Modeling Transfer Relationships Between Learning Tasks for Improved Inductive Transfer*

Eric Eaton[1], Marie desJardins[1], and Terran Lane[2]

[1] University of Maryland Baltimore County,
Department of Computer Science and Electrical Engineering
{ericeaton,mariedj}@umbc.edu
[2] University of New Mexico, Department of Computer Science
terran@cs.unm.edu

Abstract. In this paper, we propose a novel graph-based method for knowledge transfer. We model the transfer relationships between source tasks by embedding the set of learned source models in a graph using transferability as the metric. Transfer to a new problem proceeds by mapping the problem into the graph, then learning a function on this graph that automatically determines the parameters to transfer to the new learning task. This method is analogous to inductive transfer along a manifold that captures the transfer relationships between the tasks. We demonstrate improved transfer performance using this method against existing approaches in several real-world domains.

1 Introduction

Knowledge transfer from previously learned tasks to a new task is a fundamental component of human learning. Transfer enables us to learn complex tasks quickly by automatically building on our previous knowledge. Recent research efforts have shown that transfer can also improve machine learning, enabling more rapid learning or higher levels of performance.

Most machine learning methods for transfer rely on an explicit set of *source* tasks to identify a set of model parameters that can be transferred to a new *target* task. In many cases, these source tasks are hand-selected by an expert in advance. Methods for transfer may combine information from all source tasks [1, 2] or may use information from only a few tasks chosen by an automated process [3]. Accidentally transferring from irrelevant source tasks may inhibit learning and decrease performance—a phenomenon known as *negative transfer*. Our approach to transfer explicitly models the transfer relationships between the source tasks to automatically avoid this problem and transfer only relevant information.

Given a set of source tasks and a target task, our method attempts to automatically determine the knowledge to transfer in learning the target task. In our formulation, this knowledge is a vector of model parameters. We estimate the transfer relationships between the source tasks and embed them into a graph, using a notion of *transferability* to determine the edge weights. This model transfer

* To appear in ECML-2008.

graph corresponds to a discrete approximation of a high-dimensional manifold that captures the transfer relationships between the source tasks. Tasks that are close on this manifold have high transferability; tasks that are far apart have low transferability. Each task has an associated vector of model parameters that represents the knowledge at its location.

Intuitively, each location on the transfer manifold has an ideal parameter vector that should be transferred in learning a task at that location. Therefore, we can determine the knowledge to transfer to a target task by approximating the parameter vector at the target task's location on the manifold. Given a new target task, we first extend the the graph to include the new task. We define a function on the graph that determines the parameters to transfer to each location in the graph. By its construction, this transfer function respects the local geometry of the graph and, therefore, the transfer relationships among the source tasks. We learn the transfer function using the source tasks' parameters as samples of the function at various locations on the transfer manifold. Then, we evaluate the function at the new task's location to yield the parameter vector to transfer in learning the new task. We also define a reusable form of the transfer function that can be used for multiple transfer scenarios without relearning.

2 Related Work

Parameter-based transfer has been used by Marx et al. [1] to learn logistic regression models. 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 & Chellapilla [2] 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 Sect. 3 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 & Chellapilla, which combine knowledge from all given source tasks for transfer, Thrun & O'Sullivan's Task Clustering (TC) algorithm [3, 4] 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. Their approach optimizes a single distance metric for each cluster, effectively determining an average parameter vector for each cluster of tasks. 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. Our approach is similar to Thrun & O'Sullivan's in determining the transfer relationships between tasks. However, the TC algorithm transfers only a single parameter vector to all tasks in a cluster, while our flexible transfer function allows each location on the transfer surface to have a different parameter vector based on the local geometry.

Bakker & Heskes [5] take a Bayesian approach to clustering tasks, using EM to optimize the clusters. They also use a gating network, similar to that used in

the mixture-of-experts model [6], on top of the Bayesian EM framework to allow the priors to vary depending on the task's features. Pratt's Discriminability-Based Transfer method [7] 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 [8] use a more indirect approach to parameter transfer, using extracted invariances about a domain to bias the learning of model parameters. These approaches allow some of the model parameters to be dependent on the source tasks, while others are fit to the target task's data. In our approach, we allow all of the transferred model parameters to be modified in the final learned model, if the target task's data warrants such an adjustment.

3 Transfer Using Biased Logistic Regression

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 X to the same Y. The goal for learning the model for a task is to recover the true mapping $X \rightarrow Y$ from the labeled training data. Each learned model can be characterized as a vector of parameters, which can be transferred in learning a model for another task.

Our approach requires a base transfer learning algorithm to learn the models for each task. We use a biased form of logistic regression as the base learning algorithm in the experiments. Biased logistic regression penalizes deviations from a given parameter vector in $\mathbb{R}^d$, effectively biasing the learned model toward the transferred parameters. While we focus on this transfer learning algorithm, our method can utilize other parameter-based transfer learning algorithms.

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

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

$$P(y = 0|x) = 1 - P(y = 1|x) \ , \tag{2}$$

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

$$l(\beta) = \sum_{i=1}^{q} \left[y_i \log P(y_i = 1|x_i) + (1 - y_i) \log P(y_i = 0|x_i) \right] \ . \tag{3}$$

Combining ridge estimation with logistic regression[3] [10, 11] 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 $\|\beta\| = \sqrt{\sum_j \beta_j^2}$.

To use logistic regression for transfer, we penalize deviations of β from a given transferred vector β_0:

$$l^\lambda(\beta) = l(\beta) - \lambda\|\beta - \beta_0\|^2 \ . \tag{4}$$

[3] We use the Weka machine learning toolkit's implementation of this method [9].

This approach is inspired by biased regularization of support vector machines [2, 12] and the logistic regression transfer method of Marx et al. [1].

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. In practice, x and β are often augmented to include a constant term for the intercept. 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.

We use the Bayesian-optimal $\lambda = \frac{\sigma^2}{\tau^2}$ [13], where σ^2 is the variance of the model's log-likelihood errors $\{-\log P(y = y_i|x_i)\}_{i=1}^{q}$, and τ^2 is the variance of the elements of $(\beta - \beta_0)$. 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 of Marx et al. [1] uses a similar construction, in which they penalize deviations of the model parameters from a given set of normal distributions, considering both means and variances derived from the transferred parameter vectors. The method we use here (based on ridge regression) 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 application is that it is dependent on having a *set* of source tasks from which to estimate the parameter variances and thereby the regularization; in this application, we have only one source parameter vector and, therefore, no variance.

4 Modeling the Transferability Between Source Tasks

Given a set of source tasks $\{t_i\}_{i=1}^{n}$, our approach is composed of three steps:

- **Learn the base models** $\{m_i\}_{i=1}^{n}$ for the source tasks $\{t_i\}_{i=1}^{n}$ (Sect. 4.1).
- **Construct the model transfer graph** to model the transfer relationships between the source tasks (Sect. 4.2).
- **Transfer to a new task** t_{n+1} by extending the model transfer graph to include t_{n+1}, and then learning the transfer function f to determine the parameter vector v_{n+1} to transfer to t_{n+1} (Sect. 5).

4.1 Learning the Base Models

Given the set of source tasks $\{t_i\}_{i=1}^{n}$, our first step is to learn the set of base models $\{m_i\}_{i=1}^{n}$ for the source tasks. For a task t_i, we learn the corresponding model m_i using biased logistic regression without transfer, biasing the model parameters toward zero. We assume that sufficient training examples are given for each source task to learn base models that have a high degree of performance.

Each trained model m_i has an associated parameter vector $v_i \in \mathbb{R}^{\theta}$, which can be transferred in learning models for other tasks. For biased logistic regression models, $\theta = d$, with v_i corresponding to the learned β vector.

4.2 Constructing the Model Transfer Graph

Given the set of source tasks $\{t_i\}_{i=1}^n$ and their corresponding learned models $\{m_i\}_{i=1}^n$, we embed these tasks in a space that captures the transfer relationships between the tasks. Two tasks that have high transferability should be close in the space; tasks that have low or negative transferability should be far apart.

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. Although we focus on this definition of transferability, our approach is general enough to use other measures. This definition is very similar to the approach used by Thrun and O'Sullivan [3]. While their task clustering method simply looks at the change in performance for a specific number of training instances, we also consider the average transferability over the entire learning curve.

We model the space of transfer using a *model transfer graph*, with each task as a vertex in the graph. A pair of vertices are connected via an edge if they have positive transferability; this edge is weighted based on the amount of positive transferability between the tasks, which is in $(0, 1]$.

We *could* directly plot the models in $\mathbb{R}^\theta$, since each model can be characterized by its transferable parameter vector. However, this embedding ignores 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*.

The model transfer graph corresponds to a discrete approximation of the continuous transfer manifold, using transferability as the metric. The source tasks are known samples of various locations on the manifold, with each task t_i having an associated parameter vector v_i. Transfer to a new task, as described in Sect. 5, occurs by approximating the location of the new task on the manifold, then using the transfer function to determine the parameter vector to transfer.

Measuring transferability

We measure transferability from task t_i to t_j as the direct change in performance between learning with and without transfer. For a task t_j, we can generate two learning curves for the task's model: one for learning t_j's model with transfer from t_i, and one for learning the model without transfer. For learning with transfer, we use logistic regression biased toward the parameter vector v_i that characterizes t_i's base model m_i. For learning without transfer, we use standard regularized logistic regression, which is biased toward the zero vector. Any performance measure that evaluates to a real number in $[0, 1]$ can be used to compute the performance (e.g., predictive accuracy, f-measure). In our experiments, we use predictive accuracy on the held-out test set.

Let $\mathit{performance}_j(q)$ be the performance on task t_j without transfer given q training instances, and let $\mathit{performance}_{i \to j}(q)$ be the performance on task t_j with transfer from t_i given q training instances from task t_j. Then, the transferability

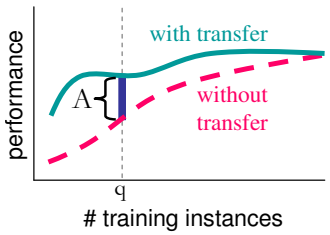


Fig. 1. The transferability measure: $transfer(q) = A$.

from task t_i to t_j is given by

$$transfer_{i \to j}(q) = performance_{i \to j}(q) - performance_{j}(q) \quad . \tag{5}$$

Note that $transfer_{i \to j} \in [-1, 1]$, with positive transfer falling in $(0, 1]$.

This definition of transferability for a given amount of training data lends itself to a definition of overall transferability from task t_i to t_j. Specifically, we can average Eqn. 5 over a range of values for q, yielding a measure of the overall transferability from t_i to t_j. By considering $transfer_{i \to j}$ across the entire learning curve, we compute the expected amount of transfer 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.

Defining the model transfer graph

The spectral graph analysis techniques we use to analyze the transfer surface (Sect. 5.2) rely on the model transfer graph being undirected. A symmetric affinity measure is the most natural representation for the transfer surface. However, transfer by its nature is directed from source knowledge to a target task. Therefore, $transfer_{i \to j}$ is *not* guaranteed to be the same as $transfer_{j \to i}$.

We define the symmetric undirected transferability between tasks t_i and t_j to be the minimum of the two directed transferabilities:

$$transfer_{i,j}(q) = \min\left(transfer_{i \to j}(q), transfer_{j \to i}(q)\right) \quad . \tag{6}$$

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 in Sect. 6.

We define the vertices of the model transfer graph to be the source tasks $V = \{t_i\}_{i=1}^{n}$, and the symmetric adjacency matrix A for q training instances as

$$A_{i,j}(q) = \begin{cases} 0 & \text{if } i = j, \\ \max\left(0, transfer_{i,j}(q)\right) & \text{otherwise.} \end{cases} \tag{7}$$

Since we need only model the positive transfer, this construction eliminates all negative edges from $A(q)$. We store multiple snapshots of the graph's adjacency matrix $\{A(q_i)\}_{q_1}^{q_k}$ at various numbers of training instances. In the experiments, we sampled the learning curve every five percent of the training data, so $k = 20$ with successive q_i's in 5% increments. To transfer to a new task, we select the current picture of the transferability space for the given number $\hat{q}$ of target task training instances, and use that version of the model transfer graph $G(\hat{q}) = (V, A(\hat{q}))$.

5 Transfer to a New Task

From Sect. 4, we can construct a model transfer graph to represent the transfer relationships among the source tasks. In this section, we describe a procedure for using the graph to determine the parameters to transfer to a new task.

Given $\hat{q}$ training instances of a new target task t_{n+1}, we can extend the model transfer graph $G(\hat{q})$ to include t_{n+1}. We then learn a transfer function on the extended graph to determine the parameter vector to transfer to the new task. This process is equivalent to interpolating the position of t_{n+1} on the transfer manifold, and then determining the transfer function's value at that point.

5.1 Extending the Model Transfer Graph

Given a small sample ($\hat{q}$ instances) of the data from t_{n+1} (much less data than was given for any other task $t_1 \ldots t_n$), we approximate task t_{n+1}'s location in the graph by computing its transferability from every other task t_i: $\{\mathit{transfer}_{i \to n+1}(\hat{q})\}_{i=1}^{n}$. This yields a set of weighted edges[4] between t_{n+1} and all other tasks $t_1 \ldots t_n$, allowing us to localize t_{n+1} in the transfer graph. Let these weights be $\hat{w}_1 \ldots \hat{w}_n$, where $\hat{w}_i = \mathit{transfer}_{i \to n+1}(\hat{q})$.

The extended model 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 given by

$$\hat{A} = \begin{bmatrix} A(\hat{q}) & \hat{w}^T \\ \hat{w} & 0 \end{bmatrix} \quad . \tag{8}$$

5.2 Learning the Transfer Function

Once the graph $G(\hat{q})$ has been extended to include the new target task, the next step is to learn the transfer function on $\hat{G}$ and use it to determine the knowledge to transfer in learning t_{n+1}. Each vertex i in the extended model transfer graph $\hat{G}$ has some associated transfer knowledge given by its parameter vector v_i. For the new target task t_{n+1}, this transfer knowledge is unknown, and the transfer function can estimate it automatically from the source tasks' parameter vectors.

The source tasks represent a known sample of the transfer surface, with the parameter vectors $\{v_i\}_{i=1}^{n}$ representing the transfer knowledge at these sample

[4] We ignore the directionality of the edges, since the transfer is one-way only.

locations on the manifold. Each parameter vector v_i is in $\mathbb{R}^\theta$. We assume that there is some function that determines the transfer knowledge for a task based on that task's location on the transfer surface. This *transfer function* $\hat{f} : \hat{V} \rightarrow \mathbb{R}^\theta$ is able to assign a parameter vector to each task located on the transfer surface.

The source tasks' parameter vectors $\{v_i\}_{i=1}^n$ represent known values of the transfer function $\hat{f}$ at various locations (given by the source tasks) on the transfer surface. Therefore, the source tasks' locations coupled with their parameter vectors provide training data for learning the transfer function $\hat{f}$. To transfer to a new task t_{n+1}, we can evaluate the learned transfer function at t_{n+1}'s location on the transfer surface to yield a parameter vector for t_{n+1}.

In order to ensure that the learned transfer function respects the transfer relationships between the tasks, we must model the transfer function in a manner that respects the model transfer graph's geometry. To do this, we define $\hat{f}$ using a set of basis functions for the graph determined by spectral graph theory.

Determining the Basis Functions

This section describes the spectral graph theory [14] techniques we use to derive the basis functions, which allow us to define a transfer function that will respect the geometry of the model transfer graph.

Let $G = (V, A)$ be the model transfer graph, which is an undirected connected weighted graph with a set of n vertices V and a weighted $n \times n$ adjacency matrix A. Recall that $A_{u,v} \neq 0$ implies that there is an edge depicting positive transferability between vertices u and v. We can denote the degree of vertex v by $d_v = \sum_{u=1}^n A_{u,v}$. Let T be the diagonal matrix where $T_{v,v} = d_v$.

Spectral graph theory allows us to define a set of basis functions for G based on the graph Laplacian, an operator defined by the Laplacian matrix. The combinatorial Laplacian matrix L for the graph is given by $L = T - A$ [14].

We can also define the *normalized Laplacian* $\mathcal{L} = I - T^{-\frac{1}{2}} A T^{-\frac{1}{2}}$, where I is the identity matrix [14]. 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.

The Laplacian L is symmetric; therefore, its eigenvalues are all real and non-negative. The eigendecomposition of L yields $L = Q \Lambda Q^T$, where Λ is the diagonal matrix of eigenvalues $[\lambda_1 \ \ldots \ \lambda_n]$ and the columns of Q are the eigenvectors $[q_1 \ \ldots \ q_n]$. Let $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ be the eigenvalues, and let eigenvector q_i correspond 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. The eigenvectors in Q form an orthonormal basis for L.

Spectral graph theory has connections to Riemannian manifolds, which we use to define the surface on which transfer occurs. The model transfer graph G represents a sample of the continuous transfer 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 (i.e., tasks that have high transferability).

Let g be a smooth function $g : \mathcal{M} \rightarrow \mathbb{R}^\theta$ representing the transfer function on the Riemannian transfer manifold $\mathcal{M}$ with Riemannian transferability metric ψ.

The Laplace-Beltrami operator Δ is defined to be the divergence of the gradient of $\mathcal{M}$, and can act on g. Hodge theory [15] implies that g has a unique spectrum based on the eigenfunctions of the Laplace-Beltrami operator on $\mathcal{M}$.

Since we have only a sample of the transfer manifold given by the source tasks, we work with a discrete form f of the true transfer function g. The graph Laplacian is a discrete form of the continuous Laplace-Beltrami operator that acts on a function $f : V \to \mathbb{R}^\theta$ defined on G. Like the continuous g, f can also be characterized by the eigenfunctions of the Laplacian; therefore, Q forms a set of basis functions which we can use to define the transfer function f.

While this analysis has focused on the original model transfer graph G, we can similarly analyze the extended graph $\hat{G}$. As with G, we can take the eigenvectors $\hat{Q}$ of $\hat{G}$'s graph Laplacian $\hat{L}$ to form a set of basis functions that can characterize the extended transfer function $\hat{f}$ on $\hat{G}$.

Modeling the transfer function

Using the basis functions for $\hat{G}$, we can model the transfer function $\hat{f} : \hat{V} \to \mathbb{R}^\theta$. 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.

We use the known parameter vectors $\{v_i\}_{i=1}^n$ as samples of the function values on the graph, defining $f = [v_1 \dots v_n]^T$, where the v_i's are column vectors. We can similarly define the basis vectors for these sample points as the corresponding rows of $\hat{Q}$: $Q = \hat{Q}_{1\dots n,*}$. The matrix Q is $n \times (n+1)$, and f is $n \times \theta$.

We fit each column of W separately using regularized least-squares by solving:

$$W_{*,i} = \arg_w \min ||f_{*,i} - Qw||^2 + \left|\left|\sqrt{\hat{\Lambda}}w\right|\right|^2 \quad , \tag{9}$$

where $\sqrt{\hat{\Lambda}}$ serves as the regularization operator in this Tikhonov regularization problem. The operator acts as a weighted penalty on the function's average second-derivative, enforcing smoothness by scaling each eigenvector's 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 $\hat{f}$—i.e., the L2 norm of the gradient of $\hat{f}$, given by $\langle \nabla \hat{f}, \nabla \hat{f} \rangle$:

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

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 $||\sqrt{\hat{\Lambda}}w||^2$ in Eqn. 9. The

solution to this least-squares problem is given by

$$W = \left(Q^T Q + \hat{\Lambda}\right)^{-1} Q^T f \ . \tag{10}$$

This process yields an $(n+1) \times \theta$ matrix for W, which can be used unaltered to form the extended transfer function $\hat{f} = \hat{Q}W$ that assigns a parameter vector to each vertex in $\hat{G}$. The extended transfer function $\hat{f}$ approximates the known parameter vectors $\{v_i\}_{i=1}^n$ at the source task vertices. At the new target task t_{n+1}, $\hat{f}$ acts as a smoothed interpolent of the source tasks' knowledge at t_{n+1}'s location, respecting the graph geometry and transferability relationships.

By the $\hat{f}$ transfer function, the transferred parameters for the target task are given by $v_{n+1} = \hat{Q}_{n+1,*}W$, where $\hat{Q}_{n+1,*}$ is the $(n+1)^{th}$ row of $\hat{Q}$. We then transfer v_{n+1} in learning t_{n+1}'s model.

5.3 Creating a Reusable Transfer Function

In the procedure we defined in Sect. 5.2, the transfer function must be relearned for each new target task based on the geometry of the extended model transfer graph. The transfer graphs used in the experiments were small enough that we could directly compute the eigendecomposition of $\hat{L}$. However, for very large transfer graphs or for repeated transfer scenarios, this process of recomputing the transfer function becomes a source of inefficiency. In this section, we describe a method for creating a reusable form of the transfer function.

First, we construct the model transfer graph G using the source tasks as described in Sect. 4. For G, we can construct the transfer function $f : V \to \mathbb{R}^\theta$ for the source tasks by solving the least-squares problem $f = QW$ for an $n \times \theta$ matrix W, where $f = [v_1 \dots v_n]^T$ and Q is the matrix of eigenvectors with eigenvalues Λ of G's graph Laplacian. The solution is given by $W = \left(Q^T Q + \Lambda\right)^{-1} Q^T f$.

While this f operates on G, applying it directly to the extended graph $\hat{G}$ would not work, because there would be more than n eigenvectors. However, we can use the Nyström method to extend G's eigenvectors to new vertices without increasing the number of eigenvectors, thereby allowing us to reuse the learned transfer function f for multiple transfer scenarios.

The Nyström method [16–18] allows us to efficiently extend a graph's eigenvectors to include a new vertex. Let $\{\hat{w}_i\}_{i=1}^n$ be the transferability edge weights between a new task t_{n+1} and all the vertices V of the original model transfer graph G. The Nyström extension allows us to extend the eigenvectors Q of G's graph Laplacian to approximate the eigenvector values at the new task t_{n+1} as

$$q_i(t_{n+1}) = \frac{1}{\lambda_i} \sum_{j=1}^{n} \hat{w}_j \, q_i(t_j) \ , \tag{11}$$

where $q_i(t_j)$ is the i^{th} eigenvector applied to task t_j.

Using these extended eigenvectors, we can form an approximation to the true eigenvectors $\hat{Q}$ of $\hat{G}$, the extended model transfer graph that includes t_{n+1}.

The Nyström approximation to the eigenvectors is given by $\tilde{Q}$, an $(n+1) \times n$ matrix. Since $\tilde{Q}$ and Q both contain n eigenvectors (recall that $\hat{Q}$ contained $n+1$ eigenvectors), we can approximate the transfer function $\hat{f}$ on $\hat{G}$ by $\tilde{f} = \tilde{Q}W$ using the same weight matrix W without relearning. Then, for the new task t_{n+1}, the transferred parameter vectors are given by $v_{n+1} = \tilde{Q}_{n+1,*}W$.

6 Evaluation

Our experiments examine transfer in two domains: letter and newsgroup recognition. The Letters data set [19] characterizes various fonts of each character using 16 features normalized to lie in $[0, 1]$. The Newsgroup experiments use the 20 newsgroups data [20], characterized by a binary vector of the 100 most discriminating words, as determined by Weka's string-to-wordvector filter [9]. Transfer is often not useful given large amounts of data, since there would be enough data to learn a model with high performance. Both original data sets are very large, so we randomly selected five percent of each to use in the experiments.

For Letters, we took the first 13 letters (A–M) and generated 13 binary tasks of each of these letters against the last 13 letters (N–Z), ensuring that each task had unique negative examples and equal class proportions. For example, the task of recognizing the letter C used 35 "C"s as positive examples and 35 random 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, since 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[5] in each major category as negative examples for the tasks given by the 13 remaining newsgroups.

The base models for each task were learned from all available data. We then 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.

For each target task, we used the task's training data to extend the transfer graph (again, computing the transfer over 10 trials of 10-fold cross-validation on the training data), learned the transfer function on the extended graph, and then used it to estimate the parameter vector for the target task. We evaluated the learned classifier with transfer on the task's held-out test data. This procedure was repeated and averaged over 20 trials of 10-fold cross-validation to generate the learning curves. Table 1 summarizes each transfer scenario used in the experiments.

Figures 2 and 3 compare the performance of our "graph transfer" approach against "hand-selected" transfer, an "average" transfer method, and the baseline

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

Table 1. Summary of transfer scenarios.

20 Newsgroups		
Target task	**# instances**	**"Hand-selected" source tasks**
comp.windows.x	100	comp.os.ms-windows.misc, comp.sys.ibm.pc.hardware, comp.sys.mac.hardware
rec.sport.baseball	100	rec.motorcycles, rec.sport.hockey
sci.space	100	sci.electronics, sci.med
talk.politics.mideast	94	talk.politics.misc, talk.religion.misc

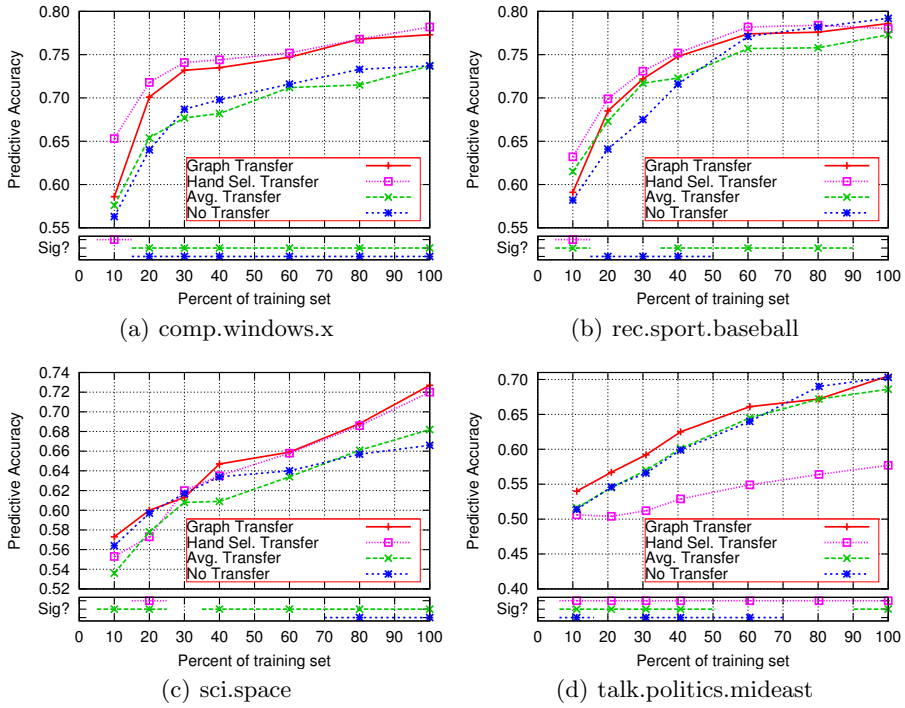
Letters		
Target task	**# instances**	**"Hand-selected" source tasks**
C	70	E, G
E	88	B, F
G	70	C
H	70	K, M
J	72	I, L
L	68	I, J

of learning without transfer. The "hand-selected" transfer computes the average parameter vector over each target task's related source tasks given in Table 1. For Newsgroups, these related tasks were chosen as the other newsgroups with the same top-level category; the related Letters tasks were chosen based on visual similarity between the letters. The average transfer method simply averages the parameter vectors from all source tasks (including irrelevant tasks), corresponding to several current transfer approaches [1, 2].

All of the Newsgroup transfer scenarios contain a mix of both relevant and irrelevant source tasks. The comp.windows.x task (Fig. 2(a)) has a higher proportion of relevant source tasks than the other Newsgroup scenarios, due both to the larger proportion of computer-related newsgroups and the broad applicability of computers to other subjects.

Our graph transfer method shows statistically significant improvement (with at least 95% confidence) over the average parameter vector on the Newsgroup tasks, demonstrating its ability to focus on information from relevant source tasks. The inclusion of irrelevant source tasks in computing the average parameter vector sometimes results in negative transfer, which our graph transfer avoids. These results support the use of localized estimates for the transfer parameters instead of averaging information from all source tasks without regard to transferability. It also shows that our approach can achieve performance near that of expensive "hand-selected" source tasks; in many cases, the performance of the graph transfer and hand-selected transfer are statistically indistinguishable.

Results on the letter transfer scenarios in Figs. 3(a)–(c) mirror the successes of our approach on the Newsgroup tasks. The letter-L transfer scenario (Fig. 3(d)) shows one case where we were unable to obtain clear improvement over the average parameter vector, although in many cases its increased performance over graph transfer is not statistically significant. This scenario also shows



(a) comp.windows.x (b) rec.sport.baseball

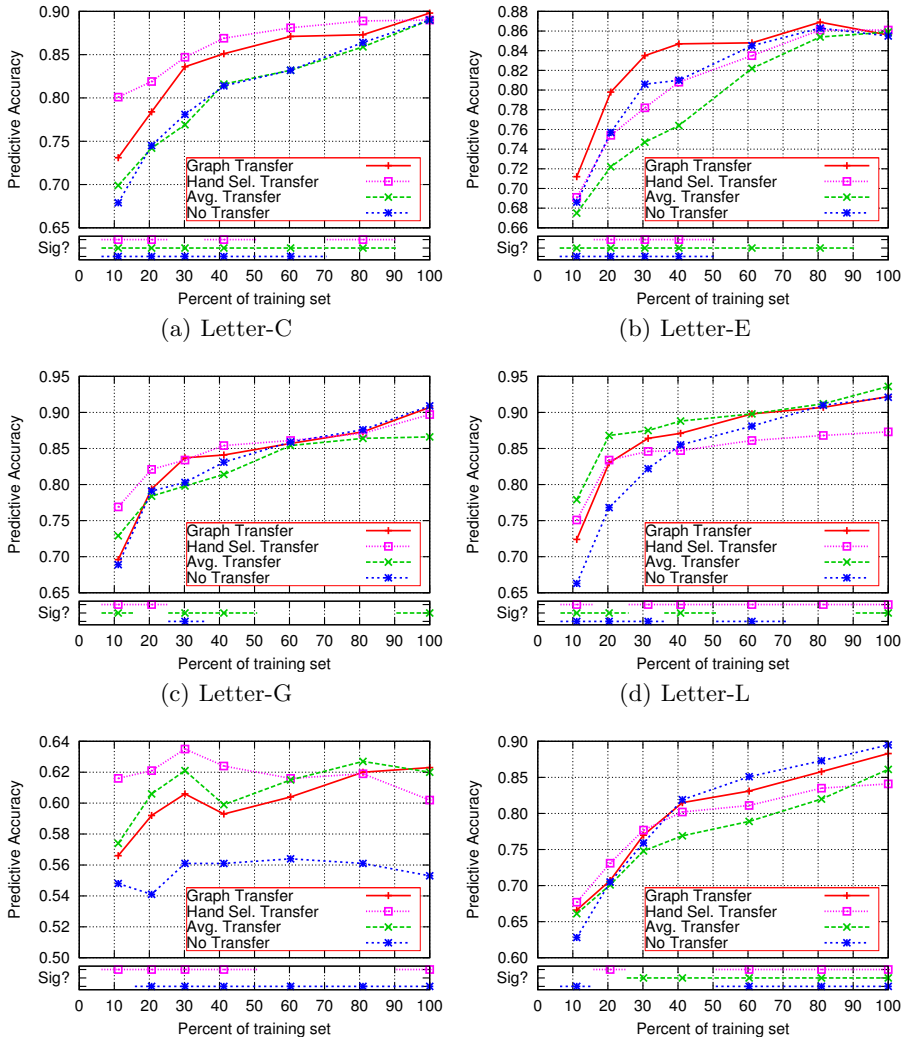
(c) sci.space (d) talk.politics.mideast

Fig. 2. Results of the Newsgroups transfer scenarios. The bottom portion of each graph depicts the range where our graph transfer approach's performance is statistically different at 95% confidence from each of the other three methods as measured by a pairwise t-test.

the pitfalls of hand-selecting source tasks, in this case based on visual similarities between the letters, in that these hand-selected tasks can unexpectedly result in negative transfer.

Figures 3(e)–(f) depict two extreme transfer scenarios that demonstrate the versatility of our approach. The complete model transfer graph for the letters domain showed that all letters had positive transfer (on average) to the letter-H task, with the exception of the letter-B task showing very slight negative transfer. For the letter-H scenario with all relevant source tasks (Fig. 3(e)), graph transfer achieves performance that is statistically indistinguishable from the average parameter transfer, correctly combining information from all source tasks. The transfer scenario in Fig. 3(f) depicts the opposite transfer scenario, with only one source task showing very slight positive transfer to the letter-J task. In this case, it is clear that any transfer would decrease performance, and our graph transfer method shows the best performance of all the transfer methods.

In working with our graph transfer approach, we did observe a few cases where learning without transfer outperformed learning with transfer when given



(a) Letter-C (b) Letter-E

(c) Letter-G (d) Letter-L

(e) All Relevant Source Tasks:Letter-H (f) No Relevant Source Tasks:Letter-J

Fig. 3. Results of the Letter transfer scenarios. Figures (e) and (f) depict extreme transfer scenarios: (e) all relevant source tasks on a difficult problem, and (f) no relevant source tasks. For explanation of the lower significance graphs, see the caption to Fig. 2.

very little training data. However, these situations disappeared when averaged over many trials and folds, as shown in our results. Biased logistic regression relies on the training data to determine the amount of regularization 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, so it could be outperformed by learning without transfer. It may also be the case that the graph

transfer method was occasionally unable to accurately localize the target task in the model transfer graph given very little data. In any case, these hindrances disappeared with the addition of slightly more training data.

We also explored a second transferability measure that was a normalized form of Eqn. 5. This measure defined transferability as the percentage improvement due to transfer against the best possible improvement. In an ideal transfer situation, the learned model's performance would immediately increase to the maximum possible performance, which may be less than 1 due to noise in the data. The percentage improvement due to transfer would then be the ratio of Eqn. 5 to the maximum possible improvement. Using this normalized transferability measure yielded similar results to those we report here, so we omit these results due to space limitations. In some cases, the normalized transferability measure performed slightly worse than the *un*normalized measure. However, a more thorough analysis involving other transfer scenarios is required to conclude whether the unnormalized measure we use in this paper is truly better.

7 Conclusion and Future Work

This paper describes a novel method for inductive transfer based on modeling the transfer relationships between the source tasks. As shown by our results, using localized estimates of the transfer values results in superior performance on most 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 approach. In this paper, we required transferability to be symmetric between two tasks. However, it has been our experience that often $transfer_{i \to j}$ is much greater than $transfer_{j \to i}$, 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 [21, 22] and using them in this transfer framework presents significant technical challenges that we leave to future work. Additionally, we are conducting a more extensive evaluation of this method, including applying this method to other domains.

Acknowledgments

Eric Eaton's and Marie desJardins' work was supported in part by NSF grant ITR-0325329; Terran Lane's work was supported by NSF grant IIS-0705681. 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. We also thank the anonymous reviewers for their detailed feedback.

References

1. Marx, Z., Rosenstein, M.T., Kaelbling, L.P., Dietterich, T.G.: Transfer learning with an ensemble of background tasks. In: NIPS 2005 Workshop on Transfer Learning, Whistler, BC, Canada (2005)
2. Kienzle, W., Chellapilla, K.: Personalized handwriting recognition via biased regularization. In: Proceedings of the Twenty-Third International Conference on Machine Learning, Pittsburgh, PA (2006)
3. Thrun, S., O'Sullivan, J.: Clustering learning tasks and the selective cross-task transfer of knowledge. Technical Report CMU-CS-95-209, Carnegie Mellon University, Pittsburgh, PA (November 1995)
4. Thrun, S., O'Sullivan, J.: Discovering structure in multiple learning tasks: the TC algorithm. In: Proceedings of the Thirteenth International Conference on Machine Learning, Morgan Kaufmann (July 1996) 489–497
5. Bakker, B., Heskes, T.: Task clustering and gating for Bayesian multitask learning. Machine Learning Research **4** (May 2003) 83–99
6. Jordan, M., Jacobs, R.: Hierarchical mixtures of experts and the EM algorithm. Neural Computation **6**(2) (1994) 181–214
7. Pratt, L.Y.: Transferring Previously Learned Back-Propagation Neural Networks to New Learning Tasks. PhD thesis, Rutgers University (June 1993)
8. Mitchell, T.M., Thrun, S.B.: Learning analytically and inductively. In: Mind Matters: A Tribute to Allen Newell. Lawrence Erlbaum Associates (1996) 85–110
9. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools with Java Implementations. Morgan Kaufmann, San Francisco, CA (2000)
10. Duffy, D.E., Santner, T.J.: On the small sample properties of norm-restricted maximum likelihood estimators for logistic regression models. Communications in Statistics: Theory and Methods **18** (1989) 959–980
11. Le Cessie, S., Van Houwelingen, J.C.: Ridge estimators in logistic regression. Applied Statistics **41**(1) (1992) 191–201
12. Schölkopf, B., Smola, A.J.: Learning with Kernels. MIT Press (2002)
13. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, New York (2001)
14. Chung, F.R.K.: Spectral Graph Theory. Number 92 in CBMS Regional Conference Series in Mathematics. American Mathematical Society, Providence, RI (1994)
15. Rosenberg, S.: The Laplacian on a Riemannian Manifold. Cambridge University Press (1997)
16. Baker, C.T.H.: The Numerical Treatment of Integral Equations. Oxford: Clarendon Press (1977)
17. Fowlkes, C., Belongie, S., Chung, F., Malik, J.: Spectral grouping using the Nyström method. IEEE Transactions on Pattern Analysis and Machine Intelligence **26**(2) (February 2004)
18. Drineas, P., Mahoney, M.W.: On the Nyström method for approximating a Gram matrix for improved kernel-based learning. Journal of Machine Learning Research **6** (December 2005) 2153–2175
19. Asuncion, A., Newman, D.: UCI machine learning repository (2007)
20. Rennie, J.: 20 Newsgroups data set, sorted by date. Available online at http://www.ai.mit.edu/∼jrennie/20Newsgroups/ (September 2003)
21. Chung, F.: Laplacians and the Cheeger inequality for directed graphs. Annals of Combinatorics **9** (2005) 1–19
22. Chung, F.: The diameter and Laplacian eigenvalues of directed graphs. Electronic Journal of Combinatorics **13**(4) (2006)