Data Min Knowl Disc (2010) 21:327–343
DOI 10.1007/s10618-010-0189-3

Predicting labels for dyadic data

Aditya Krishna Menon · Charles Elkan

Received: 30 April 2010 / Accepted: 20 June 2010 / Published online: 27 July 2010

Abstract In dyadic prediction, the input consists of a pair of items (a dyad), and the goal is to predict the value of an observation related to the dyad. Special cases of dyadic prediction include collaborative filtering, where the goal is to predict ratings associated with (user, movie) pairs, and link prediction, where the goal is to predict the presence or absence of an edge between two nodes in a graph. In this paper, we study the problem of predicting labels associated with dyad members. Special cases of this problem include predicting characteristics of users in a collaborative filtering scenario, and predicting the label of a node in a graph, which is a task sometimes called within-network classification or relational learning. This paper shows how to extend a recent dyadic prediction method to predict labels for nodes and labels for edges simultaneously. The new method learns latent features within a log-linear model in a supervised way, to maximize predictive accuracy for both dyad observations and item labels. We compare the new approach to existing methods for within-network classification, both experimentally and analytically. The experiments show, surprisingly, that learning latent features in an unsupervised way is superior for some applications to learning them in a supervised way.

Keywords Dyadic prediction · Collaborative filtering · Link prediction · Social networks · Within-network classification · Relational learning

Responsible editors: José L Balcázar, Francesco Bonchi, Aristides Gionis, Michéle Sebag.

A. K. Menon (✉) · C. Elkan
Department of Computer Science and Engineering, University of California,
La Jolla, San Diego, CA 92037, USA
e-mail: akmenon@cs.ucsd.edu

C. Elkan
e-mail: elkan@cs.ucsd.edu

1 Dyadic prediction and within-network classification

Dyadic prediction is the problem of predicting observations associated with *dyads*, or pairs of items. Typically, the only information we have about the individual dyad members is a unique identifier. This means that the input data is not endowed with explicit features, so we have to learn *latent* features from the data. The problem can be viewed as a form of matrix completion by associating the training data with a matrix whose rows and columns are indexed by the identifiers of the dyad members. For example, consider the well-studied setting of collaborative filtering. Here, we wish to predict ratings associated with (user, movie) pairs. The data can be viewed as a matrix of users by movies, with the entries being the rating for a particular (user, movie) pair. Henceforth, to ease exposition we will refer to the rows and columns of a general dyadic prediction task as "users" and "movies", and the dyad observations as "ratings."

Existing work on dyadic prediction has focused on predicting the values of missing ratings. This is a problem of practical benefit, because for example we can use it to recommend new movies to a user. A problem that has received less attention is that of predicting *labels* associated with users and/or movies. Here, we have labels associated with some subset of users and/or movies, for example whether the user is male or female, whether the movie was a commercial success. We would like to extend these labels to all users and/or movies. In this paper, we show how we can view this problem as predicting the rating for an extra "movie" corresponding to the label, but we then argue that it is beneficial to consider alternatives to merely augmenting the data matrix with this extra "label movie." We show how to extend a recent flexible dyadic prediction method to predict labels for users. An important property of the new method is that it can predict labels even when there is incomplete rating data for users, which is common for dyadic data. (The presence of missing data is one of several differences between dyadic label prediction and standard supervised learning.)

One special case where the label prediction problem has been studied previously is for network data. Here, the input consists of the adjacency matrix of a graph, and the goal is to predict labels associated with nodes of the graph; this is referred to as *within-network classification* or *relational learning*. Given the connection between dyadic prediction and within-network classification, our method is also applicable to the latter problem. An important advantage of our method is that it works in the setting where there is missing data in the network, corresponding to pairs of nodes for which we do not know if an edge exists.[1]

The closest analogs to our approach in the within-network classification literature are methods based on inferring latent features from the graph structure (Tang and Liu 2009; Zhu et al. 2007). Below, we compare our method with these existing methods both analytically and experimentally. The results reveal a number of surprising findings. First, there is little benefit in using normalizing transformations of the input adjacency matrix, which is common practice for problems such as spectral clustering

[1] An edge being missing is not the same as it being absent; the latter means *known* not to be present, while the former means *lack of knowledge* about presence.

of graph data. Second, it is common for methods to overfit on the training data when trying to predict missing labels. Third, in some instances it is preferable to learn latent features in an unsupervised manner, based only on the dyadic observations and not on the labels.

2 Background and related work

The input in a dyadic prediction problem is a data matrix $X \in \mathcal{X}^{m \times n}$. Here, m is the number of users, n is the number of movies, and $\mathcal{X}$ is the space of possible ratings augmented with a special entry "?", which denotes that a rating is missing. Remember that we use the words "user" and "movie" and "rating" merely for convenience, without loss of generality. In collaborative filtering problems, usually $\mathcal{X} = \{?, 1, \ldots, R\}$, where R is the number of possible ratings. Let us focus on predicting the missing entries in X. This is a well-studied problem in machine learning, with the most popular solution being to consider a matrix factorization of X into the product UV^T, where $U \in \mathbb{R}^{m \times k}$ and $V \in \mathbb{R}^{n \times k}$ for some small integer k. The matrices U and V can be thought of as latent feature representations of the users and movies respectively. We choose U, V to optimize the objective

$$\min_{U,V} \ell(X, UV^T) + \frac{\lambda}{2}\Omega(U, V). \tag{1}$$

Here, ℓ is a loss function that drives U and V to predict the observed values of X, while Ω is a regularization term to prevent overfitting. There are many choices for ℓ and Ω. The simplest are square-loss and L_2 regularization:

$$\min_{U,V} ||X - UV^T||_{\mathcal{O}}^2 + \frac{1}{2}\left(\lambda_U ||U||_F^2 + \lambda_V ||V||_F^2\right).$$

Here, $||\cdot||_F$ is the Frobenius norm of a matrix (sum of squared entries), and $||\cdot||_{\mathcal{O}}$ is the Frobenius norm restricted to the entries of X that are not missing, i.e. $||A||_{\mathcal{O}}^2 = \sum_{i,j} \mathbb{I}[X_{ij} \neq ?]A_{ij}^2$. With this formulation, we are performing a type of regularized singular value decomposition (SVD) with respect to the observed entries of the matrix X (Menon and Elkan 2010b).

Link prediction is the problem of predicting whether or not an edge exists between a pair of nodes in an incompletely observed graph. More generally, the task is to predict the weight of an edge if it exists. Link prediction can be viewed as a dyadic prediction problem where the nodes of the graph are users and movies, and a rating corresponds to the weight of the edge between a pair of nodes; one can let a rating of 0 denote the absence of an edge. Equivalently, however, we can cast dyadic prediction as an instance of link prediction (Huang et al. 2005). Given a ratings matrix, we can construct a bipartite graph with nodes corresponding to users and movies. There are edges only between the user and movie nodes, and the weight of an edge is the value of the rating for the particular (user, movie) pair. Predicting missing ratings is equivalent to predicting the edge weights for unobserved edges.

As an extension to the basic dyadic prediction setting, consider the case where we also have a *label matrix* $Y \in \mathcal{Y}^{m \times L}$, where $\mathcal{Y} = \{?, 0, 1\}$. The goal is to fill in the missing entries of Y, or equivalently to predict labels for all the users in the data. We allow the placement of missing entries to be arbitrary. For example, we do not require that for each l there exist at least one user i such that $Y_{il} \neq ?$. We use the word "label" to refer to the vector Y_i for a user i, and the word "tag" to refer to an individual element Y_{il} within the vector, so that each label consists of L (possibly missing) tags. In general, the prediction problem is *multilabel* for each i. However, if it is guaranteed that $\sum_l Y_{il} = 1$ for all i, then the problem is *multiclass* for each i. The guarantee must of course apply to the "true" value of missing tags. In the multiclass case it is common to encode $Y_i \in \{1, \ldots, l\}$. Note that whether multiclass or multilabel, the label prediction task considered here is a form of transductive learning, because the goal is only to make predictions for existing users.

The label prediction problem is important for practical applications of dyadic prediction, such as:

- predicting characteristics of users or movies in collaborative filtering datasets;
- scoring suspiciousness or trustworthiness of users in a social network, possibly based on side-information concerning users; and
- predicting which strains of bacteria will be observed in various food processing plants (Sarkar et al. 2008).

In the problem of within-network classification, the input is a graph where some nodes are labeled, and the goal is to predict the labels for all other nodes. Broadly, there are two main approaches to this problem. The first is to exploit the topological structure of the graph to infer labels for unobserved nodes. The idea is that if the neighbors of a node are all positive, that can help us decide with confidence whether to label the node as positive or not (depending on whether there is a positive or negative neighbor-label correlation). A popular method along these lines is wvRN (Macskassy and Provost 2003), which assumes that the neighbors of a node capture all the relevant information for predicting its label, and that the weights of edges correspond to the strength of influence of one node to another. More sophisticated extensions of this idea involve random walks on the input graph, where intuitively one assigns the label of a node based not only on its neighbors but also on nodes that are easy in some sense to reach from it.

The other broad approach is to learn latent features from the graph, and then feed these into a supervised learning algorithm. The motivation here is clear: if we were given predictive features for the nodes, then the problem would simply reduce to supervised learning. Since we do not have such features, we try to learn them from the provided link data. The general strategy is:

1. Learn a latent representation U of the nodes from the adjacency matrix X. This is analogous to the U matrix for dyadic prediction, as in Eq. 1.
2. Let the rows of U be feature representations of each node. Train a multilabel classifier to predict the labels for the unlabeled nodes.

The underlying idea is that U captures some essential information about the nodes, which we believe to be predictive for their labels.

Given this general formulation, there are at least two ways to conduct the first step. One can learn U just from the input X, independent of the labels Y; we call these *unsupervised latent features*. Or, one can use both X and Y to influence the matrix U; we call these *supervised latent features*. A popular method using unsupervised latent features is named SocDim (Tang and Liu 2009). A limitation of this approach is that we might end up learning features that are predictive for the ratings, but uninformative for the labels; see the examples discussed in Yu et al. (2006), for example. Supervised latent features are used in the supervised matrix factorization (SMF) approaches applied to network prediction in Zhu et al. (2007), and to latent semantic indexing in Yu et al. (2005). The idea of these methods is to learn U to jointly optimize least-squares reconstruction error of X and a one-versus-rest SVM classifier for the labels.

3 Dyadic label prediction: reducing labels to movies

This section shows how labels can be predicted via a reduction that converts them into additional columns in the data. This reduction has not been analyzed in previous research. Although the reduction is useful, we explain why a better strategy is to have a tradeoff between predicting missing ratings and predicting labels.

Recall that for the label prediction problem, the input is $X \in \mathcal{X}^{m \times n}$ with associated labels $Y \in \mathcal{Y}^{m \times L}$. Consider first the setting where one learns unsupervised latent features U from the input X. We then need to learn a classifier to predict Y from U. Perhaps the simplest (but naïve) solution is to perform multiple linear regression on Y, ignoring any correlation between the tags. Here, we learn a weight $W \in \mathbb{R}^{L \times k}$, where L is the number of tags and k the number of latent features, to minimize

$$\min_W ||Y - UW^T||_{\mathcal{O}}^2 + \frac{\lambda_W}{2}||W||_F^2.$$

Now suppose we decide to do this optimization jointly with U, so that the latent features in U are supervised. This is the essential idea of the SMF approaches of Zhu et al. (2007) and Yu et al. (2005), which yields the optimization problem

$$\min_{U,V,W} ||X - UV^T||_{\mathcal{O}}^2 + ||Y - UW^T||_{\mathcal{O}}^2 + \frac{1}{2}\left(\lambda_U||U||_F^2 + \lambda_V||V||_F^2 + \lambda_W||W||_F^2\right).$$

It has not been pointed out before that this is equivalent to

$$\min_{U,V,W} \left\| \begin{bmatrix} X & Y \end{bmatrix} - U \begin{bmatrix} V \\ W \end{bmatrix}^T \right\|_{\mathcal{O}}^2 + \frac{1}{2}\left(\lambda_U||U||_F^2 + \lambda_V||V||_F^2 + \lambda_W||W||_F^2\right).$$

Therefore, the joint optimization treats the problem as matrix completion of the augmented matrix $\begin{bmatrix} X & Y \end{bmatrix}$. This can be interpreted as treating the labels as new "movies" in the dataset, for which we want to make predictions as we do normally with the ratings of other movies. Notice that this setup exploits the transductive nature of the problem. We are only interested in making predictions for users known during

training, and so we use the derived latent features for each user as predictive covariates for the labels. While the latent features depend explicitly on the users with a label known during training, they also depend implicitly on other users. This is because the latent features for labelled users are used to derive the latent features for movies, which are shared by all users.

In the simple unsupervised approach there are L separate regression tasks, one for each tag. These tasks are completely disjoint, and so the results for one task do not affect those of another. However, importantly, this is *not* true in the joint optimization problem. The reason is that we are learning U jointly with W. Therefore, there are implicit interactions between the weights of the tags, because they need to be predictive for the rest of the data matrix. So, supervised latent features can capture interactions between different tags, and the supervised approach does not assume the tags are independent.

The reduction of labels to movies suggests an extension of existing dyadic prediction methods to predict labels for users. However, there are important caveats concerning this approach. First, one needs to be clear about the ultimate objective. Above, the joint optimization assumes that we want to reconstruct the augmented matrix, which means that mispredicting the labels is equally as bad as mispredicting the ratings. However, in some settings our ultimate goal might be only the accuracy of predicting the labels, with the reconstruction error of the data matrix irrelevant. The data matrix in this case is just an incomplete training set for which we are trying to predict some labels. Therefore, what we really want is a user-controllable *tradeoff*, μ, between minimizing the reconstruction error and minimizing the label training error, as suggested in the context of supervised latent semantic indexing (Yu et al. 2005):

$$\min_{U,V,W} \mu \;\; ||X - UV^T||^2_{\mathcal{O}} + (1-\mu)||Y - UW^T||^2_{\mathcal{O}} + \frac{1}{2}\left(\lambda_U ||U||^2_F + \lambda_V ||V||^2_F + \lambda_W ||W||^2_F\right). \tag{2}$$

Second, one needs to decide how to make predictions for the test data. Does one use the learned weights W, or train a classifier on the learned U in a second stage? The latter is the approach discussed earlier in the context of SocDim, except that U is now learned in a supervised manner, and so can be expected to be predictive for the labels Y. In preliminary experiments, we found that learning a separate classifier gives better performance than using the learned W, so we follow this approach.

4 Latent feature log-linear (LFL) model to predict labels

The discussion above focuses on square-loss for reconstruction and label prediction error, and assumes that we decompose the data matrix as $X = UV^T$. However, the analysis holds for any choice of loss function and any valid decomposition. Equation 2 may be expressed more generally as

$$\min_{U,V,W} \mu \ell_{\text{Recon}}(X, U, V) + (1-\mu)\ell_{\text{Label}}(Y, U, W) + \frac{1}{2}\left(\lambda_U ||U||_F^2 + \lambda_V ||V||_F^2 + \lambda_W ||W||_F^2\right).$$

Here, ℓ_{Recon} is the loss function for the reconstruction of the data matrix, while ℓ_{Label} is the loss function for the label prediction.

The dyadic prediction technique we use in this paper is the latent feature log-linear model (LFL) proposed in Menon and Elkan (2010a). Using this mode is motivated by the same goals as in the context of dyadic prediction. Briefly, we want to have well-calibrated probabilities for each of the ratings, and to use *side-information* for the users and movies when it is available. Side-information refers to extra features in addition to unique user/movie identifiers. It can be useful both for achieving higher predictive accuracy, and also for data understanding; e.g. one can discover if the education level of a user influences his suspiciousness label.

The LFL model is as follows. Suppose the input $X \in \mathbb{R}^{m \times n}$ has ratings in $\{1, \ldots, R\}$. We keep sets of weights $U^r \in \mathbb{R}^{m \times k}$ and $V^r \in \mathbb{R}^{n \times k}$ for each rating. Note that for each rating r there are separate matrices U^r and V^r. We use the log-linear model

$$p(r|i, j; U, V) = \frac{e^{U_i^r (V_j^r)^T}}{\sum_{r'} e^{U_i^{r'} (V_j^{r'})^T}}$$

for the conditional probability of rating r given user i and movie j. The notation $U_i^{r'}$ means row i of the matrix $U^{r'}$, while $(V_j^{r'})^T$ means the transpose of row j of the matrix $V^{r'}$.

The reconstruction $\mathcal{E}(X)$ of the data matrix has entries

$$\mathcal{E}(X)_{ij} = \sum_r r \cdot p(r|i, j; U, V)$$

i.e. the predicted value is the *expected value* under the probability distribution. We perform optimization using either log-likelihood or mean-squared error (MSE). For log-likelihood, the objective is

$$\ell_{\text{Recon}}(U, V) = \sum_{(i,j)\in\mathcal{O}} \log \frac{e^{U_i^{X_{ij}} \left(V_j^{X_{ij}}\right)^T}}{\sum_r e^{U_i^r \left(V_j^r\right)^T}} + \frac{1}{2}\left(\sum_r \lambda_U ||U^r||_F^2 + \lambda_V ||V^r||_F^2\right)$$

where in a slight abuse of notation, $\mathcal{O}$ denotes the set of observations in the input. For MSE, the objective is

$$\ell_{\text{Recon}}(U, V) = ||X - \mathcal{E}(X)||_{\mathcal{O}}^2 + \frac{1}{2}\left(\sum_r \lambda_U ||U^r||_F^2 + \lambda_V ||V^r||_F^2\right).$$

MSE is more appropriate when the number of ratings $R > 2$, since we need to enforce the ordinal structure of the ratings: predicting a rating of 1 incorrectly as 5 is worse than predicting it as 2. For binary ratings, $R = 2$, and for nominal (unordered) ratings, log-likelihood is an appropriate choice.

The LFL model can be applied to network data as well. To see how to adapt it to the specifics of the problem at hand, define the probability model

$$p(y|i, j; U, V, \Lambda) = \frac{e^{U_i^y \Lambda_{ij} \left(V_j^y\right)^T}}{\sum_{y'} e^{U_i^{y'} \Lambda_{ij} \left(V_j^{y'}\right)^T}}.$$

We can constrain the latent features depending on the nature of the input:

- For general dyadic data ($m \neq n$), we let $\Lambda = I$. This is valid because we can think of the low-rank approximation here as following the singular value decomposition (SVD), for which $\Lambda \succeq 0$. So, we can define $U' = U\Lambda^{1/2}$ and $V' = V\Lambda^{1/2}$, thus absorbing Λ into the latent factors.
- For asymmetric network data ($m = n$), we set $V = U$ but let Λ be an arbitrary dense matrix, as suggested in Zhu et al. (2007). Setting $V = U$ is justified because it enforces that both users and movies lie in the same latent space.
- For symmetric network data, we set $V = U$ and $\Lambda = I$. This is justified because for a symmetric matrix, the SVD of $X = U\Lambda U^T$, and as with the general dyadic case, Λ may be absorbed into U.

Finally, when applied to the binary tags in the matrix Y with log-likelihood as the loss function, the LFL performs regularized logistic regression on the observed tags:

$$\ell_{\text{Label}}(Y, U, W) = \sum_{(i,l)\in\mathcal{O}} \frac{e^{Y_{il}(W_l^T U_i)}}{1 + e^{W_l^T U_i}} + \frac{\lambda_W}{2}||W||_F^2.$$

Notice that it is also possible to predict tags in the matrix which are qualitatively different from the ratings in X. For example, if the tags are real-valued, we can replace ℓ_{Label} by a regularized linear or nonlinear regression objective.

Having defined the general LFL model, we note its salient properties:

- *Supervised latent features* As discussed earlier, learning latent features in an unsupervised manner ignores the label information, which means that the latent features may be uncorrelated with the labels and thus yield poor accuracy. The LFL model learns features to predict the labels and ratings jointly.
- *Flexible about data properties* It is desirable for a method to work for a range of different data types. For example, in the context of within-network classification, we would like for methods to work on both symmetric and asymmetric graphs. The LFL has this flexibility.
- *Handling missing data* For a general dyadic prediction task, we do not have observations for all dyads. In such settings, it is conceptually still possible to make

Table 1 Comparison of latent feature based methods for label prediction

Item	LFL	SMF	SocDim
Supervised latent features?	Yes	Yes	No
Asymmetric graphs?	Yes	Yes	No
Missing data?	Yes	No	No
Finds latent features of?	Data	Data	Modularity
Single minimum?	No	No	Yes

LFL is the model proposed in this paper

predictions about the labels of users and movies. Therefore, it is desirable for a method to handle missing data in the input matrix, which the LFL achieves seamlessly.

Table 1 summarizes the presence or absence of these qualities in the LFL model and in previous latent feature methods for within-network classification. We also note of what exactly the methods find latent features, and whether they are subject to local minima; these issues are discussed more in the next section.

5 Comparing the latent feature methods

This section looks more closely at three latent feature methods: the SocDim (Tang and Liu 2009) and SMF (Zhu et al. 2007) methods in the within-network classification literature, and the LFL method described above.

We start by comparing their objective functions. As discussed in Sect. 2, and noted in Table 1, the key difference between SocDim and the SMF and LFL methods is that the former uses unsupervised latent features. But are there other differences? To study this question, we look at the objective functions for all three methods in the *unsupervised* setting i.e. we make SMF and LFL disregard the labels during the training phase. Since SocDim and SMF were designed with network data in mind, assume that $X \in \{0, 1\}^{n \times n}$ is the adjacency matrix of an unweighted graph. Note that there are no missing entries for the purposes of this discussion. Further, since SocDim assumes that the network is symmetric (i.e. the graph is undirected), suppose that $X = X^T$. This assumption is needed because it makes the eigenvalues of X be real. The objective functions are then:

– *SocDim objective* The first step of the SocDim method is an eigendecomposition of the *modularity matrix* of X, defined as $\mathcal{Q}(X) = X - \frac{1}{2|E|} dd^T$, where d is a vector of the node degrees and $|E|$ is the number of edges in the graph. These eigenvectors are then used as a latent feature representation of the nodes in the graph. Since the eigenvectors of a symmetric matrix equal its singular vectors, up to a sign flip, we can reformulate the first step as

$$\min_{U,\Lambda} ||\mathcal{Q}(X) - U\Lambda U^T||_F^2.$$

When Λ is constrained to be diagonal, the optimal solution U^* of this equation corresponds to the eigenvectors of $\mathcal{Q}(X)$ up to rotation.

– *SMF objective* In SMF, the objective function is that of Eq 2, except that we assume that there is no missing data. When we set $\mu = 1$, corresponding to no influence of the labels on the training process, the objective is

$$\min_U ||X - U\Lambda U^T||_F^2 + \frac{\lambda_U}{2}||U||_F^2.$$

– *LFL objective* For the case of binary ratings, the LFL model reduces to

$$p(y = 1|i, j; U, V) = \sigma\left(U_i \Lambda_{ij} U_j^T\right)$$

where $\sigma(\cdot)$ is the sigmoid function. For log-likelihood, the objective is quite different from the other methods. But optimizing MSE yields

$$\min_U ||X - \sigma(UU^T)||_{\mathcal{O}}^2 + \frac{\lambda_U}{2}||U||_F^2.$$

We see that the three methods are all instantiations of the following general problem:

$$\min_{U,\Lambda} ||f(X) - g(U, \Lambda)||_F^2 + \frac{\lambda_U}{2}||U||_F^2.$$

For SocDim, $f(X) = \mathcal{Q}(X)$ and $g(U, \Lambda) = U\Lambda U^T$; for SMF, $f(X) = X$ and $g(U, \Lambda) = U\Lambda U^T$; and for LFL, $f(X) = X$ and $g(U, \Lambda) = \sigma(UU^T)$. In this general scheme, we consider a transformed version of the data matrix, and then consider a low-rank approximation that is itself passed through a transformation.

SocDim and LFL differ in terms of which of the components, the data matrix or the low-rank approximation, they choose to transform. It is not clear *a priori* which scheme is more useful. In the case of SocDim versus LFL, the transforms have different ostensible goals. The point of the sigmoidal transform is simply to make the entries lie in [0, 1]. The point of the modularity transform is to normalize the degree distributions of the nodes in the network, so that high degree nodes do not overshadow the rest of the graph. Interestingly, in collaborative filtering one can do something similar to the *regularization* part of the objective (Weimer et al. 2008):

$$\min_U ||X - \sigma(UU^T)||_{\mathcal{O}}^2 + \frac{\lambda_U}{2}\text{tr}[U^T DU].$$

Here, $D = \text{diag}(1/\sqrt{d_1}, \ldots, 1/\sqrt{d_n})$, where $d_i = \sum_j X_{ij}$. This form of regularization is used in our implementation of LFL.

SocDim and SMF perform essentially the same optimization, except that SocDim works on the modularity matrix. Of course, one can consider a variant of SMF which operates on the modularity matrix; does that imply that the solutions of the two methods will be similar when $\mu = 1 - \epsilon$ for ϵ small? This is not necessarily true, because of

the nature of the optimization process. In SocDim, we optimize the objective function using an analytic solution, namely the eigenvectors of the modularity matrix. In SMF for $\mu < 1$, we have to resort to gradient descent to optimize the objective function, since there is no closed form solution.[2] But the objective is not jointly convex in U and Λ, so one can only find a local minimum. This means that even for μ close to 1, one may not exactly recover the eigenvectors of the data matrix. So, an advantage of SocDim is that it is immune to issues of local minima, albeit at the cost of being more expensive for large datasets, since it involves an eigendecomposition, whose runtime is superlinear in the size of the data matrix.

Another question raised by the analysis above is the impact of the choice of $f(X)$. Does choosing $f(X) = X$ for SocDim give noticeably worse results? Is the use of the modularity $\mathcal{Q}$ essential, or can one use other popular transformations, such the normalized Laplacian from the spectral clustering literature, $\mathcal{L} = I - D^{-1/2} X D^{-1/2}$, where D is a diagonal matrix of node degrees. It is also interesting to see what impact these transforms have on the SMF approach.

We have claimed thus far that it is desirable to learn supervised latent features from the data, as this should give better predictive performance. This claim is intuitively reasonable, but there are two issues to be mindful of. First, one has to choose numerous parameters for a supervised factorization, the three regularization parameters $\lambda_U, \lambda_V, \lambda_W$, and the tradeoff parameter μ. Cross-validation is needed to choose these, which is expensive. Second, introducing a dependence of U on the labels also introduces a risk of overfitting on the training labels. Clearly when $\mu = 0$ the solution is useless, because it just tries to learn latent features that explain the labels of the known training examples. But even for nonzero μ, overfitting is possible, especially when the tags are sparsely populated, which is common for many multilabel problems. The similar issue is mitigated in standard collaborative filtering using ℓ_2 regularization of the weights, but in the label prediction problem there is an interplay between the tradeoff μ and the regularizer λ_W. These issues are explored in the experiments below.

6 Experimental design

This section explains the design of our experiments, covering first the issues to be explored, next the datasets to be used, and last the methods to be compared. The following questions are addressed in the experiments.

- *The value of supervised latent features* As discussed in the previous section, while supervised latent features have an intuitive advantage over their unsupervised counterparts, it is important to compare the two empirically. In particular, does supervised training of latent features exacerbate the problem of overfitting?
- *The impact of missing edges* For general dyadic prediction problems, there are often many missing dyadic observations. Existing methods for predicting labels for network data, and in particular the ones based on learning latent features, do not deal with this issue. But can we just treat the missing observations as being

[2] When we have labels for all the data, a closed form solution in terms of a generalized eigenvector is possible (Yu et al. 2005).

edges with weight 0? Does this have a noticeable impact on the accuracy of label prediction? How does this simple approach compare with the LFL approach, which is designed to handle missing dyadic observations?
- *The value of data transformation* The SocDim method involves learning latent features from the modularity matrix of the data. The idea of learning latent features from a transformation of the original data matrix has been studied in the spectral clustering literature. An interesting question is whether performance is worse when the raw data matrix is used to learn latent features. If so, how do we choose between different transformations, such as the modularity and the Laplacian?
- *Dyadic vs network data* As discussed earlier, there is a two-way reduction between dyadic prediction and link prediction. Therefore, dyadic prediction methods such as LFL can be applied to network data. However, it also follows that link prediction methods can be applied to general dyadic prediction data, such as collaborative filtering datasets. A natural question is whether, all else being equal, in particular with no missing data, both types of method perform equally well.

Experiments use the following datasets, each possessing very different properties.

- `blogcatalog`. This dataset comprises links between bloggers in the BlogCatalog directory (Tang 2010). The labels provided are the stated interests of users, which are divided into 39 possible categories. A user may have each interest independently, so this is a multilabel problem. This is an example of a network dataset where the goal is to use known links to predict something useful about the nodes. Due to time constraints, we focus on a subset of the dataset comprising a random selection of 2,500 bloggers. Preliminary results on the full dataset suggest similar patterns to the ones observed in our results.
- `senator`. This dataset comprises roll call data from the 109th session of the United States senate. It consists of the votes of 101 senators concerning 315 bills, with possible votes being "Yea" or "Nay." The dataset was studied previously in Blei and McAuliffe (2010). The data can be thought of as a binary ratings matrix for senators by bills. The goal is to predict whether or not a senator is a Republican or Democrat. This particular task is relatively easy, because the voting patterns of senators are highly predictive of their political affiliation. However, the dataset is representative of many datasets that are important in political science, where one is interested in using behavioral records to test hypotheses about the nature of political dynamics.[3]
- `usps`. This dataset consists of handwritten digits represented as grayscale 16×16 images (USPS 2010). The labels for the data are simply the true digits that the images correspond to. Learning to predict labels here is a standard supervised learning problem, but we look at the case where some entries are missing, which corresponds to some subset of pixel values being occluded. For simplicity, we make the dataset binary so that pixel values are either black or white. This dataset

[3] A few entries in the `senator` dataset are missing, which raises the point that even well-curated datasets often do have missing entries. In general it is important to have a principled way to handle these. For this dataset missing votes are treated as "Nay" for methods that cannot handle missingness directly. The impact of this choice is small for all methods.

shows how a more difficult version of a standard supervised learning task may be solved by formulating it as a dyadic label prediction problem.

The experiments compare three latent feature methods for within-network classification, namely SocDim, SMF, and the new LFL model explained above. We implement SMF and LFL ourselves, and use the code for SocDim provided by Tang (2010).

For all methods, the learned latent features are passed through a linear SVM to get final label predictions. We use LibLinear for the SVM implementation (Fan et al. 2008). Following Tang and Liu (2009), for multilabel data we assume that the number of labels are known, and we measure how well the predicted score for each tag agrees with the true label. Agreement is measured using the F1 micro and macro scores, which for true tags y_{il} and predictions $\hat{y}_{il}$ are defined as

$$\text{Micro} = 2\frac{\sum_{i,l} y_{il}\hat{y}_{il}}{\sum_{i,l} y_{il} + \hat{y}_{il}}$$

and

$$\text{Macro} = \frac{2}{L}\sum_{l}\frac{\sum_{i} y_{il}\hat{y}_{il}}{\sum_{i} y_{il} + \hat{y}_{il}}.$$

For the multiclass datasets, 0–1 error is the performance measure. In all experiments, we perform cross-validation to choose the regularization and μ parameters, and report accuracy on the training and test set to study whether overfitting is a serious issue.

An issue to note is that the `senator` and `usps` datasets are not ostensibly in the form of a graph. But as discussed in Sect. 2, we can transform general dyadic data into a bipartite graph, so this representation is used when training the SocDim and SMF methods on these datasets.

7 Experimental results

Results on the `blogcatalog` dataset are presented in Table 2, which shows F1 micro and macro scores for both training and test data; higher scores are better. The reported scores are for the parameter settings that result in the best total test set score over all folds of cross-validation. The number of latent factors k is 1, 25, or 100, denoted by a subscript for the method. Note that the LFL method assumes the input data is discrete, so it cannot be applied to the Modularity and Laplacian matrices; hence the "N/A" entry in the table.

The `blogcatalog` results reveal some surprising facts. First, all methods exhibit a strong degree of overfitting, especially as measured by the Macro score. This suggests that ℓ_2 regularization alone is not sufficient to prevent overfitting for label prediction problems. (Preliminary experiments indicate that overfitting is even worse on the AUC measure.) Second, neither SocDim and SMF benefits from working with the Laplacian or modularity matrix. In fact, we achieve the best performance with SMF on the raw adjacency matrix. Third, LFL is outperformed by SocDim and SMF. This suggests

Table 2 10-fold CV results of various methods on the `blogcatalog` dataset

	Raw data				Modularity				Laplacian			
	Train		Test		Train		Test		Train		Test	
	Micro	Macro	Micro	Macro	Micro	Macro	Micro	Macro	Micro	Macro	Micro	Macro
SocDim_1	0.1801	0.0321	0.1767	0.0293	0.1621	0.0309	0.1596	0.0296	0.1628	0.0296	0.1395	0.0195
SocDim_{25}	0.2388	0.1298	0.2043	0.0701	0.2368	0.1044	0.2105	0.0621	0.2066	0.1022	0.1632	0.0367
SocDim_{100}	0.4017	0.3954	0.2630	0.1462	0.3890	0.3619	0.2671	0.1346	0.2824	0.2674	0.1869	0.0722
LFL_1	0.1795	0.0312	0.1775	0.0311	N/A							
LFL_{25}	0.2189	0.1006	0.1872	0.0595								
LFL_{100}	0.3697	0.4184	0.2001	0.1061								
SMF_1	0.1791	0.0288	0.1474	0.0213	0.1826	0.0323	0.1707	0.0224	0.1608	0.0275	0.1585	0.0262
SMF_{25}	0.2449	0.1431	0.1973	0.0750	0.2422	0.1275	0.2126	0.0719	0.2047	0.0868	0.1624	0.0342
SMF_{100}	0.4360	0.4721	**0.2877**	**0.1822**	0.4149	0.4141	0.2874	0.1355	0.2718	0.2437	0.1667	0.0730

The bold numbers indicate the best performing methods. They are only reported for the columns labelled "Test" because that is the quantity of primary interest

Table 3 10-fold CV results of various methods on the `senator` dataset

	Raw data		Modularity		Laplacian	
	Train 0–1 error	Test 0–1 error	Train 0–1 error	Test 0–1 error	Train 0–1 error	Test 0–1 error
SocDim_1	0.0247	0.0284	0.0000	0.0544	0.0238	0.0296
SocDim_5	0.0123	0.0284	0.0000	0.0408	0.0119	0.0148
LFL_1	0.0132	0.0199	N/A			
LFL_5	0.0132	**0.0132**				
SMF_1	0.0298	0.0321	0.0152	0.0351	0.0222	0.0325
SMF_5	0.0000	0.0494	0.0152	0.0351	0.0111	0.0163

The bold numbers indicate the best performing methods. They are only reported for the columns labelled "Test" because that is the quantity of primary interest

Table 4 10-fold CV results of various methods on the `usps` dataset

	Raw data		Modularity		Laplacian	
	Train 0–1 error	Test 0–1 error	Train 0–1 error	Test 0–1 error	Train 0–1 error	Test 0–1 error
SocDim_1	0.1287	0.1304	0.1239	0.1333	0.1297	0.1360
SocDim_{25}	0.0207	0.0190	0.0249	0.0305	0.0255	0.0251
SocDim_{50}	0.0165	**0.0174**	0.0220	0.0305	0.0202	0.0235
LFL_1	0.1435	0.1405	N/A			
LFL_{25}	0.0344	0.0324				
LFL_{50}	0.0248	0.0311				
SMF_1	0.1471	0.1444	0.1411	0.1394	0.1676	0.1613
SMF_{25}	0.0205	0.0218	0.0253	0.0255	0.0244	0.0340
SMF_{50}	0.0157	0.0210	0.0215	0.0271	0.0184	0.0292

The bold numbers indicate the best performing methods. They are only reported for the columns labelled "Test" because that is the quantity of primary interest

that for this problem, there is no benefit in applying the sigmoidal transformation to ensure that predictions for the reconstruction lie in [0, 1], contrary to expectations.

Results on the `senator` dataset, presented in Table 3, are markedly different from those on `blogcatalog`. First, overfitting is not as strongly manifest, and in fact sometimes the test error is *smaller* than the training error. (The small size of the dataset likely plays a part in explaining this finding.) The LFL method does best on this dataset, while SocDim and SMF do best when operating on the Laplacian of the induced bipartite graph. There are instances where SocDim and SMF overfit badly, achieving a perfect training score. This further emphasizes that overfitting is a difficult issue for the label prediction task.

For the `usps` dataset, we first present results training on the entire dataset with no missing entries. Table 4 shows that all methods do quite well in predicting the labels of the digits, with SocDim managing the best 0–1 error of 1.7% when trained on the raw

Table 5 10-fold CV results on the `usps` dataset with 50% missing pixels

	Raw data		Modularity		Laplacian	
	Train 0–1 error	Test 0–1 error	Train 0–1 error	Test 0–1 error	Train 0–1 error	Test 0–1 error
SocDim_1	0.1337	0.1352	0.1389	0.1388	0.1261	0.1238
SocDim_{25}	0.0354	**0.0300**	0.0490	0.0522	0.0516	0.0594
SocDim_{50}	0.0320	0.0344	0.0453	0.0522	0.0470	0.0527
LFL_1	0.1452	0.1432	N/A			
LFL_{25}	0.0502	0.0472				
LFL_{50}	0.0380	0.0424				
SMF_1	0.1487	0.1471	0.1673	0.1640	0.1415	0.1390
SMF_{25}	0.0448	0.0446	0.0500	0.0440	0.0509	0.0439
SMF_{50}	0.0412	0.0504	0.0518	0.0496	0.0454	0.0447

The bold numbers indicate the best performing methods. They are only reported for the columns labelled "Test" because that is the quantity of primary interest

Table 6 10-fold CV results on the `usps` dataset with 90% missing pixels

	Raw data		Modularity		Laplacian	
	Train 0–1 error	Test 0–1 error	Train 0–1 error	Test 0–1 error	Train 0–1 error	Test 0–1 error
SocDim_1	0.1434	0.1433	0.1475	0.1339	0.1484	0.1622
SocDim_{25}	0.1162	0.1190	0.1306	0.1210	0.1346	0.1470
SocDim_{50}	0.1042	**0.1101**	0.1242	0.1226	0.1256	0.1438
LFL_1	0.1520	0.1568	N/A			
LFL_{25}	0.1295	0.1416				
LFL_{50}	0.1111	0.1336				
SMF_1	0.1585	0.1611	0.1671	0.1661	0.1515	0.1406
SMF_{25}	0.1282	0.1401	0.1297	0.1354	0.1364	0.1357
SMF_{50}	0.1166	0.1377	0.1222	0.1346	0.1272	0.1309

The bold numbers indicate the best performing methods. They are only reported for the columns labelled "Test" because that is the quantity of primary interest

data. It is not surprising that the modularity and Laplacian matrix do not give much benefit, because it is not clear that the degrees of nodes (corresponding to the number of pixel values for a particular image) are useful as a normalization of the data.

Table 5 presents results after randomly occluding 50% of pixels. The Laplacian and modularity matrix here are computed with respect to the observed pixels. For the SocDim and SMF methods, this corresponds to treating an occluded pixel value as 0, since that is the dominant class. There is only a mild degradation in the performance of all methods. Interestingly, SocDim is still superior to LFL, even though the latter is designed to fill in missing entries of the input matrix.

Finally, Table 6 shows results with 90% missing data. As expected, all methods see a noticeable drop in prediction accuracy. However, SocDim still outperforms the LFL method, despite not being designed to handle missing data. This robustness to missing data is surprising and merits further study, although it may merely be an indication that the prediction task is not difficult.

Acknowledgments The authors thank Lei Tang for gracious help with running the code for SocDim and for answering several queries regarding the same. The authors also thank David Blei for providing the `senator` dataset.

References

Blei DM, McAuliffe JD (2010) Supervised topic models. Revised version. http://arxiv.org/PS_cache/arxiv/pdf/1003/1003.0783v1.pdf

Fan RE, Chang KW, Hsieh CJ, Wang XR, Lin CJ (2008) LIBLINEAR: a library for large linear classification. J Mach Learn Res 9:1871–1874

Huang Z, Li X, Chen H (2005) Link prediction approach to collaborative filtering. In: Proceedings of the 5th ACM/IEEE-CS joint conference on digital libraries (Denver, CO, USA, June 7–11, 2005), JCDL'05. ACM, New York, NY, pp 141–142

Macskassy SA, Provost F (2003) A simple relational classifier. In: Proceedings of the second workshop on multi-relational data mining (MRDM-2003) at KDD-2003, pp 64–76

Menon AK, Elkan C (2010a) Dyadic prediction using a latent feature log-linear model. http://arxiv.org/abs/1006.2156

Menon AK, Elkan C (2010b) Fast algorithms for approximating singular value decomposition. ACM Trans Knowl Discov Data. Special issue large-scale data mining: theory appl (to appear)

Sarkar P, Chen L, Dubrawski A (2008) Dynamic network model for predicting occurrences of salmonella at food facilities. In: Proceedings of the BioSecure international workshop. Springer, Heidelberg, pp 56–63

Tang L (2010) Social dimension approach to classification in large-scale networks. http://www.public.asu.edu/~ltang9/social_dimension.html

Tang L, Liu H (2009) Relational learning via latent social dimensions. In: ACM SIGKDD international conference on knowledge discovery and data mining. ACM, Edmonton, Alberta, pp 817–826

USPS (2010) USPS dataset. Obtained from http://www-i6.informatik.rwth-aachen.de/~keysers/usps.html

Weimer M, Karatzoglou A, Smola AJ (2008) Improving maximum margin matrix factorization. In: European conference on machine learning and principles and practice of knowledge discovery in databases. pp 263–276

Yu K, Yu S, Tresp V (2005) Multi-label informed latent semantic indexing. In: ACM SIGIR conference on research and development in information retrieval. ACM, Boston, pp 258–265

Yu S, Yu K, Tresp V, Kriegel HP, Wu M (2006) Supervised probabilistic principal component analysis. In: ACM SIGKDD international conference on knowledge discovery and data mining. ACM, Philadelphia, pp 464–473

Zhu S, Yu K, Chi Y, Gong Y (2007) Combining content and link for classification using matrix factorization. In: ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, Amsterdam, pp 487–494