Unsupervised Transfer Learning

By: Sarah Yurick

1. Introduction

In unsupervised transfer learning, both the source and target domains as well as the source and target tasks are distinct but overlapping [1]. Thus, unsupervised transfer learning seeks to improve learning the target predictive function using knowledge from the source domain and source learning task, where the source and target labels are not used. In this paper, several unsupervised transfer learning techniques will be described and implemented for three target tasks (using a total of six datasets; three for the source domains and three for the target domains).

For the purpose of illustration, these unsupervised transfer learning algorithms will be described within the context of predicting character deaths from the HBO TV series Game of Thrones, given additional "auxiliary" information from the well-known Titanic survival dataset. While these datasets (as well as the other datasets used in training and testing for other tasks) will be described in further detail in the implementation section of this paper, for now it is just important to know that the source domain is from the Titanic dataset, the target domain is from the Game of Thrones dataset, the source task is to predict whether or not a given passenger survived the Titanic shipwreck, and the target task is to predict whether or not a given character survived until the end of the Game of Thrones TV show.

The source and target domains are distinct but overlapping in the fact that they have some shared features and some features which are unique to only one of the domains [1]. For example, both a Titanic survival dataset and a Game of Thrones character deaths dataset contains features for gender and age, but the Titanic dataset has the feature "embarked" to indicate from which port the passenger boarded, for which there is no equivalent in the Game of Thrones dataset.

The source and target tasks are similar in that they focus on solving the same type of unsupervised learning problem, such as clustering, dimensionality reduction, and density estimation, but with different end meanings [1]. For example, clustering can be done on both the Titanic dataset and the Game of Thrones dataset, but for the Titanic dataset the goal is to cluster based on whether or not a given passenger survived the shipwreck, while for the Game of Thrones dataset the goal is to cluster based on whether or not that character survived until the end of the TV series.

In both cases, at least one of the datasets used in learning is unlabeled [1]. Thus, although all datasets used in this paper have labels which are used for evaluation, features such as "isAlive" from the Game of Thrones dataset and "Survived" from the Titanic dataset are removed prior to dimensionality reduction and clustering.

2. Algorithms and Methods

2.1 Self-taught Clustering

Self-taught clustering is a type of unsupervised learning which aims to cluster unlabeled sparse data with the help of a large amount of unlabeled auxiliary (source) data which can be clustered easily. This approach is described by W. Dai et al [2]. The general idea is that the size of the target data is too small, meaning that traditional clustering cannot cluster it well. The target and auxiliary data may also have

different distributions. However, both the target and auxiliary data share at least some of the same features. Thus, self-taught clustering aims to cluster the target and auxiliary data simultaneously.

In order to perform self-taught clustering, let X and Y be discrete random variables corresponding to the target and auxiliary data, respectively, and let Z be a discrete random variable corresponding to the shared feature space between them. Furthermore, let X', Y', and Z' represent the clusterings on X, Y, and Z, and C_X, C_Y, and C_Z represent the functions which map the random variables to their corresponding clusters. Our objective is to find the function C_X to cluster the target data X, with the help of C_Y and C_Z.

This can be done by minimizing the loss in mutual information before and after co-clustering. The definition of mutual information between two input random variables is [3]:

$$I(X,Z) = \sum_{x\in X}\sum_{z\in Z} p(x,z) \log [\ p(x,z) / (p(x)p(z))\ ] \quad (1)$$

And thus, the objective function J to minimize is:

$$J = [\ I(X,Z) - I(X',Z')\ ] + \lambda[\ I(Y,Z) - I(Y',Z')\ ] \quad (2)$$

Where λ is a hyperparameter used to balance the influence of the target data versus the auxiliary data during training. In order to minimize the objective function, it can be reformulated into the form of Kullback-Leibler (KL) divergence, which is defined as [3]:

$$D(p||q) = \sum_x p(x) \log [\ p(x) / q(x)\ ] \quad (3)$$

The joint probability distribution of two random variables with respect to their co-clusters is also defined as [2]:

$$p'(x,z) = p(x',z')\ [\ p(x) / p(x')\ ]\ [\ p(z) / p(z')\ ] \quad (4)$$

The probability $p'(z|x')$ can be defined by using the above definition:

$$p(x',z')\ [\ p(x) / p(x')\ ]\ [\ p(z) / p(z')\ ] = p(x)\ [\ p(x',z') / p(x')\ ]\ [\ p(z) / p(z')\ ] = p(x)\ p'(z|x') \quad (5)$$

Thus J is reformulated as:

$$J = D(\ p(X,Z) \parallel p'(X,Z)\ ) + \lambda D(\ q(Y,Z) \parallel q'(Y,Z)\ ) \quad (6)$$

With this, the KL divergence with respect to joint probability distributions can be reformulated even further into:

$$D(\ p(X,Z) \parallel p'(X,Z)\ ) = \sum_{x'\in X'}\sum_{x\in X} p(x)\ D(\ p(Z|x) \parallel p'(Z|x')\ ) \quad (7)$$

Details concerning the derivation of the above equality can be found in section 3.2 of the original paper by W. Dai et al [2]. Thus minimizing $D(\ p(Z|x) \parallel p'(Z|x')\ )$ for each x will decrease the global optimization function J, and iteratively choosing the best cluster x' for each x will minimize the KL divergence. It is easy to derive similar reformulations for choosing clusters for each y and z as well. Thus, in order to minimize the objective function, the following three updates can be performed for a fixed number of iterations T:

$$C_X(x) = \text{argmin}_{x'\in X'}\ D(\ p(Z|x) \parallel p'(Z|x')\ ), \text{ for all } x\in X \quad (8)$$
$$C_Y(y) = \text{argmin}_{y'\in Y'}\ D(\ q(Z|y) \parallel q'(Z|y')\ ), \text{ for all } y\in Y \quad (9)$$

$$C_Z(z) = \text{argmin}_{z' \in Z'} \, p(z)\, D(\, p(X|z) \,||\, p'(X|z')\,) + \lambda\, q(z)\, D(\, q(Y|z) \,||\, q'(Y|z')\,), \text{ for all } z \in Z \qquad (10)$$

After T iterations, return the most recently updated C_X as the final clustering function on the target data. It is proven in the original paper (section 3.2) that iteratively updating each of the clustering functions C_X, C_Y, and C_Z as described above will monotonically decrease the objective function J. Furthermore, the time complexity of this algorithm is $O(L_1+L_2)$, where L_1 is the total number of (x,z) co-occurrences in the target dataset and L_2 is the total number of (y,z) co-occurrences in the auxiliary dataset; the space complexity is also $O(L_1+L_2)$ [2].

2.2 Transferred Dimensionality Reduction

Dimensionality reduction is desired in applications such as image recognition, in which data is high-dimensional but can be more efficiently represented in a lower dimension. Transferred dimensionality reduction aims to perform unsupervised discriminative dimensionality reduction with the help of related prior knowledge from other classes in the same type of concept [4]. In other words, task-related information from known classes is transferred to the unlabeled target class domains so that a better subspace can be found to discriminate between them. The paper by Z. Wang et al. proposes an algorithm called Transferred Discriminative Analysis, which uses clustering to generate class labels for the unlabeled target data and can then perform dimensionality reduction [4].

In order to demonstrate the purpose of Transferred Discriminative Analysis, a toy problem is provided: there are four classes of data, each which contains 50 random samples and forms a moon shape (see Figure 1 below).

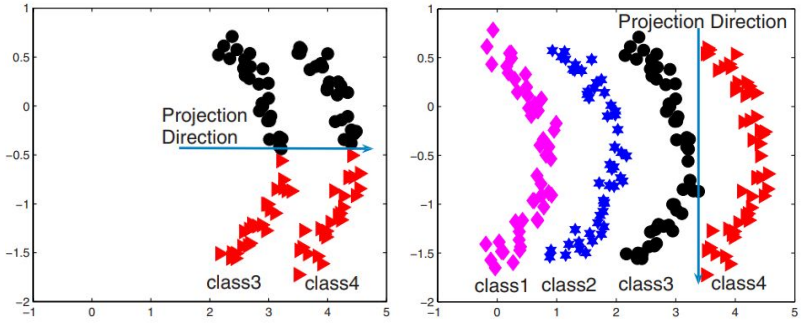

Figure 1: Toy example for intuitively illustrating the power of Transferred Discriminative Analysis (TDA). The plot on the left discriminates between class 3 and 4 using existing discriminative dimensionality reduction and clustering methods. The plot on the right discriminates between class 3 and 4 using TDA. [4]

In the above plots, classes 1 and 2 are labeled while classes 3 and 4 are not; the task is to find the best subspace to discriminate between classes 3 and 4. Using dimensionality reduction and clustering methods on classes 3 and 4 only does not discriminate between them very well, but with the help of classes 1 and 2 as seen in the right plot, the goal of Transferred Discriminative Analysis is to be better able to find the suitable subspace.

Let the labeled source dataset be denoted as $D_L = \{X_L, Y_L\}$, where X_L is the features and Y_L is the labels, and let the unlabeled target dataset be denoted as $D_U = \{X_U\}$, which have unlabeled and distinct classes from Y_L. For simplicity, assume the sample mean of $D = \{D_L, D_U\}$ is zero. In order to calculate S_b, which is the between-class information of D, measure the between-class scatter of the labeled data and an estimate of the between-class scatter of the unlabeled data, which is obtained by clustering [4]:

$$S_b = S_{bl} + S'_{bu} = \sum_{i=1}^{C} l_i m_i m_i^T + \sum_{j=1}^{K} l_j m_j' m_j^T, \quad (11)$$

Where l_i and l_j are the number of samples in that class (in other words, the labeled dataset has C classes [1, 2, …, C] and the unlabeled dataset is known to have K classes [1, 2, …, K], so for example l_C would be the number of samples in class C), and m_i is the mean of those samples. Now create a Graph Laplacian of the dataset, and define the adjacency matrix M of the neighborhood graph associated with the data [5]:

$$M_{ij} = \exp\{- ||x_i - x_j||^2 / (2\sigma^2)\} \text{ if } (x_i, x_j) \in E, \text{ 0 otherwise} \quad (12)$$

Where $(x_i, x_j) \in E$ means that the vertices x_i and x_j are neighbors and have an edge between them. The normalized Graph Laplacian L is [6]:

$$L = I - D^{-0.5} M D^{-0.5} \quad (13)$$

Where I is the identity matrix, and the diagonal matrix D satisfies $D_{ii} = d_i$ where d_i is the degree of vertex x_i as calculated with $\sum_{j=1}^{l+u} M_{ij}$. Because the sample mean is zero, the structure of the dataset can be described as:

$$(S_t + \lambda X L X^T) = X(I + \lambda L)X^T \quad (14)$$

Where S_t is the total-scatter matrix ($S_t = \sum_{i=1}^{l} x_i x_i^T$, with l labeled samples and zero sample mean, see section 2.1 of original paper for more details) and $X = \{X_L, X_U\}$. Thus, the objective function is:

$$\max \text{trace}_{W, Hu'} \left((W^T (S_t + \lambda X L X^T) W)^{-1} (W^T (S_{bl} + S_{bu}'(H_u')) W) \right) \quad (15)$$

Where H_u' represents the clustering structure of the unlabeled data and W is the projection direction for the dimensionality reduction.

In order to optimize the objective function, a clustering method such as K-means is first used to estimate the discriminative structure of the unlabeled data, the graph matrix M is constructed, and the Graph Laplacian L is calculated. Then, the data is projected into a lower dimension by a supervised method to revise the clustering result. More specifically, the optimal subspace is found using the eigenvalue decomposition for the objective function, using:

$$(S_t + \lambda_1 X L X^T + \lambda_2 I) w_j^* = \eta_j (S_{bl} + S_{bu}'(H_u')) w_j^* \quad (16)$$

Where w_j^* are the eigenvalues corresponding to the m largest eigenvalues of $(S_t + \lambda_1 X L X^T + \lambda_2 I)^{-1}(S_{bl} + S_{bu}'(H_u'))$, with a fixed H_u' and $\lambda_2 I$ as a regularization term with small real number λ_2 to ensure the nonsingularity of the matrix $S_t + \lambda_1 X L X^T$. Note that the discriminative structure of the labeled data should still be preserved, which will force the unlabeled data clustering to form a similar discriminative structure to the labeled data, which is how the transfer of information occurs. Then, cluster the unlabeled subspace using K-means; that is, fix the projection direction W and use K-means to solve $\max_{Hu', S_{bu}'(H_u')}$. The process of performing supervised dimensionality reduction and then compact clustering for the target data is repeated, and stops when the structure consistency of all of the data in the subspace and within the unlabeled data are balanced, that is when H_u' doesn't change. Thus, each sample x can be embedded into an m-dimensional subspace with $W^T x$, where $W = [w_1^*, w_2^*, \ldots, w_m^*]$.

The above algorithm can also be generalized using the kernel trick and the Representer Theorem, which is described in section 3.3 of the original paper but is omitted here for the sake of compactness. The computational complexity of the algorithm is $O(d^2 nt)$, where d is the dimension of the data, n is the

number of total data points, and t is the number of iterations, while the computational complexity using the kernel trick is $O(n^2dt)$ [4].

3. Implementation and Results

For this report, the self-taught clustering algorithm was implemented in Python 3.8 and ran in a Jupyter Notebook with Python version 3.7. The implementation utilizes three Python files: "selftaught_clustering.py" is the main file to be run for the experiment, "stc.py" consists of methods which are used during training and evaluation, and "data.py" is used to load and prepare the datasets for two of the three target tasks. All Python scripts, the Jupyter Notebooks (for Spam, Amazon, and Survival), data files, and academic papers referenced can be found in the "Sarah/" directory on csevcs; note that the datasets used for the cumulative section of the report are listed in the "data/" directory on csevcs for the convenience of the group as a whole, but in order to be run properly must be within the "Sarah/" directory. The remainder of this section discusses the specifics regarding each of the three target tasks; the first two tasks are referenced later in the cumulative section of this report for comparative analysis, and the third task was implemented out of the researcher's personal interests as well as one of several ways to go beyond the requirements of the project. The links to all datasets referenced can be found in the README.md file in the Sarah/ directory on csevcs.

The first task concerns spam filtering; the training dataset is named "task_a_labeled_train.tf" and the testing dataset is named "task_a_u00_tune.tf." These datasets do not contain raw text; instead, they are in a bag-of-words vector space representation, where attributes are the frequencies of that word. For the sake of computational time, only the first 1,000 features of the bag-of-words vector space representation were used, 1,000 examples were used as the auxiliary data (from the "task_a_labeled_train.tf" dataset), and 100 examples were used as the target data (from "task_a_u00_tune.tf"); the data is loaded using the "collect_spam_a_data" method from data.py which was implemented by co-researcher Sherry Chen.

After being loaded and the labels are separated from the features, the shared features between the datasets are discretized; for this choice, column 611 from both datasets was determined to be a good common space to use as the word associated with that feature was found to have a similar distribution between both datasets; in other words, the frequencies of feature 611 in "task_a_labeled_train.tf" were similar to the frequencies of feature 611 in "task_a_u00_tune.tf." Because of the sparseness of both of the datasets, for the sake of computational time the frequencies were then discretized to 0 (the word did not appear in that example) and 1 (the word appeared at least once in that example). After discretizing the datasets in this way, the rows of the datasets were encoded; in other words, rows of the dataset containing all of the exact same features are mapped to the same value, and rows of the dataset that are distinct from each other are mapped to different values.

After the above steps, the datasets were ready for self-taught clustering, as described above in section 2.1; first, binary clusters for X, Y, and Z were randomly initialized by using the train_test_split function from sklearn. Next, the lists representing the probability distributions for X, Y, and Z were created, as well as dictionaries representing the joint probability distributions. The joint probability distribution of the random variables with respect to their co-clusters as defined by equation 4 were also initialized. Finally, training begins (by updating cluster assignments according to equations 8-10), for which the number of iterations T=10 as suggested by the original paper was found to be sufficient (values of $T > 10$ were not found to significantly improve results) with $\lambda=10$. Values of $\lambda=0.1$, $\lambda=1$, $\lambda=10$, and $\lambda=100$ were tested on all three tasks, and $\lambda=10$ was found to yield the best results for all of them. The accuracy on the target task was measured to be 70%.

Although the original paper used entropy as a performance measure, accuracy was determined to be a sufficient metric because the comparative analysis at the end of this report uses accuracy.

The second task is rooted in sentiment analysis and concerns predicting a positive versus negative rating based on a given Amazon review. For this task, the source domain is Amazon reviews for items listed under the Toys and Games category, the target domain is reviews for the Patio, Lawn, and Garden category, and the source and target tasks are to predict whether the reviewer left a positive (greater than 2.5/5) or negative (less than 2.5/5) rating. This time, the datasets were in JSON format which included the attributes "reviewText" corresponding to the user's written review and "overall" corresponding to the user's rating out of 5. The data was parsed and loaded into an identical format as the spam data described above, using the "collect_amazon_data" method from data.py which was implemented by researcher Sarah Yurick.

Again for the sake of computational time, the first 1,000 features of the bag-of-words vector space representation were used, 1,000 Toys and Games reviews were used as the auxiliary data, and 100 Patio, Lawn, and Garden reviews were used as the target data. The datasets were discretized and encoded in the same manner as the spam data, except column 0 from both datasets was determined to be the shared feature space (using the same methodology and reasoning as before). After preparing the datasets, the self-taught algorithm was again found to be sufficient with T=10 iterations and λ=10, and the accuracy on the target task was measured to be 85%.

The third task was described in the introduction of this report, that is, using a Titanic survival dataset to predict Game of Thrones character deaths. Because both of these datasets are more "traditional" feature representations (i.e. in csv form), they were not preprocessed in the same way as the spam and Amazon datasets. Because the problem formulated by the original paper on self-taught clustering described using a very large auxiliary dataset and a very small target dataset, the Game of Thrones dataset is cut down to include roughly 50 characters (well-known characters that people actually care about!) while the Titanic dataset used in training includes 500 passengers.

The labels were then removed. After this, the year of birth, age, and number of dead relations features in the Game of Thrones dataset were rounded to the nearest tenth to make them more discrete. For the Titanic dataset, the fare and age features were rounded to the nearest tenth, the sex feature was encoded to 1 for male and 0 for female, and the features associated with sibling and parent counts were encoded to 0 (for none) and 1 (for one or more). Because both the Titanic and Game of Thrones datasets have features related to gender and age, both of these were used to make up the common feature space. Then, the rows of each dataset were encoded (without being discretized!) in the same way as for spam and Amazon reviews. This time, T=25 iterations was found to yield the best results with λ=10, which was a 68% accuracy on the target Game of Thrones task.

4. Research Extensions and Results

For the novel extensions to self-taught clustering, several dimensionality reduction techniques were implemented and tested: Principal Component Analysis (PCA), Sparse PCA, Truncated Singular Value Decomposition (SVD), Kernel PCA, and Locality Preserving Projections (LPP). All versions of PCA and SVD were implemented using sklearn, while Locality Preserving Projections were implemented with help from sklearn.neighbors and scipy.linalg; all methods for performing dimensionality reduction can be found in "dim_reduction.py" in the Sarah/ directory. In addition to performing dimensionality reduction on the auxiliary and target data separately, "transferred" dimensionality reduction was performed as well, meaning that the given dimensionality reduction technique was fitted to only the auxiliary dataset but used to transform both the auxiliary and target datasets.

Principal Component Analysis is a very famous dimensionality reduction technique, and thus only a brief overview distinguishing between the different types is provided. PCA assumes the data is centered and was generated by a Gaussian distribution. Given the set of examples, it finds an orthonormal set of vectors and the associated coefficients so that the projected points minimize the reconstruction error. The directions that maximize the variance of the data are the "principal components" used for this projection. Sparse PCA is a specialized version of PCA which is useful when the number of features is comparable or larger than the number of samples; its goal is to extract the set of sparse components that best reconstruct the data. Thus it is helpful for the spam and Amazon datasets, which have a lot of zero word frequencies, because it will select principal components that contain less nonzero values in their coefficients. Truncated Singular Value Decomposition is very similar to PCA, with the main difference being that the dataset does not have to be centered. Finally, Kernel PCA uses kernels to achieve non-linear dimensionality reduction. [7]

Locality Preserving Projections (LPP) are linear projective maps that are an alternative to PCA [8]. Given a matrix M which consists of a set of m points in n dimensions, the task is to find a transformation matrix A that maps these points to a new set of points in $l < n$ dimensions (the researcher chose l=10 to be consistent with the PCA implementations). In order to do this, the adjacency graph with m nodes is constructed such that there is an edge between two nodes if the points are among the k nearest neighbors of each other. For the implementation discussed in this paper, the author chose k=5 as suggested by various examples found on the Internet. The weights of the graph are set by using the kneighbors_graph method from sklearn. Finally, the eigenvectors and eigenvalues are found for the problem:

$$XLX^Ta = \lambda XDX^Ta \qquad (17)$$

Where D is a diagonal matrix whose entries are column sums of the weight matrix W, L = D - W is the Laplacian matrix, $[a_0, a_1, \ldots, a_{l-1}]$ the column vectors, and the λs are the eigenvalues. Thus, the first l column vectors make up the columns of the matrix A and A^Tx_i can be used to map all $[x_1, x_2, \ldots, x_m]$ points to l dimensions [8].

In "selftaught_clustering.py," dimensionality reduction is performed on the spam and Amazon datasets because the original datasets contain 1,000 features. For the sake of computational time, all experiments for the research extension of this project used only 100 auxiliary examples and 10 target examples. For normal PCA, the number of components selected is equal to the number of components which yield only a 5% reconstruction error (as calculated by sklearn), while for the sake of computational time Sparse PCA, Truncated SVD, Kernel PCA, and LPP used the top 10 components. The dimensionality reduction specified is applied after the datasets are loaded and before they are discretized, so as to perform dimensionality reduction on the actual word frequencies from the bag-of-words vector representation. "Transferred" Truncated SVD is unable to be applied to the spam datasets because too much information is lost by applying the auxiliary projection to the target domain (all rows of the target dataset end up being identical). Dimensionality reduction is not performed on the Titanic and Game of Thrones datasets because they contain discrete features, which have no real-valued space representation. See results from all trials in Table 1 below as well as in the Spam and Amazon Jupyter Notebooks.

Dimensionality Reduction Technique	*Spam datasets*	*Amazon reviews*
None (1000 source examples and 100 target examples)	70%	85%
None (100 source examples and 10 target examples)	80%	90%
PCA	60%	70%
Sparse PCA	60%	90%
Truncated SVD	80%	90%
Kernel PCA	70%	100%
“Transferred” PCA	60%	50%
“Transferred” Sparse PCA	60%	80%
“Transferred” Truncated SVD	80%	90%
“Transferred” Kernel PCA	N/A	90%
Locality Preserving Projections (LPP)	80%	90%
“Transferred” LPP	80%	90%

Table 1: Accuracy of Self-taught Clustering on the target data after performing various dimensionality reduction techniques.

5. Discussion and Conclusion

In almost every experiment, self-taught clustering performed best on the Amazon datasets, both with and without dimensionality reduction. This is likely due to how the bag-of-words vector representation was constructed for the Amazon dataset: the most frequent words are ordered by index (for example, the word “the” was the most frequent word across all datasets, so it had index 0). Meanwhile, the bag-of-words vector representation was already given by the spam dataset and thus was unordered in terms of overall word frequencies. Because of this, it can be argued that the Amazon datasets contained more meaningful information, as using the first 1,000 features was equivalent to using the top 1,000 most frequent words, whereas using the first 1,000 features in the spam dataset was equivalent to using a random selection of 1,000 words.

Self-taught clustering performed rather poorly on the Game of Thrones task, which could be explained by several reasons, the most obvious being that it was trained on less auxiliary and target data than the spam and Amazon reviews tasks. However, the fact that it performed poorly on the Game of Thrones task is unfortunate, as the initial inspiration for self-taught clustering was rooted in the problem of having a lot of auxiliary data and little target data. Thus, self-taught clustering was not found to be an appropriate solution to this problem.

When performing dimensionality reduction on the spam datasets, this never improved the accuracy results on the target task, which again is likely due to the fact that the spam dataset does not order the most

frequent words. For the Amazon task, however, dimensionality reduction still produced very strong results.

Normal PCA (both “transferred” and untransferred) did not yield strong scores for either task, which suggests that the underlying distributions are not Gaussian. However, Truncated PCA and Kernel PCA helped the clustering results, which suggests that it benefitted from some of the relaxed restrictions of PCA. Locality Preserving Projections also performed quite well and proved to be a better alternative than PCA. Interestingly enough, the results obtained after untransferred dimensionality reduction versus transferred were very similar, suggesting that using the projection for the auxiliary domain to transform the target domain was not harmful. This could be because the underlying distributions between the target and source domains were very similar after being discretized. Thus for these tasks “transferred” dimensionality reduction was not useful.

Overall, self-taught clustering was found to be a decent algorithm compared to other transfer learning techniques discussed, as discussed later in section H of this paper. Because dimensionality reduction was not found to significantly increase performance for either of the tasks, self-taught clustering without dimensionality reduction is discussed in the comparative analysis.

References

[1] S.J. Pan and Q. Yang, “A Survey on Transfer Learning,” in IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 10, Oct. 2010, pp. 1345-1359.

[2] W. Dai, Q. Yang, G. Xue, and Y. Yu, “Self-taught clustering,” in Proceedings of the 25th International Conference of Machine Learning. ACM, July 2008, pp. 200-207.

[3] T.M. Cover and J.A. Thomas, Elements of information theory. Wiley-Interscience, 1991.

[4] Z. Wang, Y. Song, and C. Zhang, “Transferred dimensionality reduction,” in Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008. Antwerp, Belgium: Springer, September 2008, pp. 550-565.

[5] O. Chapelle, B. Scholkopf, and A. Zien, Semi-Supervised Learning. Cambridge: MIT Press, 2006.

[6] F. Chung, “Spectral Graph Theory,” in CBMS Regional Conference Series in Mathematics, vol. 92, 1997.

[7] “2.5. Decomposing signals in components (matrix factorization problems),” in scikit-learn source page. 2020. https://scikit-learn.org/stable/modules/decomposition.html.

[8] X. He and P Niyogi, “Locality Preserving Projections,” in Proceedings of the Conference on Advances in Neural Information Processing Systems, 2003.