Heterogeneous Domain Adaptation for Image and Text Classification

Shih-Yen Tao
Language Technologies Institute
shihyent@andrew.cmu.edu

1 Introduction

Domain adaptation (DA) aims to solve cross-domain classification problem. In real-world learning scenario, it is often that the training (source domain) and test data (target domain) come from different distributions and different domains. For example, if the training data are low-resolution images crawled from the web while the final goal is to classify high-quality photos taken by smartphones, this domain discrepancy will harm the performance severely. How to use the source domain data and adapt the learned model to the target domains is the core problem of DA.

There are lots of DA literature. For instance, [5] aims to eliminate the domain discrepancy by learning domain-invariant feature, while [4] choose to select representative training data for performing domain adaptation. More recently, the DA research attention has shifted to *Heterogeneous* domain adaptation (HDA), in which training and testing data not only have different distribution but also are represented by different features. For instance, in cross-lingual text categorization, we must have different feature representation for the different language. Several HDA algorithms have been proposed: while [1] and [2] generalized [5] and [4] respectively for the heterogeneous setting, [3] leveraged tree structured deep neural network to perform the cross-domain HDA.

In this project, we will propose a learning algorithm for HDA. To be more specific, we aim to learn a transformation from source to target domain and an SVM model on the target domain jointly. We will follow the experiment setting in [1] and [2] to evaluate our HDA model for cross-domain object recognition and cross-lingual text categorization tasks.

2 Problem Definition: Heterogeneous Domain Adaptation (HDA)

In this work, we focus on semi-supervised HDA. That is to say, we have a sufficient number of labeled source domain data, while the target domain has few labeled data and sufficient unlabeled data. We denote the source domain data as $\mathcal{D}_S = \{\mathbf{X}_S, Y_S\} = \{\mathbf{x}_S^i, y_S^i\}_{i=1}^{n_S}$, where $\mathbf{x}_S^i \in \mathbb{R}^{d_S}$ represents the d_S-dimensional source-domain instance, and y_S^i is its label from the label set $\mathcal{L} = \{1, 2, .., C\}$. We also have $\mathcal{D}_T = \{\mathbf{X}_T, Y_T\} = \{\mathbf{x}_T^i, y_T^i\}_{i=1}^{n_T}$ as the target-domain data, where y_T^i come from the same label set $\mathcal{L}$. It is worth noting that, since we are dealing with the HDA, we have $\mathbb{R}^{d_S} \neq \mathbb{R}^{d_T}$ and $\mathcal{P}_S(\mathbf{X}_S, Y_S) \neq \mathcal{P}_T(\mathbf{X}_T, Y_T)$.

Moreover, we further divide the target domain data into labeled and unlabeled subset: $\mathcal{D}_L = \{\mathbf{X}_L, Y_L\} = \{\mathbf{x}_L^i, y_L^i\}_{i=1}^{n_L}$ and $\mathcal{D}_U = \{\mathbf{X}_U, Y_U\} = \{\mathbf{x}_U^i, y_U^i\}_{i=1}^{n_U}$, respectively. The labels $\{y_L^i\}_{i=1}^{n_L}$ are available, while $\{y_U^i\}_{i=1}^{n_U}$ are to be predicted. In a nutshell, our goal is to predict the labels of unlabeled target domain data $\{y_U^i\}_{i=1}^{n_U}$ by transfering knowledge across heterogeneous domains.

3 Our Method

Our method aims to learn a transformation matrix $\mathbf{A} \in \mathbb{R}^{d_S \times d_T}$, which projects the source domain data to the target domain, and a SVM model $\mathbf{w} \in \mathbb{R}^{d_T}$ in the target domain *jointly*. The reasons

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

doing so are as follow: As shown in [1] or [2], projecting cross-domain data on a common feature space can help the classification task. This is because, besides the original labeled target domain data, we can also leverage the projected source domain data to train a more robust classifier. The difference between our method and [1] and [2] is that after they learned the transformation, the existing SVM solver is used for classification, while our goal is to learn the SVM weight $\mathbf{w}$ at the same time.

How to leverage the source domain data properly? The most important thing is that we need to ensure that the data distribution of the projected source domain data should be "similar" enough to the target domain data distribution This is to say, $\mathcal{P}_S(\mathbf{A}^\top \mathbf{X}_S, Y_S) \approx \mathcal{P}_T(\mathbf{X}_T, Y_T)$. Since the joint distribution cannot be calculated directly, we turn out to match the conditional (class-wise) distribution $\mathcal{P}_S(\mathbf{A}^\top \mathbf{X}_S|Y_S) \approx \mathcal{P}_T(\mathbf{X}_T|Y_T)$.

Combined with the SVM loss, we can write down our objective function as

$$\begin{aligned}\min_{\mathbf{A},\mathbf{w}} \sum_{c=1}^{C} E_c(\mathbf{A}) + \sum_{i=1}^{n_S} \max(1, 1 - y_S^i \mathbf{w}^\top \mathbf{A}^\top \mathbf{x}_S^i)^2 + \sum_{i=1}^{n_L} \max(1, 1 - y_L^i \mathbf{w}^\top \mathbf{x}_L^i)^2 \\ + \frac{1}{2} \|\mathbf{A}\|^2 + \frac{1}{2} \|\mathbf{w}\|_2^2,\end{aligned} \tag{1}$$

where E_c denotes the cross-domain conditional distributions (of class c) discrepancy. As suggested in [5] and [1], we resort to *Maximum Mean Discrepancy* as the criterion of measuring distribution mismatch, where the discrepancy is approximately measured by the empirical mean difference. Accordingly, we can formulate E_c as

$$E_c(\mathbf{A}) = \| \frac{1}{n_S^c} \sum_{i=1}^{n_S^c} \mathbf{A}^\top \mathbf{x}_S^{i,c} - \frac{1}{n_T^c} \sum_{j=1}^{n_T^c} \mathbf{x}_T^{j,c} \|^2, \tag{2}$$

where $x^{i,c}$ denotes the ith instance for class c. In a nutshell, the distribution mismatch is measured by the cross-domain class-wise mean difference.

Let's return to Equation 1, we can see that we not only want to match cross-domain distributions but also aim to learn the SVM weight $\mathbf{w}$ by minimizing the hinge loss of projected source domain data and labeled target domain data. As the result, after we complete the learning process, we can use the SVM model to predict the labels of unlabeled target domain data $\{y_U^i\}_{i=1}^{n_U}$.

4 Optimization and Learning

Even though our whole objective function isn't convex for $\mathbf{w}, \mathbf{A}$ overall, it is convex for each of them separately. That is to say, the learning process can be done by updating one variable while the other is fixed. The whole process will, therefore, converge to a local minimum.

4.1 Learning Transformation A

When we fix $\mathbf{w}$, we will get a quadratic programming for $\mathbf{A}$. Since the objective function is differentiable for A, we can use traditional decent methods to solve it. In this project, we select *Gradient Descent(GD)* and *Accelerated Gradient Descent(Nesterov)*. The gradient can be computed as follow:

$$\nabla_{\mathbf{A}} f = (\mathbf{I} + 2\mathbf{S_m}\mathbf{S_m}^\top)\mathbf{A} - 2\mathbf{T_m}\mathbf{S_m}^\top - 2\sum_{i \in \mathbf{V}} (y_S^i - \mathbf{w}^\top \mathbf{A}^\top \mathbf{x}_S^i)\mathbf{x}_S^i \mathbf{w}^\top, \tag{3}$$

where $\mathbf{V} \equiv \{i | 1 - y_S^i \mathbf{w}^\top \mathbf{A}^\top \mathbf{x}_S^i > 0\}$, and $\mathbf{S_m} \in \mathbb{R}^{d_S \times C}, \mathbf{T_m} \in \mathbb{R}^{d_T \times C}$ are matrix consists of class mean vectors for source and target domain respectively. For the AGD implementation, we leverage backtracking linesearch to choose a appropriate step size. We set the Armijo condition's parameter $\alpha = 0.2$, and Nesterov's parameter $\beta = \frac{k+1}{k-2}$, where k is the epoch number. For standard DG, we use fixed step size.

4.2 Learning SVM $\mathbf{w}$

We tried three different methods for solving the SVM model $\mathbf{w}$: standard gradient descent, coordinate descent and Newton method with conjugate gradient descent. Empirically, we find that Newton method has the best convergence rate and performance. As the result, we adopt it for our final algorithm.

For gradient descent and Newton method, we need the gradient of $\mathbf{w}$ as follows:

$$\nabla_{\mathbf{w}} f = w - 2\sum_{i \in \mathbf{V}} \mathbf{x}_i (y_i - \mathbf{w}^\top \mathbf{x}_i), \tag{4}$$

where $\mathbf{V} \equiv \{i | 1 - y_i \mathbf{w}^\top \mathbf{x}_i > 0\}$.

The objective function is not twice differentiable to $\mathbf{w}$. As the result, we use the generalized Hessian for $\mathbf{w}$ instead:

$$\nabla^2_{\mathbf{w}} f = \mathbf{I} + 2\sum_{i \in \mathbf{V}} \mathbf{x}_i \mathbf{x}_i^\top. \tag{5}$$

Note that, for Newton method, we follow [8] to leverage CGD to obtain the descent direction. The reason is, when the number of features is huge, it is infeasible to store the whole Hessian matrix, not to mention do the inverse operation. In CGD, only the inner product of a given vector $\mathbf{s}$ and Hessian is needed: $\mathbf{s} + 2\sum_{i \in \mathbf{V}} \mathbf{x}_i (\mathbf{x}_i^\top \mathbf{s})$. It is worth emphasize that for the second term, we first compute the inner product between $\mathbf{x}$ and $\mathbf{s}$, which will save time and memory dramatically.

For the Newton method, we also use backtracking line search to choose step size. We set the shrinking parameter $\beta = 0.5$. As for standard GD, we fix the step size.

The coordinate descent method is very different from the above two [7]; instead of updating the whole vector $\mathbf{w}$, the algorithm updates each coordinate independently. To be more specific, for each dimension j, we aim to solve:

$$\min_z \frac{1}{2}\|\mathbf{w} + z\mathbf{e}_j\|^2 + \sum_i max(0, 1 - y_i(\mathbf{w} + z\mathbf{e}_j)^\top \mathbf{x}_i)^2. \tag{6}$$

As suggested in [7], we adopt Newton method to solve it. The gradient of the function respect to z is $\frac{\partial f}{\partial z} = \mathbf{w}_j + z - 2\sum_{i \in \mathbf{V}} y_i \mathbf{x}_{ij}(1 - y_i(\mathbf{w} + z\mathbf{e}_j)^\top \mathbf{x}_i)$, where $\mathbf{V} \equiv \{i | 1 - y_i(\mathbf{w} + z\mathbf{e}_j)^\top \mathbf{x}_i > 0\}$. Again, we can get the generalized twice derivative $\frac{\partial^2 f}{\partial z^2} = 1 + 2\sum_{i \in \mathbf{V}} x_{ij}^2$. For each dimension, we initialize $z = 0$ and do $z = z - \frac{\frac{\partial f}{\partial z}}{\frac{\partial^2 f}{\partial z^2}}$ while converge. It is worth noting that, for computing $(\mathbf{w} + z\mathbf{e}_j)^\top \mathbf{x}_i$, we first cach the value $\mathbf{w}^\top \mathbf{x}_i$ and then directly compute the value by adding $z\mathbf{x}_{ij}$. This trick could accelerate the coordinate descent algorithms greatly.

4.3 Learning Algorithms

As suggested in DA literature, leveraging *unlabeled* target domain data can be crucial for boosting the classification performance. We resort to the similar way in [1] and [2], that is, after we solve $\mathbf{w}$, we can use the model to predict the labels of those unlabeled target domain data. After that, we can use those predicted labels to re-estimate the target domain class-wise mean.

In addition, we need a good initial $\mathbf{A}$ to start the algorithm. As the result, we first let $\mathbf{w} = \mathbf{0}$, and we can have a closed form

$$\mathbf{A}_0 = (\mathbf{I} + 2\mathbf{S_m}\mathbf{S_m}^\top)^{-1} 2\mathbf{T_m}\mathbf{S_m}^\top. \tag{7}$$

Moreover, we also have a *relaxed* version of our model: in every iteration when we learn $\mathbf{A}$, the hinge loss term is ignored. That is to say, we discard the common term for $\mathbf{A}$ and $\mathbf{w}$ when learning $\mathbf{A}$. Surprisingly, this simple strategy works better than the original version. The details and possible explanation can be found in Sec 5. The overall algorithm is summarized in 1

Algorithm 1 Overall Algorithm

Require: Source-domain data $\mathcal{D}_S = \{\mathbf{x}_S^i, y_S^i\}_{i=1}^{n_S}$, labeled target-domain data $\mathcal{D}_L = \{\mathbf{x}_L^i, y_L^i\}_{i=1}^{n_L}$, and unlabeled target-domain data $\{\mathbf{x}_U^i\}_{i=1}^{n_U}$

1: Initialize $\mathbf{A}$ using equ 7
2: **while** not converge **do**
3: Use Newton method to learn $\mathbf{w}$
4: Update pseudo labels $\{\hat{y}_U^i\}_{i=1}^{n_U}$
5: Use Nestenov AGD to learn $\mathbf{A}$ (In the relaxed version, discarge the hinge loss term)
6: **end while**

Ensure: $\mathbf{A}$, $\mathbf{w}$, and $\{\hat{y}_U^i\}_{i=1}^{n_U}$

Table 1: Classification results for cross-feature object recognition.

S,T	$\mathbf{SVM}_t$	Ours	Ours(relax)
SURF to *DeCAF*$_6$			
A, A	86.4	90.0	91.8
W, W	86.4	90.8	92.1
C, C	76.7	80.3	85.6

5 Experiment-Classification

5.1 Datasets

We follow the dataset setting in [1],[2].To test our model for DA for image classification, we consider two datasets: Office + Caltech-256. As for the cross-lingual categorization task, we use Multilingual Reuters Collection. The Office dataset consists of three sub-datasets: Amazon(A) (images from the Internet), Webcam (W) (low-resolution images by webcams), and DSLR (D)(high-resolution images by digital cameras). Caltech-256 dataset itself has 256 different object classes. We select *ten* overlapping categories for these two datasets. As the result, overall we have four different domain for cross-domain image classification task. We use two different feature: SURF(800-dim) and Decaf(4096-dim) for feature representation. The source domain data contains 20 labeled instances per class, while there are only three labeled data per category in the target domain.

Multilingual Reuters Collection contains 11K articles from 6 categories in 5 languages (English, French, German, Italian and Spanish). We extract TF-IDF feature and performance PCA with 60% energy preserved. The feature dimensions of the five languages English, French, Italian, German, and Spanish are 1131, 1230, 1417, 1041, and 807, respectively. There are 100 and 10 labeled instances per category for source and target domain respectively.

5.2 Cross-Feature Object Recognition Experiment

Fro the object recognition task, we first test our method under the scenario when the source and target domain come from the same domain but have different feature representation. We let the source domain feature SURF while the target domain data is presented by Decaf. The result is shown in Table 1. Note that our baseline $\mathbf{SVM}_t$ is to train libsvm [6] using merely the labeled target domain data. We can observe that our method improve the baseline method a lot.

5.3 Cross-Domain Object Recognition Experiment

We test our algorithm on a more challenging setting: the source and target data come from the different domain and are represented by different feature. In this experiment, we use DSLR as the target domain. Again, we use SURF as source feature and Decaf as target feature. Please refer to Table 2 for the results.

5.4 Cross-Lingual Text Categorization Experiment

In this experiment, we use the Multilingual Reuters Collection dataset to conduct cross-lingual text classification task. We use Spanish as our target domain. The results can be found at Table 3. We only show the relaxed version here because the full model produces unsatisfactory results.

Table 2: Classification results for cross-domain object recognition.

S,T	$\mathbf{SVM}_t$	Ours	Ours(relax)
SURF to $DeCAF_6$			
A, D	91.3	92.4	92.5
W, D	91.3	91.8	92.0
C, D	91.3	91.9	92.3

Table 3: Classification results for cross-lingual text categorization.

S,T	$\mathbf{SVM}_t$	Ours(relax)
En, S	66.7	69.5
Fr, S	66.7	69.0
Ger, S	66.7	68.8
Ita, S	66.7	69.5

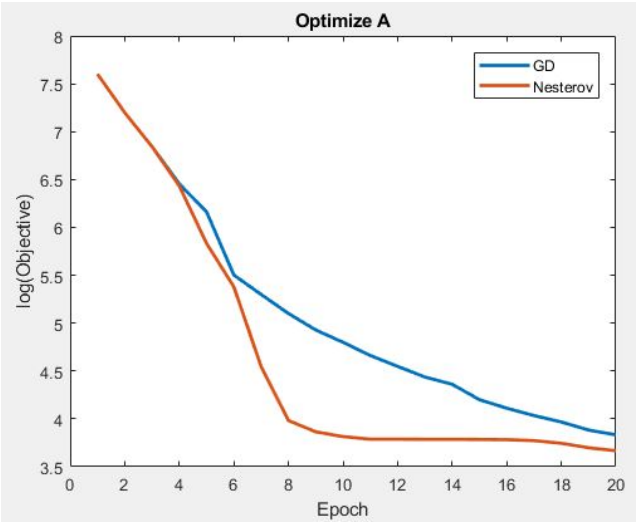

Figure 1: CG and ACG.

5.5 Evaluation

It is obvious that our relaxed model is much better than the full model. We think the reason is that if we consider the hinge loss while solving for $\mathbf{A}$, the algorithm cannot align the distribution across domain properly. As the result, after that, when we use the projected source domain data to train $\mathbf{w}$ will produce an unsatisfactory result, and the model will be locked to a bad local minimum.

Even though our full model's results are worse than the baseline model for the cross-lingual experiment, it is still better for the object classification task. Overall, our proposed method is a modest model.

6 Experiment-Optimization

We compare differnt optimization algorithms here.

6.1 Solving A

We compare GD and AGD in figure[1].

6.2 Solving W

We compare GD and coordinate descent in figure[2] and show the convergence curve for Newton method in figure[3].

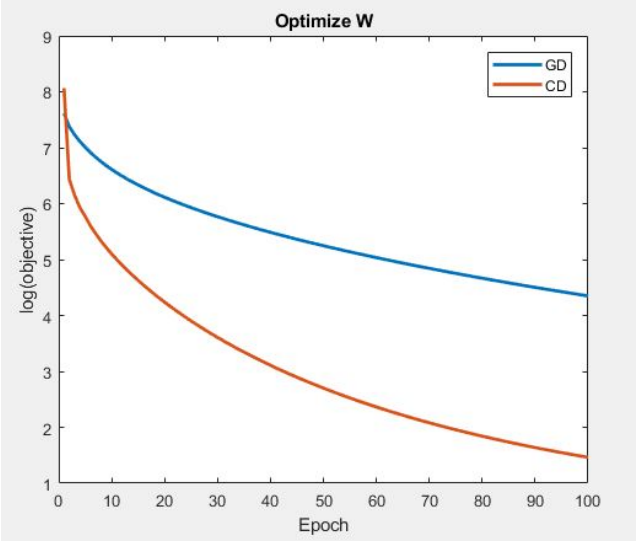

Figure 2: CG and CD.

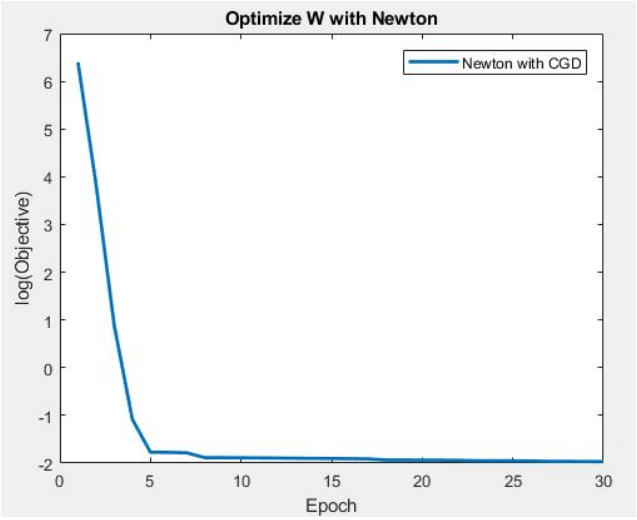

Figure 3: Newton Method

References

[1] Shih-Yen Tao, Yuan-Ting Hsieh, Yao Hung Hubert Tsai, Yi-Ren Yeh, Yu-Chiang Frank Wang

Recognizing Heterogeneous Cross-Domain Data via Generalized Joint Distribution Adaptation, in ICME 2016

[2] Yao-Hung Hubert Tsai, Yi-Ren Yeh and Yu-Chiang Frank Wang

Learning Cross-Domain Landmarks for Heterogeneous Domain Adaptation, in CVPR 2016

[3] Wei-Yu Chen, Tzu-Ming Harry Hsu, Yao-Hung Hubert Tsai, Yu-Chiang Frank Wang and Ming-Syan Chen

Transfer Neural Trees for Heterogeneous Domain Adaptation, in ECCV, 2016

[4] B. Gong et al., Connecting the dots with landmarks: Discriminatively learning domain-invariant features for unsupervised domain adaptation, in ICML 2013

[5] M. Long et al., Transfer feature learning with joint distribution adaptation, in ICCV 2013.

[6] Chang, Chih-Chung and Lin, Chih-Jen LIBSVM: A library for support vector machines, in ACM Transactions on Intelligent Systems and Technology 2011.

[7] K.-W. Chang, C.-J. Hsieh, and C.-J. Lin Coordinate Descent Method for Large-scale L2-loss Linear SVM, in JMLR 2008.

[8] R.-E. Fan et al., LIBLINEAR: A library for large linear classification in JMLR 2008.