Using Functions on a Model Graph for Inductive Transfer

Eric Eaton ERICEATON@UMBC.EDU
Marie desJardins MARIEDJ@UMBC.EDU
University of Maryland Baltimore County, Department of Computer Science and Electrical Engineering

Terran Lane TERRAN@CS.UNM.EDU
University of New Mexico, Department of Computer Science

Abstract

In this paper, we propose a novel graph-based method for knowledge transfer. We embed a set of learned background models in a graph that captures the transferability between the models. We then learn a function on this graph that automatically determines the parameters to transfer to each learning task. Transfer to a new problem proceeds by mapping the problem into the graph, then using the function to determine the parameters to transfer in learning the new model. This method is analogous to inductive transfer along a manifold that captures the transfer relationships between the tasks.

1. Introduction

Knowledge transfer from previously learned tasks to a new task is a fundamental component of human learning. Most machine learning methods for transfer rely on an explicit set of *source* tasks to identify a set of model parameters that can be applied (transferred) to a new *target* task. In many cases, these source tasks are selected by an expert in advance. Methods for transfer may combine information from all source tasks (Marx et al., 2005; Kienzle & Chellapilla, 2006) or may use information from only a few tasks that are selected by an automated process (Thrun & O'Sullivan, 1996). Our approach to transfer can adapt itself automatically to use information only from relevant source tasks when given both relevant and irrelevant source tasks.

Given a set of source tasks and a new target task, our method attempts to determine the parameter values to transfer from the background tasks in learning the target task. Our approach to knowledge transfer embeds the models learned on the source tasks into a graph, using a notion of *transferability* to determine the edge weights. This model transfer graph represents a space for models using transferability as the metric, based on samples given by the source tasks, and corresponds to a discrete approximation of a high-dimensional manifold that captures the transfer relationships between the source tasks.

We then define a function on this manifold that determines the parameters for all models. This transfer function respects the local geometry of the graph and, therefore, the transfer relationships among the source tasks. We use the parameters of the source tasks' models as samples of the function at various locations on the manifold. We learn the transfer function using these sample values and the basis functions for the graph's Laplacian (see Section 3).

Given a target task, we interpolate its position on the true manifold by extending the graph to include the new task. We construct basis functions on the extended graph using the graph Laplacian, and then use these basis vectors to estimate the transfer function's value at the target task. This yields a transferred parameter vector for the target task, which we then use to learn the new model.

We define a *task* as a mapping from an instance space $X \subset \mathbb{R}^d$ to a set of labels $Y \in \mathbb{N}$. All tasks map from the same input space to the same set of labels. The goal for learning a model for each task is to recover the true mapping $X \rightarrow Y$ from the set of labeled training instances.

Our approach to transfer assumes that all models are from a single class of learning algorithms that supports transfer from one model to another. This transfer method must represent the transferable components of the *source* model as a parameter vector in $\mathbb{R}^\theta$, and learn the *target* model using this transferred parameter vector. This formulation of transfer using a parameter vector characterizes many existing transfer algorithms. In the experiments, we use a form of logistic regression biased toward a vector of weights.

2. Related Work

Parameter-based transfer has been previously used by Marx et al. (2005) to learn a logistic regression model us-

Presented at *North East Student Colloquium on Artificial Intelligence (NESCAI)*, 2008. Copyright the authors.

ing information from the source tasks. They fit logistic regression models independently to each source task, and then estimate the prior distribution for the target model's weights *a posteriori* from the source tasks' models. Kienzle and Chellapilla (2006) use a weight vector for transfer in SVMs, biasing the regularization term toward the weight vector, instead of the zero vector as in standard SVM training. The biased logistic regression method we propose in Section 6 is based on a combination of biased regularization and Marx et al.'s logistic regression transfer.

In contrast to the approaches of Marx et al. and Kienzle et al., which combine knowledge from all source tasks for transfer, Thrun and O'Sullivan's (1996) Task Clustering (TC) algorithm groups tasks for more selective transfer. Their method also transfers parameter vectors, sharing weighted Euclidean distance metrics between k-nearest-neighbor classifiers. Transfer occurs by having one k-nearest-neighbor model use the distance metric from another model. Upon receiving a new task, the TC algorithm matches the new task to a cluster, then transfers that cluster's distance metric to the new task.

Bakker and Heskes (2003) take a Bayesian approach to clustering tasks, using EM to optimize the task clusters. A second form of their approach uses a gating network, similar to that used in the mixture-of-experts model (Jordan & Jacobs, 1994), on top of the Bayesian EM framework to allow the priors to vary depending on the task's features.

Pratt's (1993) Discriminability-Based Transfer method for neural networks selectively transfers weights from a learned network, modifying them as needed to enable learning on a target task. Explanation-Based Neural Networks (Mitchell & Thrun, 1996) use a more indirect approach to parameter transfer, using extracted invariances about a domain to bias the learning of model parameters.

3. Technical Background

This section provides an overview of the spectral graph theory (Chung, 1994) used in this paper. Let $G(V, A)$ be an undirected connected weighted graph with a set of n vertices V and a weighted $n \times n$ adjacency matrix A. $A_{u,v} \neq 0$ implies that there is an edge between vertices u and v. Let the degree of vertex v be denoted by $d_v = \sum_{u=1}^{n} A_{u,v}$. Let T be the diagonal matrix where $T_{v,v} = d_v$. The combinatorial Laplacian matrix L for the graph is given by $L = T - A$:

$$L_{u,v} = \begin{cases} d_v - A_{v,v} & \text{if } u = v, \\ -A_{u,v} & \text{if } A_{u,v} \neq 0, \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

We can also define the *normalized Laplacian* matrix $\mathcal{L}$ as $\mathcal{L} = I - T^{-\frac{1}{2}} A T^{-\frac{1}{2}}$, where I is the identity matrix. While both forms of the Laplacian are applicable to our work, we found that the combinatorial Laplacian (hereafter referred to as just the *Laplacian*) worked better in our experiments and so we focus on it for the remainder of this paper.

The Laplacian L is symmetric; therefore, its eigenvalues are all real and non-negative. The eigendecomposition of L yields $L = Q \Lambda Q^T = \sum_{i=1}^{n} \lambda_i q_i q_i^T$, where Λ is the diagonal matrix of eigenvalues $[\lambda_1 \ \ldots \ \lambda_n]$ and the columns of Q are the eigenvectors $[q_1 \ \ldots \ q_n]$. Q forms an orthonormal basis for L.

Let $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ be the eigenvalues of L contained in Λ, and let q_i denote the eigenvector corresponding to λ_i. Spectral graph theory tells us that the smallest eigenvalue λ_1 is always 0 (with multiplicity 1, since G is connected) and q_1 is constant over all vertices.

Spectral graph theory has connections to Riemannian manifolds, which we use to define the surface on which transfer occurs. A graph G can represent a sample of the manifold $\mathcal{M}$, with the vertices as points on the manifold and the edges connecting points that are close to each other on the manifold. Let f be a smooth function $f : \mathcal{M} \to \mathbb{R}$ on a Riemannian manifold $\mathcal{M}$ with Riemannian metric ψ. The Laplace-Beltrami operator Δ is defined to be the divergence of the gradient of $\mathcal{M}$, and can act on f. Hodge theory (Rosenberg, 1997) implies that f has a unique spectrum based on the eigenfunctions of the Laplace-Beltrami operator on $\mathcal{M}$.

The graph Laplacian is a discrete form of the continuous Laplace-Beltrami operator that acts on a function $g : V \to \mathbb{R}$ defined on the graph.

$$Lg(u) = \frac{1}{\sqrt{d_u}} \sum_{v : v \sim u} \left(\frac{g(u)}{\sqrt{d_u}} - \frac{g(v)}{\sqrt{d_v}} \right) \ , \quad (2)$$

where $v \sim u$ denotes that vertices v and u are adjacent in G. Like the continuous f, g can also be characterized by the eigenfunctions of the Laplacian. The smoothness of g on the graph G is given by the Dirichlet sum $S(g, G) = \sum_{u,v} A_{u,v} \left(g(u) - g(v) \right)^2$.

4. The Model Transfer Graph

Given the set of background learning tasks $\{t_1, \ldots, t_n\}$, the first step is to construct the model transfer graph. We assume that sufficient training examples are given for each background task to learn models that have a high degree of performance. Let these learned base models be denoted $\{m_1, \ldots, m_n\}$, with m_i corresponding to task t_i.

We *could* directly plot the models in $\mathbb{R}^\theta$, since each model has a transferable θ-dimensional parameter vector v. However, this embedding ignores the fact that the transferred knowledge must *improve performance* on the target task. Similarity between two parameter vectors does not imply

that models using those vectors will have similar performance on a task. Therefore, it is important to measure similarity in the transfer space based on *transferability*; that is, the degree of similarity between two models should correlate with the degree to which the transferred knowledge improves learning performance on a task. (This transferability may not be symmetric; we discuss this issue in Section 4.2.)

4.1. Computing the Transferability between Tasks

We define transferability from task t_i to t_j as the change in performance on task t_j between learning with and without transfer from t_i's model. This definition is similar to the approach used by Thrun and O'Sullivan in their task clustering framework (1996). While their method simply looks at the change in performance for a specific number of training instances, we also examine the change in performance over the entire learning curve. Although we focus on this definition of transferability, the transfer graph method is general enough to use other measures of transferability.

Let $m(t, v, q)$ denote the model learned for task t using q training instances with transfer from parameter vector v, which may be null ($\emptyset$) for learning without transfer. Let $m_i = m(t_i, \emptyset, all)$ be the model learned on task t_i without transfer using all available data. Task t_i's base model m_i has an associated parameter vector v_i, which we can transfer to learn other tasks.

To measure the transferability from task t_i to task t_j, given q training instances, we first determine the baseline performance for task t_j without transfer. We learn $m_j^q = m(t_j, \emptyset, q)$ and evaluate this model on the testing data for task t_j to generate the baseline performance $P_j(q)$. We similarly determine the transfer performance by learning a model $m_{i\to j}^q = m(t_j, v_i, q)$ using transfer from task t_i, and evaluating the model on the testing data for task t_j, yielding transfer performance $P_{i\to j}(q)$. Any performance measure that evaluates to a real number can be used to compute the transferability (e.g., predictive accuracy, f-measure). In our experiments, we use predictive accuracy on the held-out test set to evaluate performance.

In an ideal transfer situation, the transferred information would immediately increase the performance of the learned model to the maximum possible performance. Our best estimate of the maximum possible performance is simply the maximum performance we have ever observed on task t_j, across all baselines and transfer situations using all possible amounts of training data, denoted $P_{max}(t_j)$. The ideal increase due to transfer would therefore be the difference between $P_{max}(t_j)$ and $P_j(q)$.

We compute the transfer from task t_i to task t_j with q training instances to be the ratio of the actual amount of transfer to the ideal amount of transfer:

$$transfer_{i\to j}(q) = \frac{P_{i\to j}(q) - P_j(q)}{P_{max}(t_j) - P_j(q)} \quad . \tag{3}$$

This ratio forces positive transfer to be in $[0, 1]$. Negative transfer—which occurs when transfer decreases the performance from the baseline—can fall outside the range $[-1, 0]$; however, we consider only positive transfer in constructing the transfer graph, which eliminates this problem.

This definition of transferability for a given amount of training data lends itself to a definition of overall transferability for a given range of training set sizes. In particular, we can integrate and average Equation 3 over a range of values for q, yielding a single overall measure of transferability. By considering $transfer_{i\to j}$ across the entire learning curve, we compute the average amount of transfer expected for an arbitrary amount of training data. This computation assumes a uniform probability distribution over the amount of training data that will be available for a new task; it is a simple matter to scale this computation for a non-uniform probability distribution.

To compute the overall transferability, we generate the baseline learning curve without transfer, $C_j = \{P_j(q)\}$, and the transfer learning curve, $C_{i\to j} = \{P_{i\to j}(q)\}$, varying the value of q over all available training data. Ideally, the increment between successive q's should be very small; in the experiments, we generate a sampled form of the learning curve by varying q from 5% to 100% of the available training data using 5% increments.

We then measure the area $A_{i\to j}$ between these two paired curves, counting areas formed when $C_{i\to j}$ is above the baseline C_j as positive, and areas formed when $C_{i\to j}$ is below C_j as negative. The ideal area of transfer A_{ideal} is then the area between the baseline C_j and the line $P_{max}(t_j)$. Finally, we then take the ratio of $A_{i\to j}$ to the ideal amount of transfer A_{ideal} to compute the transferability. Figure 1 illustrates this step. This computation has the benefit of being invariant to the learning curve's x-axis scale.

4.2. Constructing the Transfer Graph

To ensure that our similarity metric is symmetric, we define the undirected transfer similarity between tasks t_i and t_j to be the minimum of the two directed transferabilities: $transfer_{i,j}(q) = \min\big(transfer_{i\to j}(q), transfer_{j\to i}(q)\big)$. The largest potential problem is overestimating the amount of transfer between two tasks, and using the minimum of the directed transferabilites ensures that our estimate of the transfer is as large as possible without being a potential overestimation. Using other forms of symmetrization, such as taking the average or maximum, could lead to overestimation. While this construction underestimates the amount of transfer, we show empirically that it performs well.

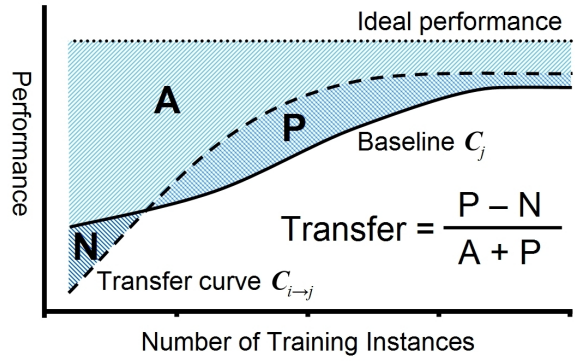


Figure 1. Graphical depiction of computing the transfer between two tasks t_i and t_j from learning curves.

We define the vertices of the model transfer graph to be the source tasks and their associated models $V = \{(t_i, m_i)\}_{i=1}^{n}$. We can construct the symmetric adjacency matrix A for the transfer graph for a given amount of training data q as

$$A_{i,j} = \begin{cases} 0 & \text{if } i = j, \\ \max\left(0, \mathit{transfer}_{i,j}(q)\right) & \text{otherwise.} \end{cases} , \quad (4)$$

providing us with a complete definition of the model transfer graph given a set of source tasks and their associated learned models. Since we need only model the positive transfer, this construction eliminates all negative edges from A. The known portion of the model transfer space is then given by the weighted graph $G = (V, A)$ with the edge weights specified in A.

5. Transfer to a Target Task

Once the model transfer graph $G = (V, A)$ has been constructed, transfer to a new target task t_{n+1} involves extending G to include t_{n+1}. Each vertex in the model transfer graph has an associated θ-dimensional parameter vector. We estimate the parameter vector for t_{n+1}'s model from the other models' parameters. This process is equivalent to interpolating the position of t_{n+1} on the manifold that models the transferability, and then determining the transfer function's value at that point.

We assume that there is some underlying hidden function $f : V \to \mathbb{R}^\theta$ that governs the assignment of the parameters to each vertex. Transfer to a target task involves determining the target model's location in the model transfer space, and then determining the parameters for the target model based on the parameters of the other models in the space. In other words, we are attempting to determine the parameter values that f would assign to the target model.

This approach requires that f be smooth over some surface, so we interpret the models as lying on some high-dimensional manifold, represented by the model transfer graph. The smooth function f acts on this manifold; therefore, we can characterize f based on the eigenfunctions of the model transfer graph's Laplacian L.

5.1. Extending the Transfer Graph

Given a small sample of the data from t_{n+1} (much less data than was given for any other task $t_1 \ldots t_n$), we approximate the model m_{n+1}'s location in the graph by computing its transferability from every other task t_i: $\mathit{transfer}_{i\to n+1}$. This yields a set of weighted edges[1] between m_{n+1} and all other models $m_1 \ldots m_n$, allowing us to localize m_{n+1} in the transfer graph. Let these weights be $\hat{w}_1 \ldots \hat{w}_n$.

The extended transfer graph that includes task t_{n+1} can now be defined by $\hat{G} = (\hat{V}, \hat{A})$, where $\hat{V} = V \bigcup \{t_{n+1}\}$ and $\hat{A}$ is the $(n+1) \times (n+1)$ extended adjacency matrix

$$\hat{A} = \begin{bmatrix} A & \hat{w}^T \\ \hat{w} & 0 \end{bmatrix} .$$

We then form the graph Laplacian $\hat{L}$ of $\hat{G}$, and take the eigenvectors $\hat{Q}$ of the Laplacian as a set of basis vectors over the extended graph.

For very large transfer graphs, the new eigenvectors could be computed efficiently using the Nyström method (Baker, 1977; Fowlkes et al., 2004; Drineas & Mahoney, 2005) to extend L's eigenvectors to the new task t_{n+1}, based on the edge weights $\hat{w}$. The Nyström extension gives:

$$q_i(t_{n+1}) = \frac{1}{\lambda_i} \sum_{j=1}^{n} \hat{w}_j \, q_i(t_j) \ , \quad (5)$$

where $q_i(t_j)$ is the i^{th} eigenvector applied to model t_j. However, the transfer graphs used in the experiments were small enough that we could directly compute the eigendecomposition of $\hat{L}$.

5.2. Determining the Transferred Parameters

We assumed earlier that there was a function $f : V \to \mathbb{R}^\theta$ that governs the assignment of parameter vectors to models in the transfer graph. To determine the parameter vector for the new model t_{n+1}, we extend f to form $\hat{f} : \hat{V} \to \mathbb{R}^\theta$.

The eigenvectors $\hat{Q}$ form an orthonormal basis for the set of all functions on $\hat{G}$; therefore, $\hat{f} = \hat{Q}W$ for some $(n+1) \times \theta$ matrix W. Using the known parameter vectors $v_1 \ldots v_n$ as samples of the function values on the graph, we fit W using least-squares. This makes $\hat{f}$ an approximation of f at the known sample points on the graph $t_1 \ldots t_n$, and a smoothed interpolant for it at t_{n+1}.

[1] The edges are also directed, although we ignore directionality, since the transfer is one-way only.

Let $\tilde{Q}$ be the rows of $\hat{Q}$ corresponding to the sampled vertices (in this case, $\tilde{Q} = \hat{Q}_{1...n,*}$). We fit each column of W separately using regularized least-squares by solving:

$$W_{*,i} = \arg_w \min \left|\left| f_{*,i} - \tilde{Q}w \right|\right|^2 + \left|\left| \sqrt{\hat{\Lambda}} w \right|\right|^2 \quad , \quad (6)$$

where $\sqrt{\hat{\Lambda}}$ serves as the regularization operator in this Tikhonov regularization problem. The regularization in this case acts as a weighted penalty on the average second-derivative of the function, enforcing smoothness by scaling each eigenvector weight by its corresponding eigenvalue λ_i, thereby increasing the regularization on higher-order eigenvectors to prevent overfitting with the high-frequency components.

We derive this expression by constraining the smoothness of $\hat{f}$—the L2 norm of the gradient of $\hat{f}$, given by $\langle \nabla \hat{f}, \nabla \hat{f} \rangle$:

$$\begin{aligned} \langle \nabla \hat{f}, \nabla \hat{f} \rangle &= \langle \hat{f}, \hat{L}\hat{f} \rangle \\ &= (\hat{Q}w)^T (\hat{L}\hat{Q}w) \\ &= w^T \hat{Q}^T (\hat{Q}\hat{\Lambda}\hat{Q}^T \hat{Q} w) \\ &= w^T I \hat{\Lambda} I w \\ &= w^T \hat{\Lambda} w \quad . \end{aligned}$$

Therefore, we can constrain the smoothness of $\hat{f}$ by penalizing the least-squares problem with $w^T \hat{\Lambda} w$, which is equivalent to the penalty $\left|\left| \sqrt{\hat{\Lambda}} w \right|\right|^2$ in Equation 6. The solution to this least-squares problem is given by

$$W_{*,i} = \left(\tilde{Q}^T \tilde{Q} + \hat{\Lambda} \right)^{-1} \tilde{Q}^T f_{*,i} \quad . \quad (7)$$

Once we have the least-squares estimate for W, the transferred parameters are given by $v_{n+1} = \hat{Q}_{n+1,*} W$, which we then use in learning the model for task t_{n+1}.

6. Transfer using Biased Logistic Regression

We use a biased form of logistic regression as the base learning algorithm in the experiments. Biased logistic regression penalizes deviations from a given weight vector, effectively biasing the learned model toward the transferred parameters.

The well known logistic regression model gives the probability of a data instance x having a binary label y as:

$$P(y = 1|x) = \frac{\exp(x\beta)}{1 + exp(x\beta)} \quad , \quad (8)$$

$$P(y = 0|x) = 1 - P(y = 1|x) \quad , \quad (9)$$

where $x \in \mathbb{R}^d$ and $\beta \in \mathbb{R}^d$. The parameter vector β is obtained in standard logistic regression by maximizing the log-likelihood of the labeled training data $\{(x_i, y_i)\}_{i=1}^n$:

$$\begin{aligned} l(\beta) = \sum_{i=1}^{n} [y_i \log P(y_i = 1|x_i) + \\ (1 - y_i) \log P(y_i = 0|x_i)] \quad . \end{aligned} \quad (10)$$

Combining ridge estimation with logistic regression[2] (Duffy & Santner, 1989; Le Cessie & Van Houwelingen, 1992) adds a penalty on the norm of β, and involves choosing β to maximize the penalized log-likelihood $l^\lambda(\beta) = l(\beta) - \lambda\|\beta\|^2$, where λ is the ridge parameter that controls the shrinkage of the norm $\|\beta\| = \sqrt{\sum_j \beta_j^2}$. Inspired by biased regularization of support vector machines (Kienzle & Chellapilla, 2006; Schölkopf & Smola, 2002) and the logistic regression transfer method of Marx et al. (2005), we penalize deviations of β from a given vector β_0:

$$l^\lambda(\beta) = l(\beta) - \lambda\|\beta - \beta_0\|^2 \quad . \quad (11)$$

Standard (non-biased) logistic regression corresponds to β_0 as the zero vector. This bias vector β_0 can be transferred from the learned β of another logistic regression model, allowing one logistic regression model to be biased toward the parameters of another model. Note that we do not transfer the constant term, allowing it to be fit individually to each problem.

When $\lambda = 0$, the bias term disappears and does not affect the learned weights; as $\lambda \to \infty$, the logistic regression learned weights approach the bias weights. In the experiments, we use the Bayesian-optimal λ, making two assumptions about the model (Hastie et al., 2001). First, we assume that the errors $\{y_i - p(x_i)\}_{i=1}^n$ are normally distributed $N(0, \sigma^2)$ with variance σ^2. Second, we assume that the parameters in β are independent and normally distributed $N(\beta_0, \tau^2)$ with mean β_0 and variance τ^2. Under these assumptions, the Bayesian-optimal lambda is given by $\lambda = \frac{\sigma^2}{\tau^2}$ (Hastie et al., 2001). Viewed from the perspective of transfer, this assumption implies a normal probability distribution over the transfer from β_0 to β.

The logistic regression transfer method proposed by Marx et al. (2005) uses a similar construction, in which they penalize the model parameters for deviating from a given set of normal distributions, considering both means and variances that were derived from the transferred parameter vectors. The method we use here (based on ridge estimation) corresponds to their method using a constant variance for all parameters, which is absorbed into λ. The major problem with using their method in this framework is that it is dependent on having a *set* of source tasks from

[2] We use the Weka machine learning toolkit's implementation of this method (Witten & Frank, 2000).

which to estimate the parameter variances and thereby the regularization parameters; in this application, we have only one source parameter vector and, therefore, no variance.

7. Evaluation

Our experiments examine transfer in two domains: letter and newsgroup recognition. The Letters data set (Asuncion & Newman, 2007) characterizes various fonts of each character using 16 features. The Newsgroup experiments use the 20 newsgroups data set (Rennie, 2003), characterized by a binary vector of the 100 most discriminating words as determined by Weka's string-to-wordvector filter (Witten & Frank, 2000). Both original data sets are very large, so we randomly selected five percent of each data set to use in the experiments.

For Letters, we took the first 13 letters (A–M) and generated 13 binary problems of each of these letters against the last 13 letters (N–Z). For example, the task of recognizing the letter A used "A"s as positive examples and all letters N–Z as negative examples. We chose this construction to yield tasks that would interfere as little as possible with each other. For example, if instead we had converted this data set into 26 one-versus-rest classification problems, there would be interference between the tasks, as one task's positive examples would appear as other tasks' negative examples, diminishing the possibility of transfer. The Newsgroups tasks are constructed similarly, using the first newsgroup in each major category as negative examples[3] and the 13 remaining newsgroups as positive examples. Additionally, we resampled the data with replacement to ensure that the class priors were approximately equal, and normalized the feature values to lie in $[0, 1]$.

We constructed model transfer graphs for both Letters and Newsgroups over 10 trials of 10-fold cross-validation over all available data on the source tasks, excluding the target task from the computations. The held-out fold was used for performance evaluation to generate the baseline and transfer learning curves. The base parameter vectors for each task were computed from all available data.

For each target task, we used 20 percent of the data for training and the remainder for testing. We used the training data to map the task to the transfer graph (again, computing the transfer over 10 trials of 10-fold cross-validation on the training data), computed the transfer function on the extended transfer graph, and then used that function to estimate the parameter vector to transfer to the target task. Then, the learned classifier was evaluated on the test data. This procedure was repeated for 50 trials.

[3]The negative newsgroups are alt.atheism, comp.graphics, misc.forsale, rec.autos, sci.crypt, soc.religion.christian, and talk.politics.guns.

Target task	Number of instances	Portion of relevant source tasks
sci.space	360	1 / 12
talk.politics.mideast	365	2 / 12
comp.windows.x	360	6 / 12
sci.med	370	8 / 12
"J"	550	1 / 12
"H"	550	11 / 12

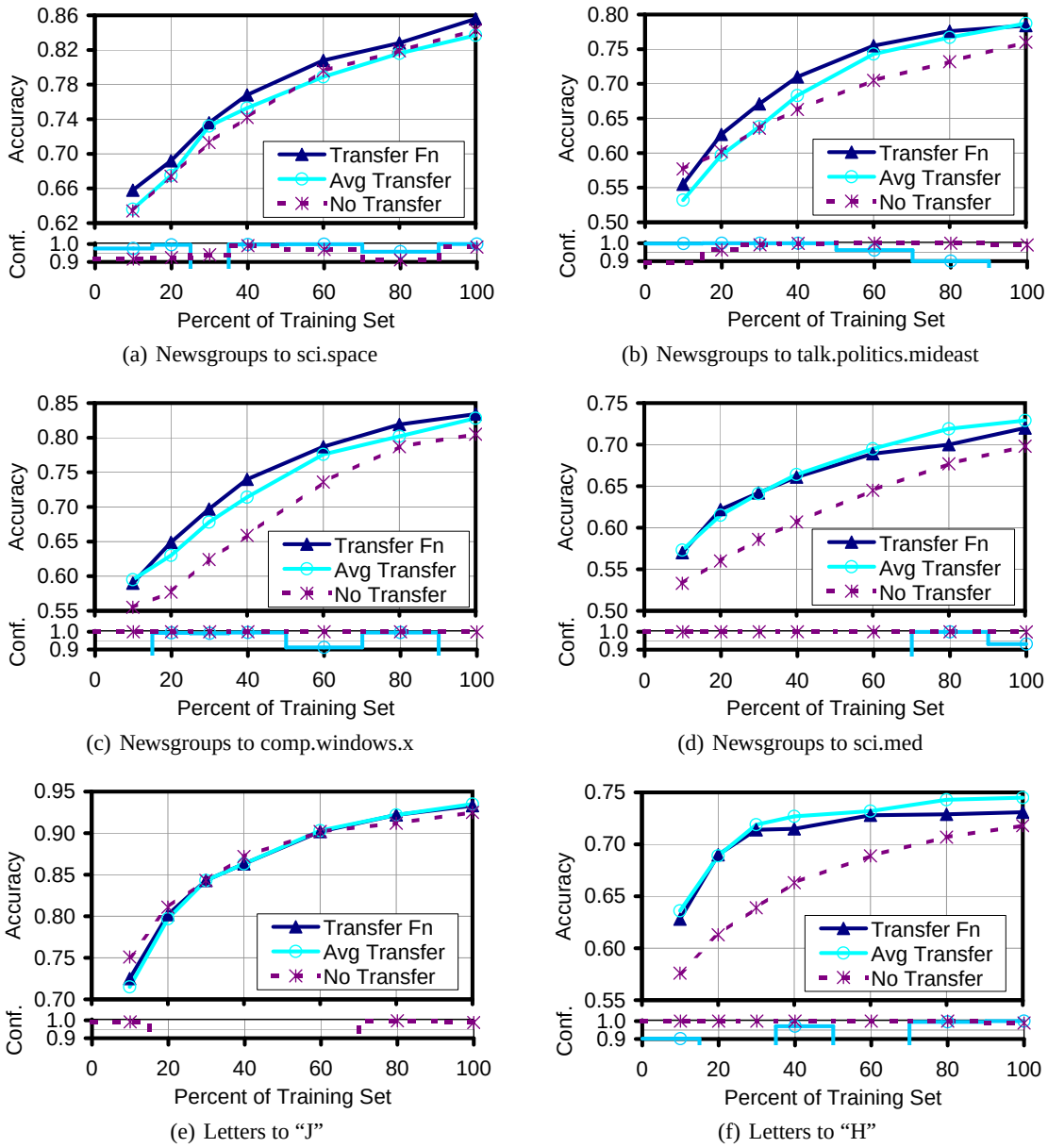
Table 1. Summary of transfer scenarios.

For every possible transfer scenario, we determined the number of relevant source tasks by generating the complete model transfer graph for all tasks in that domain (including the target tasks), and examining the (positive) incoming edges to each target task. We then chose the specific transfer scenarios for these experiments based on the ratio of relevant to irrelevant source tasks. Table 1 summarizes each transfer scenario used in the experiments.

Figure 2 compares the performance of the graph transfer function against an "average" transfer method, and against the baseline of learning without transfer. The average transfer method simply averages the parameter vectors from all source tasks (including irrelevant tasks) and transfers that average parameter vector to the target task.

Figures 2(a)–2(c) depict transfer scenarios with a mix of relevant and irrelevant source tasks. In these scenarios, the graph transfer method shows statistically significant (with at least 95% confidence) performance improvement over average parameter transfer, demonstrating our method's ability to focus on information from relevant source tasks. These results support the use of localized estimates for the transfer parameters for some problems, unlike the work of Marx et al. (2005) and Kienzle et al. (2006), which use estimates based on all the source problems (corresponding to the average transfer method).

The transfer scenarios of Figures 2(d) and 2(f) have few irrelevant source tasks, and in these cases, we see that transferring the average parameter vector works quite well. However, these results could easily be skewed by providing more irrelevant source tasks. In such a case, we hypothesize that the performance of the average parameter transfer would decrease, but that the graph transfer method would isolate the irrelevant tasks and perform equally well. Several data points in these figures show the average parameter vector outperforming the graph transfer method. This indicates that in situations where there is a majority of relevant source tasks, it may be better to transfer the average parameter vector. However, in situations where the relevance of source tasks is questionable or unknown, the graph transfer method might be a better choice, since it will automatically exclude the irrelevant source tasks.



(a) Newsgroups to sci.space

(b) Newsgroups to talk.politics.mideast

(c) Newsgroups to comp.windows.x

(d) Newsgroups to sci.med

(e) Letters to "J"

(f) Letters to "H"

Figure 2. Results on the Newsgroups and Letters transfer tasks. The top portion of each graph compares the learning curves of the graph transfer function, transfer using the average parameter vector from all source tasks, and learning without transfer. The bottom portion of each graph depicts the confidence level by which the transfer function's performance is statistically different from each of the other two methods as measured by a pairwise t-test. Typically, a difference with a confidence of 0.95 or above is considered statistically significant.

Figure 2(e) shows one scenario where we were unable to obtain improvement over the average parameter vector, despite the large number of irrelevant tasks. When we examined the model transfer graph for Letters, there were no tasks that were highly transferable to "J," unlike in the other transfer scenarios. The one task relevant to "J" had very low positive transfer. The lack of improvement from transfer is most likely due to this lack of relevant source tasks.

In two of the transfer scenarios—Figures 2(b) and 2(e)—learning without transfer outperforms learning with transfer when given very little training data. The biased logistic regression algorithm we used in these experiments relies on the training data to determine the amount of transfer from the given parameter vector. When given very little training data, the learning algorithm's estimation of the ideal amount of transfer may be inaccurate, and so it is outperformed by learning without transfer. It may also be the case that the graph transfer method is unable to accurately localize the target task in the model transfer graph given such little data. In any case, these hindrances disappear with the addition of slightly more training data.

8. Conclusion and Future Work

This paper describes a novel method for inductive transfer using a function on the transfer graph. As shown by our results, using localized estimates of the transfer values results in superior performance on some problems. The shortcut of always using the average parameter vector works well when all of the source tasks are relevant for transfer to the target task, but this involves expensive hand-selection of the source tasks. Additionally, hand-selection relies on qualitative (and sometimes incorrect) judgments that the selected tasks will transfer well to the target task.

We are exploring several extensions to our method. In this paper, we required transferability to be symmetric between two models. However, it has been our experience that often $transfer_{i \to j} \neq transfer_{j \to i}$, showing that transfer is not always symmetrical in practice. We plan to extend our method to support directed edges in the transfer graph. Techniques for spectral analysis of directed graphs have only been recently developed (Chung, 2005) and using them in this framework presents significant technical challenges that we leave to future work. Additionally, we are conducting a more extensive evaluation of this method, including applying it to other domains, such as image recognition.

Acknowledgments

This work was supported in part by NSF ITR #0325329. We thank Adam Anthony, Blazej Bulka, and Tim Oates for feedback on this work, and Josh Neil, Eduardo Corona, and Curtis Storlie for discussions on regularization.

References

Asuncion, A., & Newman, D. (2007). UCI machine learning repository.

Baker, C. T. H. (1977). *The numerical treatment of integral equations*. Oxford: Clarendon Press.

Bakker, B., & Heskes, T. (2003). Task clustering and gating for Bayesian multitask learning. *Machine Learning Research*, *4*, 83–99.

Chung, F. (2005). Laplacians and the Cheeger inequality for directed graphs. *Annals of Combinatorics*, *9*, 1–19.

Chung, F. R. K. (1994). *Spectral graph theory*. No. 92 in CBMS Regional Conference Series in Mathematics. Providence, RI: American Mathematical Society.

Drineas, P., & Mahoney, M. W. (2005). On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, *6*, 2153–2175.

Duffy, D. E., & Santner, T. J. (1989). On the small sample properties of norm-restricted maximum likelihood estimators for logistic regression models. *Communications in Statistics: Theory and Methods*, *18*, 959–980.

Fowlkes, C., Belongie, S., Chung, F., & Malik, J. (2004). Spectral grouping using the Nyström method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *26*.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning: Data mining, inference, and prediction*. New York: Springer.

Jordan, M., & Jacobs, R. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, *6*, 181–214.

Kienzle, W., & Chellapilla, K. (2006). Personalized handwriting recognition via biased regularization. *Proceedings of the Twenty-Third International Conference on Machine Learning*. Pittsburgh, PA.

Le Cessie, S., & Van Houwelingen, J. C. (1992). Ridge estimators in logistic regression. *Applied Statistics*, *41*, 191–201.

Marx, Z., Rosenstein, M. T., Kaelbling, L. P., & Dietterich, T. G. (2005). Transfer learning with an ensemble of background tasks. *NIPS 2005 Workshop on Transfer Learning*. Whistler, BC, Canada.

Mitchell, T. M., & Thrun, S. B. (1996). Learning analytically and inductively. In *Mind matters: A tribute to Allen Newell*, 85–110. Lawrence Erlbaum Associates.

Pratt, L. Y. (1993). *Transferring previously learned backpropagation neural networks to new learning tasks*. Doctoral dissertation, Rutgers University.

Rennie, J. (2003). 20 Newsgroups data set, sorted by date. Available online at http://www.ai.mit.edu/~jrennie/20Newsgroups/.

Rosenberg, S. (1997). *The Laplacian on a Riemannian manifold*. Cambridge University Press.

Schölkopf, B., & Smola, A. J. (2002). *Learning with kernels*. MIT Press.

Thrun, S., & O'Sullivan, J. (1996). Discovering structure in multiple learning tasks: the TC algorithm. *Proceedings of the Thirteenth International Conference on Machine Learning* (pp. 489–497). Morgan Kaufmann.

Witten, I. H., & Frank, E. (2000). *Data mining: Practical machine learning tools with Java implementations*. San Francisco, CA: Morgan Kaufmann.