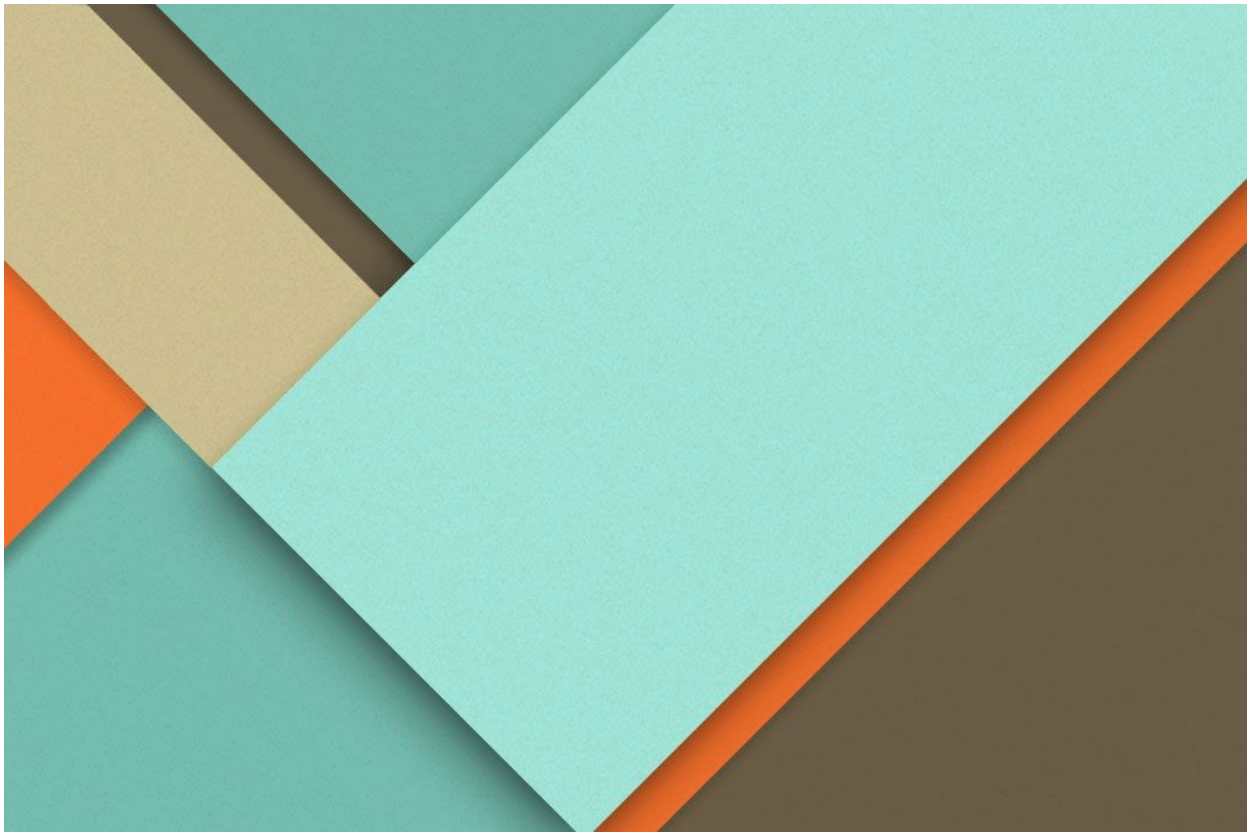


Transfer Feature Learning with Joint Distribution Adaptation

30.04.2020

Chuyue Wu, Menghong Han, Xiaofan Sun

International Business School, MSBA

Brandeis University

Contents

1. Introduction about Paper 1

1.1. Problem Definition 1

1.2. Challenge 2

1.3. Solution 2

1.4 Step by step algorithm 4

1.5 Experiments 4

2. Comments 8

3. Implementations 8

3.1. USPS, MNIST and COIL_20 8

3.2. PIE 9

3.3. OFFICE and Caltech-256 10

4. Conclusion 11

5. Reference 11

1. Introduction about Paper

1.1. Problem Definition

Transfer learning is an advanced machine learning technique, focusing on applying knowledge gained from one domain into another different but related domain. This technique is widely used nowadays. However, traditional transfer learning has some drawbacks, since most methods cannot simultaneously reduce distance of both marginal and conditional distribution between source and target domain.

Joint distribution Adaptation method is used to solve this problem. It aims to jointly reduce distance of both marginal and conditional distribution together. During our later experiments and validation, it appears to be a better algorithm than most others.
Here are some basic definitions about this topic we need to know.

Definition 1 (Domain)
A domain D is composed of an m- dimensional feature space X and a marginal probability distribution P(x), i.e., D = {X,P(x)}, where $x \in X$.

Definition 2(Task)
Given domain D, a task T is com- posed of a C -cardinality label set Y and a classifier f (x), i.e.,T ={Y,f(x)},where $y \in Y$,and $f(x)=Q(y|x)$can be interpreted as the conditional probability distribution.

Assumption 1: Xs = Xt, Ys = Yt , X and Y have same dimensions of features
Assumption 2: $Ps(xs) \neq Pt(xt)$, $Qs(ys|xs) \neq Qt(yt|xt)$, X and Y have different marginal and conditional distributions
Assumption 3: Source data are labeled, but target data are not.

1.2. Challenge

Obviously, the main challenge here is to find a way to reduce both the marginal distribution difference and conditional distribution difference between source domain and target domain.

1.3. Solution

The author's solution is to find an adaptation matrix A, to minimize the distance of both marginal and conditional distribution between source and target domain.

$$\begin{aligned}\min_{T} &\left\|\mathbb{E}_{P(\mathbf{x}_s,y_s)}\left[T\left(\mathbf{x}_s\right),y_s\right]-\mathbb{E}_{P(\mathbf{x}_t,y_t)}\left[T\left(\mathbf{x}_t\right),y_t\right]\right\|^2 \\ &\approx \left\|\mathbb{E}_{P_s(\mathbf{x}_s)}\left[T\left(\mathbf{x}_s\right)\right]-\mathbb{E}_{P_t(\mathbf{x}_t)}\left[T\left(\mathbf{x}_t\right)\right]\right\|^2 \\ &+ \left\|\mathbb{E}_{Q_s(y_s|\mathbf{x}_s)}\left[y_s|T\left(\mathbf{x}_s\right)\right]-\mathbb{E}_{Q_t(y_t|\mathbf{x}_t)}\left[y_t|T\left(\mathbf{x}_t\right)\right]\right\|^2\end{aligned}$$

The way we measure this distance is called MMD, which computes the distance between the sample means of the source and target data in the k-dimensional embeddings:

$$\mathrm{MMD}\left[\mathcal{F},X,Y\right]:=\sup_{f\in\mathcal{F}}\left(\frac{1}{m}\sum_{i=1}^{m}f(x_i)-\frac{1}{n}\sum_{i=1}^{n}f(y_i)\right)$$

The MMD distance for marginal distribution adaption shows as follows:

$$\left\|\frac{1}{n_s}\sum_{i=1}^{n_s}\mathbf{A}^{\mathrm{T}}\mathbf{x}_i-\frac{1}{n_t}\sum_{j=n_s+1}^{n_s+n_t}\mathbf{A}^{\mathrm{T}}\mathbf{x}_j\right\|^2=\mathrm{tr}\left(\mathbf{A}^{\mathrm{T}}\mathbf{X}\mathbf{M}_0\mathbf{X}^{\mathrm{T}}\mathbf{A}\right) \tag{3}$$

where $\mathbf{M}_0$ is the MMD matrix and is computed as follows

$$(M_0)_{ij}=\begin{cases}\frac{1}{n_s n_s}, & \mathbf{x}_i,\mathbf{x}_j\in\mathcal{D}_s\\ \frac{1}{n_t n_t}, & \mathbf{x}_i,\mathbf{x}_j\in\mathcal{D}_t\\ \frac{-1}{n_s n_t}, & \text{otherwise}\end{cases} \tag{4}$$

Also, when we are calculating the distance of conditional distribution, here is another problem. Without labels of target domain, we need to get pseudo labels first. For example, we can train a KNN model with labeled source data and apply that model on the target domain to get pseudo labels, and this is operated in an iterated way to constantly increase performance until convergence. After we got the pseudo labels, we can calculate class-conditional probability $P(\mathbf{x}_t|y_t)$ and then based on Bayes' theorem, we can easily calculate $P(y_t|\mathbf{A}^{\top}\mathbf{x}_t)$. The MMD distance between the class- conditional distributions is:

$$\left\|\frac{1}{n_s^{(c)}}\sum_{\mathbf{x}_i\in\mathcal{D}_s^{(c)}}\mathbf{A}^{\mathrm{T}}\mathbf{x}_i-\frac{1}{n_t^{(c)}}\sum_{\mathbf{x}_j\in\mathcal{D}_t^{(c)}}\mathbf{A}^{\mathrm{T}}\mathbf{x}_j\right\|^2=\mathrm{tr}\left(\mathbf{A}^{\mathrm{T}}\mathbf{X}\mathbf{M}_c\mathbf{X}^{\mathrm{T}}\mathbf{A}\right) \tag{5}$$

$$(M_c)_{ij}=\begin{cases}\frac{1}{n_s^{(c)}n_s^{(c)}}, & \mathbf{x}_i,\mathbf{x}_j\in\mathcal{D}_s^{(c)}\\ \frac{1}{n_t^{(c)}n_t^{(c)}}, & \mathbf{x}_i,\mathbf{x}_j\in\mathcal{D}_t^{(c)}\\ \frac{-1}{n_s^{(c)}n_t^{(c)}}, & \begin{cases}\mathbf{x}_i\in\mathcal{D}_s^{(c)},\mathbf{x}_j\in\mathcal{D}_t^{(c)}\\ \mathbf{x}_j\in\mathcal{D}_s^{(c)},\mathbf{x}_i\in\mathcal{D}_t^{(c)}\end{cases}\\ 0, & \text{otherwise}\end{cases} \tag{6}$$

Beyond that, there is some restriction on our optimization function. After adapting the matrix A, we actually still want to keep original characteristics of previous data as much as possible. In another word, we want to reduce the reconstruction error of input data. Here is how PCA works. It offers some kind of restriction to the overall optimization function, and in a mathematical way, to minimize the reconstruction error is actually to maximize the data variance below:

$$\max_{\mathbf{A}^{\mathrm{T}}\mathbf{A}=\mathbf{I}} \operatorname{tr}\left(\mathbf{A}^{\mathrm{T}}\mathbf{X}\mathbf{H}\mathbf{X}^{\mathrm{T}}\mathbf{A}\right)$$

Now we have two MMD distance to minimise and a variance to maximize, based on the gener- alized Rayleigh quotient, minimizing Equations (3) and (5) such that Equation (2) is maximized is equivalent to minimizing Equations (3) and (5) such that Equation (2) is fixed. We change this problem into a constrained optimization problem.

$$\min \quad \sum_{c=0}^{C} tr(\mathbf{A}^{\top}\mathbf{X}\mathbf{M}_c\mathbf{X}^{\top}\mathbf{A}) + \lambda\|\mathbf{A}\|_F^2 \quad \text{s.t.} \quad \mathbf{A}^{\top}\mathbf{X}\mathbf{H}\mathbf{X}^{\top}\mathbf{A} = \mathbf{I}$$

Based on the constrained optimization theory, we can use the Lagrange function to solve this problem.

note $\mathbf{\Phi} = \operatorname{diag}(\phi_1, \ldots, \phi_k) \in \mathbb{R}^{k\times k}$ as the Lagrange multiplier, and derive the Lagrange function for problem (7) as

$$\begin{aligned} L = &\operatorname{tr}\left(\mathbf{A}^{\mathrm{T}}\left(\mathbf{X}\sum\nolimits_{c=0}^{C}\mathbf{M}_c\mathbf{X}^{\mathrm{T}} + \lambda\mathbf{I}\right)\mathbf{A}\right) \\ &+ \operatorname{tr}\left(\left(\mathbf{I} - \mathbf{A}^{\mathrm{T}}\mathbf{X}\mathbf{H}\mathbf{X}^{\mathrm{T}}\mathbf{A}\right)\mathbf{\Phi}\right) \end{aligned} \tag{9}$$

Setting $\frac{\partial L}{\partial \mathbf{A}} = \mathbf{0}$, we obtain generalized eigendecomposition

$$\left(\mathbf{X}\sum\nolimits_{c=0}^{C}\mathbf{M}_c\mathbf{X}^{\mathrm{T}} + \lambda\mathbf{I}\right)\mathbf{A} = \mathbf{X}\mathbf{H}\mathbf{X}^{\mathrm{T}}\mathbf{A}\mathbf{\Phi} \tag{10}$$

Finally, finding the optimal adaptation matrix A is reduced to solving Equation (10) for the k smallest eigenvectors.

1.4 Step by step algorithm

1. Get first-round pseudo labels by classifier f
2. Constructing a MMD matrix for distance minimization
   Get the optimized adaptation matrix with restriction of PCA
3. Update source domain into $\{(\mathbf{A}^{\mathrm{T}}\mathbf{x}_i, y_i)\}_{i=1}^{n_s}$ and train a new f
4. Update pseudo labels $\{\widehat{y}_j := f(\mathbf{A}^{\mathrm{T}}\mathbf{x}_j)\}_{j=n_s+1}^{n_s+n_t}$
5. Repeat step 2 to 4 until convergence
6. In the end, return a final classifier f, An adaptation matrix A

1.5 Experiments

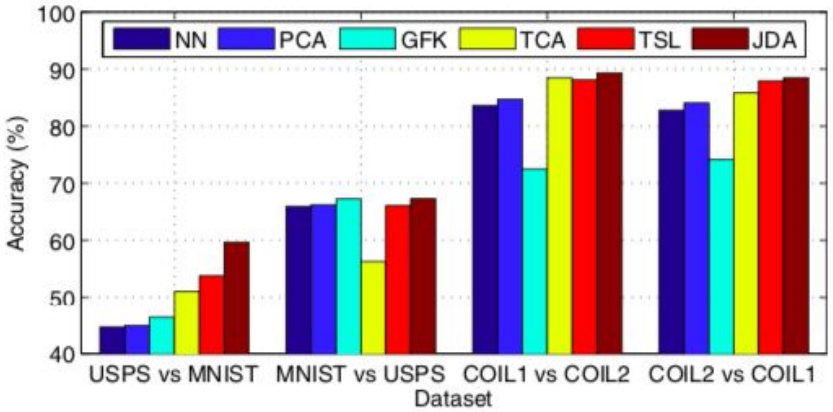
To evaluate the JDA approach, this paper conducts extensive experiments for image classification problems. There are 6 datasets being used, and they are divided into 4 groups. The first one is handwriting digits USPS and MNIST, the second one is COIL20, contains objects in different directions, the third one PIE are people's faces in right or left poses, and the last one is objects from 4 different websites.
Some baseline methods below are used to compare the results of JAD. Specifically, TCA and TSL can both be viewed as a special case of JDA with C = 0. TSL adopts Bregman divergence instead of MMD as the distance for comparing distributions.

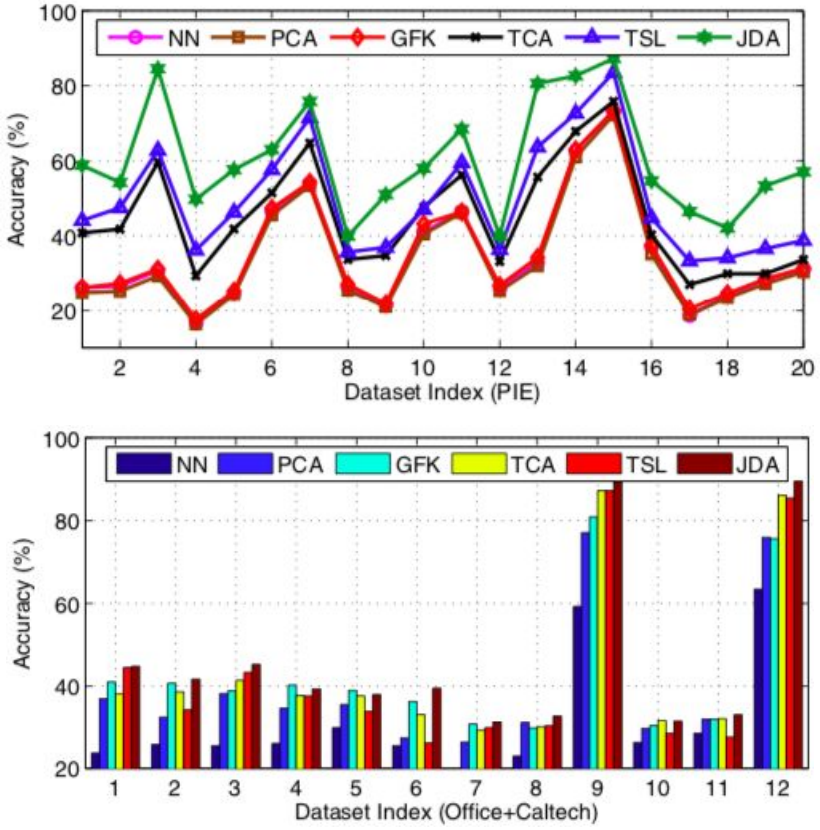
- *1-Nearest Neighbor Classifier (NN)*
- *Principal Component Analysis (PCA) + NN*
- *Geodesic Flow Kernel (GFK) [6] + NN*
- *Transfer Component Analysis (TCA) [15] + NN*
- *Transfer Subspace Learning (TSL) [22] + NN*

The average accuracy of total 36 cross-domain images is shown here. The average classification accuracy of JDA on the 36 datasets is 57.37% and the performance improvement is 7.57% compared to the best baseline method TSL.

Dataset	Standard Learning		Transfer Learning			
	NN	PCA	GFK	TCA	TSL	JDA
Average	37.46	39.84	41.19	47.22	49.80	**57.37**

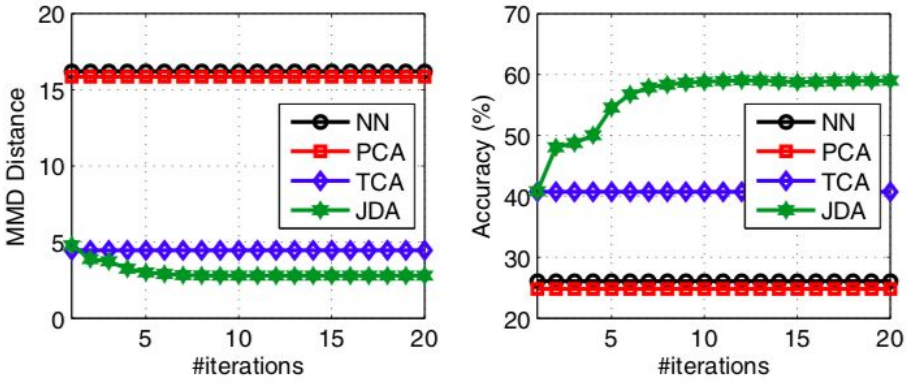
The accuracy results of different datasets are shown below. Note that the adaptation difficulty in the 36 datasets varies a lot. For example, GFK performs pretty well on Office+Caltech. Generally, JDA significantly outperforms TCA, and JDA also achieves much better performance than TSL.





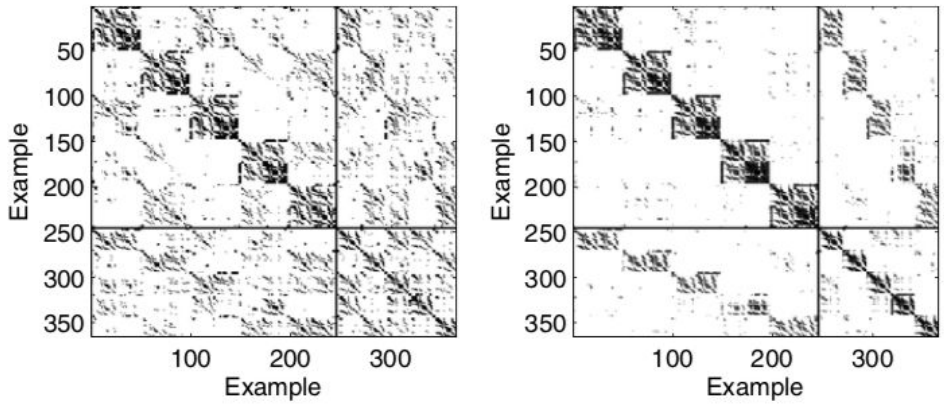
The paper further verifies the effectiveness of JDA by inspecting the distribution distance and the similarity of embeddings.

For distribution distance, (1) Without learning a feature representation, the distribution distance of NN in the original feature space is the largest. (2) PCA can learn a new representation in which the distribution distance is slightly reduced, but not much. (3) TCA can substantially reduce the distribution distance by explicitly reducing the difference in the marginal distributions, so it can achieve better classification accuracy. (4) JDA can reduce the difference in both the marginal and conditional distributions, thus it can extract a most effective and robust representation for cross-domain problems.



(a) MMD distance w.r.t. #iterations (b) Accuracy (%) w.r.t. #iterations

For similarity of embeddings, the top-left and bottom-right submatrices indicate within-domain similarity, the top-right and bottom- left submatrices indicate between-domain similarity. 1) the between-domain similarity should be high enough and the between-class similarity should be low. In this sense, we see that TCA cannot ex- tract a good embedding, but JDA can.



(c) Similarity of TCA embedding (d) Similarity of JDA embedding

Also, it tests JAD parameter sensitivity and time complexity. After running JDA with varying values of k, and plot classification accuracy with different values of k , and choose k ∈ [60, 200] to be optimal parameter values. It's the same for numda.

For time complexity, It shows that JDA is T-times worse than TCA but much better than TSL.

Method	Runtime (s)	Method	Runtime (s)	Method	Runtime (s)
NN	4.85	PCA	2.63	GFK	4.58
TCA	3.80	TSL	1789	JDA	46.32

2. Comments

In this paper, a Joint Distribution Adaptation (JDA) approach has been proposed for robust transfer learning. JDA aims to simultaneously adapt both marginal and conditional distributions in a principled dimensionality reduction procedure. Extensive experiments show that JDA is effective and robust for a variety of cross-domain problems, and can significantly outperform several state-of-the-art adaptation methods even if the distribution difference is substantially large.

There are also some directions that can be explored deeper for JDA. For example, distance measures can be extended to other representation learning methods such as Sparse Coding. In addition, time complexity of JDA algorithms is not very ideal. Maybe it can be improved by simplifying algorithms or finding other distance calculation functions. Also, some other classification tools can be combined with JDA such as KNN or SVM, that's what we want to explore in the next part.

3. Implementations

Based on the JDA algorithm in the paper, we implement our version JDA in Python on all the datasets including USPS, MNIST, COIL20, PIE, Office and Caltech-256, and compare accuracy of our implementation with the one paper provided. To try to add some improvement in our implementation, we used the basic classification model KNN instead of NN in the paper as the classifier. That's because we considered when using NN, the nodes and layers are hard to identify at the beginning. But for KNN, the k value in a small range usually doesn't make a big change. The results of our implementation are shown below.

3.1. USPS, MNIST and COIL_20

	Accuracy in Essay	**Accuracy in our JDA with KNN Model**
USPS vs MNIST	59.65	63.2
MNIST vs USPS	67.28	71.78
COIL1 vs COIL2	89.31	84.72
COIL2 vs COIL1	88.47	83.33

We first set k=20 and lumda=1, since in the grid search plot the paper provided, for USPS vs MNIST dataset, k=20 while lumda around 1 tends to have highest accuracy. After changing k=1 to k=20 for the KNN classifier, the accuracy for USPS vs MNIST and MNIST vs USPS

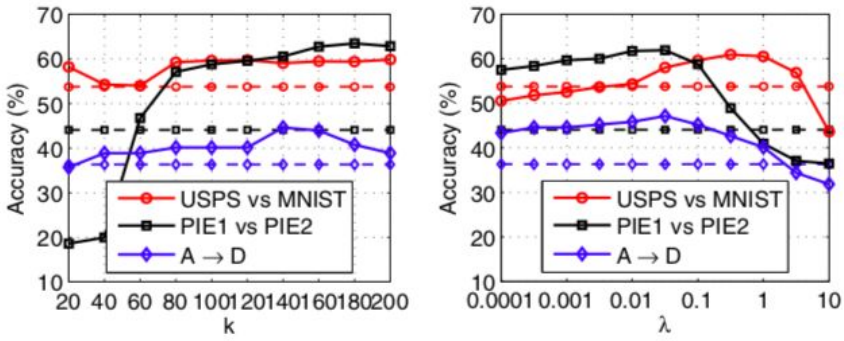
increased by nearly 4%, while the accuracy for COIL1 vs COIL2 and COIL2 vs COIL1 could not beat the accuracy in paper. We tried smaller k, for example k=5, the accuracy increased but still cannot beat the paper's result. In order to further improve the accuracy, given the efficient time, we would try grid research for different k in KNN models and try different classification models such as SVM, LGB tree, CNN and so on.

3.2. PIE

	Accuracy in Essay	Accuracy in our JDA with KNN Model
PIE1 vs PIE2 (1)	58.81	62.00
PIE1 vs PIE3 (2)	54.23	65.32
PIE1 vs PIE4 (3)	84.50	84.74
PIE1 vs PIE5 (4)	49.75	50.25
PIE2 vs PIE1 (5)	47.62	58.37
PIE2 vs PIE3 (6)	62.93	66.42
PIE2 vs PIE4 (7)	75.82	77.86
PIE2 vs PIE5 (8)	39.89	43.81
PIE3 vs PIE1 (9)	50.96	54.95
PIE3 vs PIE2 (10)	57.95	65.44
PIE3 vs PIE4 (11)	68.45	72.18
PIE3 vs PIE5 (12)	39.95	47.73
PIE4 vs PIE1 (13)	80.58	82.53
PIE4 vs PIE2 (14)	82.63	86.07
PIE4 vs PIE3 (15)	87.25	86.89
PIE4 vs PIE5 (16)	54.66	58.46
PIE5 vs PIE1 (17)	46.46	51.17
PIE5 vs PIE2 (18)	42.05	53.9
PIE5 vs PIE3 (19)	53.31	58.88
PIE5 vs PIE4 (20)	57.01	63.29

For PIE face images, we set the parameter of JDA at first. For k, although USPS or OFFICE dataset can be set as 30, for PIE dataset, it is stable when k is bigger than 80 according to

the parameter sensitivity in paper. So we set k to be 100 in our implementations. In addition, for lambda, the optimal range is [0.001, 0.05], so we changed it to be 0.01 for PIE dataset.



(a) #subspace bases k (b) regularization parameter λ

At the beginning, we set the number of neighbors to be 5 and run the model. The first eight group accuracy results are 0.5494, 0.5643, 0.8317, 0.5257, 0.5903, 0.6538, 0.6867, 0.4510. There are 5 results better than the results in paper among 8 results. But when we set k to be 1, all eight groups beat the paper results. So we decided to use n-neighbors at 1. After running the model for all the 20 combinations, the results are amazing and impressive. All the results beat the original one in the paper. It's hard to explain. I guessed it might because the face images are "more gray". The colors of these images don't have a strong contrast as USPS or MNIST data, so smaller n neighbors can make a better classification.

3.3. OFFICE and Caltech-256

	Accuracy in Essay	Accuracy in our JDA with KNN Model
C to A	44.78	43.11
C to W	41.69	32.2
C to D	45.22	40.76
A to C	39.69	37.48
A to W	37.97	31.86
A to D	39.49	35.66
W to C	31.17	28.58
W to A	32.78	34.13
W to D	89.17	74.52
D to C	31.52	30.63

D to A	33.09	30.58
D to W	89.49	66.78

For OFFICE and Caltech-256 images, we set k to 30, the lambda to 1, and n-neighbor to 5. The results are not very good compared with the results in paper.

4. Conclusion

From our re-implementing process, we found that different classification tools and different parameters can cause a bid difference, but JDA does help a lot for making good predictions. We can use grid research for different k in KNN models and even try different classification models such as SVM, LGB tree or CNN combined with the JDA algorithm, to find the best model.

In the real world, it's very common to meet this situation: we have accumulated rich experience in some domain, but with the rapid development of new technology, completely new domains are appearing one by one. With the help of JAD, we can transfer what we have in those old and mature domains to these new domains. It's really practical and useful.

5. Reference

- Mingsheng Long (MATLAB) http://ise.thss.tsinghua.edu.cn/~mlong/
- jindong Wang(Python) https://github.com/jindongwang/transferlearning/tree/master/code/traditional/JDA
- A.Gretton,K.M.Borgwardt,M.J.Rasch,B.Scholkopf,and A. J. Smola. A kernel method for the two-sample problem. In Proceedings of NIPS, 2006.
- Q. Sun, R. Chattopadhyay, S. Panchanathan, and J. Ye. A two-stage weighting framework for multi-source domain adaptation. In Proceedings of NIPS, 2011.
- S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang. Domain adaptation via transfer component analysis. IEEE TNN, 22(2):199–210, 2011.
- B. Gong, Y. Shi, F. Sha, and K. Grauman. Geodesic flow kernel for unsupervised domain adaptation. In Proceedings of CVPR, 2012.