PRE-DRAFT: On unlabeled graph classification

cep,djm,jmc,jv2

October 8, 2010

Abstract

Much data are collections of unlabeled graphs.

Inference on these collections of graphs is desirable, with the dual-purpose goal of: (1) predicting properties of new graphs, and (2) understanding the relationship between structure and function.

When graphs are unlabeled, invariants work, or solving GIP and then building classifiers on $\mathbb{P}[G, Y]$.

Solving GIP is hard, but fast approximate solutions are possible.

Sometimes, fast approximations are sufficient.

We consider classification of unlabeled graphs. We wish to assess the performance degradation due to the application of assignment methods.

This is carey's version – simplified (perhaps too much?); Joshua's connectome (i.e., "brain graph"?) interpretation will have to come from Joshua.

NB: alas, note that "labeled" is used two different ways herein; graphs may be labeled in the sense that the between-graph vertex identification is available, and graphs may be labeled in the sense that their class label is available. i'm sure you're clever enough to negotiate this issue, in the sequel, for the nonce. i solicit constructive approaches to addressing this issue.

NB: i switch willy-nilly twixt graph G and adjacency matrix A. complain if you must. but to what end?

NB: i use "NB" a lot, too. too much, maybe . . .

1 Introduction

In the current digital age, swaths of data are collected in myriad diverse fields ranging from astronomy to zoology. While these data are well characterized by networks, or graphs, the vast majority of statistical inference techniques, including the recent advances in machine learning, tend to assume the data live in finite dimensional Euclidean space. Thus, inference techniques designed to specifically to address collections or graphs could potentially yield fruitful results across domains. Furthermore, these collections of graphs often lack vertex labels, meaning no known mapping from vertices in any one graph to any other graph, is available. In such scenarios, the graphs are called "unlabeled," and this is the scenario of interest for this manuscript. Consider, for example, two food networks in different jungles. Both might have lions and tigers and bears (oh my), so it is easy to compare across the jungles. On the contrary, if one jungle only has the birds and the bees, one might still be interested in comparing the food networks, but a bird might play the same role in the second jungle as the lion does in the first, making the problem more complicated.

The exploitation task of investigation for this manuscript is that of unlabeled graph classification. More specifically, we assume that we have observed some number of unlabeled graph/class pairs, and we want to classify a novel unlabeled graph into its appropriate class. Unfortunately, because we do not directly observe the graphs, rather, only the unlabeled version of each graph, our classifiers must somehow deal with this additional complication.

We consider two general approaches. First, one can compute a large number of statistics on unlabeled graphs that are invariant to isomorphisms. That is, if g is a graph, and $\widetilde{g}$ is another graph isomorphic to g, then any function that satisfies $f(g) = f(\widetilde{g})$ is called a "graph invariant." For example, the number of edges or triangles in a graph is invariant to isomorphisms. By computing a large number of these graph invariants, one effectively projects the graph into finite Euclidean space, at which time one can apply standard machine learning tools. A second approach under consideration involves first (approximately) solving GIP for each test graph, and then applying *labeled* graph classification algorithms, such as those developed in [**?**].

Neither of these approaches are likely to dominate the other in general, as both have rather severe limitations. The graph invariant approach lacks much rigorous theoretical support. For instance, it remains an open question for which distributions can the graph invariant approach achieve the Bayes optimal classification rate (except in trivial cases for which the generative model is a function of the particular graph invariants being used). However, many graph invariants can be computed quite quickly. Further, graph invariants have both local and global properties, that is, they capture some aspect of the whole graph, by recursively considering local functions. Finally, for certain exploitation tasks, one might desire to understand the relationship between certain graph invariants and the class (for instance, are transportation grids more efficient when more triangles exist?).

On the other hand, if one first (approximately) solves GIP, then limiting results for many classifiers are readily available (upon stacking each adjacency matrix to project it onto finite Euclidean space). Unfortunately, solving GIP is known to be NP incomplete [**?**] (not known to be NP complete nor P). This means that solving GIP remains quite computationally expensive, especially as n, the number of vertices in the graphs, gets large.

Several questions therefore arise for us, as a first investigation into unlabeled graph classification. Are there distributions for which approximate GIP solutions yield good classification results? If so, can we establish any general rules under which approximating GIP is expected to perform relatively well? Alternately, are there distributions for which the graph invariant approach outperforms the GIP approach? How do the computations and performances for these two approaches scale with respect to one another? In this study, we first describe precisely how one might proceed upon using either of the above two approaches. Then, we conduct a number of in simulu experiments that we believe are informative with regard to some of these questions. We conclude with some discussion and outlines for possible future work and applications.

2 Methods

2.1 Preliminaries

Let G be a graph-valued random variable, $G : \Omega \mapsto \mathcal{G}$, where $\mathcal{G}$ is the set of all possible graphs, $g \in \mathcal{G}$ (and Ω is the universal sample space). A graph is assumed to be a triple, (V, A, Q), where $V = \{V_i\} = V_i \, \forall i \in [n]$, where $[n] = \{1, \ldots, n\}$, $A = \{A_{ij}\}$ is the adjacency matrix, where each edge is binary (an easily relaxed assumption), and Q is a permutation matrix (a matrix with a single one in each column and row, and zeros otherwise). Let $\mathcal{Q}$ denote the set of permutation matrices.

Further, let Y be a binary-valued random variable, $Y : \Omega \mapsto \mathcal{Y} = \{0, 1\}$, which can be easily generalized to any multinomial space (or continuous spaces with a bit more work). Graphs and classes are sampled jointly and exchangeably from some unknown but true distribution, $\mathbb{P}[G, Y; \theta] \in \mathcal{P}$, where $\mathcal{P}$ is the family of distributions under consideration, $\mathcal{P} = \{\mathbb{P}[G, Y; \theta] : \theta \in \Theta\}$, where θ is the parameter of the distribution, and Θ is the set of all possible parameters (in the sequel, we often drop the θ from $\mathbb{P}[G, Y; \theta]$).

Formally, we assume that $(G, Y), \{(G_l, Y_l)\} \sim \mathbb{P}[G, Y; \theta]$, where (G, Y) is the graph/class test sample pair, and $\mathcal{T}_s = \{G_l, Y_l\} = \{(G_1, Y_1), \ldots, (G_s, Y_s)\}$ is the training sample set, where s is the total number of training samples. The goal is to impute the latent test sample class, $Y = y$, given the training data. While we do not believe that $\mathbb{P}[G, Y; \theta]$ is in fact true, we do believe that the true distribution (if one exists) can be approximated reasonably well by some distributions in $\mathcal{P}$, such that we have hope to achieve misclassification rates better than chance on at least some data sets.

Unfortunately, we do not observe the graphs in their entirety, but rather, we only see the adjacency matrices, and not the permutation matrices. That is, instead of observing $G_l = (A_l, Q_l)$, we observe only A_l. Throughout, we will use the notation that $\widetilde{A}_l = Q_l A_l Q_l^{\mathsf{T}}$, so, had we observed $\widetilde{A}_l$'s, then we would have observed all there is to perform the classification.

Graph invariants can therefore be defined as functions on A, not requiring Q for computing their values.

2.2 Graph invariant based classifiers

Many graph invariants are available from the literature (see `http://en.wikipedia.org/wiki/Graph_invariant` for a list of some of the most popular ones). A subfield of random graph models, collectively called exponential family random graph models (ERGMs), typically models a graph by its triangles, k-stars, and the like [**?**]. Recent work from Pao et al. analyzed the power of various graph invariants for a special case of hypothesis testing [**?**]. Because of the close relationship between hypothesis testing and classification (see B&D, pg. XXX [**?**]), we consider the same set of graph invariants here. In particular, we consider:

1. size: $\#(A)$ = the number of edges in the graph
2. maximum degree: $\delta(G) = \max_{i \in [n]} d(V_i)$, where $d(V_i)$ indicates degree of vertex i, that is, the number of edges incident to V_i.
3. maximum average degree: $MAD(G) = \max_{\Omega_G} \bar{d}(\Omega_G)$, where $\bar{d}(G)$ is the average degree of a graph, and Ω_G is all subgraphs of G. Because computing $MAD(G)$ exactly is computationally taxing (how much???), we instead compute the maximum eigenvalue MAD [**?**].
4. scan statistics: the k-th scan statistic is the maximum number of edges over all k-th order neighborhoods, $S_k(G) = \max_{i \in [n]} \#(\Omega_{N_k[i;G]})$, where $N_k[i;G] = \{j \in [n] : l(V_i, V_j) \leq k\}$, and $l(V_i, V_j)$ is the distance between any two vertices, that is, the minimum number of edges traversals between them. We assume that $l(V_i, V_i) = 0 \, \forall i$, and $l(V_i, V_j) = \infty$ if no path exists between V_i and V_j.
5. number of triangles: a triangle exists whenever there is an edge from V_i to V_j, one from V_j to V_k, and one from V_k back to V_i
6. clustering coefficient: a measure that captures the degree to which nodes tend to cluster together
7. average path length: typically defined by $\sum_{i,j} l(V_i, V_j)/(n^2 - n)$, but to exclude the infinities, we let $\infty \mapsto 2 \max l(V_i, V_j)$.

Let $\boldsymbol{f}(\cdot) : \mathcal{G} \mapsto \mathbb{R}^k$ be the function that takes a graph as input, and outputs a set of k graph invariants. Having defined such a function, one can then use standard machine learning algorithms to perform classification. We plug the results into a "standard" suite of machine learning tools: (i) a linear classifier—linear discriminant analysis (LDA), (ii) a quadratic classifier—quadratic discriminant analysis (QDA), (iii) a support vector machine (SVM), (iv) random forests (RF), and (v) k_n nearest neighbor classifier. Note that the dimensionality of the data is quite large ($\mathcal{O}(n^2)$), we first reduce the dimensionality of the data using principal components analysis (PCA). The dimensionality of the data to keep is chosen using the method of [**?**].

2.3 Likelihood based approach

The Bayes optimal classifier is:

$$\hat{y} = \operatorname*{argmax}_{y \in \{0,1\}} \mathbb{P}[G, Y] = \operatorname*{argmax}_{y \in \{0,1\}} \mathbb{P}[G|Y]\mathbb{P}[Y] \tag{1}$$

where $\mathbb{P}[G|Y]$ is the *likelihood* of observing a graph given its class, and $\mathbb{P}[Y]$ is the prior probability of each class. By obtaining estimates of these two terms, one can build a plugin classifier:

$$\hat{y} = \operatorname*{argmax}_{y \in \{0,1\}} \widehat{\mathbb{P}}[G|Y]\widehat{\mathbb{P}}[Y] \tag{2}$$

where $\widehat{\mathbb{P}}[G|Y]$ and $\widehat{\mathbb{P}}[Y]$ are the plugin estimates of the likelihood and prior, respectively. Estimating the prior is trivial, one can simply use the maximum likelihood estimator, $\hat{\pi} = \widehat{\mathbb{P}}[Y = 1]$ and $1 - \hat{\pi} = \widehat{\mathbb{P}}[Y = 0]$. The likelihood term, however, is more complex. It can be expanded:

$$\mathbb{P}[G|Y] = \mathbb{P}[A, Q|Y] = \mathbb{P}[A|Y]\mathbb{P}[Q] \tag{3}$$

as the adjacency matrix, A, and permutation matrix Q, are conditionally independent given the class label, Y. Further, the permutation matrix is assumed to be independent of the class label. Thus, if Q were observed, then estimating $\mathbb{P}[A|Y]$ would be all there is to it; unfortunately, Q is latent, and thus must be imputed for each example.

To proceed, we define an explicit likelihood model, which we will use in the sequel. Because each element of the adjacency matrix is Binary, edges are Bernoulli random variables. We assume that each edge is independent, but not identical. If the permutation matrix was known to be the identity matrix, we would have:

$$\mathbb{P}[A, Q|Y] = \prod_{ij} \text{Bernoulli}(a_{ij}; \eta_{ij}^{y}) \tag{4}$$

However, because the permutation matrix is in general, not the identity, we instead have:

$$\mathbb{P}[A, Q|Y] = \prod_{ij} \text{Bernoulli}\bigg(\sum_{ij} q_{ij} q_{ji} a_{ij}; \eta_{ij}^{y} \bigg) \mathbb{P}[Q] \tag{5}$$

The permutation matrix has the following distribution:

$$\mathbb{P}[Q] = \sum_{Q \in \mathcal{Q}} \delta_Q w_Q \tag{6}$$

where δ_Q is an indicator function taking unity value whenever $Q' = Q$ and zero otherwise, and w_Q is the likelihood of any particular permutation matrix, where $w_Q \leq 0$ and $\sum_Q w'_Q = 1$. The simplest assumption is that each permutation matrix is equally likely, that is: $w'_Q = 1/n!$, because $n!$ is the number of $n \times n$ element permutation matrices.

The log-likelihood of $\mathbb{P}[A, Q|Y]$, ignoring the $\mathbb{P}[Q]$ term is:

$$\mathcal{L}(\boldsymbol{\eta}, Q) = \sum_{ij} \bigg\{ \Big[\sum_{ij} q_{ij} q_{ji} a_{ij} \ln \eta_{ij} \Big] + \Big[\Big(1 + \sum_{ij} q_{ij} q_{ji} a_{ij} \Big) \ln(1 - \eta_{ij}) \Big] \bigg\} \tag{7}$$

The goal is then to maximize the log-likelihood with respect to both $\boldsymbol{\eta}$ and Q:

$$(\hat{\eta}, \hat{Q}) = \underset{\boldsymbol{\eta} \in (0,1)^{n \times n}, Q \in \mathcal{Q}}{\text{argmax}} \mathcal{L}(\boldsymbol{\eta}, Q), \tag{8}$$

which is difficult in part due to the constraint on Q. But one can relax that constraint to allow Q to be any doubly-stochastic matrix, which is a natural relaxation, given that permutation matrices are the extreme points of the set of doubly stochastic matrices. A matrix is said to be doubly stochastic if both all the rows and all the columns sum to unity. Thus, to maximize the log-likelihood, we have:

$$(\hat{\eta}, \hat{Q}) \approx \underset{\eta \in (0,1)^{n \times n}, Q \in \mathcal{D}}{\text{argmax}} \mathcal{L}(\boldsymbol{\eta}, Q), \tag{9}$$

where $\mathcal{D}$ is the set of doubly stochastic matrices. Using Lagrange multipliers, we can rewrite :

$$(\hat{\eta}, \hat{Q}) \approx \underset{\eta \in (0,1)^{n \times n}}{\text{argmax}} \mathcal{L}(\boldsymbol{\eta}, Q) - \lambda(1 - \tfrac{1}{n} \mathbf{1}^{\mathsf{T}} Q) - \lambda(1 - \tfrac{1}{n} \mathbf{1}^{\mathsf{T}} Q^{\mathsf{T}}), \tag{10}$$

where the additional terms impose the constraint that the columns and rows sum to one, respectively. Now, the likelihood term is concave in η, and the constraints are linear, and therefore concave in Q, thus, one can use a coordinate ascend strategy, optimizing Q followed by optimizing η, until convergence. This ignores the Q terms in the likelihood. Perhaps somebody (else) could figure out how to (at least approximately) account for those. Regardless, this is an approximate solution to finding the most likely permutation matrix for each sample, as well as the independent edge probabilities.

2.4 Assignment problem approach

Instead of the likelihood based approach outlined above, one could consider the permutation matrices, Q^l, "nuisance parameters," and construct *some* way to estimate them, and plug that value in, assuming it is "correct." Note that the disadvantage of this approach relative to the likelihood based approach is that it is less "natural" from a statistical point of view, modeling averaging is not quite as obvious, incorporating prior information might also be less obvious. The advantage, however, is that much work as already been devoted to solving assignment problems, and we can therefore piggyback on the shoulders of giants.

More specifically, assume we have the same model assumed above in Section , and assume that η is known, such that we must only estimate the permutation matrix. This problem is closely related to the graph isomorphism problem (GIP), in which, given any pair of graphs, one searches for the permutation matrix that makes them as similar as possible:

$$\hat{Q} = \underset{Q \in \mathcal{Q}}{\text{argmin}} \left\| QAQ^{\mathsf{T}} - B \right\| \tag{11}$$

where A and B are two arbitrary adjacency matrices. Unfortunately, because the set of permutation matrices, $\mathcal{Q}$ is discrete, optimizing over them is difficult. In fact, solving GIP is known to be NP incomplete (meaning it is not known to be in either P or NP) [**?**]. Therefore, instead of trying to solve this exactly, we make approximations to make it easier.

First, as mentioned above, a natural relaxation of the set of permutation matrices is the set of doubly stochastic matrices, of which the permutation matrices are a proper subset (the extreme points, in fact). Thus, QIP can be relaxed:

$$\hat{Q} = \operatorname*{argmin}_{Q \in \mathcal{D}} \mathcal{L}(Q) = \operatorname*{argmin}_{Q \in \mathcal{D}} \left\| QAQ^{\mathsf{T}} - B \right\| \tag{12}$$

While less studied, it is clear that this is not a convex problem, because the Hessian of $\left\| QAQ^{\mathsf{T}} - B \right\|$ is given by:

$$\nabla^2_Q \mathcal{L} = B \otimes A + B^{\mathsf{T}} \otimes A^{\mathsf{T}}, \tag{13}$$

where $\otimes$ indicates the Kronecker product which is not positive definite (in general), meaning that gradient ascent is not guaranteed to yield a global optimum, rather only a local optimum. The Frank-Wolfe (FW) algorithm is a successive linear programming algorithm specifically designed to solve quadratic programming problems with linear constraints [**?**]. Here, the problem is not quadratic in Q, rather, it is a 4th order polynomial function of Q. Thus, FW will not (in general) achieve the global optimal. Regardless, it proceeds as follows:

Algorithm 1 (Frank-Wolfe Algorithm (FW))**.** *Given $Q^{(1)} \in \mathcal{D}_m$.*

Step 0: Let k = 1.

Step 1: Let $\nabla_Q^{(k-1)} = AQ^{(k-1)}B^{\mathsf{T}} + A^{\mathsf{T}}Q^{(k-1)}B$.

Step 2: Compute $W^{(k)}$ by solving the linear assignment problem:

$$W^{(k)} = \operatorname*{argmin}_{W^{(k)} \in \mathcal{D}} \sum_{i,j=1}^{m} \left(\nabla_Q^{(k-1)} \circ W^{(k)} \right)_{ij} \tag{14}$$

Step 3: Let $d^{(k)} = W^{(k)} - Q^{(k-1)}$.

Step 4: Find $\alpha^{(k)} \in [0,1]$ such that

$$\alpha^{(k)} = \operatorname*{argmin}_{\alpha^{(k)}} \mathcal{L}(Q^{(k)} + \alpha^{(k)} d^{(k)}).$$

Step 5: Let

$$Q^{(k+1)} = Q^{(k)} + \alpha^{(k)} d^{(k)}.$$

Step 6: If $\|d\|_P^{(k)} \| \leq \| \epsilon$ or $\alpha^{(k)} = 0$ or $k = k_{\max}$ then stop; else k = k + 1 and go to step 1.

The above algorithm is merely Netwon's method, applied to the this problem, with linear constraints. With each step, $Q^{(k)}$ is not (in general) a permutation matrix. One can project $Q^{(k)}$ into the space of permutation matrices at each step, or one can merely project the final $Q^{(k_{max})}$

Remark 1.

At each iteration, $Q^{(k)}$ is an extreme point of the constraint set, a permutation matrix. However, in general, the iterates $Q^{(k)}$ remain in the interior of the constraint set. ((XXX: i'm confused by this. $Q^{(k)}$ should *be an extreme point, not is an extreme point, yes?))*

Step 3 is a "line search;" however, as the objective function is linear (XXX: right?) the optimum α can be found exactly.

The above algorithm is simply Newton's method, applied to some non-convex objective function, with linear constraints.

Once FW has terminated the nearest permutation to the double stochastic matrix, $X^{(k)}$ can be found as given by the following Lemma.

3 other

Finding the best alignment of vertices has been called the graph isomorphism problem (GIP) [**?**], and is known to be computationally equivalent to the quadratic alignment problem (QAP) [**?**]. Given to graphs, A and B, the QAP can be formalized as follows:

$$\hat{P} = \underset{P \in \Pi}{\operatorname{argmin}} \left\| PAP^{\mathsf{T}} - B \right\| \tag{15}$$

where Π is the set of all permutation matrices, that is, all matrices with a single one in each row and column, and zeros otherwise.

We therefore investigate building classifiers on the space of collections of unlabeled graphs. Specifically, we consider two distinct approaches. First, we build classifiers using "graph invariants," that is, statistics of the graphs that are invariant to isomorphic translations. Second, we first approximately solve the graph isomorphism problem,

Data today are unlabeled graphs. examples (mostly brain-graph examples)....

Previous work classifying unlabeled graphs..... Problem: consistency for what model?

Previous work classifying labeled graphs.... Problem: graph isomorphism

Previous work solving graph isomorphism problem.

Paper organization...

4 Preliminaries

Consider $(G, Y), \{(G_i, Y_i)\}_{i=1}^{s} \overset{iid}{\sim} \mathbb{P}_{GY}$, with classes $Y : \Omega \to \{0, 1\}$ and (labeled) graphs $G : \Omega \to \mathcal{G}_n$, where $\mathcal{G}_n$ denotes the collection of simple (labeled) graphs on $V = [n]$.

NB: We consider simple graphs – unweighted, undirected, with no loops, so the adjacency matrices are binary, symmetric, and hollow; our connectome applications may involve directed, loopy, weighted, attributed, multi graphs . . . (And I use directed loopy graphs in my example Section 7 . . . perhaps we should use directed loopy throughout? or undirected unloopy? what say you, John???)

The collection $\mathcal{T} \equiv \{(G_i, Y_i)\}_{i=1}^{s}$ is the training sample and s is the training sample size; G is the graph to be classified, and Y is the true but unobserved class label. For simplicity, we will assume that the prior probability of class membership $\pi \equiv P[Y = 1]$ is known to be $1/2$, and the class-conditional sample sizes $S_y \equiv \sum_{i=1}^{s} I\{Y = y\}$ are fixed (s_y) rather than random variables (S_y) with s even and $s_0 = s_1 = s/2$.

We consider the independent edge model (IE), so that for $y \in \{0, 1\}$ the class-conditional distribution $F_{G|Y=y}$ is parameterized by a (symmetric, hollow) $n \times n$ matrix P_y with entries $p_{y;u,v} \in [0, 1]$.

(NB: We will eventually *generalize* the independent edge model to RDPG . . . and beyond! But first, herein, we will *simplify* to independent edge block model (IEBM), for mathematical expediency.)

For this IE model, the Bayes optimal classifier for observed graph G (equivalently, for observed (symmetric, hollow) $n \times n$ adjacency matrix $A = A(G) = [a_{u,v}]$) is given by

$$g^*(G) \quad = \quad \arg\max_{y} \prod_{(u,v) \in \binom{V}{2}} f(a_{u,v}; p_{y;u,v}), \tag{16}$$

where the Bernoulli probability $f(a; p)$ is given by $f(a; p) = p^a(1 - p)^{1-a}$.

Alas, the graph G is not observed; rather, we observe the *unlabeled* version. (right. we don't observe the class label. but here i mean "unlabeled" in the "assignment problem" sense. too.) That is, rather than observing the adjacency matrix A, we observe $\widetilde{A} \equiv QAQ^T$ for some unknown permutation matrix Q.
(NB: sorry John. because of all the Ps, i'm gonna use Q for permutation matrices. deal with it?)

5 An Assignment Problem Application

(This is Assignment Problem Application #1. E.g., voxels = vertices, and voxels have been morphed to anatomical regions as per JV's description. If we assume for #1 that this region assignment is perfect, we have the within-region vertex assignment problem. This is this . . . we disussed adding distance $d(u, u')$ and assuming that true assignment is more likely for smaller distances; we discussed relaxing the perfect region assignment to $P[u \in \text{correct region}]$; we discussed other generalizations?)

Consider the observed $\widetilde{A} \equiv QAQ^T$. For $i \in [s]$, let

$$\widehat{Q}_i = \arg\min_{Q'} ||Q'^T \widetilde{A} Q' - A_i||_F. \tag{17}$$

For each pair (u, v), let $\sigma_i(u, v)$ be the reassignment through Q and $\widehat{Q}_i$. That is, entries $a_{u,v}$ in A are out of place due to unlabeledness in $\widetilde{A}$, and the assignment minimization attempts to put them back into place; $\sigma_i(u, v)$ is the result of this attempt – the assignment provided by $\widehat{Q}_i QAQ^T \widehat{Q}_i^T$.

Definition 1. *The* naive assignment classifier *for the observed $\widetilde{A}$ is given by*

$$g(G; \mathcal{T}) = \arg\max_y \max_{i:Y_i=y} \prod_{(u,v)\in\binom{V}{2}} f(a_{\sigma_i(u,v)}; p_{y;u,v}). \tag{18}$$

Note 1: The classifier $g(G; \mathcal{T})$ presented in Definition 1 assumes that the class-conditional edge probabilities $p_{y;u,v}$ are known; however, these probabilities are not used in the assignment equation 17. (They could be? But they're not!)

Note 2: We could employ a plug-in classifier, using estimates in both the classifier and the assignment; e.g., $\widehat{p}_{y;u,v} = (s_y)^{-1} \sum_{i:Y_i=y} a_{i;u,v}$ and $\overline{A}_y = (s_y)^{-1} \sum_{i:Y_i=y} A_i$. However, we would want to use *smoothed* estimates in the classifier (to avoid degeneracy when $\widehat{p}_{y;u,v}$ equals 0 or 1) and *unsmoothed* estimates in the assignment. This complicates the evaluation analysis; we leave this more complicated (and more realistic) investigation for the future.

Note 3: The classifier $g(G; \mathcal{T})$ presented in Definition 1 uses only the single probability-maximizing training assignment for each class. This could be generalized either in the assignment (using $\overline{A}_y$) or by processing the collection $\{\prod_{(u,v)\in\binom{V}{2}} f(a_{\sigma_i(u,v)}; p_{y;u,v})\}_{i:Y_i=y}$ with methods more elaborate than the simple maximum.

Note 4: We could also use nearest neighbor classifier … but i think this would be less tractable.

The advantage of the classifier $g(G; \mathcal{T})$ presented in Definition 1 is that the assignment methodology and only the assignment methodology is on trial! Better classifiers could be considered, but I'm trying to design a tractable investigation of assignment methodologies …

Under IE, the difference between $F_{G|Y=1}$ and $F_{G|Y=0}$ is wholly captured by the collection of marginal "signal edges" probabilities

$$\mathcal{E} \equiv \{(u, v) \in \binom{V}{2} : p_{0;u,v} \neq p_{1;u,v}\}.$$

This collection $\mathcal{E}$ might be all of $\binom{V}{2}$. We hereby simplify IE to independent edge block model (IEBM), for mathematical expediency. Let $\mathcal{E} = \binom{V'}{2}$ for a collection V' of $1 \leq m \leq n$ vertices, and define IEBM(n, p, m) to be the model $\mathbb{P}_{GY}$ defined by class-conditional probabilities $p_{0;u,v} = p \neq 1/2$ and $p_{1;u,v} = 1 - p$ for all $(u, v) \in \mathcal{E}$ and $p_{0;u,v} = p_{1;u,v} = 1/2$ for all $(u, v) \in \binom{V}{2} \setminus \mathcal{E}$. (In this case, all signal edges are created equally and all noise edges are created equally.) Notice that IEBM(n, p, m) requires that $\mathcal{E}$ is a block – the signal edges consist of precisely the potential interconnections between a set of m vertices.

Let $s = 2$ (one single training observation from each class). In this case, since all signal edges are created equally and all noise edges are created equally, the performance of the classifier $g(G; \mathcal{T})$ is monotonic in the number of signal edges recovered by σ_1 and σ_2.

Let the random variable $L(g) \equiv P[g(G; \mathcal{T}) \neq Y|\mathcal{T}]$ be the probability of misclassification for classifier g conditioned on the training sample [see DGL 1996]. Under IEBM(n, p, m) with $s = 2$, we have that $L(g)$ depends on only n, p, m. Define $L^* \equiv L(g^*)$.

Theorem 1. *For $\mathbb{P}_{GY} \in$ IEBM(n, p, m), $L(g|T = t) < L(g|T = t - 1)$ for all $t \in [2m - 1]$, where*

$$T_i \equiv |\{(u, v) \in \mathcal{E} : \sigma_i(u, v) \in \mathcal{E}\}| \tag{19}$$

and

$$T \equiv T_1 + T_2.$$

Proof: (Proof of this (alleged) monotonicity requires but a simple modification to Henry's proof?)

So, for this simple case, we need concern ourselves with only the performance of the assignment algorithm in terms of T_i.

Theorem 2. *$T_1 =^{\mathcal{L}} T_2$ for this simple case.*

Proof: (By construction? it's suppose to . . . that's why i set up the class-conditional edge probabilities $p_{y;u,v}$ to be reflective about $1/2$ in y.)

6 Performance Degradation

What is the performance degradation due to unlabeled-ness?

Case I:

Theorem 3. *$P[\widehat{Q}_i = Q] = 1$ for all i implies $L(g) = L^*$.*

Proof: If $P[\widehat{Q}_i = Q] = 1$ for all i (that is, if the assignment algorithm gets the *right* answer – not to be confused with the *best* answer in terms of the optimization) then $T_1 = T_2 = |\mathcal{E}|$ and hence $L(g) = L^*$.

Case II:

How about when the assignment algorithm gets the *best* answer in terms of the optimization? Perhaps we can work this out – a theorem/proof?

Theorem 4. *$L_{n,p,\mathcal{E}}(g) =$ some *identifiable* function of $n, p, \mathcal{E}$?*

According to my calculations, the only tricky bit is $I\{T_i(g_1, g_2) = t\}$ given two graphs $g_1, g_2 \in \mathcal{G}_n$. That is, given two graphs (no randomness), T_i either equals t or it doesn't (after (arbitrarily?) accounting for non-uniqueness of assignment solution $\widehat{Q}$). This looks like i could do it for $n = 3$. But then the combinatorics get silly. BUT: maybe this is one of those things for which generating function folks could derive a *generating function* . . . ? at least for my simple stochastic block model class-conditional distributions?

After Case I (the assignment algorithm gets the *right* answer – not to be confused with the *best* answer in terms of the optimization) and Case II (the assignment algorithm gets the *best* answer in terms of the optimization), we will investigate approximation assignment algorithms based on the trade-off between (1) computational complexity and (2) classification performance (either $L(g)$ directly, or in terms of the distribution of T_i).

The Monte Carlo example presented below (Section 7) demonstrates significant but not complete performance degradation due to a particular algorithm (**lp.assign** in **R** package **lpSolve**).

7 Example

A Monte Carlo experiment (20000 paired replications) demonstrates that, (using **directed loopy** graphs for simplicity) with $n = 10, p = 0.25$, and $|\mathcal{E}| = 9$ (where $\mathcal{E}$ is in fact a 3×3 block) the performance degradation due to the application of **lp.assign** in **R** package **lpSolve** (with $Q = I$ specifying starting point) is from $\widehat{L} = 0.0476 \approx L^* = 1 - F_{Binomial(9,0.25)}(4) = 0.04892731\ldots$ to $\widehat{L} = 0.28855$. So better-than-chance classification is achieved for our unlabeled scenario using this assignment algorithm, but performance is significantly degraded.

NB: should also report performance in terms of T_i. and objective value at solution?

Code for this example is provided in the Appendix.

NB: LAP is not QAP; i'd be happy to have QAP R code . . .

8 Proposal

I propose that we investigate, via theory, simulation, and experiment, the trade-off between computational complexity and performance, and also identify the relationship between the explicit assignment objective function and the exploitation task (classification) objective function.

Except for Cases I (the assignment algorithm gets the *right* answer – not to be confused with the *best* answer in terms of the optimization) and II (the assignment algorithm gets the *best* answer in terms of the optimization), I'm not sure what we'll be able to prove. Perhaps

Theorem 5. *LAP is as good as QAP $\iff$ model $\in$ IEBM?*

But we can do simulation analysis: first, for my simple scenaro; then, generalizing to (perhaps) approach experimental settings?

NB: perhaps we should be doing hypothesis testing instead???

9 20 statements

```
allow me to try to state some things that it seems we agree on, and some things that seem t
with the hope of clarifying at least where we are, and where we'd like to be.  if anything

(1) The "quadratic assignment problem" (QAP) is the following:
give two matrices, A and B,
find a Q and P such that
min ||QAP' - B||
where both Q and P are permutation matrices.


JMC: Actually, this is the bi-linear assignment problem for the QAP P=Q.
The bilinear problem has the pleasant property that the relaxed version
(replacing the permutation constraint with the doubly stochastic constraint)
has its solutions on the vertices.  That is when you solve the
bilinear assignment
your are guaranteed to find a vertex.  The solution may be a local optimum and
multiple starts may still be necessary.

JoVo: modified first two points
(1) The "bilinear assignment problem" (BAP) is the following:
give two matrices, A and B,
find a Q and P such that
min ||QAP' - B||
where both Q and P are permutation matrices.

(2) The Quadratic Assignment Problem (QAP) is the following:

given two adjacency matrices, A and B,
find a P such that
min || PAP' - B||
where P is a permutation matrix.
thus, QAP is a constrained BAP.

(3) the graph isomorphism problem is a QAP


(2) QAP is NP-hard

(3) The Graph Isomorphism Problem (GIP) is the following:
given two adjacency matrices, A and B,
find a P such that
min || PAP' - B||
where P is a permutation matrix.
thus, GIP is a constrained QAP.
```

```
(4) GIP has weird complexity, somewhere between NP-complete and P

(4) The Linear Assignment Problem (LAP) is the following:
given two adjacency matrices, A and B,
find a P such that
min || PA - B||
where P is a permutation matrix.

(5) LAP can be solved in O(n^3) by the hungarian algorithm

(6) The Frank-Wolfe (FW) algorithm is an algorithm designed to *solve* quadratic problems w
min f(x) = 0.5* x' E x + h' x
where x is in some feasible region defined by a set of linear constraints, that is Ax \leq

(7) FW is an iterative algorithm that works as follows

(i) initialize x with x_0
(ii) compute the first order taylor expansion around f(x_k)
(iii) compute the gradient of f(x_k), call that g(x_k)
(iv) solve the following subproblem:
min f(x_k) + g(x_k) xbar_k, where xbar_k is in the feasible region
(v) compute the step size, lambda that solves
min f(x_k + lambda (xbar_l - x_k)) subject to lambda is in (0,1)
(vi) let x_{k+1} = x_k + lambda*(xbar_k - x_k), and let k=k+1

we stop if ever g(x_k)=0 or lambda=0.

(8) FW can be quite slow.  although the first few iterations are often fast.

(9) doubly stochastic matrices have the following properties:

(i) each row sums to 1
(ii) each column sums to 1
(iii) each element is a non-negative real number

these can each be written as linear constraints (linear equalities, in fact).

(10) the set of doubly stochastic matrices is a convex polytope, and is the convex hull of

(11) permutation matrices are square matrices with exactly one 1 in each row and one 1 in e

(12) thus, one can solve the following relaxation of GIP, called the Doubly Stochastic Appr
Yhat = min || YAY' - B||
where Y is the set of doubly stochastic matrices.
in particular, FW can solve this problem in linear time.

(13) it is possible that the solution to DSA is in fact also a solution to GIP, because the

(14) if Yhat is not a permutation matrix, then one can find the closest permutation matrix
min || Yhat - X ||
which can be solved in polynomial time using the hungarian algorithm.

(15) we have found a case for which we can show in simulo:
L* ~ E[L_{labeled}] << E[L_{LAP}] << 1/2
```

(16) question 1: where does E[L_{QAP}] fit in for this simulation. this is a question we c

(17) question 2: where does E[L_{DSA}] fit in, where L_{DSA} is the misclassification rate

(18) question 3: since DSA takes a long time, let DSA_k indicate the solution to DSA from t

(19) question 4: is it the case that the solution to DSA_1always equals the solution to LAI

(20) question 5: how does conroy's algorithm fit into all of this? it seems that FW will ev

i hope this missive at least clarifies what i believe and don't understand. i also hope it

shabbat shalom to all,
jovo

10 email from jmc

John Conroy <conroyjohnm@gmail.com> Sat, Sep 18, 2010 at 11:15 PM
To: Carey Priebe <cep@jhu.edu>, joshuav <joshuav@jhu.edu>
Hi Carey and Joshua,
Here's a rough draft description of the FW method for QAP. I adapted from the
original late 90's unpublished paper by Lou, Steve, and I. I suspect that
once this is merged into the paper much can be cut out and some may be appropriate
for an appendix.   I think it would be good to include the performance of the FW method
on the QAP test set to justify the method.

Here's a couple of points relative to our discussion on Thursday:

1. Indeed, the subproblem
FW solves is a linearization of the quadratic function, a Taylor series about $X^{(k)}$.
This sub-problem's solution then produces a vertex which give a
search direction for the "exact line search."
2. Furthermore, I am convinced that the
relaxed QAP is NOT in general a convex optimization problem. The Hessian is
kron(A,B)+kron(A',B') and in general this will not be a positive definite matrix.
(A sufficient condition is for A and B to be symmetric positive definite or for them
both to be symmetric negative definite.)

I sent a copy of this draft to Lou and Steve as well
with some background where we intend to
go next.

Best regards,
Johnny

Appendix

Code producing the example results presented in Section 7.

```
library(lpSolve)
cuc = function(thisclass,myseed=1)
# Classification of Unlabeled Connectomes
{
set.seed(myseed)
n=10
nmc=10000
# directed, with loops
P1 = P2 = matrix(0.5,n,n) # class-conditional distribution specification
P1[1:3,1:3] = 0.25
P2[1:3,1:3] = 0.75
label = labeltilde = tie = tietilde = rep(0,nmc)
Qhat1lpobjval = Qhat2lpobjval = NULL
for(mc in 1:nmc)
{
G1 = matrix(rbinom(n^2,1,P1),n,n)
G2 = matrix(rbinom(n^2,1,P2),n,n)
if(thisclass == 1 )
 G  = matrix(rbinom(n^2,1,P1),n,n)
if(thisclass == 2 )
 G  = matrix(rbinom(n^2,1,P2),n,n)
Q = matrix(0,n,n)
diag(Q) = 1 # Q == I
Gtilde = Q %*% G %*% t(Q)
C1 = C2 = matrix(0,n,n) # cost
for(i in 1:n) for(j in 1:n)
 {
 C1[i,j] = sum(abs(Gtilde[i,]-G1[j,]))
 C2[i,j] = sum(abs(Gtilde[i,]-G2[j,]))
 }
Qhat1lp = lp.assign(C1)
Qhat2lp = lp.assign(C2)
Qhat1 = Qhat1lp$solution
Qhat2 = Qhat2lp$solution
Qhat1lpobjval[mc] = Qhat1lp$objval
Qhat2lpobjval[mc] = Qhat2lp$objval
sigma1 = t(Qhat1) %*% Gtilde
sigma2 = t(Qhat2) %*% Gtilde
# now ... classify G and Gtilde
p1 = prod( (P1^G) * ((1-P1)^(1-G)) )
p2 = prod( (P2^G) * ((1-P2)^(1-G)) )
p1tilde = prod( (P1^sigma1) * ((1-P1)^(1-sigma1)) )
p2tilde = prod( (P2^sigma2) * ((1-P2)^(1-sigma2)) )
if(p1>p2) label[mc]=1
if(p1==p2) tie[mc]=1
if(p1tilde>p2tilde) labeltilde[mc]=1
if(p1tilde==p2tilde) tietilde[mc]=1
}
return(list(label,tie,labeltilde,tietilde))
}

cuc1 = cuc(1)
cuc2 = cuc(2)
```

```
sum(cuc1[[1]])
sum(cuc1[[2]])
sum(cuc1[[3]])
sum(cuc1[[4]])
# [1] 9524
# [1] 0
# [1] 6986
# [1] 121

sum(cuc2[[1]])
sum(cuc2[[2]])
sum(cuc2[[3]])
sum(cuc2[[4]])
# [1] 476
# [1] 0
# [1] 2759
# [1] 117

(10000 - sum(cuc1[[3]]) - .5*sum(cuc1[[4]]) + sum(cuc2[[3]]) + .5*sum(cuc2[[4]]))/20000
# [1] 0.28855

# L*:
# > 1-pbinom(4,9,.25)
# [1] 0.04892731
```

10.1 move this . . .

Under IE, the difference between $F_{G|Y=1}$ and $F_{G|Y=0}$ is wholly captured by the collection of marginal "signal edges" probabilities

$$\mathcal{E} \equiv \{(u,v) \in \binom{V}{2} : p_{0;u,v} \neq p_{1;u,v}\}.$$

$$\begin{aligned} g^*(G) &= \arg\max_y \prod_{(u,v)\in\binom{V}{2}} f(a_{u,v}; p_{y;u,v}) && (20) \\ &= \arg\max_y \prod_{(u,v)\in\mathcal{E}} f(a_{u,v}; p_{y;u,v}), && (21) \end{aligned}$$

If we estimate $p_{y;u,v}$ from the training data, we may consider classifiers

$$g_{NB}(G;\mathcal{T}) = \arg\max_y \prod_{(u,v)\in\binom{V}{2}} f(a_{u,v}; \widehat{p}_{y;u,v}) \tag{22}$$

and

$$g_{\mathcal{E}}(G;\mathcal{T}) = \arg\max_y \prod_{(u,v)\in\mathcal{E}} f(a_{u,v}; \widehat{p}_{y;u,v}). \tag{23}$$

NB: requires *smoothed* estimates $\widehat{p}_{y;u,v}$, to avoid degeneracy when $\widehat{p}_{y;u,v}$ equals 0 or 1.

The latter classifier, $g_{\mathcal{E}}(G;\mathcal{T})$, is the best we can hope for – it considers the signal edges and only the signal edges; the former can be swamped by noise from non-signal edges.

If the estimates $\widehat{p}_{y;u,v}$ are consistent (converge to $\widehat{p}_{y;u,v}$ as $s \rightarrow \infty$), then both of these classifiers are consistent (converge to Bayes optimal); that is, $L(g) \rightarrow L(g^*) \equiv L^*$, where the random variable $L(g) \equiv P[g(G) \neq Y|\{(G_i,Y_i)\}_{i=1}^s]$ is the probability of misclassification for classifier g conditioned on the training sample [see DGL 1996]. Note that $g_{\mathcal{E}}(G;\mathcal{T})$ should dominate $g_{NB}(G;\mathcal{T})$.

11 Simulations

```
n = 10; p = 0.5; m = 5; q0 = 0.2; q1 = 0.8; s(test) = 100; s(train) = 2;
```

For each of the 100 test samples, we first tried to solve GIP with respect to each of the two training samples. Then, given the solution, we tried to classify. Note that because the number of training samples is 2, the parameter estimates are degenerate, we therefore add or subtract epsilon to ensure they are not 0 or 1. The MAP estimates would likely perform better. I will have that code working soon.

12 Results

13 Some misc thoughts

Classifying labeled graphs has already been solved (VP10a, VLP10b).

Two broadly different approaches to classifying unlabeled graphs: i) first (approximately) solve graph isomorphism and then use standard graph classification problems, and ii) classify using graph invariants.

The first approach can be universally consistent, interpretable, etc. Yet, it might take too long. The second approach might have good performance, but lacks consistency results.

Given two unlabeled graphs with n vertices, A, and B, the graph isomorphism problem can be written as:

$$\hat{P} = \operatorname*{argmin}_{P\in\pi} \left\|PAP^{\mathsf{T}} - B\right\| \tag{24}$$

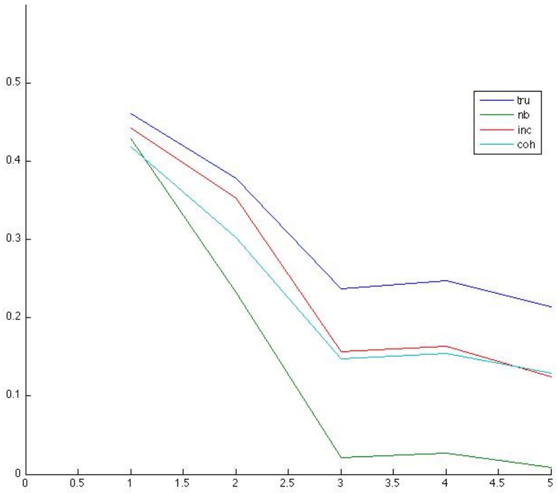

Figure 1: caption

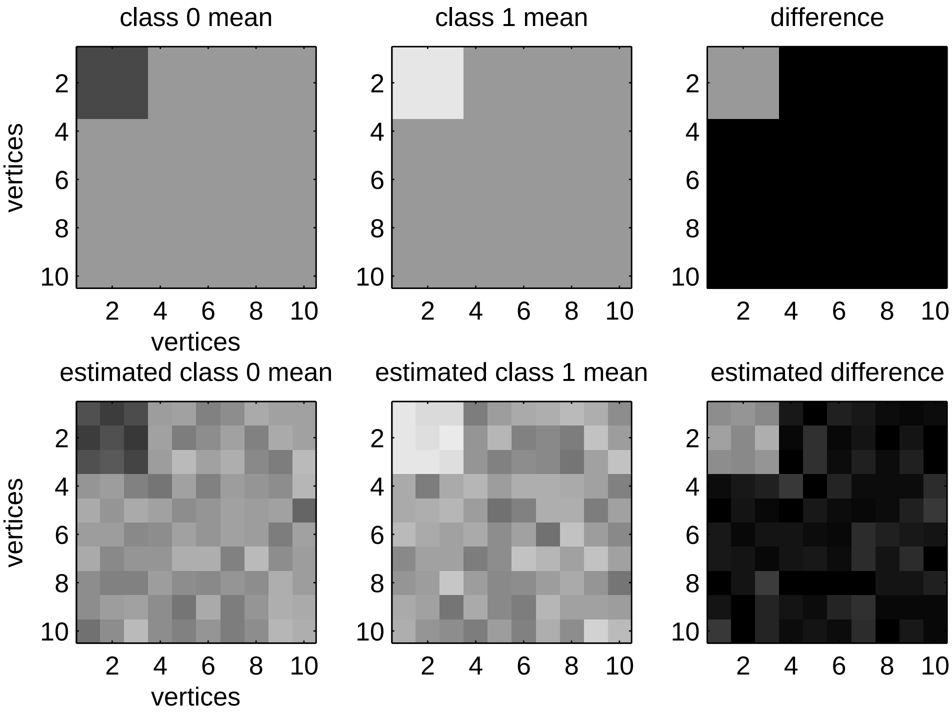

Figure 2: Top row: true parameters for simulation 1. Bottom row: estimates using all training data (assuming vertices are labeled)

where π is the set of permutation matrices ($0-1$ matrices with a single 1 in each row and column). Some people know (ref?) that this problem is nearly NP, and the fastest known algorithms to solve this exactly scale exponentially with n. One could instead solve:

$$\hat{X} = \operatorname*{argmin}_{X \in \mathcal{D}} \left\| XAX^{\mathsf{T}} - B \right\| \tag{25}$$

where $\mathcal{D}$ is the set of doubly stochastic matrices (matrices with non-negative elements whose rows and columns sum to one), which is a superset of permutation matrices. In fact, the permutation matrices are the extreme points of the set of doubly stochastic matrices. Thus, sometimes solving (25) gives you the solution to (24). So, one way to approximately solve (24) is to first solve (25), and then project $\hat{X}$ onto the set of permutation matrices:

$$\hat{P} = \operatorname*{argmin}_{P \in \pi} \left\| P - \hat{X} \right\| \tag{26}$$

Both (25) and (26) can be solved in $\mathcal{O}(n^3)$ using the Hungarian algorithm. However, sometimes $\hat{X}$ is far from any extreme point in π, thus the projection might end up somewhere bad.

According to JCM, (25) is not log-concave, so the initial starting point matters significantly. Thus, multiple restarts, each time starting with a doubly stochastic matrix, can improve results.

JCM fills in here how his algorithm works, as I don't yet understand it.

For classification, we wonder: for what models do multiple restarts help?

In the kidney-egg classification problem, $n - m$ of the vertices can be arbitrarily permuted without loss of classification accuracy (as they are exchangeable). However, in the more general independent edge model, no vertices may be exchangeable with one another, and in such cases, multiple restarts may help significantly.