↑)	52.79 (↑)	75.13 (↑)	75.63 (○)	**75.80**
1-6	84.04 (↑)	50.83 (↑)	88.52 (↑)	90.00 (↑)	**90.96**
1-7	55.96 (↑)	34.13 (↑)	54.86 (↑)	68.23 (↑)	**71.30**
1-8	**44.90** (↓)	35.37 (↑)	27.21 (↑)	39.12 (○)	39.33
1-9	39.15 (↑)	25.96 (↑)	53.83 (↑)	56.60 (↑)	**66.34**
1-10	51.11 (↑)	31.83 (↑)	50.83 (↑)	53.58 (↑)	**59.37**
A-C	40.25 (↑)	35.17 (↑)	40.25 (↑)	**47.82** (↓)	45.38
A-D	39.49 (↑)	32.48 (↑)	36.31 (↑)	43.95 (↑)	**44.44**
A-W	41.69 (↑)	35.93 (↑)	40.00 (↑)	**46.44** (↓)	44.60
C-A	44.47 (↑)	39.25 (↑)	41.02 (↑)	56.78 (↑)	**57.01**
C-D	46.50 (↑)	45.86 (↑)	41.40 (↑)	50.96 (↑)	**57.96**
C-W	43.05 (↑)	33.56 (↑)	40.68 (↑)	52.88 (↑)	**54.17**
D-A	32.05 (↑)	26.10 (↑)	32.05 (↑)	42.69 (↑)	**43.89**
D-C	30.72 (↑)	29.12 (↑)	30.10 (↑)	31.17 (↑)	**34.11**
D-W	87.46 (↑)	84.07 (↑)	84.41 (↑)	**91.19** (↓)	88.01
W-A	30.17 (↑)	33.09 (↑)	31.84 (↑)	42.59 (↑)	**42.97**
W-C	30.37 (↑)	29.03 (↑)	30.72 (↑)	30.01 (↑)	**31.74**
W-D	**91.72** (↓)	84.71 (↑)	87.90 (↑)	91.08 (↓)	89.60
M-U	56.44 (↑)	37.17 (↑)	64.33 (↑)	71.06 (↑)	**80.21**
U-M	37.75 (↑)	30.95 (↑)	44.50 (↑)	63.25 (↑)	**65.62**

benchmark methods aim to reduce the distribution difference in the original feature space where the domain divergence is difficult to reduce due to feature distortion. In contrast, GFK performs feature transformation based on manifold structure, but it does not explicitly reduce distribution differences. Thus, GFK is also significantly worse than P-MEDA on 21 out of the 23 cases.

Among the four benchmark methods, MEDA is the most similar one to P-MEDA. MEDA firstly uses GFK to transform the data, and then reduces distribution distances based on the transformed data. Thus, MEDA achieves the best classification performance among the four benchmark algorithms. However, in comparison with P-MEDA, MEDA is significantly worse than P-MEDA on 16 out of the 23 cases. There are two main reasons for the dominance of P-MEDA. Firstly, P-MEDA outputs an ensemble of classifiers which is expected to be more general than a single classifier generated by MEDA. Secondly, P-MEDA utilizes a population-based searching mechanism that allows P-MEDA to avoid being stuck at local optima.

To examine the searching ability of MEDA and P-MEDA, their fitness values are recorded during the optimization process. The fitness comparison between the two algorithms on six representative cases is shown in Fig. 1. The patterns on the other cases are similar and are not shown here due to the space limit. Note that since P-MEDA uses a set of classifiers to initialize its population, its starting fitness is usually better (smaller) than that of MEDA. As can be seen from the figure, MEDA usually converges after ten evaluations, while P-MEDA keeps evolving until 100 evaluations are executed. The premature convergence of MEDA is a consequence of its gradient-based search mechanism. During the evolutionary

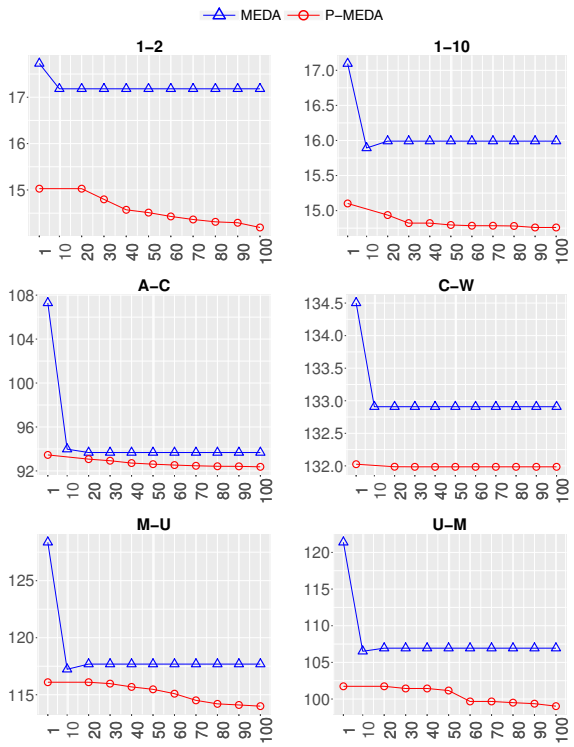

Figure 1: Fitness comparison between MEDA and P-MEDA. The vertical and horizontal axises represent the fitness values and the number of evaluations, respectively.

process, P-MEDA also ensures that the later generation is never worse than the previous one. In contrast, MEDA replaces the current candidate solution with a newly generated solution without considering their fitness values. As a result, on many cases such as 1-10, M-U, and U-M, the fitness value of the later generation is worse than that of the previous generation.

The experimental results show that the population-based mechanism assists P-MEDA to improve its global search ability and the generalization of its evolved classifier over MEDA.

5.3 Parameter Sensitivity

5.3.1 MinSize and RandRate. The two parameters *MinSize* and *RandRate* control whether the re-initialization generates the new candidate solutions randomly or based on the archive set $\mathcal{A}$. The sensitivity of the two parameters is examined through experiments.

Fig. 2 shows the comparison between three different values of *MinSize*: {2, 6, 10}. In the figure, the horizontal axis is the *MinSize* values, and the vertical axis is the obtained classification accuracy. For better visualization, each problem is represented by one subfigure that contains comparisons on at most four cases. As can be seen from the figure, on most cases *MinSize* = 10 has the best accuracy while *MinSize* = 2 has the worst one. The results show that it is better to have a large enough archive set before using it in the re-initialization process. However, the results also show that the accuracy differences between these different *MinSize* are actually quite small which means that P-MEDA is quite robust to different values of *MinSize*.

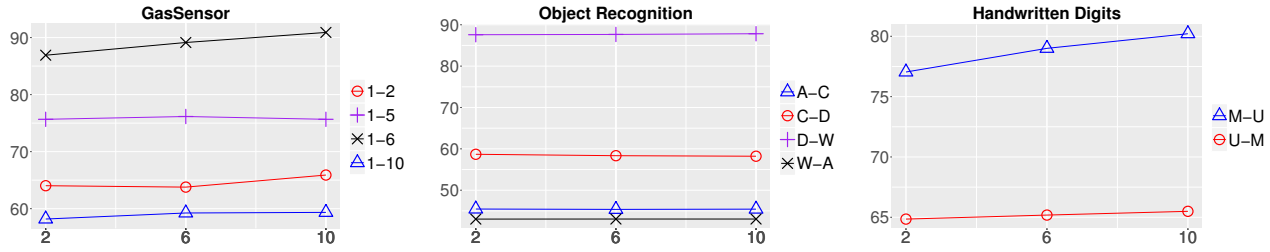

Figure 2: Classification accuracy of different *MinSize*.

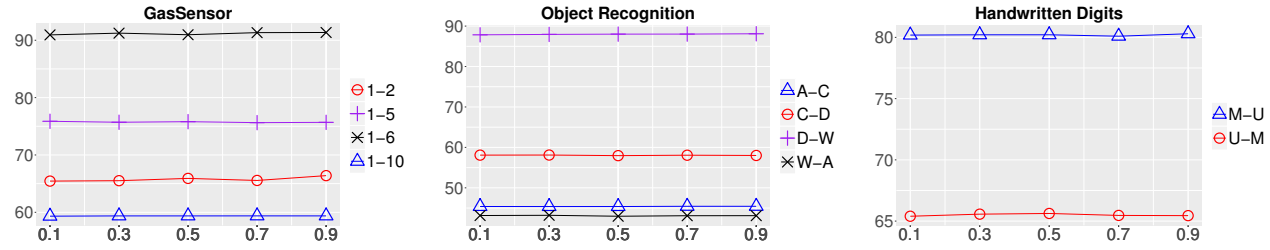

Figure 3: Classification accuracy of different *RandRate*.

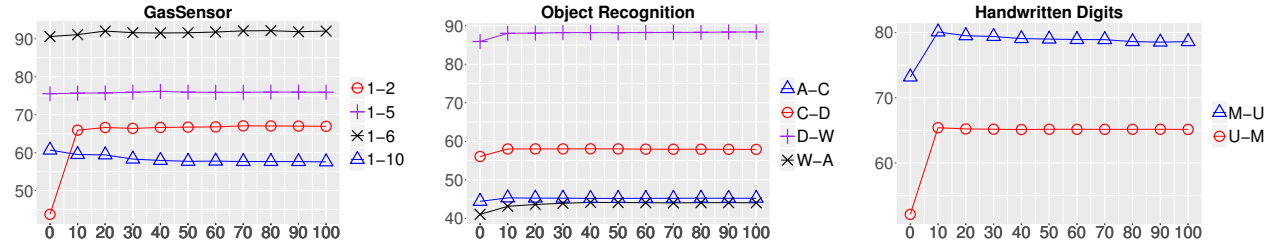

Figure 4: Classification accuracy of different maximum iterations.

Fig. 3 shows the comparison between five different values of *RandRate*: {0.1, 0.3, 0.5, 0.7, 0.9}. As can be seen from the figure, there is no significant difference between the classification performances obtained by the five values, which shows that P-MEDA is robust to different values of *RandRate*. Thus, *RandRate* can be selected without knowledge in real-world applications.

5.3.2 Maximum number of iterations. P-MEDA is examined with ten different maximum numbers of iterations ranging from 10 to 100. Fig. 4 presents the obtained classification performance of the ten numbers of iterations on the three real-world problems. It can be seen that the classification performance is quickly improved in the first ten iterations. In the later iterations, the classification performance does not change or change very little. It illustrates that for such a small population size of 10, the algorithm can converge within ten iterations. An interesting pattern can be seen in 1-10, M-U, and U-M, where increasing the number of iterations reduces the classification performance. Due to a large number of iterations, many archive members become very similar, which reduces the diversity of the final ensemble of classifiers.

6 CONCLUSIONS

In this work, a population-based algorithm is developed to achieve unsupervised domain adaptation. The proposed algorithm is based on MEDA — a state-of-the-art transfer classifier induction algorithm. The population-based mechanism is embedded to enhance the global search ability. Once a population member gets stuck at a local optimum, the member is restarted at a new promising position which is generated based on the good locations explored by the population so far. More importantly, an ensemble of classifiers is evolved instead of a single classifier to improve the generalization. The experimental results show that the proposed algorithm, called P-MEDA, can achieve a significantly better classification performance than four state-of-the-art domain adaptation algorithms. A further analysis illustrates the robustness of the proposed algorithm regarding its parameters.

Although the proposed algorithm achieves better performance than the state-of-the-art competitors, its performance can be further improved. For example, we will investigate how to well initialize the population without classifiers, which is promising but not an easy task since it is problem-dependent. Another direction is to enhance the communication between population members. Currently, the population members indirectly exchange their information via the archive set, and more effective ways will be investigated.

REFERENCES

[1] Thomas Bäck, Ulrich Hammel, and Hans-Paul Schwefel. 1997. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation* 1, 1 (1997), 3–17.

[2] Mahsa Baktashmotlagh, Mehrtash Harandi, and Mathieu Salzmann. 2016. Distribution-matching embedding for visual domain adaptation. *The Journal of Machine Learning Research* 17, 1 (2016), 3760–3789.

[3] Mahsa Baktashmotlagh, Mehrtash T Harandi, Brian C Lovell, and Mathieu Salzmann. 2013. Unsupervised domain adaptation by domain invariant projection. In *Proceedings of the IEEE International Conference on Computer Vision*. 769–776.

[4] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. 2006. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *The Journal of Machine Learning Research* 7, Nov (2006), 2399–2434.

[5] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. 2007. Analysis of representations for domain adaptation. In *Advances in Neural Information Processing Systems*. 137–144.

[6] Karsten M Borgwardt, Arthur Gretton, Malte J Rasch, Hans-Peter Kriegel, Bernhard Schölkopf, and Alex J Smola. 2006. Integrating structured biological data by kernel maximum mean discrepancy. *Bioinformatics* 22, 14 (2006), e49–e57.

[7] Yue Cao, Mingsheng Long, and Jianmin Wang. 2018. Unsupervised Domain Adaptation with Distribution Matching Machines. In *AAAI Conference on Artificial Intelligence*.

[8] Shi Cheng, Bin Liu, Yuhui Shi, Yaochu Jin, and Bin Li. 2016. Evolutionary computation and big data: key challenges and future directions. In *International Conference on Data Mining and Big Data*. Springer, 3–14.

[9] Lawrence Davis. 1991. Handbook of genetic algorithms. (1991).

[10] Thomas G Dietterich. 2000. Ensemble methods in machine learning. In *International Workshop on Multiple Classifier Systems*. Springer, 1–15.

[11] Basura Fernando, Amaury Habrard, Marc Sebban, and Tinne Tuytelaars. 2013. Unsupervised visual domain adaptation using subspace alignment. In *Proceedings of the IEEE International Conference on Computer Vision*. 2960–2967.

[12] Wenlong Fu, Bing Xue, Mengjie Zhang, and Xiaoying Gao. 2017. Transductive transfer learning in genetic programming for document classification. In *Asia-Pacific Conference on Simulated Evolution and Learning*. Springer, 556–568.

[13] Boqing Gong, Kristen Grauman, and Fei Sha. 2013. Connecting the dots with landmarks: Discriminatively learning domain-invariant features for unsupervised domain adaptation. In *International Conference on Machine Learning*. 222–230.

[14] Boqing Gong, Yuan Shi, Fei Sha, and Kristen Grauman. 2012. Geodesic flow kernel for unsupervised domain adaptation. In *Proceedings of the IEEE International Conference on Computer Vision*. IEEE, 2066–2073.

[15] Raghuraman Gopalan, Ruonan Li, and Rama Chellappa. 2011. Domain adaptation for object recognition: An unsupervised approach. In *Proceedings of the IEEE International Conference on Computer Vision*. IEEE, 999–1006.

[16] Gregory Griffin, Alex Holub, and Pietro Perona. 2007. Caltech-256 object category dataset. (2007).

[17] Muhammad Iqbal, Harith Al-Sahaf, Bing Xue, and Mengjie Zhang. 2019. Genetic programming with transfer learning for texture image classification. *Soft Computing* (2019), 1–13.

[18] James Kennedy. 2011. Particle swarm optimization. In *Encyclopedia of Machine Learning*. Springer, 760–766.

[19] Mingsheng Long, Jianmin Wang, Guiguang Ding, Sinno Jialin Pan, and S Yu Philip. 2014. Adaptation regularization: A general framework for transfer learning. *IEEE Transactions on Knowledge and Data Engineering* 26, 5 (2014), 1076–1089.

[20] Mingsheng Long, Jianmin Wang, Guiguang Ding, Jiaguang Sun, and Philip S Yu. 2013. Transfer feature learning with joint distribution adaptation. In *Proceedings of the IEEE International Conference on Computer Vision*. 2200–2207.

[21] Yishay Mansour, Mehryar Mohri, and Afshin Rostamizadeh. 2009. Domain Adaptation: Learning Bounds and Algorithms. *CoRR* abs/0902.3430 (2009).

[22] Bach Hoai Nguyen, Bing Xue, and Peter Andreae. 2018. A particle swarm optimization based feature selection approach to transfer learning in classification. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 37–44.

[23] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. 2011. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks* 22, 2 (2011), 199–210.

[24] Sinno Jialin Pan, Qiang Yang, et al. 2010. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (2010), 1345–1359.

[25] Si Si, Dacheng Tao, and Bo Geng. 2010. Bregman divergence-based regularization for transfer subspace learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 7 (2010), 929.

[26] Baochen Sun and Kate Saenko. 2015. Subspace Distribution Alignment for Unsupervised Domain Adaptation.. In *BMVC*. 24–1.

[27] Qian Sun, Rita Chattopadhyay, Sethuraman Panchanathan, and Jieping Ye. 2011. A two-stage weighting framework for multi-source domain adaptation. In *Advances in Neural Information Processing Systems*. 505–513.

[28] Jafar Tahmoresnezhad and Sattar Hashemi. 2016. An Efficient yet Effective Random Partitioning and Feature Weighting Approach for Transfer Learning. *International Journal of Pattern Recognition and Artificial Intelligence* 30, 02 (2016), 1651003.

[29] Vladimir Vapnik. 1998. *Statistical learning theory. 1998*. Vol. 3. Wiley, New York.

[30] Alexander Vergara, Shankar Vembu, Tuba Ayhan, Margaret A Ryan, Margie L Homer, and Ramón Huerta. 2012. Chemical gas sensor drift compensation using classifier ensembles. *Sensors and Actuators B: Chemical* 166 (2012), 320–329.

[31] Jindong Wang, Wenjie Feng, Yiqiang Chen, Han Yu, Meiyu Huang, and Philip S Yu. 2018. Visual Domain Adaptation with Manifold Embedded Distribution Alignment. In *ACM Multimedia Conference on Multimedia Conference*. ACM, 402–410.

[32] Yonghui Xu, Sinno Jialin Pan, Hui Xiong, Qingyao Wu, Ronghua Luo, Huaqing Min, and Hengjie Song. 2017. A Unified Framework for Metric Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* 29, 6 (2017), 1158–1171.

[33] Qiang Yang, Sinno Jialin Pan, and Vincent Wenchen Zheng. 2008. Estimating Location Using Wi-Fi. *IEEE Intelligent Systems* 23, 1 (2008), 8–13.