Semi-Supervised Learning with Graphs

Xiaojin Zhu
May 2005
CMU-LTI-05-192

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
zhuxj@cs.cmu.edu

DOCTORAL THESIS

THESIS COMMITTEE
John Lafferty, Co-chair
Ronald Rosenfeld, Co-chair
Zoubin Ghahramani
Tommi Jaakkola, MIT

Abstract

In traditional machine learning approaches to classification, one uses only a labeled set to train the classifier. Labeled instances however are often difficult, expensive, or time consuming to obtain, as they require the efforts of experienced human annotators. Meanwhile unlabeled data may be relatively easy to collect, but there has been few ways to use them. Semi-supervised learning addresses this problem by using large amount of unlabeled data, together with the labeled data, to build better classifiers. Because semi-supervised learning requires less human effort and gives higher accuracy, it is of great interest both in theory and in practice.

We present a series of novel semi-supervised learning approaches arising from a graph representation, where labeled and unlabeled instances are represented as vertices, and edges encode the similarity between instances. They address the following questions: How to use unlabeled data? (label propagation); What is the probabilistic interpretation? (Gaussian fields and harmonic functions); What if we can choose labeled data? (active learning); How to construct good graphs? (hyperparameter learning); How to work with kernel machines like SVM? (graph kernels); How to handle complex data like sequences? (kernel conditional random fields); How to handle scalability and induction? (harmonic mixtures). An extensive literature review is included at the end.

Acknowledgments

First I would like to thank my thesis committee members. Roni Rosenfeld brought me into the wonderful world of research. He not only gave me valuable advices in academics, but also helped my transition into a different culture. John Lafferty guided me further into machine learning. I am always impressed by his mathematical vigor and sharp thinking. Zoubin Ghahramani has been a great mentor and collaborator, energetic and full of ideas. I wish he could stay in Pittsburgh more! Tommi Jaakkola helped me by asking insightful questions, and giving me thoughtful comments on the thesis. I enjoyed working with them, and benefited enormously from the interactions with them.

I spent nearly seven years in Carnegie Mellon University. I thank the following collaborators, faculties, staffs, fellow students and friends, who made my graduate life a very memorable experience: Maria Florina Balcan, Paul Bennett, Adam Berger, Michael Bett, Alan Black, Avrim Blum, Dan Bohus, Sharon Burks, Can Cai, Jamie Callan, Rich Caruana, Arthur Chan, Peng Chang, Shuchi Chawla, Lifei Cheng, Stanley Chen, Tao Chen, Pak Yan Choi, Ananlada Chotimongicol, Tianjiao Chu, Debbie Clement, William Cohen, Catherine Copetas, Derek Dreyer, Dannie Durand, Maxine Eskenazi, Christos Faloutsos, Li Fan, Zhaohui Fan, Marc Fasnacht, Stephen Fienberg, Robert Frederking, Rayid Ghani, Anna Goldenberg, Evandro Gouvea, Alexander Gray, Ralph Gross, Benjamin Han, Thomas Harris, Alexander Hauptmann, Rose Hoberman, Fei Huang, Pu Huang, Xiaoqiu Huang, Yi-Fen Huang, Jianing Hu, Changhao Jiang, Qin Jin, Rong Jin, Rosie Jones, Szu-Chen Jou, Jaz Kandola, Chris Koch, John Kominek, Leonid Kontorovich, Chad Langley, Guy Lebanon, Lillian Lee, Kevin Lenzo, Hongliang Liu, Yan Liu, Xiang Li, Ariadna Font Llitjos, Si Luo, Yong Lu, Matt Mason, Iain Matthews, Andrew McCallum, Uwe Meier, Tom Minka, Tom Mitchell, Andrew W Moore, Jack Mostow, Ravishankar Mosur, Jon Nedel, Kamal Nigam, Eric Nyberg, Alice Oh, Chris Paciorek, Brian Pantano, Yue Pan, Vasco Calais Pedro, Francisco Pereira, Yanjun Qi, Bhiksha Raj, Radha Rao, Pradeep Ravikumar, Nadine Reaves, Max Ritter, Chuck Rosenberg, Steven Rudich, Alex Rudnicky, Mugizi Robert Rwebangira, Kenji Sagae, Barbara Sandling, Henry Schneiderman, Tanja Schultz, Teddy

Seidenfeld, Michael Seltzer, Kristie Seymore, Minglong Shao, Chen Shimin, Rita Singh, Jim Skees, Richard Stern, Diane Stidle, Yong Sun, Sebastian Thrun, Stefanie Tomko, Laura Mayfield Tomokiyo, Arthur Toth, Yanghai Tsin, Alex Waibel, Lisha Wang, Mengzhi Wang, Larry Wasserman, Jeannette Wing, Weng-Keen Wong, Sharon Woodside, Hao Xu, Mingxin Xu, Wei Xu, Jie Yang, Jun Yang, Ke Yang, Wei Yang, Yiming Yang, Rong Yan, Rong Yan, Stacey Young, Hua Yu, Klaus Zechner, Jian Zhang, Jieyuan Zhang, Li Zhang, Rong Zhang, Ying Zhang, Yi Zhang, Bing Zhao, Pei Zheng, Jie Zhu. I spent some serious effort finding everyone from archival emails. My apologies if I left your name out. In particular, I thank you if you are reading this thesis.

Finally I thank my family. My parents Yu and Jingquan endowed me with the curiosity about the natural world. My dear wife Jing brings to life so much love and happiness, making thesis writing an enjoyable endeavor. Last but not least, my ten-month-old daughter Amanda helped me ty pe the ,manuscr ihpt .

Contents

Chapter 1

Introduction

1.1 What is Semi-Supervised Learning?

The field of machine learning has traditionally been divided into three sub-fields:

- unsupervised learning. The learning system observes an unlabeled set of items, represented by their features $\{x_1, \ldots, \mathbf{x}_n\}$. The goal is to organize the items. Typical unsupervised learning tasks include clustering that groups items into clusters; outlier detection which determines if a new item x is significantly different from items seen so far; dimensionality reduction which maps x into a low dimensional space, while preserving certain properties of the dataset.

- supervised learning. The learning system observes a labeled training set consisting of (feature, label) pairs, denoted by $\{(x_1, y_1), \ldots, (x_n, y_n)\}$. The goal is to predict the label y for any new input with feature x. A supervised learning task is called regression when $y \in \mathbb{R}$, and classification when y takes a set of discrete values.

- reinforcement learning. The learning system repeatedly observes the environment x, performs an action a, and receives a reward r. The goal is to choose the actions that maximize the future rewards.

This thesis focuses on classification, which is traditionally a supervised learning task. To train a classifier one needs the labeled training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$. However the labels y are often hard, expensive, and slow to obtain, because it may require experienced human annotators. For instance,

- Speech recognition. Accurate transcription of speech utterance at phonetic level is extremely time consuming (as slow as 400$\times$RT, i.e. 400 times longer

than the utterance duration), and requires linguistic expertise. Transcription at word level is still time consuming (about 10×RT), especially for conversational or spontaneous speech. This problem is more prominent for foreign languages or dialects with less speakers, when linguistic experts of that language are hard to find.

- Text categorization. Filtering out spam emails, categorizing user messages, recommending Internet articles – many such tasks need the user to label text document as 'interesting' or not. Having to read and label thousands of documents is daunting for average users.

- Parsing. To train a good parser one needs sentence / parse tree pairs, known as treebanks. Treebanks are very time consuming to construct by linguists. It took the experts several years to create parse trees for only a few thousand sentences.

- Video surveillance. Manually labeling people in large amount of surveillance camera images can be time consuming.

- Protein structure prediction. It may take months of expensive lab work by expert crystallographers to identify the 3D structure of a single protein.

On the other hand, unlabeled data x, without labels, is usually available in large quantity and costs little to collect. Utterances can be recorded from radio broadcast; Text documents can be crawled from the Internet; Sentences are everywhere; Surveillance cameras run 24 hours a day; DNA sequences of proteins are readily available from gene databases. The problem with traditional classification methods is: **they cannot use unlabeled data to train classifiers**.

The question *semi-supervised learning* addresses is: given a relatively small labeled dataset $\{(x, y)\}$ and a large unlabeled dataset $\{x\}$, can one devise ways to learn from both for classification? The name "semi-supervised learning" comes from the fact that the data used is between supervised and unsupervised learning. Semi-supervised learning promises higher accuracies with less annotating effort. It is therefore of great theoretic and practical interest. A broader definition of semi-supervised learning includes regression and clustering as well, but we will not pursued that direction here.

1.2 A Short History of Semi-Supervised Learning

There has been a whole spectrum of interesting ideas on how to learn from both labeled and unlabeled data. We give a highly simplified history of semi-supervised

learning in this section. Interested readers can skip to Chapter 11 for an extended literature review. It should be pointed out that semi-supervised learning is a rapidly evolving field, and the review is necessarily incomplete.

Early work in semi-supervised learning assumes there are two classes, and each class has a Gaussian distribution. This amounts to assuming the complete data comes from a mixture model . With large amount of unlabeled data, the mixture components can be identified with the expectation-maximization (EM) algorithm. One needs only a single labeled example per component to fully determine the mixture model. This model has been successfully applied to text categorization.

A variant is self-training : A classifier is first trained with the labeled data. It is then used to classify the unlabeled data. The most confident unlabeled points, together with their predicted labels, are added to the training set. The classifier is re-trained and the procedure repeated. Note the classifier uses its own predictions to teach itself. This is a 'hard' version of the mixture model and EM algorithm. The procedure is also called self-teaching , or bootstrapping[1] in some research communities. One can imagine that a classification mistake can reinforce itself.

Both methods have been used since long time ago. They remain popular because of their conceptual and algorithmic simplicity.

Co-training reduces the mistake-reinforcing danger of self-training. This recent method assumes that the features of an item can be split into two subsets. Each sub-feature set is sufficient to train a good classifier; and the two sets are conditionally independent given the class. Initially two classifiers are trained with the labeled data, one on each sub-feature set. Each classifier then iteratively classifies the unlabeled data, and teaches the other classifier with its predictions.

With the rising popularity of support vector machines (SVMs), transductive SVMs emerge as an extension to standard SVMs for semi-supervised learning. Transductive SVMs find a labeling for all the unlabeled data, and a separating hyperplane, such that maximum margin is achieved on both the labeled data and the (now labeled) unlabeled data. Intuitively unlabeled data guides the decision boundary away from dense regions.

Recently graph-based semi-supervised learning methods have attracted great attention. Graph-based methods start with a graph where the nodes are the labeled and unlabeled data points, and (weighted) edges reflect the similarity of nodes. The assumption is that nodes connected by a large-weight edge tend to have the same label, and labels can propagation throughout the graph. Graph-based methods enjoy nice properties from spectral graph theory. This thesis mainly discusses graph-based semi-supervised methods.

We summarize a few representative semi-supervised methods in Table 1.1.

[1] Not to be confused with the resample procedure with the same name in statistics.

Method	Assumptions
mixture model, EM	generative mixture model
transductive SVM	low density region between classes
co-training	conditionally independent and redundant features splits
graph methods	labels smooth on graph

Table 1.1: Some representative semi-supervised learning methods

1.3 Structure of the Thesis

The rest of the thesis is organized as follows:

Chapter 2 starts with the simple *label propagation* algorithm, which propagates class labels on a graph. This is the first semi-supervised learning algorithm we will encounter. It is also the basis for many variations later.

Chapter 3 discusses how one constructs a graph. The emphasis is on the intuition – what graphs make sense for semi-supervised learning? We will give several examples on various datasets.

Chapter 4 formalizes label propagation in a probabilistic framework with Gaussian random fields. Concepts like graph Laplacian and harmonic function are introduced. We will explore interesting connections to electric networks, random walk, and spectral clustering. Issues like the balance between classes, and inclusion of external classifiers are also discussed here.

Chapter 5 assumes that one can choose a data point and ask an oracle for the label. This is the standard active learning scheme. We show that active learning and semi-supervised learning can be naturally combined.

Chapter 6 establishes the link to Gaussian processes. The kernel matrices are shown to be the smoothed inverse graph Laplacian.

Chapter 7 no longer assumes the graph is given and fixed. Instead, we parameterize the graph weights, and learn the optimal hyperparameters. We will discuss several methods: evidence maximization, entropy minimization, and minimum spanning tree.

Chapter 8 turns semi-supervised learning problem into kernel learning. We show a natural family of kernels derived from the graph Laplacian, and find the best kernel via convex optimization.

Chapter 9 discusses kernel conditional random fields, and its potential application in semi-supervised learning, for sequences and other complex structures.

Chapter 10 explores scalability and induction for semi-supervised learning.

Chapter 11 reviews the literatures on semi-supervised learning.

Chapter 2

Label Propagation

In this chapter we introduce our first semi-supervised learning algorithm: Label Propagation. We formulate the problem as a form of propagation on a graph, where a node's label propagates to neighboring nodes according to their proximity. In this process we fix the labels on the labeled data. Thus labeled data act like sources that push out labels through unlabeled data.

2.1 Problem Setup

Let $\{(x_1, y_1) \dots (x_l, y_l)\}$ be the labeled data, $y \in \{1 \dots C\}$, and $\{x_{l+1} \dots x_{l+u}\}$ the unlabeled data, usually $l \ll u$. Let $n = l + u$. We will often use L and U to denote labeled and unlabeled data respectively. We assume the number of classes C is known, and all classes are present in the labeled data. In most of the thesis we study the *transductive* problem of finding the labels for U. The inductive problem of finding labels for points outside of $L \cup U$ will be discussed in Chapter 10.

Intuitively we want data points that are similar to have the same label. We create a graph where the nodes are all the data points, both labeled and unlabeled. The edge between nodes i, j represents their similarity. For the time being let us assume the graph is fully connected with the following weights:

$$w_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{\alpha^2}\right) \tag{2.1}$$

where α is a bandwidth hyperparameter. The construction of graphs will be discussed in later Chapters.

2.2 The Algorithm

We propagate the labels through the edges. Larger edge weights allow labels to travel through more easily. Define a $n \times n$ probabilistic transition matrix P

$$P_{ij} = P(i \to j) = \frac{w_{ij}}{\sum_{k=1}^{n} w_{ik}} \tag{2.2}$$

where P_{ij} is the probability of transit from node i to j. Also define a $l \times C$ label matrix Y_L, whose ith row is an indicator vector for $y_i, i \in L$: $Y_{ic} = \delta(y_i, c)$. We will compute soft labels f for the nodes. f is a $n \times C$ matrix, the rows can be interpreted as the probability distributions over labels. The initialization of f is not important. We are now ready to present the algorithm.

The label propagation algorithm is as follows:

1. Propagate $f \leftarrow Pf$
2. Clamp the labeled data $f_L = Y_L$.
3. Repeat from step 1 until f converges.

In step 1, all nodes propagate their labels to their neighbors for one step. Step 2 is critical: we want persistent label sources from labeled data. So instead of letting the initially labels fade away, we clamp them at Y_L. With this constant 'push' from labeled nodes, the class boundaries will be pushed through high density regions and settle in low density gaps. If this structure of data fits the classification goal, then the algorithm can use unlabeled data to help learning.

2.3 Convergence

We now show the algorithm converges to a simple solution. Let $f = \begin{pmatrix} f_L \\ f_U \end{pmatrix}$. Since f_L is clamped to Y_L, we are solely interested in f_U. We split P into labeled and unlabeled sub-matrices

$$P = \begin{bmatrix} P_{LL} & P_{LU} \\ P_{UL} & P_{UU} \end{bmatrix} \tag{2.3}$$

It can be shown that our algorithm is

$$f_U \leftarrow P_{UU} f_U + P_{UL} Y_L \tag{2.4}$$

which leads to

$$f_U = \lim_{n\to\infty} (P_{UU})^n f_U^0 + \left(\sum_{i=1}^{n} (P_{UU})^{(i-1)} \right) P_{UL} Y_L \tag{2.5}$$

where f_U^0 is the initial value for f_U. We need to show $(P_{UU})^n f_U^0 \to 0$. Since P is row normalized, and P_{UU} is a sub-matrix of P, it follows

$$\exists \gamma < 1, \sum_{j=1}^{u} (P_{UU})_{ij} \leq \gamma, \forall i = 1 \ldots u \tag{2.6}$$

Therefore

$$\begin{aligned} \sum_j (P_{UU})^n{}_{ij} &= \sum_j \sum_k (P_{UU})^{(n-1)}{}_{ik} (P_{UU})_{kj} && (2.7)\\ &= \sum_k (P_{UU})^{(n-1)}{}_{ik} \sum_j (P_{UU})_{kj} && (2.8)\\ &\leq \sum_k (P_{UU})^{(n-1)}{}_{ik} \gamma && (2.9)\\ &\leq \gamma^n && (2.10) \end{aligned}$$

Therefore the row sums of $(P_{UU})^n$ converges to zero, which means $(P_{UU})^n f_U^0 \to 0$. Thus the initial value f_U^0 is inconsequential. Obviously

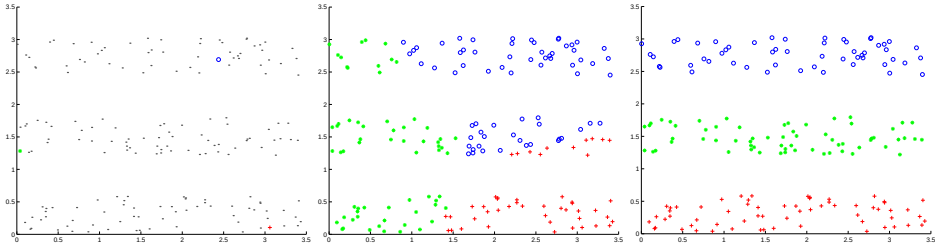
$$f_U = (I - P_{UU})^{-1} P_{UL} Y_L \tag{2.11}$$

is a fixed point. Therefore it is the unique fixed point and the solution to our iterative algorithm. This gives us a way to solve the label propagation problem directly without iterative propagation.

Note the solution is valid only when $I - P_{UU}$ is invertible. The condition is satisfied, intuitively, when every connected component in the graph has at least one labeled point in it.

2.4 Illustrative Examples

We demonstrate the properties of the Label Propagation algorithm on two synthetic datasets. Figure 2.1(a) shows a synthetic dataset with three classes, each being a narrow horizontal band. Data points are uniformly drawn from the bands. There are 3 labeled points and 178 unlabeled points. 1-nearest-neighbor algorithm, one of the standard supervised learning methods, ignores the unlabeled data and thus the

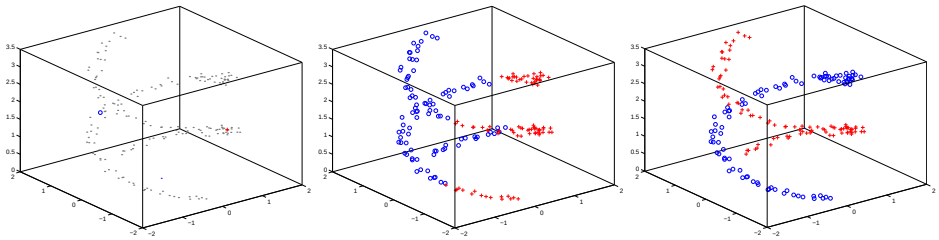


(a) The data (b) 1NN (c) Label Propagation

Figure 2.1: The Three Bands dataset. Labeled data are marked with color symbols, and unlabeled data are black dots in (a). 1NN ignores unlabeled data structure (b), while Label Propagation takes advantage of it (c).

band structure (b). On the other hand, the Label Propagation algorithm takes into account the unlabeled data (c). It propagates labels along the bands. In this example, we used $\alpha = 0.22$ from the minimum spanning tree heuristic (see Chapter 7).

Figure 2.2 shows a synthetic dataset with two classes as intertwined three-dimensional spirals. There are 2 labeled points and 184 unlabeled points. Again, 1NN fails to notice the structure of unlabeled data, while Label Propagation finds the spirals. We used $\alpha = 0.43$.



(a) The data (b) 1NN (c) Label Propagation

Figure 2.2: The Springs dataset. Again 1NN ignores unlabeled data structure, while Label Propagation takes advantage of it.

Chapter 3

What is a Good Graph?

In Label Propagation we need a graph , represented by the weight matrix W. How does one construct a graph? What is a good graph? In this chapter we give several examples on different datasets. The goal is not to rigorously define 'good' graphs, but to illustrate the assumptions behind graph based semi-supervised learning.

A good graph should reflect our prior knowledge about the domain. At the present time, its design is more of an art than science. It is the practitioner's responsibility to feed a good graph to graph-based semi-supervised learning algorithms, in order to expect useful output. The algorithms in this thesis do not deal directly with the design of graphs (with the exception of Chapter 7).

3.1 Example One: Handwritten Digits

Our first example is optical character recognition (OCR) for handwritten digits. The handwritten digits dataset originates from the Cedar Buffalo binary digits database (Hull, 1994). The digits were initially preprocessed to reduce the size of each image down to a 16×16 grid by down-sampling and Gaussian smoothing, with pixel values in 0 to 255 (Le Cun et al., 1990). Figure 3.1 shows a random sample of the digits. In some of the experiments below they are further scaled down to 8×8 by averaging 2×2 pixel bins.

We show why graphs based on pixel-wise Euclidean distance make sense for digits semi-supervised learning. Euclidean distance by itself is a bad similarity measure. For example the two images in Figure 3.2(a) have a large Euclidean distance although they are in the same class. However Euclidean distance is a good 'local' similarity measure. If it is small, we can expect the two images to be in the same class. Consider a k-nearest-neighbor graph based on Euclidean distance. Neighboring images have small Euclidean distance. With large amount

Figure 3.1: some random samples of the handwritten digits dataset

(a) two images of '2' with large Euclidean distance

(b) a path in an Euclidean distance kNN graph between them

Figure 3.2: Locally similar images propagate labels to globally dissimilar ones.

of unlabeled images of 2s, there will be many *paths* connecting the two images in (a). One such path is shown in Figure 3.2(b). Note adjacent pairs are similar to each other. Although the two images in (a) are not directly connected (not similar in Euclidean distance), Label Propagation can propagate along the paths, marking them with the same label.

Figure 3.3 shows a symmetrized [1] 2NN graph based on Euclidean distance. The small dataset has only a few 1s and 2s for clarity. The actual graphs used in the OCR experiments are too large to show.

It should be mentioned that our focus is on semi-supervised learning methods, not OCR handwriting recognizers. We could have normalized the image intensity, or used edge detection or other invariant features instead of Euclidean distance. These should be used for any real applications, as the graph should represent domain knowledge. The same is true for all other tasks described below.

[1] Symmetrization means we connect nodes i, j if i is in j's kNN or vice versa, and therefore a node can have more than k edges.

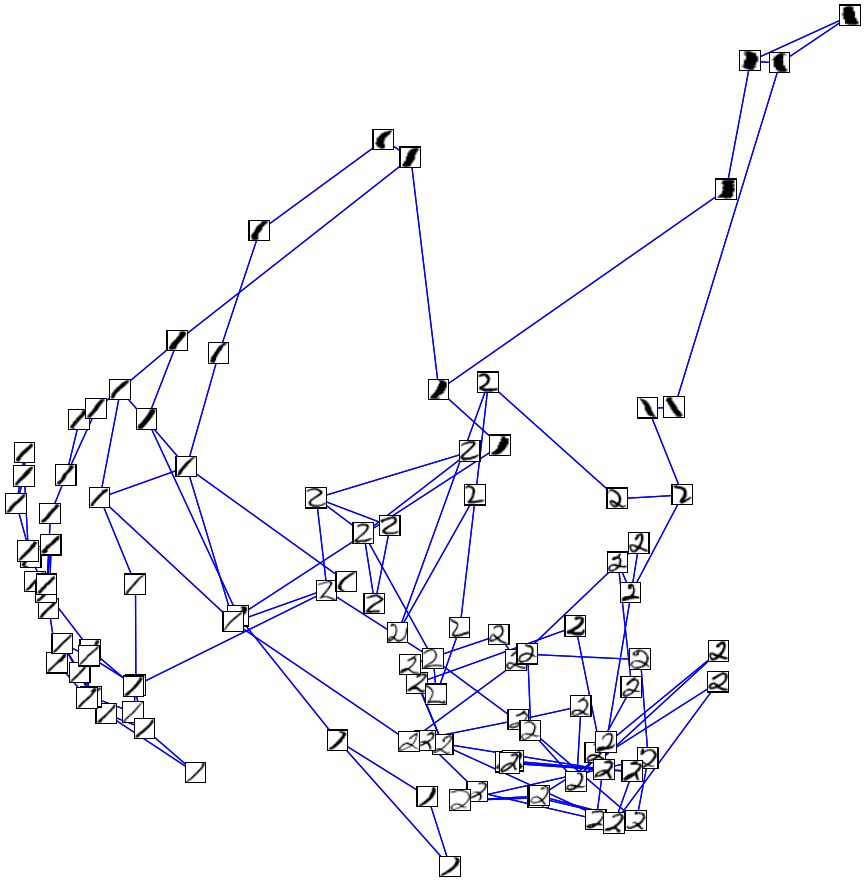


Figure 3.3: A symmetrized Euclidean 2NN graph on some 1s and 2s. Label Propagation on this graph works well.

3.2 Example Two: Document Categorization

Our second example is document categorization on 20 newsgroups dataset [2] . Each document has no header except 'From' and 'Subject' lines. Each document is minimally processed into a $tf.idf$ vector, without frequency cutoff, stemming, or a stopword list. The 'From' and 'Subject' lines are included. We measure the similarity between two documents u, v with the cosine similarity $cs(u, v) = \frac{u^\top v}{|u||v|}$. Like Euclidean distance, cosine similarity is not a good 'global' measure: two documents from the same class can have few common words. However it is a good 'local' measure.

A graph based on cosine similarity in this domain makes good sense. Documents from the same thread (class) tend to quote one another, giving them high cosine similarities. Many paths in the graph are quotations. Even though the first and last documents in a thread share few common words, them can be classified in the same class via the graph.

The full graphs are again too large to visualize. We show the few nearest neighbors of document 60532 in comp.sys.ibm.pc.hardware vs. comp.sys.mac.hardware sub-dataset in Figure 3.4. The example is typical in the whole graph. Nevertheless we note that not all edges are due to quotation.

3.3 Example Three: The FreeFoodCam

The Carnegie Mellon University School of Computer Science has a lounge, where leftover pizza from various meetings converge, to the delight of students. In fact a webcam (the FreeFoodCam [3]) was set up in the lounge, so that people can see whether food is available. The FreeFoodCam provides interesting research opportunities. We collect webcam images of 10 people over a period of several months. The data is used for 10-way people recognition, i.e. identify the name of person in FreeFoodCam images. The dataset consists of 5254 images with one and only one person in it. Figure 3.5 shows some random images in the dataset. The task is not trivial:

1. The images of each person were captured on multiple days during a four month period. People changed clothes, had hair cut, one person even grew a beard. We simulate a video surveillance scenario where a person is manually labeled at first, and needs to be recognized on later days. Therefore we choose labeled data within the first day of a person's appearance, and test on

[2] http://www.ai.mit.edu/people/jrennie/20Newsgroups/, '18828 version'

[3] http://www-2.cs.cmu.edu/∼coke/, Carnegie Mellon internal access.

```
From: rash@access.digex.com (Wayne Rash)
Subject: Re: 17" Monitors
mikey@sgi.com (Mike Yang) writes:
>In article <1qslfs$bm1@access.digex.net> rash@access.digex.com (Wayne Rash) writes:
>>I also reviewed a new Nanao, the F550iW, which has just
>>been released.
>What's the difference between the F550i and the new F550iW?  I'm
>about to buy a Gateway system and was going to take the F550i
>upgrade.  Should I get the F550iW instead?
>-------------------------------------------------------------------------
>                Mike Yang        Silicon Graphics, Inc.
>              mikey@sgi.com           415/390-1786
The F550iW is optimized for Windows.  It powers down when the screen
blanker appears, it powers down with you turn your computer off, and it
meets all of the Swedish standards.  It's also protected against  EMI from
adjacent monitors.
Personally, I think the F550i is more bang for the buck right now.
```

(a) document 60532. Its nearest neighbors are shown below.

```
From: mikey@eukanuba.wpd.sgi.com (Mike Yang)
Subject: Re: 17" Monitors
In article <1qulqa$hp2@access.digex.net>, rash@access.digex.com (Wayne Rash) writes:
|> The F550iW is optimized for Windows.  It powers down when the screen
|> blanker appears, it powers down with you turn your computer off, and it
|> meets all of the Swedish standards.  It's also protected against  EMI from
|> adjacent monitors.
Thanks for the info.
|> Personally, I think the F550i is more bang for the buck right now.
How much more does the F550iW cost?
-------------------------------------------------------------------------
                Mike Yang        Silicon Graphics, Inc.
              mikey@sgi.com           415/390-1786
```

(b) The nearest neighbor 60538. It quotes a large portion of 60532.

```
From: rash@access.digex.com (Wayne Rash)
Subject: Re: 17" Monitors
mikey@eukanuba.wpd.sgi.com (Mike Yang) writes:
>In article <1qulqa$hp2@access.digex.net>, rash@access.digex.com (Wayne Rash) writes:
>|> The F550iW is optimized for Windows.  It powers down when the screen
>|> blanker appears, it powers down with you turn your computer off, and it
>|> meets all of the Swedish standards.  It's also protected against  EMI from
>|> adjacent monitors.
>Thanks for the info.
>|> Personally, I think the F550i is more bang for the buck right now.
>How much more does the F550iW cost?
>-------------------------------------------------------------------------
>                Mike Yang        Silicon Graphics, Inc.
>              mikey@sgi.com           415/390-1786
I think the difference is about 400 dollars, but I could be wrong.  These
things change between press time and publication.
```

(c) The 2nd nearest neighbor 60574. It also quotes 60532.

Figure 3.4: (continued on next page)

```
From: mikey@sgi.com (Mike Yang)
Subject: Re: 17" Monitors
In article <1qslfs$bm1@access.digex.net> rash@access.digex.com (Wayne Rash) writes:
>I also reviewed a new Nanao, the F550iW, which has just
>been released.
What's the difference between the F550i and the new F550iW?  I'm
about to buy a Gateway system and was going to take the F550i
upgrade.  Should I get the F550iW instead?
--------------------------------------------------------------------------
                Mike Yang        Silicon Graphics, Inc.
              mikey@sgi.com           415/390-1786
```

(d) The 3rd nearest neighbor 60445, quoted by 60532.

```
From: goyal@utdallas.edu (MOHIT K GOYAL)
Subject: Re: 17" Monitors
>the Mitsubishi.  I also reviewed a new Nanao, the F550iW, which has just
>been released.  Last year for the May '92 issue of Windows, I reviewed
Do you have the specs for this monitor?  What have they changed from the
F550i?
Do you know if their is going to be a new T560i soon? (a T560iW?)
Thanks.
```

(e) The 4th nearest neighbor 60463. It and 60532 quote the same source.

```
From: mikey@eukanuba.wpd.sgi.com (Mike Yang)
Subject: Gateway 4DX2-66V update
I just ordered my 4DX2-66V system from Gateway.  Thanks for all the net
discussions which helped me decide among all the vendors and options.
Right now, the 4DX2-66V system includes 16MB of RAM.  The 8MB upgrade
used to cost an additional $340.
--------------------------------------------------------------------------
                Mike Yang        Silicon Graphics, Inc.
              mikey@sgi.com           415/390-1786
```

(f) The 5th nearest neighbor 61165. It has a different subject than 60532, but the same author signature appears in both.

Figure 3.4: The nearest neighbors of document 60532 in the 20newsgroups dataset, as measured by cosine similarity. Notice many neighbors either quote or are quoted by the document. Many also share the same subject line.

Figure 3.5: A few FreeFoodCam image examples

the remaining images of the day and all other days. It is harder than testing only on the same day, or allowing labeled data to come from all days.

2. The FreeFoodCam is a low quality webcam. Each frame is 640×480 so faces of far away people are small; The frame rate is a little over 0.5 frame per second; Lighting in the lounge is complex and changing.

3. The person could turn the back to the camera. About one third of the images have no face.

Since only a few images are labeled, and we have all the test images, it is a natural task to apply semi-supervised learning techniques. As computer vision is not the focus of the paper, we use only primitive image processing methods to extract the following features:

Time. Each image has a time stamp.

Foreground color histogram. A simple background subtraction algorithm is applied to each image to find the foreground area. The foreground area is assumed to be the person (head and body). We compute the color histogram (hue, saturation and brightness) of the foreground pixels. The histogram is a 100 dimensional vector.

Face image. We apply a face detector (Schneiderman, 2004b) (Schneiderman, 2004a) to each image. Note it is *not* a face recognizer (we do not use a face recognizer for this task). It simply detects the presence of frontal or profile faces. The output is the estimated center and radius of the detected face. We take a square area around the center as the face image. If no face is detected, the face image is empty.

One theme throughout the thesis is that the graph should reflect domain knowledge of similarity. The FreeFoodCam is a good example. The nodes in the graph are all the images. An edge is put between two images by the following criteria:

1. *Time edges* People normally move around in the lounge in moderate speed, thus adjacent frames are likely to contain the same person. We represent this belief in the graph by putting an edge between images i, j whose time difference is less than a threshold t_1 (usually a few seconds).

2. *Color edges* The color histogram is largely determined by a person's clothes. We assume people change clothes on different days, so color histogram is unusable across multiple days. However it is an informative feature during a shorter time period (t_2) like half a day. In the graph for every image i, we find the set of images having a time difference between (t_1, t_2) to i, and connect i with its k_c-nearest-neighbors (in terms of cosine similarity on histograms) in the set. k_c is a small number, e.g. 3.

3. *Face edges* We resort to face similarity over longer time spans. For every image i with a face, we find the set of images more than t_2 apart from i, and connect i with its k_f-nearest-neighbor in the set. We use pixel-wise Euclidean distance between face images (the pair of face images are scaled to the same size).

The final graph is the union of the three kinds of edges. The edges are unweighted in the experiments (one could also learn different weights for different kinds of edges. For example it might be advantageous to give time edges higher weights). We used $t_1 = 2$ second, $t_2 = 12$ hours, $k_c = 3$ and $k_f = 1$ below. Incidentally these parameters give a connected graph. It is impossible to visualize the whole graph. Instead we show the neighbors of a random node in Figure 3.6.

3.4 Common Ways to Create Graphs

Sometimes one faces a dataset with limited domain knowledge. This section discusses some common ways to create a graph as a starting point.

image 4005 | neighbor 1: time edge

neighbor 2: color edge | neighbor 3: color edge

neighbor 4: color edge | neighbor 5: face edge

Figure 3.6: A random image and its neighbors in the graph

Fully connected graphs One can create a fully connected graph with an edge between all pairs of nodes. The graph needs to be weighted so that similar nodes have large edge weight between them. The advantage of a fully connected graph is in weight learning – with a differentiable weight function, one can easily take the derivatives of the graph w.r.t. weight hyperparameters. The disadvantage is in computational cost as the graph is dense (although sometimes one can apply fast approximate algorithms like N-body problems). Furthermore we have observed that empirically fully connect graphs performs worse than sparse graphs.

Sparse graphs One can create kNN or ϵNN graphs as shown below, where each node connects to only a few nodes. Such sparse graphs are computationally fast. They also tend to enjoy good empirical performance. We surmise it is because spurious connections between dissimilar nodes (which tend to be in different classes) are removed. With sparse graphs, the edges can be unweighted or weighted. One disadvantage is weight learning – a change in weight hyperparameters will likely change the neighborhood, making optimization awkward.

kNN graphs Nodes i, j are connected by an edge if i is in j's k-nearest-neighborhood or vice versa. k is a hyperparameter that controls the density of the graph. kNN has the nice property of "adaptive scales," because the neighborhood radius is different in low and high data density regions. Small k may result in disconnected graphs. For Label Propagation this is not a problem if each connected component has some labeled points. For other algorithms introduced later in the thesis, one can smooth the Laplacian.

ϵNN graphs Nodes i, j are connected by an edge, if the distance $d(i, j) \leq \epsilon$. The hyperparameter ϵ controls neighborhood radius. Although ϵ is continuous, the search for the optimal value is discrete, with at most $O(n^2)$ values (the edge lengths in the graph).

$\tanh$-weighted graphs $w_{ij} = (\tanh(\alpha_1(d(i, j) - \alpha_2)) + 1)/2$. The hyperbolic tangent function is a 'soft step' function that simulates ϵNN in that when $d(i, j) \gg \alpha_2$, $w_{ij} \approx 0$; $d(i, j) \ll \alpha_2$, $w_{ij} \approx 1$. The hyperparameters α_1, α_2 controls the slope and cutoff value respectively. The intuition is to create a soft cutoff around distance α_2, so that close examples (presumably from the same class) are connected and examples from different classes (presumably with large distance) are nearly disconnected. Unlike ϵNN, $\tanh$-weighted graph is continuous with respect to α_1, α_2 and is amenable to learning with gradient methods.

$\exp$-**weighted graphs** $w_{ij} = \exp(-d(i,j)^2/\alpha^2)$. Again this is a continuous weighting scheme, but the cutoff is not as clear as $\tanh()$. Hyperparameter α controls the decay rate. If d is e.g. Euclidean distance, one can have one hyperparameter per feature dimension.

These weight functions are all potentially useful when we do not have enough domain knowledge. However we observed that weighted kNN graphs with a small k tend to perform well empirically. All the graph construction methods have hyperparameters. We will discuss graph hyperparameter learning in Chapter 7.

A graph is represented by the $n \times n$ weight matrix W, $w_{ij} = 0$ if there is no edge between node i, j. We point out that W does not have to be positive semi-definite. Nor need it satisfy metric conditions. As long as W's entries are non-negative and symmetric, the graph Laplacian, an important quantity defined in the next chapter, will be well defined and positive semi-definite.

Chapter 4

Gaussian Random Fields and Harmonic Functions

In this chapter we formalize label propagation with a probabilistic framework. Without loss of generality we assume binary classification $y \in \{0, 1\}$. We assume the $n \times n$ weight matrix W is given, which defines the graph. W has to be symmetric with non-negative entries, but otherwise need not to be positive semi-definite. Intuitively W specifies the 'local similarity' between points. Our task is to assign labels to unlabeled nodes.

4.1 Gaussian Random Fields

Our strategy is to define a continuous random field on the graph. First we define a real function over the nodes $f : L \cup U \longrightarrow \mathbb{R}$. Notice f can be negative or larger than 1. Intuitively, we want unlabeled points that are similar (as determined by edge weights) to have similar labels. This motivates the choice of the quadratic *energy* function

$$E(f) = \frac{1}{2} \sum_{i,j} w_{ij} \left(f(i) - f(j)\right)^2 \tag{4.1}$$

Obviously E is minimized by constant functions. But since we have observed some labeled data, we constrain f to take values $f(i) = y_i, i \in L$ on the labeled data. We assign a probability distribution to functions f by a *Gaussian random field*

$$p(f) = \frac{1}{Z} e^{-\beta E(f)} \tag{4.2}$$

where β is an "inverse temperature" parameter, and Z is the partition function

$$Z = \int_{f_L = Y_L} \exp\left(-\beta E(f)\right) \, df \tag{4.3}$$

which normalizes over functions constrained to Y_L on the labeled data. We are interested in the inference problem $p(f_i|Y_L), i \in U$, or the mean $\int_{-\infty}^{\infty} f_i p(f_i|Y_L) \, df_i$.

The distribution $p(f)$ is very similar to a standard Markov Random field with discrete states (the Ising model, or Boltzmann machines (Zhu & Ghahramani, 2002b)). In fact the only difference is the relaxation to real-valued states. However this relaxation greatly simplify the inference problem. Because of the quadratic energy, $p(f)$ and $p(f_U|Y_L)$ are both multivariate Gaussian distributions. This is why p is called a *Gaussian* random field. The marginals $p(f_i|Y_L)$ are univariate Gaussian too, and have closed form solutions.

4.2 The Graph Laplacian

We now introduce an important quantity: the *combinatorial Laplacian* Δ. Let D be the diagonal degree matrix, where $D_{ii} = \sum_j W_{ij}$ is the degree of node i. The Laplacian is defined as

$$\Delta \equiv D - W \tag{4.4}$$

For the time being the Laplacian is useful shorthand for the energy function: One can verify that

$$E(f) = \frac{1}{2} \sum_{i,j} w_{ij} \left(f(i) - f(j)\right)^2 = f^\top \Delta f \tag{4.5}$$

The Gaussian random field can be written as

$$p(f) = \frac{1}{Z} e^{-\beta f^\top \Delta f} \tag{4.6}$$

where the quadratic form becomes obvious. Δ plays the role of the precision (inverse covariance) matrix in a multivariate Gaussian distribution. It is always positive semi-definite if W is symmetric and non-negative. The Laplacian will be further explored in later chapters.

4.3 Harmonic Functions

It is not difficult to show that the minimum energy function $f = \arg\min_{f_L = Y_L} E(f)$ is *harmonic*; namely, it satisfies $\Delta f = 0$ on unlabeled data points U, and is equal to Y_L on the labeled data points L. We use h to represent this harmonic function.

The harmonic property means that the value of $h(i)$ at each unlabeled data point i is the average of its neighbors in the graph:

$$h(i) = \frac{1}{D_{ii}} \sum_{j \sim i} w_{ij} h(j), \text{ for } i \in U \tag{4.7}$$

which is consistent with our prior notion of smoothness with respect to the graph. Because of the maximum principle of harmonic functions (Doyle & Snell, 1984), h is unique and satisfies $0 \leq h(i) \leq 1$ for $i \in U$ (remember $h(i) = 0$ or 1 for $i \in L$).

To compute the harmonic solution, we partition the weight matrix W (and similarly D, Δ, etc.) into 4 blocks for L and U:

$$W = \begin{bmatrix} W_{LL} & W_{LU} \\ W_{UL} & W_{UU} \end{bmatrix} \tag{4.8}$$

The harmonic solution $\Delta h = 0$ subject to $h_L = Y_L$ is given by

$$\begin{aligned} h_U &= (D_{UU} - W_{UU})^{-1} W_{UL} Y_L & (4.9) \\ &= -(\Delta_{UU})^{-1} \Delta_{UL} Y_L & (4.10) \\ &= (I - P_{UU})^{-1} P_{UL} Y_L & (4.11) \end{aligned}$$

The last representation is the same as equation (2.11), where $P = D^{-1}W$ is the transition matrix on the graph. The Label Propagation algorithm in Chapter 2 in fact computes the harmonic function.

The harmonic function minimizes the energy and is thus the mode of (4.2). Since (4.2) defines a Gaussian distribution which is symmetric and unimodal, the mode is also the mean.

4.4 Interpretation and Connections

The harmonic function can be viewed in several fundamentally different ways, and these different viewpoints provide a rich and complementary set of techniques for reasoning about this approach to the semi-supervised learning problem.

4.4.1 Random Walks

Imagine a random walk on the graph. Starting from an unlabeled node i, we move to a node j with probability P_{ij} after one step. The walk stops when we hit a labeled node. Then $h(i)$ is the probability that the random walk, starting from node i, hits a labeled node with label 1. Here the labeled data is viewed as an "absorbing boundary" for the random walk. The random walk interpretation is shown in Figure 4.1.

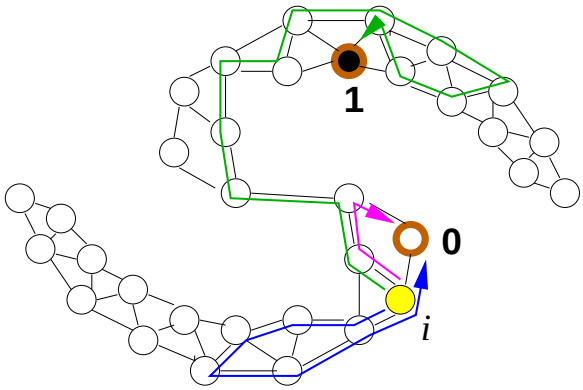


Figure 4.1: Harmonic function as random walk on the graph

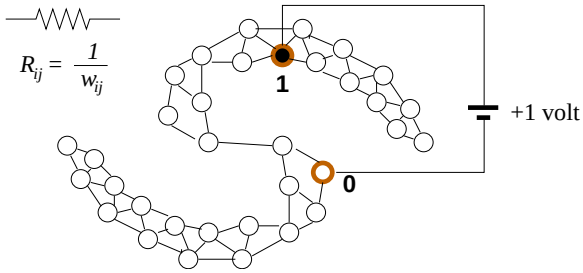


Figure 4.2: Harmonic function as electric network graph

4.4.2 Electric Networks

We can also view the framework as electrical networks. Imagine the edges of the graph to be resistors with conductance W. Equivalently the resistance between nodes i, j is $1/w_{ij}$. We connect positive labeled nodes to a $+1$ volt source, and negative labeled nodes to the ground. Then h_U is the voltage in the resulting electric network on each of the unlabeled nodes (Figure 4.2). Furthermore h_U minimizes the energy dissipation, in the form of heat, of the electric network. The energy dissipation is exactly $E(h)$ as in (4.1). The harmonic property here follows from Kirchoff's and Ohm's laws, and the maximum principle then shows that this is precisely the same solution obtained in (4.11).

4.4.3 Graph Mincut

The harmonic function can be viewed as a soft version of the graph mincut approach by Blum and Chawla (2001). In graph mincut the problem is cast as one

of finding a minimum st-cut. The minimum st-cuts minimize the same energy function (4.1) but with discrete labels 0,1. Therefore they are the modes of a standard Boltzmann machine. It is difficult to compute the mean. One often has to use Monte Carlo Markov Chain or use approximation methods. Furthermore, the minimum st-cut is not necessarily unique. For example, consider a linear chain graph with n nodes. Let $w_{i,i+1} = 1$ and other edges zero. Let node 1 be labeled positive, node n negative. Then a cut on any one edge is a minimum st-cut. In contrast, the harmonic solution has a closed form, unique solution for the mean, which is also the mode.

The Gaussian random fields and harmonic functions also have connection to graph spectral clustering, and kernel regularization. These will be discussed later.

4.5 Incorporating Class Proportion Knowledge

To go from f to class labels, the obvious decision rule is to assign label 1 to node i if $h(i) > 0.5$, and label 0 otherwise. We call this rule *0.5-threshold*. In terms of the random walk interpretation if $h(i) > 0.5$, then starting at i, the random walk is more likely to reach a positively labeled point before a negatively labeled point. This decision rule works well when the classes are well separated. However in practice, 0.5-threshold tends to produce unbalanced classification (most points in one of the classes). The problem stems from the fact that W, which specifies the data manifold, is often poorly estimated in practice and does not reflect the classification goal. In other words, we should not "fully trust" the graph structure.

Often we have the knowledge of class proportions, i.e. how many unlabeled data are from class 0 and 1 respectively. This can either be estimated from the labeled set, or given by domain experts. This is a valuable piece of complementary information.

We propose a heuristic method called *class mass normalization* (CMN) to incorporate the information as follows. Let's assume the desirable proportions for classes 1 and 0 are q and $1 - q$ respectively. Define the mass of class 1 to be $\sum_i h_U(i)$, and the mass of class 0 to be $\sum_i (1 - h_U(i))$. Class mass normalization scales these masses to match q and $1 - q$. In particular an unlabeled point i is classified as class 1 iff

$$q \frac{h_U(i)}{\sum_i h_U(i)} > (1 - q) \frac{1 - h_U(i)}{\sum_i (1 - h_U(i))} \tag{4.12}$$

CMN extends naturally to the general multi-label case. It is interesting to note CMN's potential connection to the procedures in (Belkin et al., 2004a). Further research is needed to study whether the heuristic (or its variation) can be justified in theory.

4.6 Incorporating Vertex Potentials on Unlabeled Instances

We can incorporate the knowledge on individual class label of unlabeled instances too. This is similar to using a "assignment cost" for each unlabeled instance. For example, the external knowledge may come from an external classifier which is constructed on labeled data alone (It could come from domain expert too). The external classifier produces labels g_U on the unlabeled data; g can be 0/1 or soft labels in $[0, 1]$. We combine g with the harmonic function h by a simple modification of the graph. For each unlabeled node i in the original graph, we attach a "dongle" node which is a labeled node with value g_i. Let the transition probability from i to its dongle be η, and discount other transitions from i by $1 - \eta$. We then compute the harmonic function on this augmented graph. Thus, the external classifier introduces assignment costs to the energy function, which play the role of vertex potentials in the random field. It is not difficult to show that the harmonic solution on the augmented graph is, in the random walk view,

$$h_U = (I - (1 - \eta)P_{UU})^{-1} ((1 - \eta)P_{UL}Y_L + \eta g_U) \tag{4.13}$$

We note that up to now we have assumed the labeled data to be noise free, and so clamping their values makes sense. If there is reason to doubt this assumption, it would be reasonable to attach dongles to labeled nodes as well, and to move the labels to these dongles. An alternative is to use Gaussian process classifiers with a noise model, which will be discussed in Chapter 6.

4.7 Experimental Results

We evaluate harmonic functions on the following tasks. For each task, we gradually increase the labeled set size systematically. For each labeled set size, we perform 30 random trials. In each trial we randomly sample a labeled set with the specific size (except for the Freefoodcam task where we sample labeled set from the first day only). However if a class is missing from the sampled labeled set, we redo the random sampling. We use the remaining data as the unlabeled set and report the classification accuracy with harmonic functions on them.

To compare the harmonic function solution against a standard supervised learning method, we use a Matlab implementation of SVM (Gunn, 1997) as the baseline. Notice the SVMs are not semi-supervised: the unlabeled data are merely used as test data. For c-class multiclass problems, we use a one-against-all scheme which creates c binary subproblems, one for each class against the rest classes, and select the class with the largest margin. We use 3 standard kernels for each task: linear $K(i, j) = \langle x_i, x_j \rangle$, quadratic $K(i, j) = (\langle x_i, x_j \rangle + 1)^2$, and radial basis function

(RBF) $K(i,j) = \exp\left(-\|x_i - x_j\|^2/2\sigma^2\right)$. The slack variable upper bound (usually denoted by C) for each kernel, as well as the bandwidth σ for RBF, are tuned by 5 fold cross validation for each task.

1. **1 vs. 2**. Binary classification for OCR handwritten digits "1" vs. "2". This is a subset of the handwritten digits dataset. There are 2200 images, half are "1"s and the other half are "2"s.

   The graph (or equivalently the weight matrix W) is the single most important input to the harmonic algorithm. To demonstrate its importance, we show the results of not one but six related graphs:

   (a) 16×16 full. Each digit image is 16×16 gray scale with pixel values between 0 and 255. The graph is fully connected, and the weights decrease exponentially with Euclidean distance:

   $$w_{ij} = \exp\left(-\sum_{d=1}^{256} \frac{(x_{i,d} - x_{j,d})^2}{380^2}\right) \tag{4.14}$$

   The parameter 380 is chosen by evidence maximization (see Section 7.1). This was the graph used in (Zhu et al., 2003a).

   (b) 16×16 10NN weighted. Same as '16×16 full', but i, j are connected only if i is in j's 10-nearest-neighbor or vice versa. Other edges are removed. The weights on the surviving edges are unchanged. Therefore this is a much sparser graph. The number 10 is chosen arbitrarily and not tuned for semi-supervised learning.

   (c) 16×16 10NN unweighted. Same as '16×16 10NN weighted' except that the weights on the surviving edges are all set to 1. This represents a further simplification of prior knowledge.

   (d) 8×8 full. All images are down sampled to 8×8 by averaging 2×2 pixel bins. Lowering resolution helps to make Euclidean distance less sensitive to small spatial variations. The graph is fully connected with weights

   $$w_{ij} = \exp\left(-\sum_{d=1}^{64} \frac{(x'_{i,d} - x'_{j,d})^2}{140^2}\right) \tag{4.15}$$

   (e) 8×8 10NN weighted. Similar to '16×16 10NN weighted'.

   (f) 8×8 10NN unweighted. Ditto.

The classification accuracy with these graphs are shown in Figure 4.3(a). Different graphs give very different accuracies. This should be a reminder that the quality of the graph determines the performance of harmonic function (as well as semi-supervised learning methods based on graphs in general). 8×8 seems to be better than 16×16. Sparser graphs are better than fully connected graphs. The better graphs outperform SVM baselines when labeled set size is not too small.

2. **ten digits**. 10-class classification for 4000 OCR handwritten digit images. The class proportions are intentionally chosen to be skewed, with 213, 129, 100, 754, 970, 275, 585, 166, 353, and 455 images for digits "1,2,3,4,5,6,7,8,9,0" respectively. We use 6 graphs constructed similarly as in **1 vs. 2**. Figure 4.3(b) shows the result, which is similar to **1 vs. 2** except the overall accuracy is lower.

3. **odd vs. even**. Binary classification for OCR handwritten digits "1,3,5,7,9" vs. "0,2,4,6,8". Each digit has 400 images, i.e. 2000 per class and 4000 total. We show only the 8×8 graphs in Figure 4.3(c), which do not outperform the baseline.

4. **baseball vs. hockey** Binary document classification for rec.sport.baseball vs. rec.sport.hockey in the 20newsgroups dataset (18828 version). The processing of documents into tf.idf vectors has been described in section 3.2. The classes have 994 and 999 documents respectively. We report the results of three graphs in Figure 4.3(d):

   (a) full. A fully connected graph with weights

   $$w_{ij} = \exp\left(-\frac{1}{0.03}\left(1 - \frac{\langle d_i, d_j\rangle}{|d_i||d_j|}\right)\right) \tag{4.16}$$

   so that the weights decreases with the cosine similarity between document d_i, d_j.

   (b) 10NN weighted. Only symmetrized 10-nearest-neighbor edges are kept in the graph, with the same weights above. This was the graph in (Zhu et al., 2003a).

   (c) 10NN unweighted. Same as above except all weights are set to 1.

5. **PC vs. MAC** Binary classification on comp.sys.ibm.pc.hardware (number of documents 982) vs. comp.sys.mac.hardware (961) in the 20 newsgroups dataset. The three graphs are constructed in the same way as **baseball vs. hockey**. See Figure 4.3(e).

6. **religion vs. atheism** Binary classification on talk.religion.misc (628) vs. alt.atheism (799). See Figure 4.3(f). The three 20newsgroups tasks have increasing difficulty.

7. **isolet** This is the ISOLET dataset from the UCI data repository (Blake & Merz, 1998). It is a 26-class classification problem for isolated spoken English letter recognition. There are 7797 instances. We use the Euclidean distance on raw features, and create a 100NN unweighted graph. The result is in Figure 4.3(g).

8. **freefoodcam** The details of the dataset and graph construction are discussed in section 3.3. The experiments need special treatment compared to other datasets. Since we want to recognize people across multiple days, we only sample the labeled set from the first days of a person's appearance. This is harder and more realistic than sampling labeled set from the whole dataset. We show two graphs in Figure 4.3(h), one with $t_1 = 2$ seconds, $t_2 = 12$ hours, $k_c = 3$, $k_f = 1$, the other the same except $k_c = 1$.

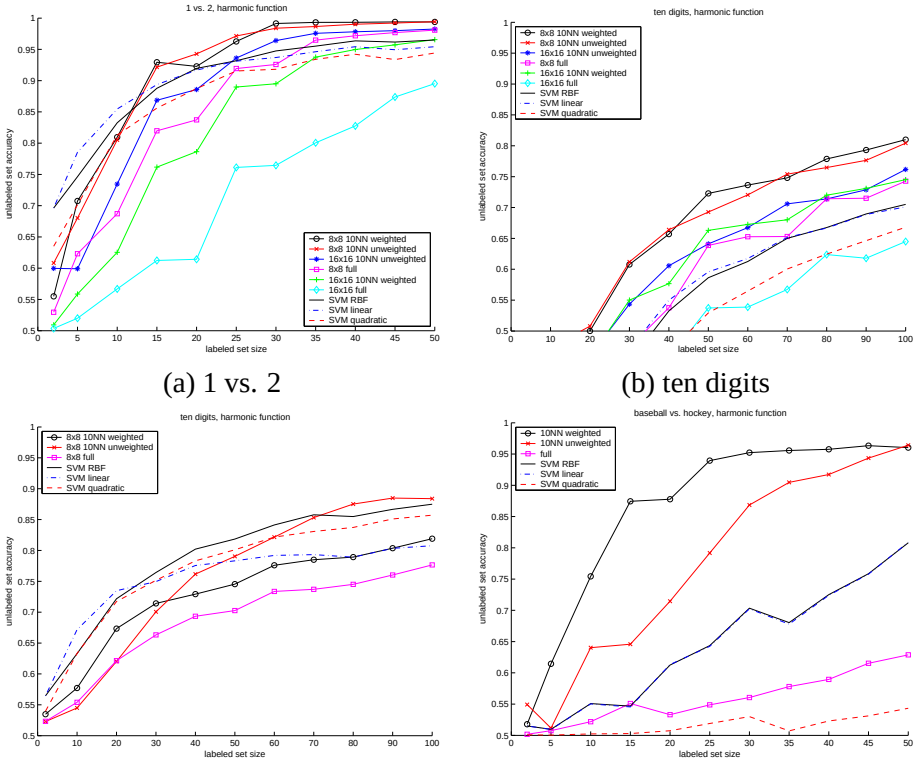
   The kernel for SVM baseline is optimized differently as well. We use an interpolated linear kernel $K(i,j) = w_t K_t(i,j) + w_c K_c(i,j) + w_f K_f(i,j)$, where K_t, K_c, K_f are linear kernels (inner products) on time stamp, color histogram, and face sub-image (normalized to 50×50 pixels) respectively. If an image i contains no face, we define $K_f(i,\cdot) = 0$. The interpolation weights w_t, w_c, w_f are optimized with cross validation.

The experiments demonstrate that the performance of harmonic function varies considerably depending on the graphs. With certain graphs, the semi-supervised learning method outperforms SVM, a standard supervised learning method. In particular sparse nearest-neighbor graphs, even unweighted, tend to outperform fully connected graphs. We believe the reason is that in fully connected graphs the edges between different classes, even with relatively small weights, create unwarrantedly strong connections across the classes. This highlights the sensitivity to the graph in graph-based semi-supervised learning methods.

It is also apparent from the results that the benefit of semi-supervised learning deminishes as the labeled set size grows. This suggests that semi-supervised learning is most helpful when the cost of getting labels is prohibitive.

CMN: Incorporating Class Proportion Knowledge

The harmonic function accuracy can be significantly improved, if we incorporate class proportion knowledge with the simple CMN heuristic. The class proportion is estimated from labeled data with Laplace (add one) smoothing. All the graphs and



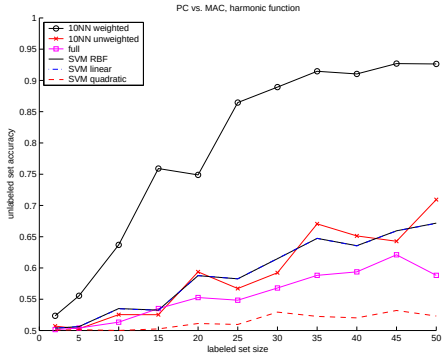
(a) 1 vs. 2

(b) ten digits

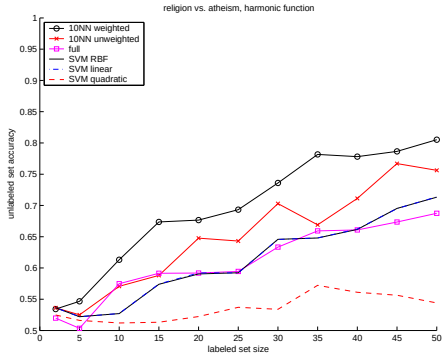
(c) odd vs. even

(d) baseball vs. hockey

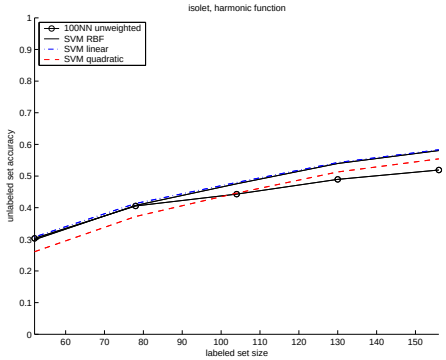
Figure 4.3: harmonic function accuracy



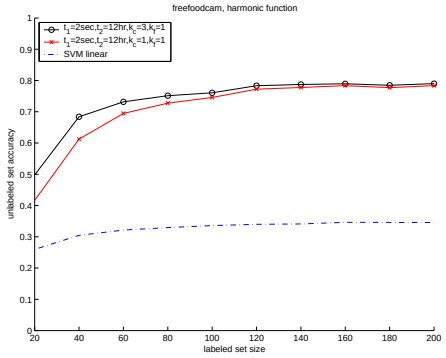
(e) PC vs. MAC



(f) religion vs. atheism



(g) isolet



(h) freefoodcam

Figure 4.3: harmonic function accuracy (continued)

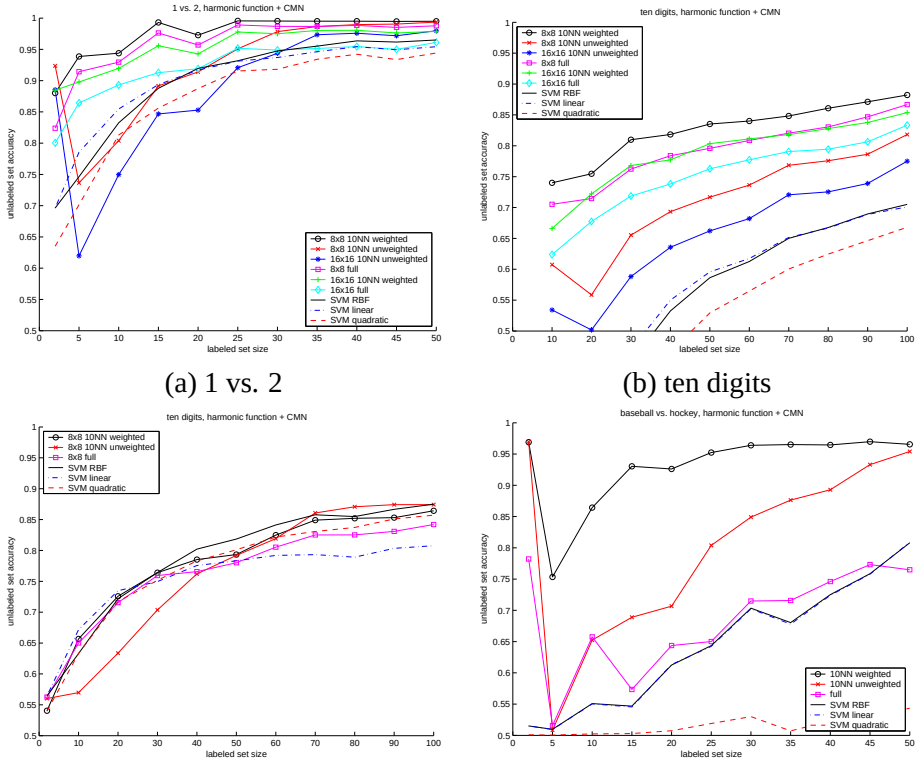
other settings are the same as in section 4.7. The CMN results are shown in Figure 4.4. Compared to Figure 4.3 we see that in most cases CMN helps to improve accuracy.

For several tasks, CMN gives a huge improvement for the smallest labeled set size. The improvement is so large that the curves become 'V' shaped at the left hand side. This is an artifact: we often use the number of classes as the smallest labeled set size. Because of our sampling method, there will be one instance from each class in the labeled set. The CMN class proportion estimation is thus uniform. Incidentally, many datasets have close to uniform class proportions. Therefore the CMN class proportion estimation is close to the truth for the smallest labeled set size, and produces large improvement. On the other hand, intermediate labeled set size tends to give the worst class proportion estimates and hence little improvement.

In conclusion, it is important to incorporate class proportion knowledge to assist semi-supervised learning. However for clarity, CMN is not used in the remaining experiments.

Dongles: Incorporating External Classifier

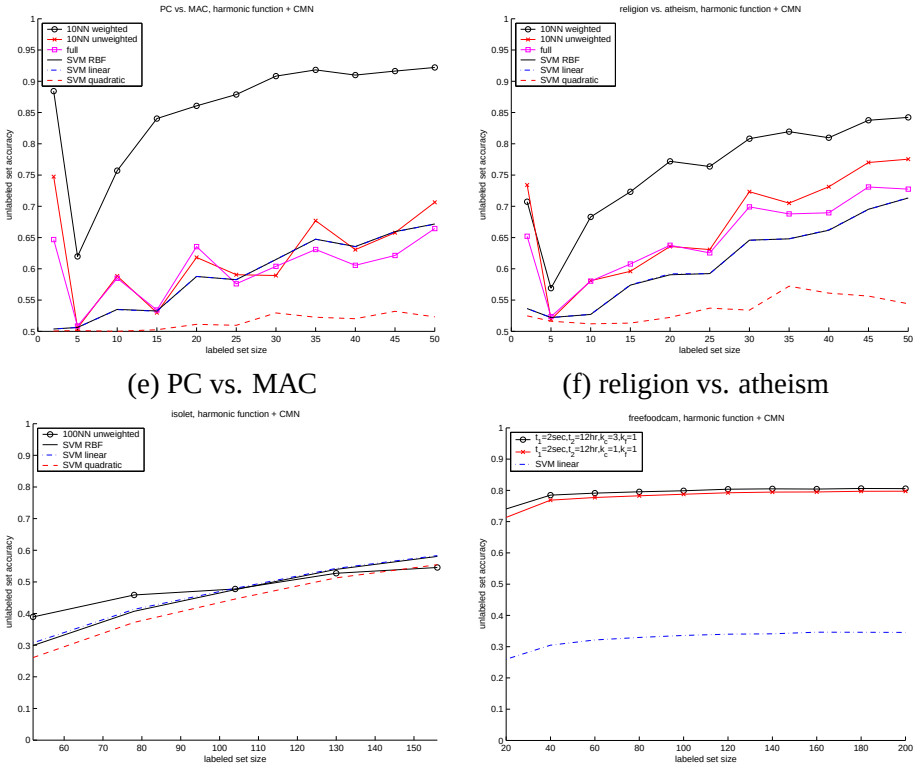
We use the **odd vs. even** task, where the RBF SVM baseline is sometimes better than the harmonic function with a 10NN unweighted graph. We augment the graph with a dongle on each unlabeled node. We use the hard (0/1) labels from the RBF SVM (Figure 4.3) on the dongles. The dongle transition probability η is set to 0.1 by cross validation. As before, we experiment on different labeled set sizes, and 30 random trials per size. In Figure 4.5, we compare the average accuracy of incorporating the external classifier (**dongle**) to the external classifier (**SVM**) or the harmonic function (**harmonic**) alone. The combination results in higher accuracy than either method alone, suggesting there is complementary information used by each.



(a) 1 vs. 2 (b) ten digits

(c) odd vs. even (d) baseball vs. hockey

Figure 4.4: CMN accuracy



(e) PC vs. MAC (f) religion vs. atheism

(g) isolet (h) freefoodcam

Figure 4.4: CMN accuracy (continued)

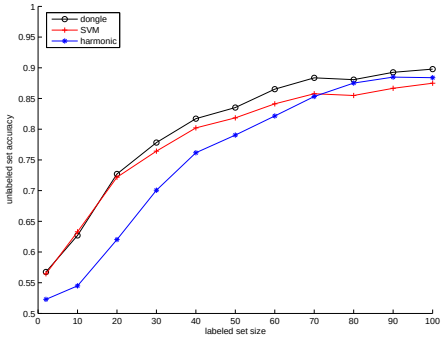


Figure 4.5: Incorporating external classifier with dongles

Chapter 5

Active Learning

In this chapter, we take a brief detour to look at the active learning problem. We combine semi-supervised learning and active learning naturally and efficiently.

5.1 Combining Semi-Supervised and Active Learning

So far, we assumed the labeled data set is given and fixed. In practice, it may make sense to utilize *active learning* in conjunction with semi-supervised learning. That is, we might allow the learning algorithm to pick unlabeled instances to be labeled by a domain expert. The expert returns the label, which will then be used as (or to augment) the labeled data set. In other words, if we have to label a few instances for semi-supervised learning, it may be attractive to let the learning algorithm tell us which instances to label, rather than selecting them randomly. We will limit the range of query selection to the unlabeled data set, a practice known as pool-based active learning or selective sampling.

There has been a great deal of research in active learning. For example, Tong and Koller (2000) select queries to minimize the version space size for support vector machines; Cohn et al. (1996) minimize the variance component of the estimated generalization error; Freund et al. (1997) employ a committee of classifiers, and query a point whenever the committee members disagree. Most of the active learning methods do not take further advantage of the large amount of unlabeled data once the queries are selected. The work by McCallum and Nigam (1998b) is an exception, where EM with unlabeled data is integrated into active learning. Another exception is (Muslea et al., 2002), which uses a semi-supervised learning method during training. In addition to this body of work from the machine learning community, there is a large literature on the closely related topic of experimental design in statistics; Chaloner and Verdinelli (1995) give a survey of experimental

design from a Bayesian perspective.

The Gaussian random fields and harmonic functions framework allows a natural combination of active learning and semi-supervised learning. In brief, the framework allows one to efficiently estimate the expected generalization error after querying a point, which leads to a better query selection criterion than naively selecting the point with maximum label ambiguity. Then, once the queries are selected and added to the labeled data set, the classifier can be trained using both the labeled and remaining unlabeled data. Minimizing the estimated generalization error was first proposed by Roy and McCallum (2001). We independently discovered the same idea (Zhu et al., 2003b), and the effective combination of semi-supervised learning and active learning is novel.

We perform active learning with the Gaussian random field model by greedily selecting queries from the unlabeled data to minimize the *risk* of the harmonic energy minimization function. The risk is the estimated generalization error of the Bayes classifier, and can be computed with matrix methods. We define the *true risk* $\mathcal{R}(h)$ of the Bayes classifier based on the harmonic function h to be

$$\mathcal{R}(h) \quad = \quad \sum_{i=1}^{n} \sum_{y_i=0,1} [\text{sgn}(h_i) \neq y_i] \; p^*(y_i)$$

where $\text{sgn}(h_i)$ is the Bayes decision rule with threshold 0.5, such that (with a slight abuse of notation) $\text{sgn}(h_i) = 1$ if $h_i > 0.5$ and $\text{sgn}(h_i) = 0$ otherwise. Here $p^*(y_i)$ is the unknown true label distribution at node i, given the labeled data. Because of this, $\mathcal{R}(h)$ is not computable. In order to proceed, it is necessary to make assumptions. We begin by assuming that we can estimate the unknown distribution $p^*(y_i)$ with the mean of the Gaussian field model:

$$p^*(y_i = 1) \approx h_i$$

Intuitively, recalling h_i is the probability of reaching 1 in a random walk on the graph, our assumption is that we can approximate the distribution using a biased coin at each node, whose probability of heads is h_i. With this assumption, we can compute the *estimated risk* $\widehat{\mathcal{R}}(h)$ as

$$\begin{aligned} \widehat{\mathcal{R}}(h) \quad &= \quad \sum_{i=1}^{n} [\text{sgn}(h_i) \neq 0] \, (1 - h_i) + [\text{sgn}(h_i) \neq 1] \, h_i \\ &= \quad \sum_{i=1}^{n} \min(h_i, 1 - h_i) \end{aligned} \tag{5.1}$$

If we perform active learning and query an unlabeled node k, we will receive an answer y_k (0 or 1). Adding this point to the training set and retraining, the Gaussian

field and its mean function will of course change. We denote the new harmonic function by $h^{+(x_k,y_k)}$. The estimated risk will also change:

$$\widehat{\mathcal{R}}(h^{+(x_k,y_k)}) \quad = \quad \sum_{i=1}^{n} \min(h_i^{+(x_k,y_k)}, 1 - h_i^{+(x_k,y_k)})$$

Since we do not know what answer y_k we will receive, we again assume the probability of receiving answer $p^*(y_k = 1)$ is approximately h_k. The *expected* estimated risk after querying node k is therefore

$$\widehat{\mathcal{R}}(h^{+x_k}) \quad = \quad (1 - h_k)\, \widehat{\mathcal{R}}(h^{+(x_k,0)}) + h_k\, \widehat{\mathcal{R}}(h^{+(x_k,1)})$$

The active learning criterion we use in this paper is the greedy procedure of choosing the next query k that minimizes the expected estimated risk:

$$k \quad = \quad \arg\min_{k'} \widehat{\mathcal{R}}(h^{+x_{k'}}) \tag{5.2}$$

To carry out this procedure, we need to compute the harmonic function $h^{+(x_k,y_k)}$ after adding (x_k, y_k) to the current labeled training set. This is the retraining problem and is computationally intensive in general. However for Gaussian fields and harmonic functions, there is an efficient way to retrain. Recall that the harmonic function solution is

$$h_U = -\Delta_{UU}^{-1} \Delta_{UL} Y_L$$

What is the solution if we fix the value y_k for node k? This is the same as finding the conditional distribution of all unlabeled nodes, given the value of y_k. In Gaussian fields the conditional on unlabeled data is multivariate Normal distributions $\mathcal{N}(h_U, \Delta_{UU}^{-1})$. A standard result (a derivation is given in Appendix A) gives the mean of the conditional once we fix y_k:

$$h_U^{+(x_k,y_k)} \quad = \quad h_U + (y_k - h_k) \frac{(\Delta_{UU}^{-1})_{\cdot k}}{(\Delta_{UU}^{-1})_{kk}}$$

where $(\Delta_{UU}^{-1})_{\cdot k}$ is the k-th column of the inverse Laplacian on unlabeled data, and $(\Delta_{UU}^{-1})_{kk}$ is the k-th diagonal element of the same matrix. Both are already computed when we compute the harmonic function h. This is a linear computation and therefore can be carried out efficiently.

To summarize, the active learning algorithm is shown in Figure 5.1. The time complexity to find the best query is $O(n^2)$. As a final word on computational efficiency, we note that after adding query x_k and its answer to L, in the next iteration we will need to compute $((\Delta_{UU})_{\neg k})^{-1}$, the inverse of the Laplacian on unlabeled data, with the row/column for x_k removed. Instead of naively taking the inverse, there are efficient algorithms to compute it from $(\Delta_{UU})^{-1}$; a derivation is given in Appendix B.

Input: L, U, weight matrix W
While more labeled data required:
  Compute harmonic h using (4.11)
  Find best query k using (5.2)
  Query point x_k, receive answer y_k
  Add (x_k, y_k) to L, remove x_k from U
end
Output: L and classifier h.

Figure 5.1: The active learning algorithm

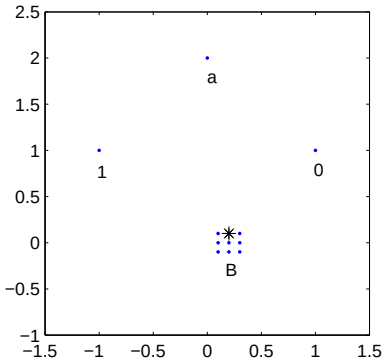


Figure 5.2: Entropy Minimization selects the most uncertain point a as the next query. Our method will select a point in B, a better choice.

5.2 Why not Entropy Minimization

We used the estimated generalization error to select queries. A different query selection criterion, *entropy minimization* (or selecting the most uncertain instance), has been suggested in some papers. We next show why it is inappropriate when the loss function is based on individual instances. Such loss functions include the widely used *accuracy* for classification and *mean squared error* for regression.

To illustrate the idea, Figure 5.2 shows a synthetic dataset with two labeled data (marked '1', '0'), an unlabeled point 'a' in the center above and a cluster of 9 unlabeled points 'B' below. 'B' is slighted shifted to the right. The graph is fully connected with weights $w_{ij} = \exp(-d_{ij}^2)$, where d_{ij} is the Euclidean distance between i, j. In this configuration, we have the most uncertainty in 'a': the harmonic function at node 'a' is $h(a) = 0.43$. Points in 'B' have their harmonic func-

tion values around 0.32. Therefore entropy minimization will pick 'a' as the query. However, the risk minimization criterion picks the upper center point (marked with a star) in 'B' to query, instead of 'a'. In fact the estimated risk is $\widehat{\mathcal{R}}(a) = 2.9$, and $\widehat{\mathcal{R}}(b \in B) \approx 1.1$. Intuitively knowing the label of one point in B let us know the label of all points in B, which is a larger gain. Entropy minimization is worse than risk minimization in this example.

The root of the problem is that entropy does not account for the loss of making a large number of *correlated mistakes*. In a pool-based incremental active learning setting, given the current unlabeled set U, entropy minimization finds the query $q \in U$ such that the conditional entropy $H(U \setminus q|q)$ is minimized. As $H(U \setminus q|q) = H(U) - H(q)$, it amounts to selecting q with the largest entropy, or the most ambiguous unlabeled point as the query. Consider another example where $U = \{a, b_1, \ldots, b_{100}\}$. Let $P(a = +) = P(a = -) = 0.5$ and $P(b_i = +) = 0.51, P(b_i = -) = 0.49$ for $i = 1 \ldots 100$. Furthermore let $b_1 \ldots b_{100}$ be perfectly correlated so they always take the same value; Let a and b_i's be independent. Entropy minimization will select a as the next query since $H(a) = 1 > H(b_i) = 0.9997$. If our goal were to reduce uncertainty about U, such query selection is good: $H(b_1 \ldots b_{100}|a) = 0.9997 < H(a, b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_{100}|b_i) = H(a|b_i) = 1$. However if our loss function is the accuracy on the remaining instances in U, the picture is quite different. After querying a, $P(b_i = +)$ remains at 0.51, so that each b_i incurs a Bayes error of 0.49 by always predict $b_i = +$. The problem is that the individual error adds up, and the overall accuracy is $0.51 * 100/100 = 0.51$. On the other hand if we query b_1, we know the labels of $b_2 \ldots b_{100}$ too because of their perfect correlation. The only error we might make is on a with Bayes error of 0.5. The overall accuracy is $(0.5 + 1 * 99)/100 = 0.995$. The situation is analogous to speech recognition in which one can measure the 'word level accuracy' or 'sentence level accuracy' where a sentence is correct if all words in it are correct. The sentence corresponds to the whole U in our example. Entropy minimization is more aligned with sentence level accuracy. Nevertheless since most active learning systems use instance level loss function, it can leads to suboptimal query choices as we show above.

5.3 Experiments

Figure 5.3 shows a check-board synthetic dataset with 400 points. We expect active learning to discover the pattern and query a small number of representatives from each cluster. On the other hand, we expect a much larger number of queries if queries are randomly selected. We use a fully connected graph with weight $w_{ij} = \exp(-d_{ij}^2/4)$. We perform 20 random trials. At the beginning of each trial we

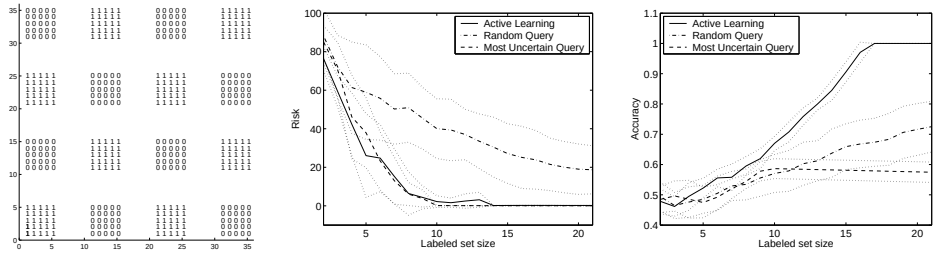
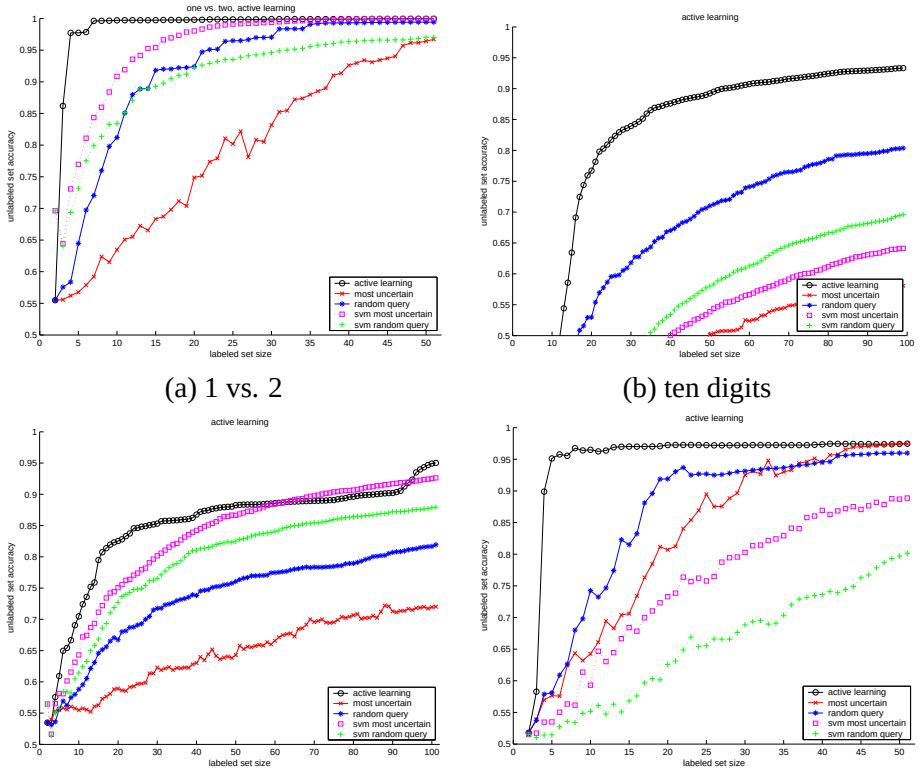


Figure 5.3: A check-board example. Left: dataset and true labels; Center: estimated risk; Right: classification accuracy.

randomly select a positive example and a negative example as the initial training set. We then run active learning and compare it to two baselines: (1) "Random Query": randomly selecting the next query from U; (2) "Most Uncertain Query": selecting the most uncertain instance in U, i.e. the one with h closest to 0.5. In each case, we run for 20 iterations (queries). At each iteration, we plot the estimated risk (5.1) of the selected query (center), and the classification accuracy on U (right). The error bars are ± 1 standard deviation, averaged over the random trials. As expected, with risk minimization active learning we reduce the risk more quickly than random queries or the most uncertain queries. In fact, risk minimization active learning with about 15 queries (plus 2 initial random points) learns the correct concept, which is nearly optimal given that there are 16 clusters. Looking at the queries, we find that active learning mostly selects the central points within the clusters.

Next, we ran the risk minimization active learning method on several tasks (marked **active learning** in the plots). We compare it with several alternative ways of picking queries:

- **random query**. Randomly select the next query from the unlabeled set. Classification on the unlabeled set is based on the harmonic function. Therefore, this method consists of no active learning, but only semi-supervised learning.
- **most uncertain**. Pick the most ambiguous point (h closest to 0.5 for binary problems) as the query. Classification is based on the harmonic function.
- **SVM random query**. Randomly select the next query from the unlabeled set. Classification with SVM. This is neither active nor semi-supervised learning.
- **SVM most uncertain**. Pick the query closest to the SVM decision boundary.



(a) 1 vs. 2 (b) ten digits

(c) odd vs. even (d) baseball vs. hockey

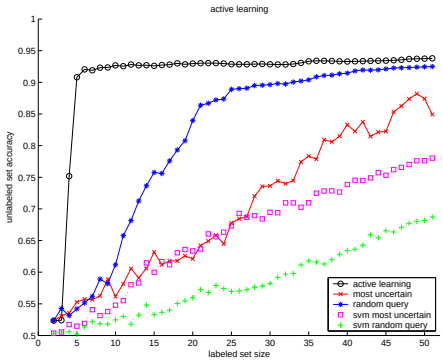
Figure 5.4: Active learning accuracy

Classification with SVM.

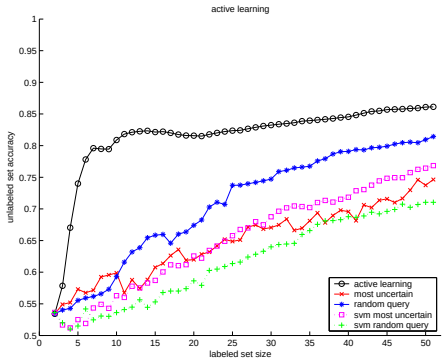
For each task, we use the best graph for harmonic functions, and the best kernel for SVM, as in section 4.7. We run 30 trials and the plots are the average. In each trial, we start from a randomly selected labeled set, so that each class has exactly one labeled example. The query selection methods mentioned above are used independently to grow the labeled set until a predetermined size. We plot the classification accuracy on the remaining unlabeled data in Figure 5.4. For the **FreeFoodCam** task, there are two experiments: 1. We allow the queries to come from all days; 2. From only the first days of a person's first appearance.

It is interesting to see what queries are selected by different methods. Figures 5.5 and 5.6 compare the first few queries for the **1 vs. 2** and **ten digits** tasks. In each case, the initial labeled set is the same.

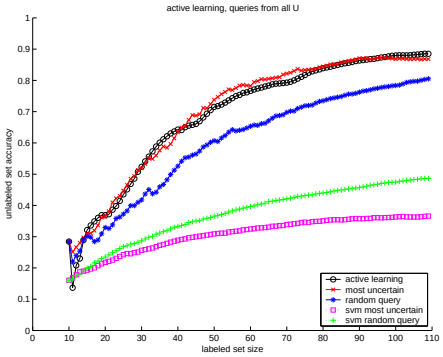
The combined semi-supervised learning and risk minimization active learning method performs well on the tasks. Compared to the results reported in (Roy &



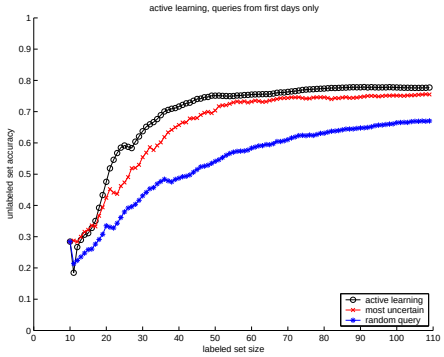
(e) PC vs. MAC



(f) religion vs. atheism



(g) freefoodcam, query from all days



(h) freefoodcam, query from the first days

Figure 5.4: Active learning accuracy (continued)

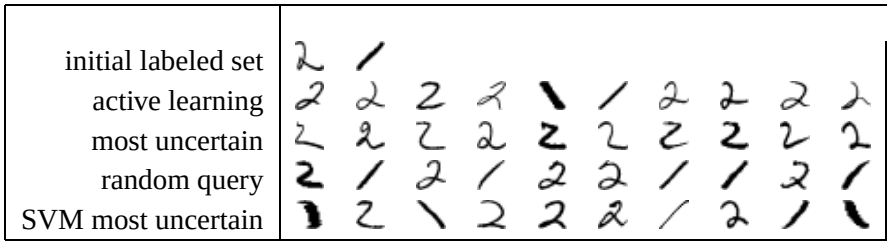


Figure 5.5: The first few queries selected by different active learning methods on the **1 vs. 2** task. All methods start with the same initial labeled set.

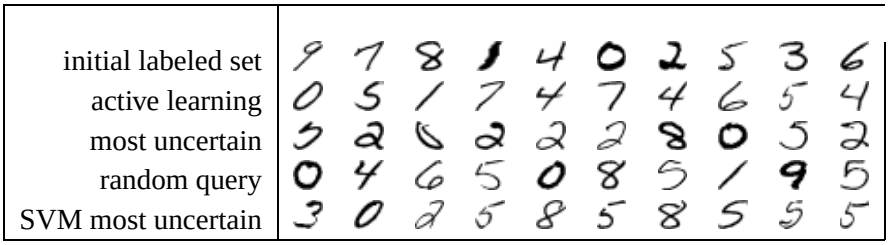


Figure 5.6: The first few queries selected by different active learning methods on the **ten digits** task. All methods start with the same initial labeled set.

McCallum, 2001), we think that good semi-supervised learning algorithm is a key to the success of the active learning scheme.

Chapter 6

Connection to Gaussian Processes

A Gaussian process define a prior $p(f(x))$ over function values $f(x)$, where x ranges over an infinite input space. It is an extension to an n-dimensional Gaussian distribution as n goes to infinity. A Gaussian process is defined by its mean function $\mu(x)$ (usually taken to be zero everywhere), and a covariance function $C(x, x')$. For any finite set of points $x_1, \ldots, x_m$, the Gaussian process on the set reduces to an m-dimensional Gaussian distribution with a covariance matrix $C_{ij} = C(x_i, x_j)$, for $i, j = 1 \ldots m$. More information can be found in Chapter 45 of (MacKay, 2003).

Gaussian random fields are equivalent to Gaussian processes that are restricted to a finite set of points. Thus, the standard machineries for Gaussian processes can be used for semi-supervised learning. Through this connection, we establish the link between the graph Laplacian and kernel methods in general.

6.1 A Finite Set Gaussian Process Model

Recall for any real-valued function f on the graph, the energy is defined as

$$E(f) = \frac{1}{2} \sum_{i,j} w_{ij} \left(f(i) - f(j)\right)^2 = f^\top \Delta f \tag{6.1}$$

the corresponding Gaussian random field is

$$p(f) = \frac{1}{Z} e^{-\beta E(f)} = \frac{1}{Z} e^{-\beta f^\top \Delta f} \tag{6.2}$$

The Gaussian random field is nothing but a multivariate Gaussian distribution on the nodes. Meanwhile a Gaussian process restricted to finite data is a multivariate Gaussian distribution too (MacKay, 1998). This indicates a connection between

Gaussian random fields and finite set Gaussian processes. Notice the 'finite set Gaussian processes' are not real Gaussian processes, since the kernel matrix is only defined on $L \cup U$, not the whole input space X.

Equation (6.2) can be viewed as a Gaussian process restricted to $L \cup U$ with covariance matrix $(2\beta\Delta)^{-1}$. However the covariance matrix is an improper prior. The Laplacian Δ by definition has a zero eigenvalue with constant eigenvector $\mathbf{1}$. To see this note that the degree matrix D is the row sum of W. This makes Δ singular: we cannot invert Δ to get the covariance matrix. To make a proper prior out of the Laplacian, we can smooth its spectrum to remove the zero eigenvalues, as suggested in (Smola & Kondor, 2003). In particular, we choose to transform the eigenvalues λ according to the function $r(\lambda) = \lambda + 1/\sigma^2$ where $1/\sigma^2$ is a small smoothing parameter. This gives the *regularized Laplacian*

$$\Delta + I/\sigma^2 \tag{6.3}$$

Using the regularized Laplacian, we define a zero mean prior as

$$p(f) \propto \exp\left(-\frac{1}{2} f^\top \tilde{\Delta} f\right) \tag{6.4}$$

which corresponds to a kernel with Gram matrix (i.e. covariance matrix)

$$K = \tilde{\Delta}^{-1} = \left(2\beta(\Delta + I/\sigma^2)\right)^{-1} \tag{6.5}$$

We note several important aspects of the resulting finite set Gaussian process:

- $f \sim \mathcal{N}\left(\mathbf{0}, \tilde{\Delta}^{-1}\right)$;
- Unlike Δ, $\tilde{\Delta}$ gives a proper covariance matrix.
- The parameter β controls the overall sharpness of the distribution; large β means $p(f)$ is more peaked around its mean.
- The parameter σ^2 controls the amount of spectral smoothing; large σ smoothes less.
- The kernel (covariance) matrix $K = \tilde{\Delta}^{-1}$ is the inverse of a function of the Laplacian Δ. Therefore the covariance between any two point i, j in general depends on *all the points*. This is how unlabeled data influences the prior.

The last point warrants further explanation. In many standard kernels, the entries are 'local'. For example, in a radial basis function (RBF) kernel K, the matrix entry $k_{ij} = \exp\left(-d_{ij}^2/\alpha^2\right)$ only depends on the distance between i, j and *not any other*

points. In this case unlabeled data is useless because the influence of unlabeled data in K is marginalized out. In contrast, the entries in kernel (6.4) depends on all entries in Δ, which in turn depends on all edge weights W. Thus, unlabeled data will influence the kernel, which is desirable for semi-supervised learning. Another way to view the difference is that in RBF (and many other) kernels we parameterize the covariance matrix directly, while with graph Laplacians we parameterize the *inverse covariance matrix*.

6.2 Incorporating a Noise Model

In moving from Gaussian fields to finite set Gaussian processes, we no longer assume that the soft labels f_L for the labeled data are fixed at the observed labels Y_L. Instead we now assume the data generation process is $x \rightarrow f \rightarrow y$, where $f \rightarrow y$ is a noisy label generation process. We use a sigmoid noise model between the hidden soft labels f_i and observed labels y_i:

$$P(y_i|f_i) = \frac{e^{\gamma f_i y_i}}{e^{\gamma f_i y_i} + e^{-\gamma f_i y_i}} = \frac{1}{1 + e^{-2\gamma f_i y_i}} \tag{6.6}$$

where γ is a hyperparameter which controls the steepness of the sigmoid. This assumption allows us to handle noise in training labels, and is a common practice in Gaussian process classification.

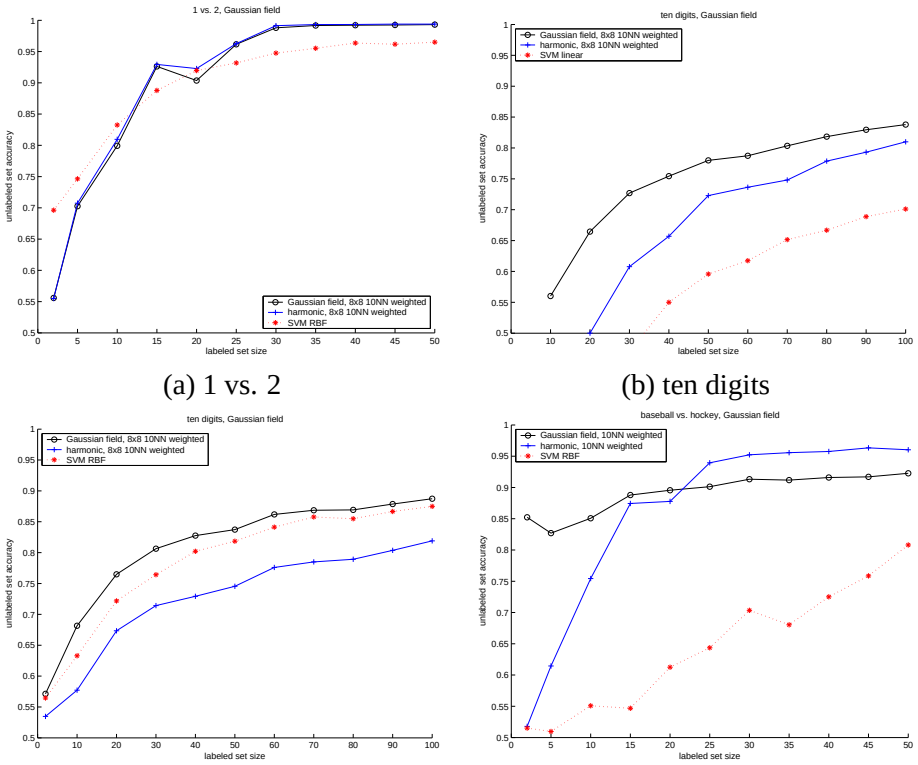
We are interested in $p(Y_U|Y_L)$, the labels for unlabeled data. We first need to compute the posterior distribution $p(f_L, f_U|Y_L)$. By Bayes' theorem,

$$p(f_L, f_U|Y_L) = \frac{\prod_{i=1}^{l} P(y_i|f_i)p(f_L, f_U)}{P(Y_L)} \tag{6.7}$$

Because of the noise model, the posterior is not Gaussian and has no closed form solution. There are several ways to approximate the posterior. For simplicity we use the Laplace approximation to find the approximate $p(f_L, f_U|Y_L)$. A derivation can be found in Appendix C, which largely follows (Herbrich, 2002) (B.7). Bayesian classification is based on the posterior distribution $p(Y_U|Y_L)$. Since under the Laplace approximation this distribution is also Gaussian, the classification rule depends only on the sign of the mean (which is also the mode) of f_U.

6.3 Experiments

We compare the accuracy of Gaussian process classification with the 0.5-threshold harmonic function (without CMN). To simplify the plots, we use the same graphs



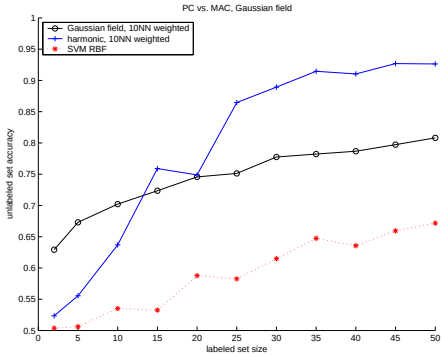
(a) 1 vs. 2 (b) ten digits

(c) odd vs. even (d) baseball vs. hockey

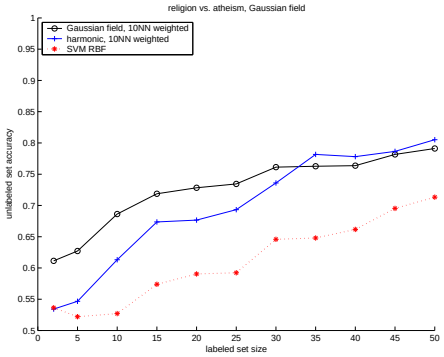
Figure 6.1: Gaussian process accuracy

that give the best harmonic function accuracy (except **FreeFoodCam**). To aid comparison we also show SVMs with the best kernel among linear, quadratic or RBF. In the experiments, the inverse temperature parameter β, smoothing parameter σ and noise model parameter γ are tuned with cross validation for each task. The results are in Figure 6.1.

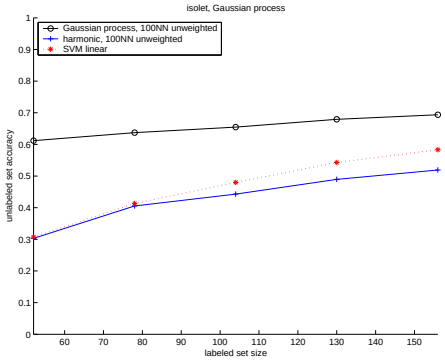
For **FreeFoodCam** we also use two other graphs with no face edges at all ($k_f = 0$). The first one limits color edges to within 12 hours ($t_2 = 12$ hour), thus the first days that contain the labeled data is disconnected from the rest. The second one allows color edges on far away images ($t_2 = \infty$). Neither has good accuracy, indicating that face is an important feature to use.



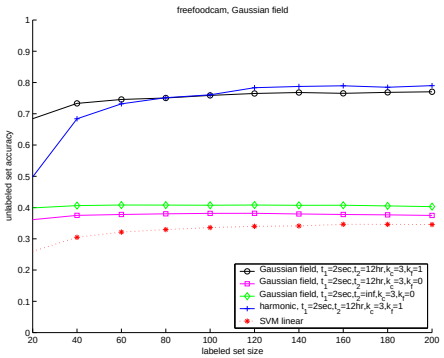
(e) PC vs. MAC



(f) religion vs. atheism



(g) isolet



(h) freefoodcam

Figure 6.1: Gaussian process accuracy (continued)

6.4 Extending to Unseen Data

We have so far restricted ourselves to the $L \cup U$ nodes in the graph. In this finite case Gaussian processes are nothing but n-dimensional multivariate normal distributions, and are equivalent to Gaussian random fields. However Gaussian fields, by definition, cannot handle unseen instances. Any new data points need to become additional nodes in the graph. The Laplacian and kernel matrices need to be re-computed, which is expensive. We would like to extend the framework to allow arbitrary new points. Equivalently, this is the problem of induction instead of transduction.

The simplest strategy is to divide the input space into Voronoi cells. The Voronoi cells are centered on instances in $L \cup U$. We classify any new instance x by the Voronoi cell it falls into. Let $x^* \in L \cup U$ be the point closest to x:

$$x^* = \arg\max_{z \in L \cup U} w_{xz} \tag{6.8}$$

where closeness is measured by weights w_{xz}. From an algorithmic point of view, we classify x by its 1-nearest-neighbor x^*. When the unlabeled data size is large, the approximation is reasonable.

We will discuss more inductive methods in Chapter 10.

Chapter 7

Graph Hyperparameter Learning

Previously we assumed that the weight matrix W is given and fixed. In this chapter we investigate *learning* the weights from both labeled and unlabeled data. We present three methods. The first one is evidence maximization in the context of Gaussian processes. The second is entropy minimization, and the third one is based on minimum spanning trees. The latter ones are heuristic but also practical.

7.1 Evidence Maximization

We assume the edge weights are parameterized with hyperparameters Θ. For instance the edge weights can be

$$w_{ij} = \exp\left(-\sum_{d=1}^{D} \frac{(x_{i,d} - x_{j,d})^2}{\alpha_d^2}\right)$$

and $\Theta = \{\alpha_1, \ldots, \alpha_D\}$. To learn the weight hyperparameters in a Gaussian process, one can choose the hyperparameters that maximize the log likelihood: $\Theta^* = \arg\max_\Theta \log p(y_L|\Theta)$. $\log p(y_L|\Theta)$ is known as the evidence and the procedure is also called evidence maximization . One can also assume a prior on Θ and find the maximum a posteriori (MAP) estimate $\Theta^* = \arg\max_\Theta \log p(y_L|\Theta) + \log p(\Theta)$. The evidence can be multimodal and usually gradient methods are used to find a mode in hyperparameter space. This requires the derivatives $\partial \log p(y_L|\Theta)/\partial\Theta$. A complete derivation is given in Appendix D.

In a full Bayesian setup, one would average over all hyperparameter values (weighted by the posterior $p(\Theta|y_L)$) instead of using a point estimate Θ^*. This usually involves Markov Chain Monte Carlo techniques, and is not pursued in this paper.

	regularized evidence		accuracy	
task	before	after	before	after
1 vs. 2	-24.6	-23.9	0.973	0.982
7 vs. 9	-40.5	-39.9	0.737	0.756

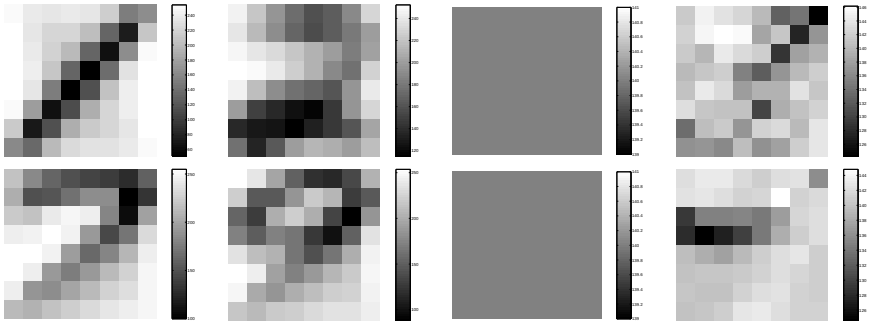
Table 7.1: the regularized evidence and classification before and after learning α's for the two digits recognition tasks

We use binary OCR handwritten digits recognition tasks as our example, since the results are more interpretable. We choose two tasks: "1 vs. 2" which has been presented previously, and " 7 vs. 9" which are the two most confusing digits in terms of Euclidean distance. We use fully connected graphs with weights

$$w_{ij} = \exp\left(-\sum_{d=1}^{64} \frac{(x_{i,d} - x_{j,d})^2}{\alpha_d^2}\right) \tag{7.1}$$

The hyperparameters are the 64 length scales α_d for each pixel dimension on 8×8 images. Intuitively they determine which pixel positions are salient for the classification task: if α_d is close to zero, a difference at pixel position d will be magnified; if it is large, pixel position d will be essentially ignored. The weight function is an extension to eq (4.15) by giving each dimension its own length scale. For each task there are 2200 images, and we run 10 trials, in each trial we randomly pick 50 images as the labeled set. The rest is used as unlabeled set. For each trial we start at $\alpha_i = 140, i = 1 \ldots 64$, which is the same as in eq (4.15). We compute the gradients for α_i for evidence maximization. However since there are 64 hyperparameters and only 50 labeled points, regularization is important. We use a Normal prior on the hyperparameters which is centered at the initial value: $p(\alpha_i) \sim \mathcal{N}(140, 30^2), i = 1 \ldots 64$. We use a line search algorithm to find a (possibly local) optimum for the α's.

Table 7.1 shows the regularized evidence and classification before and after learning α's for the two tasks. Figure 7.1 compares the learned hyperparameters with the mean images of the tasks. Smaller (darker) α's correspond to feature dimensions in which the learning algorithm pays more attention. It is obvious, for instance in the **7 vs. 9** task, that the learned hyperparameters focus on the 'gap on the neck of the image', which is the distinguishing feature between 7's and 9's.



a b c d

Figure 7.1: Graph hyperparameter learning. The upper row is for the **1 vs. 2** task, and the lower row for **7 vs. 9**. The four images are: (a,b) Averaged digit images for the two classes; (c) The 64 initial length scale hyperparameters α, shown as an 8×8 array; (d) Learned hyperparameters.

7.2 Entropy Minimization

Alternatively, we can use *average label entropy* as a heuristic criterion for parameter learning [1]. This heuristic uses only the harmonic function and does not depend on the Gaussian process setup.

The average label entropy $H(h)$ of the harmonic function h is defined as

$$H(h) = \frac{1}{u} \sum_{i=l+1}^{l+u} H_i(h(i)) \tag{7.2}$$

where $H_i(h(i)) = -h(i) \log h(i) - (1-h(i)) \log(1-h(i))$ is the Shannon entropy of individual unlabeled data point i. Here we use the random walk interpretation of h, relying on the maximum principle of harmonic functions which guarantees that $0 \leq h(i) \leq 1$ for $i \in U$. Small entropy implies that $h(i)$ is close to 0 or 1; this captures the intuition that a good W (equivalently, a good set of hyperparameters Θ) should result in a *confident* labeling. There are of course many arbitrary labelings of the data that have low entropy, which might suggest that this criterion will not work. However, it is important to point out that we are constraining h on the labeled data—most of these arbitrary low entropy labelings are inconsistent with this constraint. In fact, we find that the space of low entropy labelings achievable by harmonic function is small and lends itself well to tuning the hyperparameters.

[1] We could have used the estimated risk, cf. Chapter 5. The gradient will be more difficult because of the min function.

As an example, let us consider the case where weights are parameterized as (7.1). We can apply entropy minimization but there is a complication, namely H has a minimum at 0 as $\alpha_d \to 0$. As the length scale approaches zero, the tail of the weight function (7.1) is increasingly sensitive to the distance. In the end, the label predicted for an unlabeled example is dominated by its nearest neighbor's label, which results in the following equivalent labeling procedure: (1) starting from the labeled data set, find the unlabeled point x_u that is closest to some labeled point x_l; (2) label x_u with x_l's label, put x_u in the labeled set and repeat. Since these are hard labels, the entropy is zero. This solution is desirable only when the classes are well separated, and is inferior otherwise. This complication can be avoided by smoothing the transition matrix. Inspired by analysis of the PageRank algorithm in (Ng et al., 2001b), we smooth the transition matrix P with the uniform matrix $\mathcal{U}$: $\mathcal{U}_{ij} = 1/n$. The smoothed transition matrix is $\tilde{P} = \epsilon\mathcal{U} + (1-\epsilon)\,P$.

We use gradient descent to find the hyperparameters α_d that minimize H. The gradient is computed as

$$\frac{\partial H}{\partial \alpha_d} = \frac{1}{u} \sum_{i=l+1}^{l+u} \log\left(\frac{1-h(i)}{h(i)}\right) \frac{\partial h(i)}{\partial \alpha_d} \tag{7.3}$$

where the values $\partial h(i)/\partial\alpha_d$ can be read off the vector $\partial h_U/\partial\alpha_d$, which is given by

$$\frac{\partial h_U}{\partial \alpha_d} \;=\; (I - \tilde{P}_{UU})^{-1} \left(\frac{\partial \tilde{P}_{UU}}{\partial \alpha_d} h_U + \frac{\partial \tilde{P}_{UL}}{\partial \alpha_d} Y_L\right) \tag{7.4}$$

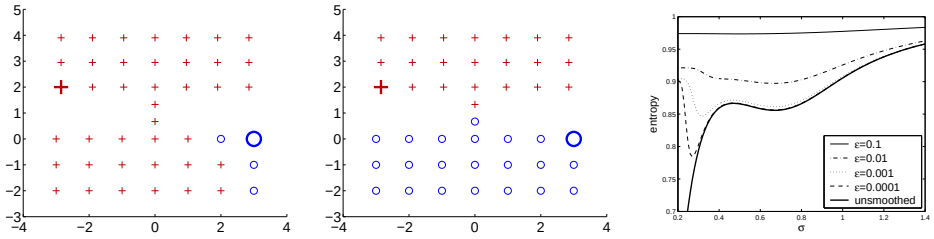
using the fact that $dX^{-1} = -X^{-1}(dX)X^{-1}$. Both $\partial\tilde{P}_{UU}/\partial\alpha_d$ and $\partial\tilde{P}_{UL}/\partial\alpha_d$ are sub-matrices of $\partial\tilde{P}/\partial\alpha_d = (1-\epsilon)\frac{\partial P}{\partial\alpha_d}$. Since the original transition matrix P is obtained by normalizing the weight matrix W, we have that

$$\frac{\partial p_{ij}}{\partial \alpha_d} \;=\; \frac{\frac{\partial w_{ij}}{\partial\alpha_d} - p_{ij}\sum_{n=1}^{l+u}\frac{\partial w_{in}}{\partial\alpha_d}}{\sum_{n=1}^{l+u} w_{in}} \tag{7.5}$$

Finally, $\frac{\partial w_{ij}}{\partial\alpha_d} = 2w_{ij}(x_{di} - x_{dj})^2/\alpha_d^3$.

In the above derivation we use h_U as label probabilities directly; that is, $p(y_i = 1) = h_U(i)$. If we incorporate class proportion information, or combine the harmonic function with other classifiers, it makes sense to minimize entropy on the combined probabilities. For instance, if we incorporate class proportions using CMN, the probability is given by

$$h'(i) = \frac{q(u - \sum h_U)h_U(i)}{q(u - \sum h_U)h_U(i) + (1-q)\sum h_U(1-h_U(j))} \tag{7.6}$$



(a) (b) (c)

Figure 7.2: The effect of parameter α on the harmonic function. (a) If not smoothed, $H \rightarrow 0$ as $\alpha \rightarrow 0$, and the algorithm performs poorly. (b) Result at optimal $\alpha = 0.67$, smoothed with $\epsilon = 0.01$ (c) Smoothing helps to remove the entropy minimum.

and we use this probability in place of $h(i)$ in (7.2). The derivation of the gradient descent rule is a straightforward extension of the above analysis.

We use a toy dataset in Figure 7.2 as an example for Entropy Minimization. The upper grid is slightly tighter than the lower grid, and they are connected by a few data points. There are two labeled examples, marked with large symbols. We learn the optimal length scales for this dataset by minimizing entropy on unlabeled data.

To simplify the problem, we first tie the length scales in the two dimensions, so there is only a single parameter α to learn. As noted earlier, without smoothing, the entropy approaches the minimum at 0 as $\alpha \rightarrow 0$. Under such conditions, the harmonic function is usually undesirable, and for this dataset the tighter grid "invades" the sparser one as shown in Figure 7.2(a). With smoothing, the "nuisance minimum" at 0 gradually disappears as the smoothing factor ϵ grows, as shown in Figure 7.2(c). When we set $\epsilon = 0.01$, the minimum entropy is 0.898 bits at $\alpha = 0.67$. The harmonic function under this length scale is shown in Figure 7.2(b), which is able to distinguish the structure of the two grids.

If we allow separate α's for each dimension, parameter learning is more dramatic. With the same smoothing of $\epsilon = 0.01$, α_x keeps growing toward infinity (we use $\alpha_x = 10^{16}$ for computation) while α_y stabilizes at 0.65, and we reach a minimum entropy of 0.619 bits. In this case $\alpha_x \rightarrow \infty$ is legitimate; it means that the learning algorithm has identified the x-direction as irrelevant, based on both the labeled and unlabeled data. The harmonic function under these hyperparameters gives the same classification as shown in Figure 7.2(b).

7.3 Minimum Spanning Tree

If the graph edges are $\exp$-weighted with a single hyperparameter α (Section 3.4), we can set the hyperparameter α with the following heuristic. We construct a minimum spanning tree over all data points with Kruskal's algorithm (Kruskal, 1956). In the beginning no node is connected. During tree growth, the edges are examined one by one from short to long. An edge is added to the tree if it connects two separate components. The process repeats until the whole graph is connected. We find the first tree edge that connects two components with different labeled points in them. We regard the length of this edge d^0 as a heuristic to the minimum distance between different class regions. We then set $\alpha = d^0/3$ following the 3σ rule of Normal distribution, so that the weight of this edge is close to 0, with the hope that local propagation is then mostly within classes.

7.4 Discussion

Other ways to learn the weight hyperparameters are possible. For example one can try to maximize the kernel alignment to labeled data. This criterion will be used to learn a spectral transformation from the Laplacian to a graph kernel in Chapter 8. There the graph weights are fixed, and the hyperparameters are the eigenvalues of the graph kernel. It is possible that one can instead fix a spectral transformation but learn the weight hyperparameters, or better yet jointly learn both. The hope is the problem can be formulated as convex optimization. This remains future research.

Chapter 8

Kernels from the Spectrum of Laplacians

We used the inverse of a smoothed Laplacian as kernel matrix in Chapter 6. In fact, one can construct a whole family of graph kernels from the spectral decomposition of graph Laplacians. These kernels combine labeled and unlabeled data in a systematic fashion. In this chapter we devise the best one (in a certain sense) for semi-supervised learning.

8.1 The Spectrum of Laplacians

Let us denote the Laplacian Δ's eigen-decomposition by $\{\lambda_i, \phi_i\}$, so that $\Delta = \sum_{i=1}^{n} \lambda_i \phi_i \phi_i^\top$. We assume the eigenvalues are sorted in non-decreasing order. The Laplacian Δ has many interesting properties (Chung, 1997); For example Δ has exactly k zero eigenvalues $\lambda_1 = \cdots = \lambda_k = 0$, where k is the number of connected subgraphs. The corresponding eigenvectors $\phi_1, \ldots, \phi_k$ are constant over the individual subgraphs and zero elsewhere. Perhaps the most important property of the Laplacian related to semi-supervised learning is the following: a smaller eigenvalue λ corresponds to a smoother eigenvector ϕ over the graph; that is, the value $\sum_{ij} w_{ij}(\phi(i) - \phi(j))^2$ is small. Informally, a smooth eigenvector has the property that two elements of the vector have similar values if there are many large weight paths between the nodes in the graph. In a physical system, the smoother eigenvectors correspond to the major vibration modes. Figure 8.1(top) shows a simple graph consisting of two linear segments. The edges have the same weight 1. Its Laplacian spectral decomposition is shown below, where the eigenvalues are sorted from small to large. The first two eigenvalues should be zero – there are numerical errors in Matlab eigen computation. As the eigenvalues increase, the

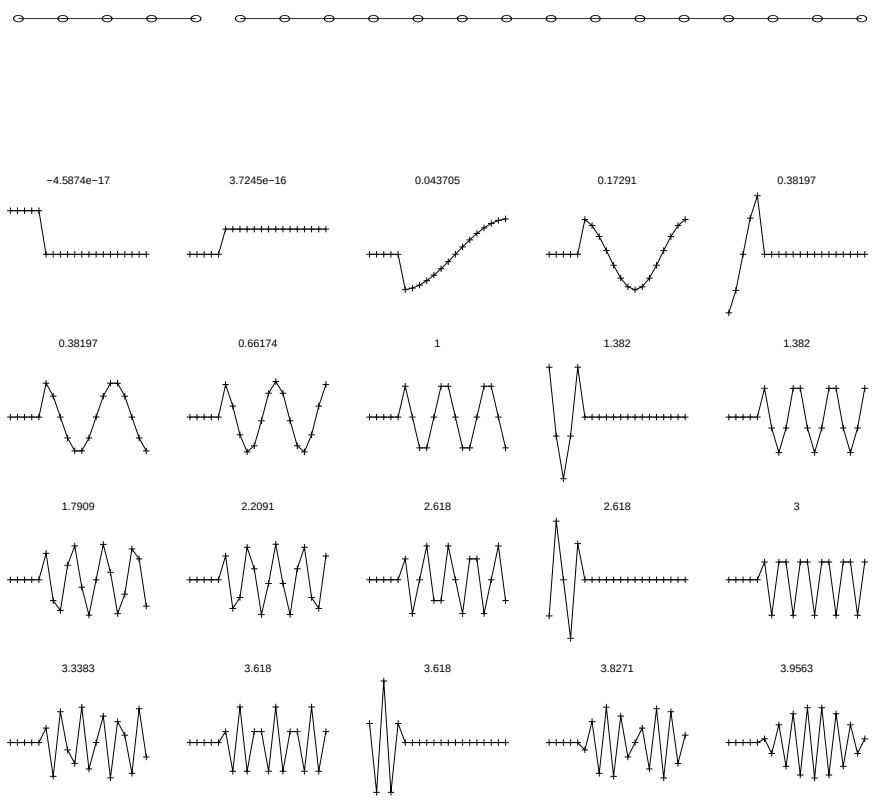


Figure 8.1: A simple graph with two segments, and its Laplacian spectral decomposition. The numbers are the eigenvalues, and the zigzag shapes are the corresponding eigenvectors.

corresponding eigenvectors become less and less smooth.

8.2 From Laplacians to Kernels

Kernel-based methods are increasingly being used for data modeling and prediction because of their conceptual simplicity and good performance on many tasks. A promising family of semi-supervised learning methods can be viewed as constructing kernels by transforming the spectrum (i.e. eigen-decomposition) of the graph Laplacian. These kernels, when viewed as regularizers, penalize functions that are not smooth over the graph (Smola & Kondor, 2003).

Assuming the graph structure is correct, from a regularization perspective we

want to encourage smooth functions, to reflect our belief that labels should vary slowly over the graph. Specifically, Chapelle et al. (2002) and Smola and Kondor (2003) suggest a general principle for creating a family of semi-supervised kernels K from the graph Laplacian Δ: transform the eigenvalues λ into $r(\lambda)$, where the *spectral transformation* r is a non-negative and usually decreasing function[1]

$$K = \sum_{i=1}^{n} r(\lambda_i)\, \phi_i \phi_i^\top \tag{8.1}$$

Note it may be that r reverses the order of the eigenvalues, so that smooth ϕ_i's have *larger* eigenvalues in K. With such a kernel, a "soft labeling" function $f = \sum c_i \phi_i$ in a kernel machine has a penalty term in the RKHS norm given by $\Omega(||f||_K^2) = \Omega(\sum c_i^2 / r(\lambda_i))$. If r is decreasing, a greater penalty is incurred for those terms of f corresponding to eigenfunctions that are less smooth.

In previous work r has often been chosen from a parametric family. For example, the diffusion kernel (Kondor & Lafferty, 2002) corresponds to

$$r(\lambda) = \exp(-\frac{\sigma^2}{2}\lambda) \tag{8.2}$$

The regularized Gaussian process kernel in Chapter 6 corresponds to

$$r(\lambda) = \frac{1}{\lambda + \sigma} \tag{8.3}$$

Figure 8.2 shows such a regularized Gaussian process kernel, constructed from the Laplacian in Figure 8.1 with $\sigma = 0.05$. Cross validation has been used to find the hyperparameter σ for these spectral transformations. Although the general principle of equation (8.1) is appealing, it does not address the question of which *parametric family* to use for r. Moreover, the degree of freedom (or the number of hyperparameters) may not suit the task, resulting in overly constrained kernels.

We address these limitations with a nonparametric method. Instead of using a parametric transformation $r(\lambda)$, we allow the transformed eigenvalues $\mu_i = r(\lambda_i), i = 1 \ldots n$ to be almost independent. The only additional condition is that μ_i's have to be non-increasing, to encourage smooth functions over the graph. Under this condition, we find the set of optimal spectral transformation μ that maximizes the kernel alignment to the labeled data. The main advantage of using kernel alignment is that it gives us a convex optimization problem, and does not suffer from poor convergence to local minima. The optimization problem in general is solved using semi-definite programming (SDP) (Boyd & Vandenberge, 2004);

[1] We use a slightly different notation where r is the inverse of that in (Smola & Kondor, 2003).

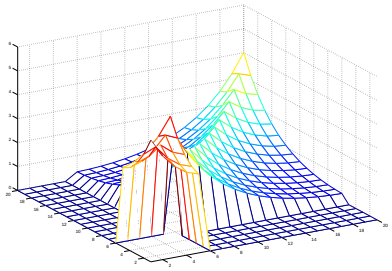


Figure 8.2: The kernel constructed from the Laplacian in Figure 8.1, with spectrum transformation $r(\lambda) = 1/(\lambda + 0.05)$.

however, in our approach the problem can be formulated in terms of quadratically constrained quadratic programming (QCQP), which can be solved more efficiently than a general SDP. We review QCQP next.

8.3 Convex Optimization using QCQP

Let $K_i = \phi_i \phi_i^\top, i = 1 \cdots n$ be the outer product matrices of the Laplacian's eigenvectors. Our kernel K is a linear combination

$$K = \sum_{i=1}^{n} \mu_i K_i \tag{8.4}$$

where $\mu_i \geq 0$. We formulate the problem of finding the optimal spectral transformation as one that finds the interpolation coefficients $\{r(\lambda_i) = \mu_i\}$ by optimizing some convex objective function on K. To maintain the positive semi-definiteness constraint on K, one in general needs to invoke SDPs (Boyd & Vandenberge, 2004). Semi-definite optimization can be described as the problem of optimizing a linear function of a symmetric matrix subject to linear equality constraints and the condition that the matrix be positive semi-definite. The well known linear programming problem can be generalized to a semi-definite optimization by replacing the vector of variables with a symmetric matrix, and replacing the non-negativity constraints with a positive semi-definite constraints. This generalization inherits several properties: it is convex, has a rich duality theory and allows theoretically efficient solution algorithms based on iterating interior point methods to either follow a central path or decrease a potential function. However, a limitation of SDPs is their computational complexity (Boyd & Vandenberge, 2004), which has restricted their application to small-scale problems (Lanckriet et al., 2004). However, an important special case of SDPs are *quadratically constrained quadratic programs*

(QCQP) which are computationally more efficient. Here both the objective function and the constraints are quadratic as illustrated below,

$$\text{minimize} \qquad \frac{1}{2}x^\top P_0 x + q_0^\top x + r_0 \tag{8.5}$$

$$\text{subject to} \qquad \frac{1}{2}x^\top P_i x + q_i^\top x + r_i \leq 0 \quad i = 1 \cdots m \tag{8.6}$$

$$Ax = b \tag{8.7}$$

where $P_i \in \mathcal{S}_+^n,\ i = 1, \ldots, m$, where $\mathcal{S}_+^n$ defines the set of square symmetric positive semi-definite matrices. In a QCQP, we minimize a convex quadratic function over a feasible region that is the intersection of ellipsoids. The number of iterations required to reach the solution is comparable to the number required for linear programs, making the approach feasible for large datasets. However, as observed in (Boyd & Vandenberge, 2004), not all SDPs can be relaxed to QCQPs. For the semi-supervised kernel learning task presented here solving an SDP would be computationally infeasible.

Recent work (Cristianini et al., 2001a; Lanckriet et al., 2004) has proposed *kernel target alignment* that can be used not only to assess the relationship between the feature spaces generated by two different kernels, but also to assess the similarity between spaces induced by a kernel and that induced by the labels themselves. Desirable properties of the alignment measure can be found in (Cristianini et al., 2001a). The crucial aspect of alignment for our purposes is that its optimization can be formulated as a QCQP. The objective function is the empirical kernel alignment score:

$$\hat{A}(K_{tr}, T) = \frac{\langle K_{tr}, T \rangle_F}{\sqrt{\langle K_{tr}, K_{tr} \rangle_F \langle T, T \rangle_F}} \tag{8.8}$$

where K_{tr} is the kernel matrix restricted to the training points, $\langle M, N \rangle_F$ denotes the Frobenius product between two square matrices $\langle M, N \rangle_F = \sum_{ij} m_{ij} n_{ij} = \text{trace}(MN^\top)$, and T is the target matrix on training data, with entry T_{ij} set to $+1$ if $y_i = y_j$ and -1 otherwise. Note for binary $\{+1, -1\}$ training labels Y_L this is simply the rank one matrix $T = Y_L Y_L^\top$. K is guaranteed to be positive semi-definite by constraining $\mu_i \geq 0$. Our kernel alignment problem is special in that the K_i's were derived from the graph Laplacian with the goal of semi-supervised learning. We require smoother eigenvectors to receive larger coefficients, as shown in the next section.

8.4 Semi-Supervised Kernels with Order Constraints

As stated above, we would like to maintain a decreasing order on the spectral transformation $\mu_i = r(\lambda_i)$ to encourage smooth functions over the graph. This

motivates the set of *order constraints*

$$\mu_i \geq \mu_{i+1}, \quad i = 1 \cdots n-1 \tag{8.9}$$

We can specify the desired semi-supervised kernel as follows.

Definition 1 *An order constrained semi-supervised kernel K is the solution to the following convex optimization problem:*

$$\begin{aligned} \textit{max}_K \quad & \hat{A}(K_{tr}, T) && (8.10)\\ \textit{subject to} \quad & K = \textstyle\sum_{i=1}^{n} \mu_i K_i && (8.11)\\ & \mu_i \geq 0 && (8.12)\\ & trace(K) = 1 && (8.13)\\ & \mu_i \geq \mu_{i+1}, \quad i = 1 \cdots n-1 && (8.14) \end{aligned}$$

where T is the training target matrix, $K_i = \phi_i \phi_i^\top$ and ϕ_i's are the eigenvectors of the graph Laplacian.

The formulation is an extension to (Lanckriet et al., 2004) with order constraints, and with special components K_i's from the graph Laplacian. Since $\mu_i \geq 0$ and K_i's are outer products, K will automatically be positive semi-definite and hence a valid kernel matrix. The trace constraint is needed to fix the scale invariance of kernel alignment. It is important to notice the order constraints are convex, and as such the whole problem is convex. This problem is equivalent to:

$$\begin{aligned} \text{max}_K \quad & \langle K_{tr}, T \rangle_F && (8.15)\\ \text{subject to} \quad & \langle K_{tr}, K_{tr} \rangle_F \leq 1 && (8.16)\\ & K = \textstyle\sum_{i=1}^{n} \mu_i K_i && (8.17)\\ & \mu_i \geq 0 && (8.18)\\ & \mu_i \geq \mu_{i+1}, \ \forall i && (8.19) \end{aligned}$$

Let $vec(A)$ be the column vectorization of a matrix A. Defining a $l^2 \times m$ matrix

$$M = \left[vec(K_{1,tr}) \cdots vec(K_{m,tr}) \right] \tag{8.20}$$

it is not hard to show that the problem can then be expressed as

$$\begin{aligned} \text{max}_\mu \quad & vec(T)^\top M \mu && (8.21)\\ \text{subject to} \quad & ||M\mu|| \leq 1 && (8.22)\\ & \mu_i \geq 0 && (8.23)\\ & \mu_i \geq \mu_{i+1}, \quad i = 1 \cdots n-1 && (8.24) \end{aligned}$$

The objective function is linear in μ, and there is a simple cone constraint, making it a quadratically constrained quadratic program (QCQP) [2].

An improvement of the above order constrained semi-supervised kernel can be obtained by taking a closer look at the Laplacian eigenvectors with zero eigenvalues. As stated earlier, for a graph Laplacian there will be k zero eigenvalues if the graph has k connected subgraphs. The k eigenvectors are piecewise constant over individual subgraphs, and zero elsewhere. This is desirable when $k > 1$, with the hope that subgraphs correspond to different classes. However if $k = 1$, the graph is connected. The first eigenvector ϕ_1 is a constant vector over all nodes. The corresponding K_1 is a constant matrix, and acts as a bias term in (8.1). In this situation we do not want to impose the order constraint $\mu_1 \geq \mu_2$ on the constant bias term, rather we let μ_1 vary freely during optimization:

Definition 2 *An improved order constrained semi-supervised kernel K is the solution to the same problem in Definition 1, but the order constraints (8.14) apply only to non-constant eigenvectors:*

$$\mu_i \geq \mu_{i+1}, \quad i = 1 \cdots n-1, \text{ and } \phi_i \text{ not constant} \tag{8.25}$$

In practice we do not need all n eigenvectors of the graph Laplacian, or equivalently all n K_i's. The first $m < n$ eigenvectors with the smallest eigenvalues work well empirically. Also note we could have used the fact that K_i's are from orthogonal eigenvectors ϕ_i to further simplify the expression. However we neglect this observation, making it easier to incorporate other kernel components if necessary.

It is illustrative to compare and contrast the order constrained semi-supervised kernels to other semi-supervised kernels with different spectral transformation. We call the original kernel alignment solution in (Lanckriet et al., 2004) a *maximal-alignment* kernel. It is the solution to Definition 1 without the order constraints (8.14). Because it does not have the additional constraints, it maximizes kernel alignment among all spectral transformation. The hyperparameters σ of the Diffusion kernel and Gaussian fields kernel (described earlier) can be learned by maximizing the alignment score too, although the optimization problem is not necessarily convex. These kernels use different information in the original Laplacian eigenvalues λ_i. The maximal-alignment kernels ignore λ_i altogether. The order constrained semi-supervised kernels only use the *order* of λ_i and ignore their actual values. The diffusion and Gaussian field kernels use the actual values. In terms of the degree of freedom in choosing the spectral transformation μ_i's, the maximal-alignment kernels are completely free. The diffusion and Gaussian field

[2] An alternative formulation results in a quadratic program (QP), which is faster than QCQP. Details can be found at `http://www.cs.cmu.edu/~zhuxj/pub/QP.pdf`

kernels are restrictive since they have an implicit parametric form and only one free parameter. The order constrained semi-supervised kernels incorporates desirable features from both approaches.

8.5 Experiments

We evaluate the order constrained kernels on seven datasets. **baseball-hockey** (1993 instances / 2 classes), **pc-mac** (1943/2) and **religion-atheism** (1427/2) are document categorization tasks taken from the 20-newsgroups dataset. The distance measure is the standard cosine similarity between tf.idf vectors. **one-two** (2200/2), **odd-even** (4000/2) and **ten digits** (4000/10) are handwritten digits recognition tasks. **one-two** is digits "1" vs. "2"; **odd-even** is the artificial task of classifying odd "1, 3, 5, 7, 9" vs. even "0, 2, 4, 6, 8" digits, such that each class has several well defined internal clusters; **ten digits** is 10-way classification. **isolet** (7797/26) is isolated spoken English alphabet recognition from the UCI repository. For these datasets we use Euclidean distance on raw features. We use 10NN unweighted graphs on all datasets except isolet which is 100NN. For all datasets, we use the smallest $m = 200$ eigenvalue and eigenvector pairs from the graph Laplacian. These values are set arbitrarily without optimizing and do not create a unfair advantage to the proposed kernels. For each dataset we test on five different labeled set sizes. For a given labeled set size, we perform 30 random trials in which a labeled set is randomly sampled from the whole dataset. All classes must be present in the labeled set. The rest is used as unlabeled (test) set in that trial. We compare 5 semi-supervised kernels (improved order constrained kernel, order constrained kernel, Gaussian field kernel, diffusion kernel[3] and maximal-alignment kernel), and 3 standard supervised kernels (RBF (bandwidth learned using 5-fold cross validation),linear and quadratic). We compute the spectral transformation for order constrained kernels and maximal-alignment kernels by solving the QCQP using standard solvers (SeDuMi/YALMIP). To compute accuracy we use these kernels in a standard SVM. We choose the bound on slack variables C with cross validation for all tasks and kernels. For multiclass classification we perform one-against-all and pick the class with the largest margin.

Table 8.1 through Table 8.7 list the results. There are two rows for each cell: The upper row is the average *test set accuracy* with one standard deviation; The lower row is the average *training set kernel alignment*, and in parenthesis the average *run time in seconds* for QCQP on a 2.4GHz Linux computer. Each number is averaged over 30 random trials. To assess the statistical significance of the re-

[3]The hyperparameters σ are learned with the **fminbnd()** function in Matlab to maximize kernel alignment.

	semi-supervised kernels					standard kernels		
Training set size	Improved Order	Order	Gaussian Field	Diffusion	Max-align	RBF $\sigma = 200$	Linear	Quadratic
10	**95.7** $\pm$ 8.9	**93.9** $\pm$12.0	63.1 $\pm$15.8	65.8 $\pm$22.8	**93.2** $\pm$ 6.8	53.6 $\pm$ 5.5	68.1 $\pm$ 7.6	68.1 $\pm$ 7.6
	0.90 (2)	0.69 (1)	0.35	0.44	0.95 (1)	0.11	0.29	0.23
30	**98.0** $\pm$ 0.2	**97.3** $\pm$ 2.1	91.8 $\pm$ 9.3	59.1 $\pm$17.9	96.6 $\pm$ 2.2	69.3 $\pm$11.2	78.5 $\pm$ 8.5	77.8 $\pm$10.6
	0.91 (9)	0.67 (9)	0.25	0.39	0.93 (6)	0.03	0.17	0.11
50	**97.9** $\pm$ 0.5	**97.8** $\pm$ 0.6	96.7 $\pm$ 0.6	93.7 $\pm$ 6.8	97.0 $\pm$ 1.1	77.7 $\pm$ 8.3	84.1 $\pm$ 7.8	75.6 $\pm$14.2
	0.89 (29)	0.63 (29)	0.22	0.36	0.90 (27)	0.02	0.15	0.09
70	**97.9** $\pm$ 0.3	**97.9** $\pm$ 0.3	96.8 $\pm$ 0.6	**97.5** $\pm$ 1.4	97.2 $\pm$ 0.8	83.9 $\pm$ 7.2	87.5 $\pm$ 6.5	76.1 $\pm$14.9
	0.90 (68)	0.64 (64)	0.22	0.37	0.90 (46)	0.01	0.13	0.07
90	**98.0** $\pm$ 0.5	**98.0** $\pm$ 0.2	97.0 $\pm$ 0.4	97.8 $\pm$ 0.2	97.6 $\pm$ 0.3	88.5 $\pm$ 5.1	89.3 $\pm$ 4.4	73.3 $\pm$16.8
	0.89 (103)	0.63 (101)	0.21	0.36	0.89 (90)	0.01	0.12	0.06

Table 8.1: Baseball vs. Hockey

	semi-supervised kernels					standard kernels		
Training set size	Improved Order	Order	Gaussian Field	Diffusion	Max-align	RBF $\sigma = 100$	Linear	Quadratic
10	**87.0** $\pm$ 5.0	**84.9** $\pm$ 7.2	56.4 $\pm$ 6.2	57.8 $\pm$11.5	71.1 $\pm$ 9.7	51.6 $\pm$ 3.4	63.0 $\pm$ 5.1	62.3 $\pm$ 4.2
	0.71 (1)	0.57 (1)	0.32	0.35	0.90 (1)	0.11	0.30	0.25
30	**90.3** $\pm$ 1.3	**89.6** $\pm$ 2.3	76.4 $\pm$ 6.1	79.6 $\pm$11.2	85.4 $\pm$ 3.9	62.6 $\pm$ 9.6	71.8 $\pm$ 5.5	71.2 $\pm$ 5.3
	0.68 (8)	0.49 (8)	0.19	0.23	0.74 (6)	0.03	0.18	0.13
50	**91.3** $\pm$ 0.9	90.5 $\pm$ 1.7	81.1 $\pm$ 4.6	87.5 $\pm$ 2.8	88.4 $\pm$ 2.1	67.8 $\pm$ 9.0	77.6 $\pm$ 4.8	75.7 $\pm$ 5.4
	0.64 (31)	0.46 (31)	0.16	0.20	0.68 (25)	0.02	0.14	0.10
70	**91.5** $\pm$ 0.6	90.8 $\pm$ 1.3	84.6 $\pm$ 2.1	90.5 $\pm$ 1.2	89.6 $\pm$ 1.6	74.7 $\pm$ 7.4	80.2 $\pm$ 4.6	74.3 $\pm$ 8.7
	0.63 (70)	0.46 (56)	0.14	0.19	0.66 (59)	0.01	0.12	0.08
90	**91.5** $\pm$ 0.6	**91.3** $\pm$ 1.3	86.3 $\pm$ 2.3	**91.3** $\pm$ 1.1	90.3 $\pm$ 1.0	79.0 $\pm$ 6.4	82.5 $\pm$ 4.2	79.1 $\pm$ 7.3
	0.63 (108)	0.45 (98)	0.13	0.18	0.65 (84)	0.01	0.11	0.08

Table 8.2: PC vs. MAC

	semi-supervised kernels					standard kernels		
Training set size	Improved Order	Order	Gaussian Field	Diffusion	Max-align	RBF $\sigma = 130$	Linear	Quadratic
10	**72.8** $\pm$11.2	70.9 $\pm$10.9	55.2 $\pm$ 5.8	60.9 $\pm$10.7	60.7 $\pm$ 7.5	55.8 $\pm$ 5.8	60.1 $\pm$ 7.0	61.2 $\pm$ 4.8
	0.50 (1)	0.42 (1)	0.31	0.31	0.85 (1)	0.13	0.30	0.26
30	**84.2** $\pm$ 2.4	83.0 $\pm$ 2.9	71.2 $\pm$ 6.3	80.3 $\pm$ 5.1	74.4 $\pm$ 5.4	63.4 $\pm$ 6.5	63.7 $\pm$ 8.3	70.1 $\pm$ 6.3
	0.38 (8)	0.31 (6)	0.20	0.22	0.60 (7)	0.05	0.18	0.15
50	**84.5** $\pm$ 2.3	83.5 $\pm$ 2.5	80.4 $\pm$ 4.1	83.5 $\pm$ 2.7	77.4 $\pm$ 6.1	69.3 $\pm$ 6.5	69.4 $\pm$ 7.0	70.7 $\pm$ 8.5
	0.31 (28)	0.26 (23)	0.17	0.20	0.48 (27)	0.04	0.15	0.11
70	**85.7** $\pm$ 1.4	85.3 $\pm$ 1.6	83.0 $\pm$ 2.9	**85.4** $\pm$ 1.8	82.3 $\pm$ 3.0	73.1 $\pm$ 5.8	75.7 $\pm$ 6.0	71.0 $\pm$10.0
	0.29 (55)	0.25 (42)	0.16	0.19	0.43 (51)	0.03	0.13	0.10
90	**86.6** $\pm$ 1.3	**86.4** $\pm$ 1.5	84.5 $\pm$ 2.1	**86.2** $\pm$ 1.6	82.8 $\pm$ 2.6	77.7 $\pm$ 5.1	74.6 $\pm$ 7.6	70.0 $\pm$11.5
	0.27 (86)	0.24 (92)	0.15	0.18	0.40 (85)	0.02	0.12	0.09

Table 8.3: Religion vs. Atheism

	semi-supervised kernels					standard kernels		
Training set size	Improved Order	Order	Gaussian Field	Diffusion	Max-align	RBF $\sigma = 1000$	Linear	Quadratic
10	**96.2** $\pm$ 2.7	90.6 $\pm$14.0	58.2 $\pm$17.6	59.4 $\pm$18.9	85.4 $\pm$11.5	78.7 $\pm$14.3	85.1 $\pm$ 5.7	85.7 $\pm$ 4.8
	0.87 (2)	0.66 (1)	0.43	0.53	0.95 (1)	0.38	0.26	0.30
20	**96.4** $\pm$ 2.8	**93.9** $\pm$ 8.7	87.0 $\pm$16.0	83.2 $\pm$19.8	94.5 $\pm$ 1.6	90.4 $\pm$ 4.6	86.0 $\pm$ 9.4	90.9 $\pm$ 3.7
	0.87 (3)	0.64 (4)	0.38	0.50	0.90 (3)	0.33	0.22	0.25
30	**98.2** $\pm$ 2.1	97.2 $\pm$ 2.5	**98.1** $\pm$ 2.2	**98.1** $\pm$ 2.7	96.4 $\pm$ 2.1	93.6 $\pm$ 3.1	89.6 $\pm$ 5.9	92.9 $\pm$ 2.8
	0.84 (8)	0.61 (7)	0.35	0.47	0.86 (6)	0.30	0.17	0.24
40	98.3 $\pm$ 1.9	96.5 $\pm$ 2.4	98.9 $\pm$ 1.8	**99.1** $\pm$ 1.4	96.3 $\pm$ 2.3	94.0 $\pm$ 2.7	91.6 $\pm$ 6.3	94.9 $\pm$ 2.0
	0.84 (13)	0.61 (15)	0.36	0.48	0.86 (11)	0.29	0.18	0.21
50	98.4 $\pm$ 1.9	95.6 $\pm$ 9.0	99.4 $\pm$ 0.5	**99.6** $\pm$ 0.3	96.6 $\pm$ 2.3	96.1 $\pm$ 2.4	93.0 $\pm$ 3.6	95.8 $\pm$ 2.3
	0.83 (31)	0.60 (37)	0.35	0.46	0.84 (25)	0.28	0.17	0.20

Table 8.4: One vs. Two

	semi-supervised kernels					standard kernels		
Training set size	Improved Order	Order	Gaussian Field	Diffusion	Max-align	RBF $\sigma = 1500$	Linear	Quadratic
10	**69.6** $\pm$ 6.5	**68.8** $\pm$ 6.1	65.5 $\pm$ 8.9	**68.4** $\pm$ 8.5	55.7 $\pm$ 4.4	65.0 $\pm$ 7.0	63.1 $\pm$ 6.9	65.4 $\pm$ 6.5
	0.45 (1)	0.41 (1)	0.32	0.34	0.86 (1)	0.23	0.25	0.27
30	**82.4** $\pm$ 4.1	**82.0** $\pm$ 4.0	79.6 $\pm$ 4.1	**83.0** $\pm$ 4.2	67.2 $\pm$ 5.0	77.7 $\pm$ 3.5	72.4 $\pm$ 6.1	76.5 $\pm$ 5.1
	0.32 (6)	0.28 (6)	0.21	0.23	0.56 (6)	0.10	0.11	0.16
50	87.6 $\pm$ 3.5	87.5 $\pm$ 3.4	85.9 $\pm$ 3.8	**89.1** $\pm$ 2.7	76.0 $\pm$ 5.3	81.8 $\pm$ 2.7	74.4 $\pm$ 9.2	81.3 $\pm$ 3.1
	0.29 (24)	0.26 (25)	0.19	0.21	0.45 (26)	0.07	0.09	0.12
70	89.2 $\pm$ 2.6	89.0 $\pm$ 2.7	89.0 $\pm$ 1.9	**90.3** $\pm$ 2.8	80.9 $\pm$ 4.4	84.4 $\pm$ 2.0	73.6 $\pm$10.0	83.8 $\pm$ 2.8
	0.27 (65)	0.24 (50)	0.17	0.20	0.39 (51)	0.06	0.07	0.12
90	**91.5** $\pm$ 1.5	**91.4** $\pm$ 1.6	90.5 $\pm$ 1.4	**91.9** $\pm$ 1.7	85.4 $\pm$ 3.1	86.1 $\pm$ 1.8	66.1 $\pm$14.8	85.5 $\pm$ 1.6
	0.26 (94)	0.23 (97)	0.16	0.19	0.36 (88)	0.05	0.07	0.11

Table 8.5: Odd vs. Even

	semi-supervised kernels					standard kernels		
Training set size	Improved Order	Order	Gaussian Field	Diffusion	Max-align	RBF $\sigma = 2000$	Linear	Quadratic
50	**76.6** $\pm$ 4.3	71.5 $\pm$ 5.0	41.4 $\pm$ 6.8	49.8 $\pm$ 6.3	70.3 $\pm$ 5.2	57.0 $\pm$ 4.0	50.2 $\pm$ 9.0	66.3 $\pm$ 3.7
	0.47 (26)	0.21 (26)	0.15	0.16	0.51 (25)	-0.62	-0.50	-0.25
100	**84.8** $\pm$ 2.6	83.4 $\pm$ 2.6	63.7 $\pm$ 3.5	72.5 $\pm$ 3.3	80.7 $\pm$ 2.6	69.4 $\pm$ 1.9	56.0 $\pm$ 7.8	77.2 $\pm$ 2.3
	0.47 (124)	0.17 (98)	0.12	0.13	0.49 (100)	-0.64	-0.52	-0.29
150	**86.5** $\pm$ 1.7	**86.4** $\pm$ 1.3	75.1 $\pm$ 3.0	80.4 $\pm$ 2.1	84.5 $\pm$ 1.9	75.2 $\pm$ 1.4	56.2 $\pm$ 7.2	81.4 $\pm$ 2.2
	0.48 (310)	0.18 (255)	0.11	0.13	0.50 (244)	-0.66	-0.53	-0.31
200	**88.1** $\pm$ 1.3	**88.0** $\pm$ 1.3	80.4 $\pm$ 2.5	84.4 $\pm$ 1.6	86.0 $\pm$ 1.5	78.3 $\pm$ 1.3	60.8 $\pm$ 7.3	84.3 $\pm$ 1.7
	0.47 (708)	0.16 (477)	0.10	0.11	0.49 (523)	-0.65	-0.54	-0.33
250	**89.1** $\pm$ 1.1	**89.3** $\pm$ 1.0	84.6 $\pm$ 1.4	87.2 $\pm$ 1.3	87.2 $\pm$ 1.3	80.4 $\pm$ 1.4	61.3 $\pm$ 7.6	85.7 $\pm$ 1.3
	0.47 (942)	0.16 (873)	0.10	0.11	0.49 (706)	-0.65	-0.54	-0.33

Table 8.6: Ten Digits (10 classes)

	semi-supervised kernels					standard kernels		
Training set size	Improved Order	Order	Gaussian Field	Diffusion	Max-align	RBF $\sigma = 30$	Linear	Quadratic
50	**56.0** $\pm$ 3.5	42.0 $\pm$ 5.2	41.2 $\pm$ 2.9	29.0 $\pm$ 2.7	50.1 $\pm$ 3.7	28.7 $\pm$ 2.0	30.0 $\pm$ 2.7	23.7 $\pm$ 2.4
	0.27 (26)	0.13 (25)	0.03	0.11	0.31 (24)	-0.89	-0.80	-0.65
100	**64.6** $\pm$ 2.1	59.0 $\pm$ 3.6	58.5 $\pm$ 2.9	47.4 $\pm$ 2.7	63.2 $\pm$ 1.9	46.3 $\pm$ 2.4	46.6 $\pm$ 2.7	42.0 $\pm$ 2.9
	0.26 (105)	0.10 (127)	-0.02	0.08	0.29 (102)	-0.90	-0.82	-0.69
150	**67.6** $\pm$ 2.6	65.2 $\pm$ 3.0	65.4 $\pm$ 2.6	57.2 $\pm$ 2.7	**67.9** $\pm$ 2.5	57.6 $\pm$ 1.5	57.3 $\pm$ 1.8	53.8 $\pm$ 2.2
	0.26 (249)	0.09 (280)	-0.05	0.07	0.27 (221)	-0.90	-0.83	-0.70
200	71.0 $\pm$ 1.8	70.9 $\pm$ 2.3	70.6 $\pm$ 1.9	64.8 $\pm$ 2.1	**72.3** $\pm$ 1.7	63.9 $\pm$ 1.6	64.2 $\pm$ 2.0	60.5 $\pm$ 1.6
	0.26 (441)	0.08 (570)	-0.07	0.06	0.27 (423)	-0.91	-0.83	-0.72
250	71.8 $\pm$ 2.3	73.6 $\pm$ 1.5	73.7 $\pm$ 1.2	69.8 $\pm$ 1.5	**74.2** $\pm$ 1.5	68.8 $\pm$ 1.5	69.5 $\pm$ 1.7	66.2 $\pm$ 1.4
	0.26 (709)	0.08 (836)	-0.07	0.06	0.27 (665)	-0.91	-0.84	-0.72

Table 8.7: ISOLET (26 classes)

sults, we perform paired t-test on test accuracy. We highlight the best accuracy in each row, and those that cannot be determined as different from the best, with paired t-test at significance level 0.05. The semi-supervised kernels tend to outperform standard supervised kernels. The improved order constrained kernels are consistently among the best. Figure 8.3 shows the spectral transformation μ_i of the semi-supervised kernels for different tasks. These are for the 30 trials with the largest labeled set size in each task. The x-axis is in increasing order of λ_i (the original eigenvalues of the Laplacian). The mean (thick lines) and ± 1 standard deviation (dotted lines) of only the top 50 μ_i's are plotted for clarity. The μ_i values are scaled vertically for easy comparison among kernels. As expected the maximal-alignment kernels' spectral transformation is zigzagged, diffusion and Gaussian field's are very smooth, while order constrained kernels' are in between. The order constrained kernels (green) have large μ_1 because of the order constraint. This seems to be disadvantageous — the spectral transformation tries to balance it out by increasing the value of other μ_i's so that the constant K_1's relative influence is smaller. On the other hand the improved order constrained kernels (black) allow μ_1 to be small. As a result the rest μ_i's decay fast, which is desirable.

In conclusion, the method is both computationally feasible and results in improvements to classification performance when used with support vector machines.

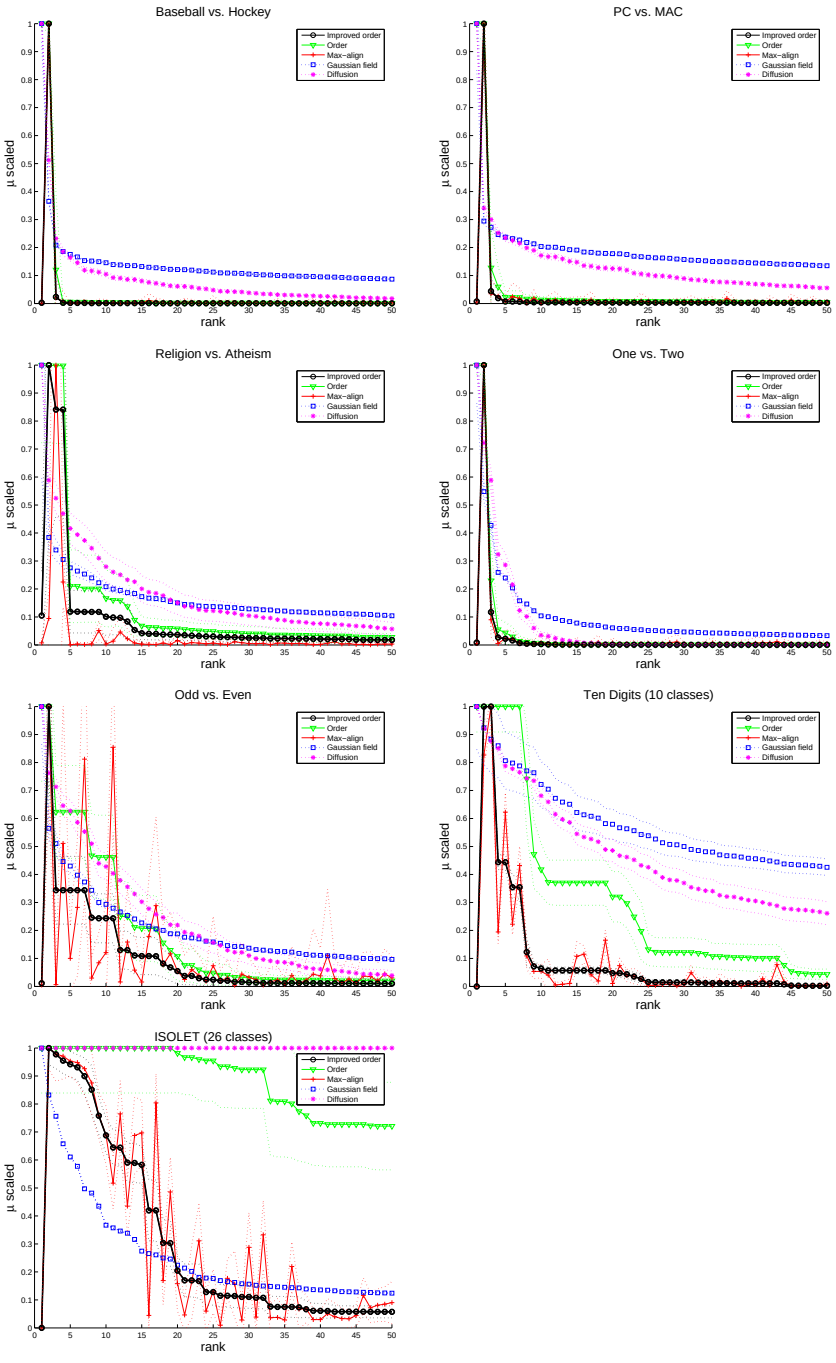


Figure 8.3: Spectral transformation of the 5 semi-supervised kernels.

Chapter 9

Sequences and Beyond

So far, we have treated each data point individually. However in many problems the data has complex structures. For example in speech recognition the data is sequential. Most semi-supervised learning methods have not addressed this problem. We use sequential data as an example in the following discussion because it is simple. Nevertheless the discussion applies to other complex data structures like grids, trees etc.

It is important to clarify the setting. By sequential data we do not mean each data item x is a sequence and we give a *single label* y to the whole sequence. Instead we want to give individual labels to the constituent data points in the sequence.

There are generative and discriminative methods that can be used for semi-supervised learning on sequences.

The Hidden Markov Model (HMM) is such a generative methods. Specifically the standard EM training with forward-backward algorithm (also known as Baum-Welch (Rabiner, 1989)) is a sequence semi-supervised learning algorithm, although it is usually not presented that way. The training data typically consists of a small labeled set with l labeled sequences $\{X_L, Y_L\} = \{(\mathbf{x}_1, \mathbf{y}_1) \dots (\mathbf{x}_l, \mathbf{y}_l)\}$, and a much larger unlabeled set of sequences $X_U = \{\mathbf{x}_{l+1} \dots \mathbf{x}_{l+u}\}$. We use bold font $\mathbf{x}_i$ to represent the i-th sequence with length m_i, whose elements are $x_{i1} \dots x_{im_i}$. Similarly $\mathbf{y}_i$ is a sequence of labels $y_{i1} \dots y_{im_i}$. The labeled set is used to estimate initial HMM parameters. The unlabeled data is then used to run the EM algorithm on, to improve the HMM likelihood $P(X_U)$ to a local maximum. The trained HMM parameters thus are determined by both the labeled and unlabeled sequences. This parallels the mixture models and EM algorithm in the *i.i.d.* case. We will not discuss it further in the thesis.

For discriminative methods one strategy is to use a kernel machine for se-

quences, and introduce semi-supervised dependency via the kernels in Chapter 8. Recent kernel machines for sequences and other complex structures include Kernel Conditional Random Fields (KCRFs) (Lafferty et al., 2004) and Max-Margin Markov Networks (Taskar et al., 2003), which are generalization of logistic regression and support vector machines respectively to structured data. These kernel machines by themselves are not designed specifically for semi-supervised learning. However we can use a semi-supervised kernel, for example the graph kernels in Chapter 8, with the kernel machines. This results in semi-supervised learning methods on sequential data.

The idea is straightforward. The remainder of the chapter focuses on KCRFs, describing the formalism and training issues, with a synthetic example on semi-supervised learning.

9.1 Cliques and Two Graphs

Before we start, it is useful to distinguish two kinds of graphs in KCRF for semi-supervised learning. The first graph ($\mathbf{g}_s$) represents the conditional random field *structure*, for example a linear chain graph for sequences. In this case the size of $\mathbf{g}_s$ is the length of the sequence. In general let $\mathbf{x}$ be the features on $\mathbf{g}_s$'s nodes and $\mathbf{y}$ the labels. A *clique* c is a subset of the nodes which is fully connected, with any pair of nodes joined by an edge. Let $\mathbf{y}_c$ be the labels on the clique. We want Mercer kernels K to compare cliques in different graphs,

$$K((\mathbf{g}_s, \mathbf{x}, c, \mathbf{y}_c), (\mathbf{g}'_s, \mathbf{x}', c', \mathbf{y}'_{c'})) \in \mathbb{R} \tag{9.1}$$

Intuitively, this assigns a measure of similarity between a labeled clique in one graph and a labeled clique in a (possibly) different graph. We denote by $\mathcal{H}_K$ the associated reproducing kernel Hilbert space, and by $\|\cdot\|_K$ the associated norm.

In the context of semi-supervised learning, we are interested in kernels with the special form:

$$K((\mathbf{g}_s, \mathbf{x}, c, \mathbf{y}_c), (\mathbf{g}'_s, \mathbf{x}', c', \mathbf{y}'_{c'})) = \psi\left(K'(\mathbf{x}_c, \mathbf{x}'_c), \mathbf{g}_s, \mathbf{y}_c, \mathbf{g}'_s, \mathbf{y}'_{c'}\right) \tag{9.2}$$

i.e. some function ψ of a kernel K', where K' depends only on the features, not the labels. This is where the second graph (denoted $\mathbf{g}_k$) comes in. $\mathbf{g}_k$ is the semi-supervised graph discussed in previous chapters. Its nodes are the cliques $\mathbf{x}_c$ in both labeled and unlabeled data, and edges represent similarity between the cliques. The size of $\mathbf{g}_k$ is the total number of cliques in the whole dataset. It however does not represent the sequence structure. $\mathbf{g}_k$ is used to derive the Laplacian and ultimately the kernel matrix $K'(\mathbf{x}_c, \mathbf{x}'_c)$, as in Chapter 8.

9.2 Representer Theorem for KCRFs

We start from a function f which, looking at a clique (c) in graph ($\mathbf{g}_s, \mathbf{x}$) and an arbitrary labeling of the clique ($\mathbf{y}_c$), computes a 'compatibility' score. That is, $f(\mathbf{g}_s, \mathbf{x}, c, \mathbf{y}_c) \rightarrow \mathbb{R}$. We define a conditional random field

$$p(\mathbf{y}|\mathbf{g}_s, \mathbf{x}) = Z^{-1}(\mathbf{g}_s, \mathbf{x}, f) \exp\left(\sum_c f(\mathbf{g}_s, \mathbf{x}, c, \mathbf{y}_c)\right) \tag{9.3}$$

The normalization factor is

$$Z(\mathbf{g}_s, \mathbf{x}, f) = \sum_{\mathbf{y}'} \exp\left(\sum_c f(\mathbf{g}_s, \mathbf{x}, c, \mathbf{y}'_c)\right) \tag{9.4}$$

Notice we sum over all possible labelings of all cliques. The conditional random field induces a loss function, the *negative log loss*

$$\begin{aligned} & \phi(\mathbf{y}|\mathbf{g}_s, \mathbf{x}, f) && (9.5)\\ = \; & -\log p(\mathbf{y}|\mathbf{g}_s, \mathbf{x}) && (9.6)\\ = \; & -\sum_c f(\mathbf{g}_s, \mathbf{x}, c, \mathbf{y}_c) + \log \sum_{\mathbf{y}'} \exp\left(\sum_c f(\mathbf{g}_s, \mathbf{x}, c, \mathbf{y}'_c)\right) && (9.7) \end{aligned}$$

We now extend the standard "representer theorem" of kernel machines (Kimeldorf & Wahba, 1971) to conditional graphical models. Consider a regularized loss function (i.e. risk) of the form

$$R_\phi(f) = \sum_{i=1}^{l} \phi\left(\mathbf{y}^{(i)}|\mathbf{g}_s^{(i)}, \mathbf{x}^{(i)}, f\right) + \Omega\left(\|f\|_K\right) \tag{9.8}$$

on a labeled training set of size l. Ω is a strictly increasing function. It is important to note that the risk depends on all possible assignments $\mathbf{y}_c$ of labels to each clique, not just those observed in the labeled data $\mathbf{y}^{(i)}$. This is due to the normalization factor in the negative log loss. We have the following representer theorem for KCRFs:

Proposition (Representer theorem for CRFs). *The minimizer $f^\star$ of the risk* (9.8)*, if it exists, has the form*

$$f^\star(\mathbf{g}_s, \mathbf{x}, c, \mathbf{y}_c) \; = \; \sum_{i=1}^{l} \sum_{c'} \sum_{\mathbf{y}'} \alpha_{c'}^{(i)}(\mathbf{y}') \, K((\mathbf{g}_s^{(i)}, \mathbf{x}^{(i)}, c', \mathbf{y}'), (\mathbf{g}_s, \mathbf{x}, c, \mathbf{y}_c)) \tag{9.9}$$

where the sum $\mathbf{y}'$ is over all labelings of clique c'. The key property distinguishing this result from the standard representer theorem is that the "dual parameters" $\alpha_{c'}^{(i)}(\mathbf{y}')$ now depend on *all* assignments of labels. That is, for each training graph i, and each clique c' within the graph, and *each labeling* $\mathbf{y}'$ of the clique, not just the labeling in the training data, there is a dual parameter α.

The difference between KCRFs and the earlier non-kernel version of CRFs is the representation of f. In a standard non-kernel CRF, f is represented as a sum of weights times feature functions

$$f(\mathbf{g}_s, \mathbf{x}, c, \mathbf{y}_c) = \Lambda^\top \Phi(\mathbf{g}_s, \mathbf{x}, c, \mathbf{y}_c) \tag{9.10}$$

where Λ is a vector of weights (the "primal parameters"), and Φ is a set of fixed feature functions. Standard CRF learning finds the optimal Λ. Therefore one advantage of KCRFs is the use of kernels which can correspond to infinite features. In addition if we plug in a semi-supervised learning kernel to KCRFs, we obtain a semi-supervised learning algorithm on structured data.

Let us look at two special cases of KCRF. In the first case let the cliques be the vertices v, and with a special kernel

$$K((\mathbf{g}_s, \mathbf{x}, v, \mathbf{y}_v), (\mathbf{g}'_s, \mathbf{x}', v', \mathbf{y}'_{v'})) = K'(x_v, x'_{v'})\delta(y_v, y'_{v'}) \tag{9.11}$$

The representer theorem states that

$$f^\star(x, y) = \sum_{i=1}^{l} \sum_{v \in \mathbf{g}_s^{(i)}} \alpha_v^{(i)}(y) K'(x, x_v^{(i)}) \tag{9.12}$$

Under the probabilistic model 9.3, this is simply kernel logistic regression. It has no ability to model sequences.

In the second case let the cliques be edges connecting two vertices v_1, v_2. Let the kernel be

$$\begin{aligned} & K((\mathbf{g}_s, \mathbf{x}, v_1 v_2, y_{v_1} y_{v_2}), (\mathbf{g}'_s, \mathbf{x}', v'_1 v'_2, y'_{v_1} y'_{v_2})) & (9.13) \\ = \; & K'(x_{v_1}, x'_{v_1})\delta(y_{v_1}, y'_{v_1}) + \delta(y_{v_1}, y'_{v_1})\delta(y_{v_2}, y'_{v_2}) & (9.14) \end{aligned}$$

and we have

$$f^\star(x_{v_1}, y_{v_1} y_{v_2}) = \sum_{i=1}^{l} \sum_{u \in \mathbf{g}_s^{(i)}} \alpha_u^{(i)}(y_{v_1}) K'(x_{v_1}, x_u^{(i)}) + \alpha(y_{v_1}, y_{v_2}) \tag{9.15}$$

which is a simple type of semiparametric CRF. It has rudimentary ability to model sequences with $\alpha(y_{v_1}, y_{v_2})$, similar to a transition matrix between states. In both cases, we can use a graph kernel K' on both labeled and unlabeled data for semi-supervised learning.

9.3 Sparse Training: Clique Selection

The representer theorem shows that the minimizing function f is supported by labeled cliques over the training examples; however, this may result in an extremely large number of parameters. We therefore pursue a strategy of incrementally selecting cliques in order to greedily reduce the risk. The resulting procedure is parallel to forward stepwise logistic regression, and to related methods for kernel logistic regression (Zhu & Hastie, 2001).

Our algorithm will maintain an *active set* $\left\{(\mathbf{g}_s^{(i)}, \mathbf{x}^{(i)}, c, \mathbf{y}_c)\right\}$, each item uniquely specifies a labeled clique. Again notice the labelings $\mathbf{y}_c$ are not necessarily those appearing in the training data. Each labeled clique can be represented by a basis function $h(\cdot) = K((\mathbf{g}_s^{(i)}, \mathbf{x}^{(i)}, c, \mathbf{y}_c), \cdot) \in \mathcal{H}_K$, and is assigned a parameter $\alpha_h = \alpha_c^{(i)}(\mathbf{y}_c)$. We work with the regularized risk

$$R_\phi(f) = \sum_{i=1}^{l} \phi\left(\mathbf{y}^{(i)} | \mathbf{g}_s^{(i)}, \mathbf{x}^{(i)}, f\right) + \frac{\lambda}{2}\|f\|_K^2 \tag{9.16}$$

where ϕ is the negative log loss of equation (9.5). To evaluate a candidate h, one strategy is to compute the *gain* $\sup_\alpha R_\phi(f) - R_\phi(f + \alpha h)$, and to choose the candidate h having the largest gain. This presents an apparent difficulty, since the optimal parameter α cannot be computed in closed form, and must be evaluated numerically. For sequence models this would involve forward-backward calculations for each candidate h, the cost of which is prohibitive.

As an alternative, we adopt the functional gradient descent approach, which evaluates a small change to the current function. For a given candidate h, consider adding h to the current model with small weight ε; thus $f \mapsto f + \varepsilon h$. Then $R_\phi(f + \varepsilon h) = R_\phi(f) + \varepsilon dR_\phi(f, h) + O(\varepsilon^2)$, where the functional derivative of R_ϕ at f in the direction h is computed as

$$dR_\phi(f, h) \quad = \quad E_f[h] - \widetilde{E}[h] + \lambda\langle f, h\rangle_K \tag{9.17}$$

where $\widetilde{E}[h] = \sum_i \sum_c h(\mathbf{g}_s^{(i)}, \mathbf{x}^{(i)}, c, \mathbf{y}_c^{(i)})$ is the empirical expectation and $E_f[h] = \sum_i \sum_{\mathbf{y}} \sum_c p(\mathbf{y}|\mathbf{x}^{(i)}, f) h(\mathbf{g}_s^{(i)}, \mathbf{x}^{(i)}, c, \mathbf{y}_c)$ is the model expectation conditioned on $\mathbf{x}$. The idea is that in directions h where the functional gradient $dR_\phi(f, h)$ is large, the model is mismatched with the labeled data; this direction should be added to the model to make a correction. This results in the greedy clique selection algorithm, as summarized in Figure 9.1.

An alternative to the functional gradient descent algorithm above is to estimate parameters α_h for each candidate. When each candidate clique is a vertex, the

Initialize with $f = 0$, and iterate:

1. For each candidate $h \in \mathcal{H}_K$, supported by a single labeled clique, calculate the functional derivative $dR_\phi(f, h)$.
2. Select the candidate $h = \arg\max_h |dR_\phi(f, h)|$ having the largest gradient direction. Set $f \mapsto f + \alpha_h h$.
3. Estimate parameters α_f for each active f by minimizing $R_\phi(f)$.

Figure 9.1: Greedy Clique Selection. Labeled cliques encode basis functions h which are greedily added to the model, using a form of functional gradient descent.

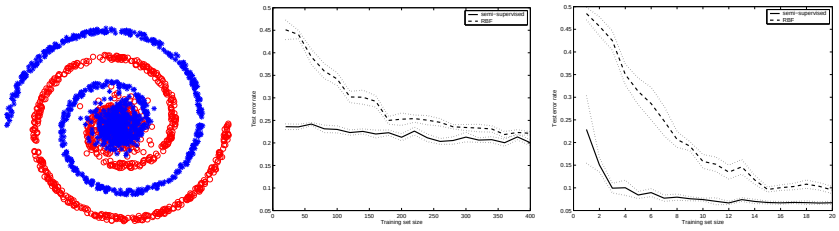


Figure 9.2: Left: The galaxy data is comprised of two interlocking spirals together with a "dense core" of samples from both classes. Center: Kernel logistic regression comparing two kernels, RBF and a graph kernel using the unlabeled data. Right: Kernel conditional random fields, which take into account the sequential structure of the data.

gain can be efficiently approximated using a mean field approximation. Under this approximation, a candidate is evaluated according to the approximate gain

$$R_\phi(f) - R_\phi(f + \alpha h) \tag{9.18}$$

$$\approx \sum_i \sum_v Z(f, \mathbf{x}^{(i)})^{-1} p(\mathbf{y}_v^{(i)} | \mathbf{x}^{(i)}, f) \exp(\alpha h(\mathbf{x}^{(i)}, \mathbf{y}_v^{(i)})) + \lambda \langle f, h \rangle \tag{9.19}$$

which is a logistic approximation. Details can be found in Appendix E.

9.4 Synthetic Data Experiments

In the experiments reported below for sequences, the marginal probabilities $p(\mathbf{y}_v = 1|\mathbf{x})$ and expected counts for the state transitions are required; these are computed

using the forward-backward algorithm, with log domain arithmetic to avoid underflow. A quasi-Newton method (BFGS, cubic-polynomial line search) is used to estimate the parameters in step 3 of Figure 9.1.

To work with a data set that will distinguish a semi-supervised graph kernel from a standard kernel, and a sequence model from a non-sequence model, we prepared a synthetic data set ("galaxy") that is a variant of spirals, see Figure 9.2 (left). Note data in the dense core come from both classes.

We sample 100 sequences of length 20 according to an HMM with two states, where each state emits instances uniformly from one of the classes. There is a 90% chance of staying in the same state, and the initial state is uniformly chosen. The idea is that under a sequence model we should be able to use the context to determine the class of an example at the core. However, under a non-sequence model without the context, the core region will be indistinguishable, and the dataset as a whole will have about 20% Bayes error rate. Note the choice of semi-supervised vs. standard kernels and sequence vs. non-sequence models are orthogonal; the four combinations are all tested on.

We construct the semi-supervised graph kernel by first building an unweighted 10-nearest neighbor graph. We compute the associated graph Laplacian Δ, and then the graph kernel $K = 10\left(\Delta + 10^{-6}I\right)^{-1}$. The standard kernel is the radial basis function (RBF) kernel with an optimal bandwidth $\sigma = 0.35$.

First we apply both kernels to a non-sequence model: kernel logistic regression (9.12), see Figure 9.2 (center). The sequence structure is ignored. Ten random trials were performed with each training set size, which ranges from 20 to 400 points. The error intervals are one standard error. As expected, when the labeled set size is small, the RBF kernel results in significantly larger test error than the graph kernel. Furthermore, both kernels saturate at the 20% Bayes error rate.

Next we apply both kernels to a KCRF sequence model 9.15. Experimental results are shown in Figure 9.2 (right). Note the x-axis is the number of training sequences: Since each sequence has 20 instances, the range is the same as Figure 9.2 (center). The kernel CRF is capable of getting below the 20% Bayes error rate of the non-sequence model, with both kernels and sufficient labeled data. However the graph kernel is able to learn the structure much faster than the RBF kernel. Evidently the high error rate for small label data sizes prevents the RBF model from effectively using the context.

Finally we examine clique selection in KCRFs. For this experiment we use 50 training sequences. We use the mean field approximation and only select vertex cliques. At each iteration the selection is based on the estimated change in risk for each candidate vertex (training position). We plot the estimated change in risk for the first four iterations of clique selection, with the graph kernel and RBF kernel re-

spectively in Figure 9.3. Smaller values (lower on z-axis) indicate good candidates with potentially large reduction in risk if selected. For the graph kernel, the first two selected vertices are sufficient to reduce the risk essentially to the minimum (note in the third iteration the z-axis scale is already 10^{-6}). Such reduction does not happen with the RBF kernel.

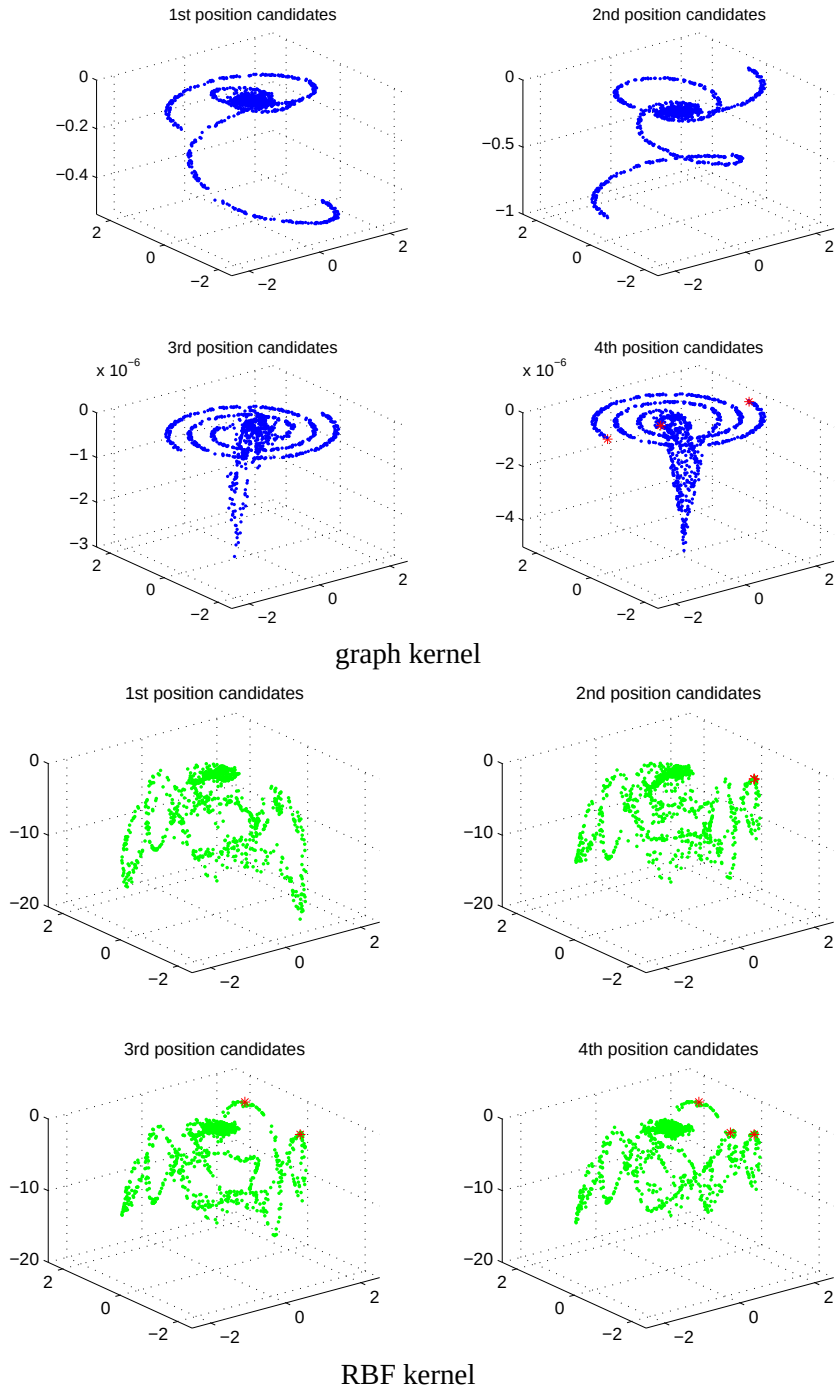


Figure 9.3: Mean field estimate of the change in loss function with the graph kernel (top) and the RBF kernel (bottom) for the first four iterations of clique selection on the galaxy dataset. For the graph kernel the endpoints of the spirals are chosen as the first two cliques.

Chapter 10

Harmonic Mixtures: Handling Unseen Data and Reducing Computation

There are two important questions to graph based semi-supervised learning methods:

1. The graph is constructed only on the labeled and unlabeled data. Many such methods are transductive in nature. How can we handle unseen new data points?

2. They often involve expensive manipulation on large matrices, for example matrix inversion, which can be $O(n^3)$. Because unlabeled data is relatively easy to obtain in large quantity, the matrix could be too big to handle. How can we reduce computation when the unlabeled dataset is large?

In this chapter we address these questions by combining graph method with a mixture model.

Mixture model has long been used for semi-supervised learning, e.g. Gaussian mixture model (GMM) (Castelli & Cover, 1996) (Ratsaby & Venkatesh, 1995), and mixture of multinomial (Nigam et al., 2000). Training is typically done with the EM algorithm. It has several advantages: The model is inductive and handles unseen points naturally; It is a parametric model with a small number of parameters. However when there is underlying manifold structure in the data, EM may have difficulty making the *labels* follow the manifold: An example is given in Figure 10.1. The desired behavior is shown in Figure 10.2, which can be achieved by the *harmonic mixture* method discussed in this Chapter.

Mixture models and graph based semi-supervised learning methods make different assumptions about the relation between unlabeled data and labels. Nevertheless, they are not mutually exclusive. It is possible that the data fits the component model (e.g. Gaussian) *locally*, while the manifold structure appears *globally*. We combine the best from both. From a graph method point of view, the resulting model is a much smaller (thus computationally less expensive) 'backbone graph' with 'supernodes' induced by the mixture components; From a mixture model point of view, it is still inductive and naturally handles new points, but also has the ability for labels to follow the data manifold. Our approach is related to graph regularization in (Belkin et al., 2004b), and is an alternative to the induction method in (Delalleau et al., 2005). It should be noted that we are interested in mixture models with a large number (possibly more than the number of labeled points) of components, so that the manifold structure can appear, which is different from previous works.

10.1 Review of Mixture Models and the EM Algorithm

In typical mixture models for classification, the generative process is the following. One first picks a class y, then chooses a mixture component $m \in \{1 \dots M\}$ by $p(m|y)$, and finally generates a point x according to $p(x|m)$. Thus $p(x, y) = \sum_{m=1}^{M} p(y)p(m|y)p(x|m)$. In this paper we take a different but equivalent parameterization,

$$p(x, y) = \sum_{m=1}^{M} p(m)p(y|m)p(x|m) \tag{10.1}$$

We allow $p(y|m) > 0$ for all y, enabling classes to share a mixture component.

The standard EM algorithm learns these parameters to maximize the log likelihood of observed data:

$$\begin{aligned} \mathcal{L}(\Theta) &= \log p(x_L, x_U, y_L|\Theta) \qquad (10.2)\\ &= \sum_{i\in L} \log p(x_i, y_i|\Theta) + \sum_{i\in U} \log p(x_i|\Theta) \\ &= \sum_{i\in L} \log \sum_{m=1}^{M} p(m)p(y_i|m)p(x_i|m) + \sum_{i\in U} \log \sum_{m=1}^{M} p(m)p(x_i|m) \end{aligned}$$

We introduce arbitrary distributions $q_i(m|i)$ on mixture membership, one for each

i. By Jensen's inequality

$$\begin{aligned}
\mathcal{L}(\Theta) &= \sum_{i\in L}\log\sum_{m=1}^{M} q_i(m|x_i,y_i)\frac{p(m)p(y_i|m)p(x_i|m)}{q_i(m|x_i,y_i)} && (10.3)\\
&\quad + \sum_{i\in U}\log\sum_{m=1}^{M} q_i(m|x_i)\frac{p(m)p(x_i|m)}{q_i(m|x_i)} \\
&\geq \sum_{i\in L}\sum_{m=1}^{M} q_i(m|x_i,y_i)\log\frac{p(m)p(y_i|m)p(x_i|m)}{q_i(m|x_i,y_i)} && (10.4)\\
&\quad + \sum_{i\in U}\sum_{m=1}^{M} q_i(m|x_i)\log\frac{p(m)p(x_i|m)}{q_i(m|x_i)} \\
&\equiv \mathcal{F}(\mathbf{q},\Theta) && (10.5)
\end{aligned}$$

The EM algorithm works by iterating coordinate-wise ascend on $\mathbf{q}$ and Θ to maximize $\mathcal{F}(\mathbf{q},\Theta)$. The E step fixes Θ and finds the $\mathbf{q}$ that maximizes $\mathcal{F}(\mathbf{q},\Theta)$. We denote the fixed Θ at iteration t by $p(m)^{(t)}$, $p(y|m)^{(t)}$ and $p(x|m)^{(t)}$. Since the terms of $\mathcal{F}$ has the form of KL divergence, it is easy to see that the optimal $\mathbf{q}$ are the posterior on m:

$$\begin{aligned}
q_i^{(t)}(m|x_i,y_i) &= p(m|x_i,y_i) = \frac{p(m)^{(t)}p(y_i|m)^{(t)}p(x_i|m)^{(t)}}{\sum_{k=1}^{M} p(k)^{(t)}p(y_i|k)^{(t)}p(x_i|k)^{(t)}}\ , i\in L\\
q_i^{(t)}(m|x_i) &= p(m|x_i) = \frac{p(m)^{(t)}p(x_i|m)^{(t)}}{\sum_{k=1}^{M} p(k)^{(t)}p(x_i|k)^{(t)}}\ , i\in U && (10.6)
\end{aligned}$$

The M step fixes $\mathbf{q}^{(t)}$ and finds $\Theta^{(t+1)}$ to maximize $\mathcal{F}$. Taking the partial derivatives and set to zero, we find

$$\begin{aligned}
p(m)^{(t+1)} &\propto \sum_{i\in L\cup U} q_i(m)^{(t)} && (10.7)\\
\theta_m^{(t+1)} \equiv p(y=1|m)^{(t+1)} &= \frac{\sum_{i\in L,\ y_i=1} q_i(m)^{(t)}}{\sum_{i\in L} q_i(m)^{(t)}} && (10.8)\\
\sum_{i\in L\cup U} q_i(m)^{(t)}\frac{1}{p(x_i|m)}\frac{\partial p(x_i|m)}{\partial\Theta_x} &= 0 && (10.9)
\end{aligned}$$

The last equation needs to be reduced further with the specific generative model

for x, e.g. Gaussian or multinomial. For Gaussian, we have

$$\mu_m^{(t+1)} = \frac{\sum_{i \in L \cup U} q_i(m)^{(t)} x_i}{\sum_{i \in L \cup U} q_i(m)^{(t)}} \tag{10.10}$$

$$\Sigma_m^{(t+1)} = \frac{\sum_{i \in L \cup U} q_i(m)^{(t)} (x_i - \mu_m^{(t)})(x_i - \mu_m^{(t)})^\top}{\sum_{i \in L \cup U} q_i(m)^{(t)}} \tag{10.11}$$

In practice one can smooth the ML estimate of covariance to avoid degeneracy:

$$\Sigma_m^{(t+1)} = \frac{\epsilon \mathbf{I} + \sum_{i \in L \cup U} q_i(m)^{(t)} (x_i - \mu_m^{(t)})(x_i - \mu_m^{(t)})^\top}{\epsilon + \sum_{i \in L \cup U} q_i(m)^{(t)}} \tag{10.12}$$

After EM converges, the classification of a new point x is done by

$$\begin{aligned} p(y = 1|x) &= \sum_{m=1}^{M} p(y = 1|m) p(m|x) \\ &= \frac{\sum_{m=1}^{M} p(y = 1|m) p(x|m) p(m)}{\sum_{m=1}^{M} p(x|m) p(m)} \end{aligned} \tag{10.13}$$

10.2 Label Smoothness on the Graph

Graph-based semi-supervised learning methods enforce label smoothness over a graph, so that neighboring labels tend to have the same label. The graph has n nodes $L \cup U$. Two nodes are connected by an edge with higher weights if they are more likely to be in the same class. The graph is represented by the $n \times n$ symmetric weight matrix W, and is assumed given.

Label smoothness can be expressed in different ways. We use the energy of the label posterior as the measure,

$$E(f) = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (f_i - f_j)^2 = f^\top \Delta f \tag{10.14}$$

where f is the label posterior vector, defined as

$$f_i = \begin{cases} \delta(y_i, 1) & i \in L \\ p(y_i = 1|x_i, \Theta) & i \in U \end{cases} \tag{10.15}$$

That is, f_i is the probability that point i having label 1 under the mixture model Θ. The energy is small when f varies smoothly on the graph. $\Delta = D - W$ is the combinatorial Laplacian matrix, and D is the diagonal degree matrix with $D_{ii} = \sum_j w_{ij}$. See Chapter 4 for more details. Other smoothness measures are possible too, for example those derived from the normalized Laplacian (Zhou et al., 2004a) or spectral transforms (Zhu et al., 2005).

10.3 Combining Mixture Model and Graph

We want to train a mixture model that maximizes the data log likelihood (10.3) and minimizes the graph energy (10.14) at the same time. One way of doing so is to learn the parameters $p(m), p(x|m), p(y|m)$ to maximize the objective

$$\mathcal{O} = \alpha\mathcal{L} - (1-\alpha)E \tag{10.16}$$

where $\alpha \in [0, 1]$ is a coefficient that controls the relative strength of the two terms. The E term may look like a prior $e^{-f^\top \Delta f}$ on the parameters. But it involves the observed labels y_L, and is best described as a discriminative objective, while $\mathcal{L}$ is a generative objective. This is closely related to, but different from, the graph regularization framework of (Belkin et al., 2004b). Learning all the parameters together however is difficult. Because of the E term, it is similar to conditional EM training which is more complicated than the standard EM algorithm. Instead we take a two-step approach:

- Step 1: Train all parameters $p(m), p(x|m), p(y|m)$ with standard EM, which maximizes $\mathcal{L}$ only;
- Step 2: Fix $p(m)$ and $p(x|m)$, and only learn $p(y|m)$ to maximize (10.16).

It is suboptimal in terms of optimizing the objective function. However it has two advantages: We created a concave optimization problem in the second step (see section 10.3.2); Moreover, we can use standard EM without modification. We call the solution *harmonic mixtures*.

We focus on step 2. The free parameters are $p(y|m)$ for $m = 1 \dots M$. To simplify the notation, we use the shorthand $\theta_m \equiv p(y = 1|m)$, and $\theta \equiv (\theta_1, \dots, \theta_M)^\top$. We first look at the special case with $\alpha = 0$ in the objective function (10.16), as it has a particularly simple closed form solution and interpretation. Notice although $\alpha = 0$, the generative objective $\mathcal{L}$ still influences θ through $p(m)$ and $p(x|m)$ learned in step 1.

10.3.1 The Special Case with $\alpha = 0$

We need to find the parameters θ that minimize E. θ are constrained in $[0, 1]^M$. However let us look at the *unconstrained* optimization problem first. Applying the chain rule:

$$\frac{\partial E}{\partial \theta_m} = \langle \frac{\partial E}{\partial f_U}, \frac{\partial f_U}{\partial \theta_m} \rangle \tag{10.17}$$

The first term is

$$\begin{aligned}\frac{\partial E}{\partial f_U} &= \frac{\partial}{\partial f_U}\left(f^\top \Delta f\right) && (10.18)\\ &= \frac{\partial}{\partial f_U}\left(f_L^\top \Delta_{LL} f_L + 2 f_L^\top \Delta_{LU} f_U + f_U^\top \Delta_{UU} f_U\right) && (10.19)\\ &= 2\Delta_{LU} f_L + 2\Delta_{UU} f_U && (10.20)\end{aligned}$$

where we partitioned the Laplacian matrix into labeled and unlabeled parts respectively. The second term is

$$\frac{\partial f_U}{\partial \theta_m} = \left(p(m|x_{l+1}), \ldots, p(m|x_{l+u})\right)^\top \equiv \mathbf{R}_m \quad (10.21)$$

where we defined a $u \times M$ *responsibility matrix* $\mathbf{R}$ such that $\mathbf{R}_{im} = p(m|x_i)$, and $\mathbf{R}_m$ is its m-th column. We used the fact that for $i \in U$,

$$\begin{aligned}f_i &= p(y_i = 1|x_i, \Theta) && (10.22)\\ &= \frac{\sum_m p(m)p(y_i = 1|m)p(x_i|m)}{\sum_m p(m)p(x_i|m)} && (10.23)\\ &= \sum_m p(m|x_i)p(y_i = 1|m) && (10.24)\\ &= \sum_m p(m|x_i)\theta_m && (10.25)\end{aligned}$$

Notice we can write $f_U = \mathbf{R}\theta$. Therefore

$$\begin{aligned}\frac{\partial E}{\partial \theta_m} &= \mathbf{R}_m^\top \left(2\Delta_{UU} f_U + 2\Delta_{UL} f_L\right) && (10.26)\\ &= \mathbf{R}_m^\top \left(2\Delta_{UU}\mathbf{R}\theta + 2\Delta_{UL} f_L\right) && (10.27)\end{aligned}$$

When we put all M partial derivatives in a vector and set them to zero, we find

$$\frac{\partial E}{\partial \theta} = \mathbf{R}^\top \left(2\Delta_{UU}\mathbf{R}\theta + 2\Delta_{UL} f_L\right) = \mathbf{0} \quad (10.28)$$

where $\mathbf{0}$ is the zero vector of length M. This is a linear system and the solution is

$$\theta = -\left(\mathbf{R}^\top \Delta_{UU}\mathbf{R}\right)^{-1} \mathbf{R}^\top \Delta_{UL} f_L \quad (10.29)$$

Notice this is the solution to the unconstrained problem, where some θ might be out of the bound $[0, 1]$. If it happens, we set out-of-bound θ's to their corresponding boundary values of 0 or 1, and use them as starting point in a constrained convex

optimization (the problem is convex, as shown in the next section) to find the global solution. In practice however we found most of the time the closed form solution for the unconstrained problem is already within bounds. Even when some components are out of bounds, the solution is close enough to the constrained optimum to allow quick convergence.

With the component class membership θ, the soft labels for the unlabeled data are given by

$$f_U = -\mathbf{R}\theta \tag{10.30}$$

Unseen new points can be classified similarly.

We can compare (10.29) with the (completely graph based) harmonic function solution (Zhu et al., 2003a). The former is $f_U = -\mathbf{R}\left(\mathbf{R}^\top \Delta_{UU}\mathbf{R}\right)^{-1}\mathbf{R}^\top \Delta_{UL} f_L$; The latter is $f_U = -\Delta_{UU}^{-1}\Delta_{UL} f_L$. Computationally the former only needs to invert a $M \times M$ matrix, which is much cheaper than the latter of $u \times u$ because typically the number of mixture components is much smaller than the number of unlabeled points. This reduction is possible because f_U are now tied together by the mixture model.

In the special case where R corresponds to hard clustering, we just created a much smaller *backbone graph* with *supernodes* induced by the mixture components. In this case $R_{im} = 1$ for cluster m to which point i belongs, and 0 for all other $M - 1$ clusters. The backbone graph has the same L labeled nodes as in the original graph, but only M unlabeled supernodes. Let w_{ij} be the weight between nodes i, j in the original graph. By rearranging the terms it is not hard to show that in the backbone graph, the equivalent weight between supernodes $s, t \in \{1 \dots M\}$ is

$$\tilde{w}_{st} = \sum_{i,j \in U} R_{is} R_{jt} w_{ij} \tag{10.31}$$

and the equivalent weight between a supernode s and a labeled node $l \in L$ is

$$\tilde{w}_{sl} = \sum_{i \in U} R_{is} w_{il} \tag{10.32}$$

θ is simply the harmonic function on the supernodes in the backbone graph. For this reason $\theta \in [0, 1]^M$ is guaranteed. Let $c(m) = \{i | R_{im} = 1\}$ be the cluster m. The equivalent weight between supernodes s, t reduces to

$$\tilde{w}_{st} = \sum_{i \in c(s),\ j \in c(t)} w_{ij} \tag{10.33}$$

The supernodes are the clusters themselves. The equivalent weights are the sum of edges between the clusters (or the cluster and a labeled node). One can easily

Input: initial mixture model $p(m), p(x|m), p(y|m), m = 1 \dots M$
data x_L, y_L, x_U
graph Laplacian Δ
1. Run standard EM on data and get converged model $p(m), p(x|m), p(y|m)$
2. Fix $p(m), p(x|m)$. Compute $\theta_m \equiv p(y = 1|m) = -\left(\mathbf{R}^\top \Delta_{UU} \mathbf{R}\right)^{-1} \mathbf{R}^\top \Delta_{UL} f_L$
3. Set out-of-bound θ's to 0 or 1, run constrained convex optimization
Output: mixture model $p(m), p(x|m), p(y|m), m = 1 \dots M$

Table 10.1: The harmonic mixture algorithm for the special case $\alpha = 0$

create such a backbone graph by e.g. k-means clustering. In the general case when R is soft, the solution deviates from that of the backbone graph.

The above algorithm is listed in Table 10.1. In practice some mixture components may have little or no responsibility ($p(m) \approx 0$). They should be excluded from (10.29) to avoid numerical problems. In addition, if R is rank deficient we use the pseudo inverse in (10.29).

10.3.2 The General Case with $\alpha > 0$

The objective (10.16) is concave in θ. To see this, we first write $\mathcal{L}$ as

$$\begin{aligned} \mathcal{L}(\Theta) &= \sum_{i \in L} \log \sum_{m=1}^{M} p(m) p(y_i|m) p(x_i|m) + const \qquad (10.34) \\ &= \sum_{\substack{i \in L \\ y_i = 1}} \log \sum_{m=1}^{M} p(m) p(x_i|m) \theta_m + \sum_{\substack{i \in L \\ y_i = -1}} \log \sum_{m=1}^{M} p(m) p(x_i|m)(1 - \theta_m) + const \end{aligned}$$

Since we fix $p(m)$ and $p(x|m)$, the term within the first sum has the form $\log \sum_m a_m \theta_m$. We can directly verify the Hessian

$$H = \left[\frac{\partial \log \sum_m a_m \theta_m}{\partial \theta_i \partial \theta_j} \right] = -\frac{1}{(\sum_m a_m \theta_m)^2} a a^\top \preceq 0 \qquad (10.35)$$

is negative semi-definite. Therefore the first term ($i \in L$ and $y_i = 1$) is concave. Similarly the Hessian for the second term is

$$H = \left[\frac{\partial \log \sum_m a_m (1 - \theta_m)}{\partial \theta_i \partial \theta_j} \right] = -\frac{a a^\top}{(\sum_m a_m (1 - \theta_m))^2} \preceq 0 \qquad (10.36)$$

$\mathcal{L}$ is the non-negative sum of concave terms and is concave. Recall $f_U = R\theta$, the graph energy can be written as

$$\begin{aligned} E &= f^\top \Delta f && (10.37)\\ &= f_L^\top \Delta_{LL} f_L + 2 f_L^\top \Delta_{LU} f_U + f_U^\top \Delta_{UU} f_U && (10.38)\\ &= f_L^\top \Delta_{LL} f_L + 2 f_L^\top \Delta_{LU} R\theta + \theta^\top R^\top \Delta_{UU} R\theta && (10.39) \end{aligned}$$

The Hessian is $2R^\top \Delta_{UU} R \succeq 0$ because $\Delta_{UU} \succeq 0$. Therefore E is convex in θ. Putting them together, $\mathcal{O}$ is concave in θ.

As θ_m is in $[0, 1]$, we perform constrained convex optimization in the general case with $\alpha > 0$. The gradient of the objective is easily computed:

$$\frac{\partial \mathcal{O}}{\partial \theta_m} = \alpha \frac{\partial \mathcal{L}}{\partial \theta_m} - (1-\alpha)\frac{\partial E}{\partial \theta_m} \quad (10.40)$$

$$\frac{\partial \mathcal{L}}{\partial \theta_m} \quad (10.41)$$

$$= \sum_{\substack{i \in L \\ y_i = 1}} \frac{p(m)p(x_i|m)}{\sum_{k=1}^M p(k)p(x_i|k)\theta_k} - \sum_{\substack{i \in L \\ y_i = -1}} \frac{p(m)p(x_i|m)}{\sum_{k=1}^M p(k)p(x_i|k)(1-\theta_k)} \quad (10.42)$$

and $\partial E / \partial \theta$ was given in (10.28). One can also use the sigmoid function to transform it into an unconstrained optimization problem with

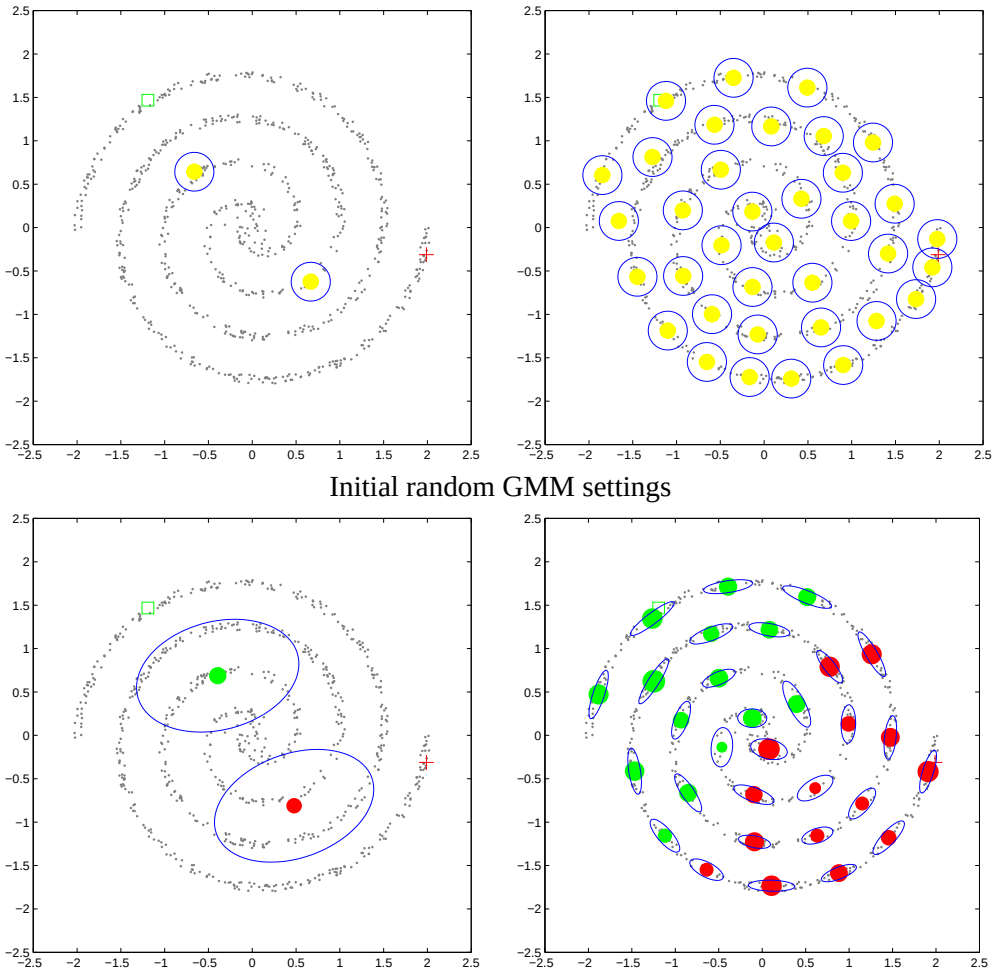
$$\theta_m = \sigma(\gamma_m) = \frac{1}{e^{-\gamma_m} + 1} \quad (10.43)$$

and optimize the γ's.

Although the objective is concave, a good starting point for θ is still important to reduce the computation time until convergence. We find a good initial value for θ by solving an one-dimensional concave optimization problem first. We have two parameters at hand: θ_{em} is the solution from the standard EM algorithm in step 1, and $\theta_{special}$ is the special case solution in section 10.3.1. We find the optimal interpolated coefficient $\epsilon \in [0, 1]$

$$\theta_{init} = \epsilon \theta_{em} + (1-\epsilon)\theta_{special} \quad (10.44)$$

that maximizes the objective (the optimal ϵ in general will not be α). Then we start from θ_{init} and use a quasi-Newton algorithm to find the global optimum for θ.



Initial random GMM settings

After EM converges

(a) $M = 2$ Gaussian components (b) $M = 36$ Gaussian components

Figure 10.1: Gaussian mixture models learned with the standard EM algorithm cannot make labels follow the manifold structure in an artificial dataset. Small dots are unlabeled data. The two labeled points are marked with red $+$ and green $\square$. The left panel has $M = 2$ and right $M = 36$ mixture components. Top plots show the initial settings of the GMM. Bottom plots show the GMM after EM converges. The ellipses are the contours of covariance matrices. The colored central dots have sizes proportional to the component weight $p(m)$. Components with very small $p(m)$ are not plotted. The color stands for component class membership $\theta_m \equiv p(y = 1|m)$: red for $\theta = 1$, green for $\theta = 0$, and intermediate yellow for values in between – which did not occur in the converged solutions. Notice in the bottom-right plot, although the density $p(x)$ is estimated well by EM, θ does not follow the manifold.

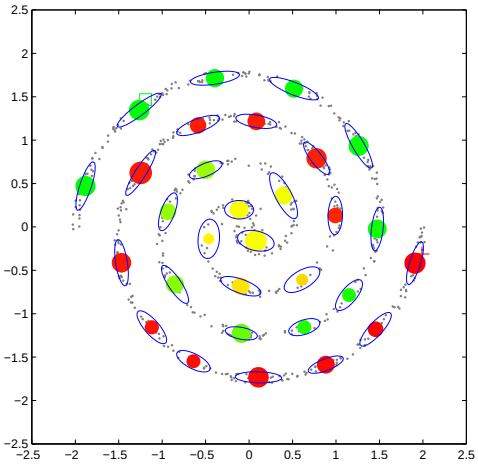


Figure 10.2: The GMM with the component class membership θ learned as in the special case $\alpha = 0$. θ, color coded from red to yellow and green, now follow the structure of the unlabeled data.

10.4 Experiments

We test harmonic mixture on synthetic data, image and text classification. The emphases are on how harmonic mixtures perform on unlabeled data compared to EM or the harmonic function; how they handle unseen data; and whether they can reduce the problem size. Unless otherwise noted, the harmonic mixtures are computed with $\alpha = 0$.

10.4.1 Synthetic Data

First we look at a synthetic dataset in Figure 10.1. It has a Swiss roll structure, and we hope the labels can follow the spiral arms. There is one positive and one negative labeled point, at roughly the opposite ends. We use $u = 766$ unlabeled points and an additional 384 points as unseen test data.

The mixture model and standard EM. We start with Figure 10.1(a, top), the initial setting for a Gaussian mixture model with $M = 2$ components. The initial means are set by running a k-means algorithm. The initial covariances are identity, thus the circles. The initial θ are all set to 0.5, represented by the yellow color. (a, bottom) shows the GMM after EM converges. Obviously it is a bad model because M is too small.

Next we consider a Gaussian mixture model (GMM) with $M = 36$ compo-

nents, each with full covariance. Figure 10.1(b, top) shows the initial GMM and (b, bottom) the converged GMM after running EM. The GMM models the manifold density $p(x)$ well. However the component class membership $\theta_m \equiv p(y = 1|m)$ (red and green colors) does not follow the manifold. In fact θ takes the extreme values of 0 or 1 along a somewhat linear boundary instead of following the spiral arms, which is undesirable. The classification of data points will not follow the manifold either.

The graph and harmonic mixtures. Next we combine the mixture model with a graph to compute the harmonic mixtures, as in the special case $\alpha = 0$. We construct a fully connected graph on the $L \cup U$ data points with weighted edges $w_{ij} = \exp\left(-||x_i - x_j||^2/0.01\right)$. We then reestimate θ, which are shown in Figure 10.2. Note θ now follow the manifold as it changes from 0 (green) to approximately 0.5 (yellow) and finally 1 (red). This is the desired behavior.

The particular graph-based method we use needs extra care. The harmonic function solution f is known to sometimes skew toward 0 or 1. This problem is easily corrected if we know or have an estimate of the proportion of positive and negative points, with the Class Mass Normalization heuristic (Zhu et al., 2003a). In this paper we use a similar but simpler heuristic. Assuming the two classes are about equal in size, we simply set the decision boundary at the median. That is, let $f(l+1), \ldots, f(n)$ be the soft label values on the unlabeled nodes. Let $m(f) =$ median$(f(l+1), \ldots, f(n))$. We classify point i as positive if $f(i) > m(f)$, and negative otherwise.

Sensitivity to M. If the number of mixture components M is too small, the GMM is unable to model $p(x)$ well, let alone θ. In other words, the harmonic mixture is sensitive to M. M has to be larger than a certain threshold so that the manifold structure can appear. In fact M may need to be larger than the number of labeled points l, which is unusual in traditional mixture model methods for semi-supervised learning. However once M is over the threshold, further increase should not dramatically change the solution. In the end the harmonic mixture may approach the harmonic function solution when $M = u$.

Figure 10.3(a) shows the classification accuracy on U as we change M. We find that the threshold for harmonic mixtures is $M = 35$, at which point the accuracy ('HM') jumps up and stabilizes thereafter. This is the number of mixture components needed for harmonic mixture to capture the manifold structure. The harmonic function on the complete graph ('graph') is not a mixture model and appears flat. The EM algorithm ('EM') fails to discover the manifold structure regardless of the number of mixtures M.

Computational savings. The harmonic mixtures perform almost as well as the harmonic function on the complete graph, but with a much smaller problem size. As Figure 10.3(a) shows, we only need to invert a 35×35 matrix instead of a

766×766 one as required by the harmonic function solution. The difference can be significant if the unlabeled set size is even larger. There is of course the overhead of EM training.

Handling unseen data. Because the harmonic mixture model is a mixture model, it naturally handles unseen points. On 384 new test points harmonic mixtures perform similarly to Figure 10.3(a), with accuracies around 95.3% after $M \geq 35$.

10.4.2 Image Recognition: Handwritten Digits

We use the '1vs2' dataset which contains equal number of images of handwritten digit of 1s and 2s. Each gray scale image is 8×8, which is represented by a 64 dimensional vector of pixel values. We use $l + u = 1600$ images as the labeled and unlabeled set, and 600 additional images as unseen new data to test induction.

The mixture model. We use Gaussian mixture models. To avoid data sparseness problem, we model each Gaussian component with a spherical covariance, i.e. diagonal covariance matrix with the same variance in all dimensions. Different components may have different variances. We set the initial means and variances of the GMM with k-means algorithm before running EM.

The graph. We use a symmetrized 10-nearest-neighbor weighted graph on the 1600 images. That is, images i, j are connected if i is within j's 10NN or vice versa, as measured by Euclidean distance. The weights are $w_{ij} = \exp\left(-||x_i - x_j||^2/140^2\right)$.

Sensitivity to M. As illustrated in the synthetic data, the number of mixture components M needs to be large enough for harmonic mixture to work. We vary M and observe the classification accuracies on the unlabeled data with different methods. For each M we perform 20 trials with random L/U split and plot the mean and standard deviation of classification accuracies in Figure 10.3(b). The experiments were performed with labeled set size fixed at $l = 10$. We conclude that harmonic mixtures need only $M \approx 100$ components to match the performance of the harmonic function method.

Computational savings. In terms of graph method computation, we invert a 100×100 matrix instead of the original 1590×1590 matrix for harmonic function. This is good saving with little sacrifice in accuracy. We fix $M = 100$ in the experiments that follow.

Handling unseen data. We systematically vary labeled set size l. For each l we run 20 random trials. The classification accuracy on U (with 1600-l points) and unseen data (600 points) are listed in Table 10.2. On U, harmonic mixtures ('HM') achieve the same accuracy as harmonic function ('graph'). Both are not sensitive to l. The GMM trained with EM ('EM') also performs well when l is not too small, but suffers otherwise. On the unseen test data, the harmonic mixtures maintain high accuracy.

The general case $\alpha > 0$. We also vary the parameter α between 0 and 1, which balances the generative and discriminative objectives. In our experiments $\alpha = 0$ always gives the best accuracies.

10.4.3 Text Categorization: PC vs. Mac

We perform binary text classification on the two groups comp.sys.ibm.pc.hardware vs. comp.sys.mac.hardware (982 and 961 documents respectively) in the 18828 version of the 20-newsgroups data. We use rainbow (McCallum, 1996) to preprocess the data, with the default stopword list, no stemming, and keep words that occur at least 5 times. We represent documents by *tf.idf* vectors with the Okapi TF formula (Zhai, 2001), which was also used in (Zhu et al., 2003a). Of the 1943 documents, we use 1600 as $L \cup U$ and the rest as unseen test data.

The mixture model. We use multinomial mixture models (bag-of-words naive Bayes model), treating *tf.idf* as 'pseudo word counts' of the documents. We found this works better than using the raw word counts. We use k-means to initialize the models.

The graph. We use a symmetrized 10NN weighted graph on the 1600 documents. The weight between documents u, v is $w_{uv} = \exp\left(-(1 - c_{uv})/0.03\right)$, where $c_{uv} = \langle u, v \rangle / \left(||u|| \cdot ||v||\right)$ is the cosine between the $tf.idf$ vectors u, v.

Sensitivity to M. The accuracy on U with different number of components M is shown in Figure 10.3(c). l is fixed at 10. Qualitatively the performance of harmonic mixtures increases when $M > 400$. From the plot it may look like the 'graph' curve varies with M, but this is an artifact as we used different randomly sampled L, U splits for different M. The error bars on harmonic mixtures are large. We suspect the particular mixture model is bad for the task.

Computational savings. Unlike the previous tasks, we need a much larger M around 600. We still have a smaller problem than the original $u = 1590$, but the saving is limited.

Handling unseen data. We fix $M = 600$ and vary labeled set size l. For each l we run 20 random trials. The classification accuracy on U (with 1600-l documents) and unseen data (343 documents) are listed in Table 10.3. The harmonic mixture model has lower accuracies than the harmonic function on the $L \cup U$ graph. The harmonic mixture model performs similarly on U and on unseen data.

10.5 Related Work

Recently Delalleau et al. (2005) use a small random subset of the unlabeled data to create a small graph. This is related to the Nyström method in spectral clustering

l	HM	EM	graph
		on U:	
2	98.7 ± 0.0	86.7 ± 5.7	98.7 ± 0.0
5	98.7 ± 0.0	90.1 ± 4.1	98.7 ± 0.1
10	98.7 ± 0.1	93.6 ± 2.4	98.7 ± 0.1
20	98.7 ± 0.2	96.0 ± 3.2	98.7 ± 0.2
30	98.7 ± 0.2	97.1 ± 1.9	98.8 ± 0.2
		on unseen:	
2	96.1 ± 0.1	87.1 ± 5.4	-
5	96.1 ± 0.1	89.8 ± 3.8	-
10	96.1 ± 0.1	93.2 ± 2.3	-
20	96.1 ± 0.1	95.1 ± 3.2	-
30	96.1 ± 0.1	96.8 ± 1.7	-

Table 10.2: Image classification 1 vs. 2: Accuracy on U and unseen data. $M = 100$. Each number is the mean and standard deviation of 20 trials.

l	HM	EM	graph
		on U:	
2	75.9 ± 14.3	54.5 ± 6.2	84.6 ± 10.9
5	74.5 ± 16.6	53.7 ± 5.2	87.9 ± 3.9
10	84.5 ± 2.1	55.7 ± 6.5	89.5 ± 1.0
20	83.3 ± 7.1	59.5 ± 6.4	90.1 ± 1.0
40	85.7 ± 2.3	61.8 ± 6.1	90.3 ± 0.6
		on unseen:	
2	73.6 ± 13.0	53.5 ± 6.0	-
5	73.2 ± 15.2	52.3 ± 5.9	-
10	82.9 ± 2.9	55.7 ± 5.7	-
20	82.0 ± 6.5	58.9 ± 6.1	-
40	84.7 ± 3.3	60.4 ± 5.9	-

Table 10.3: Text classification PC vs. Mac: Accuracy on U and unseen data. $M = 600$. Each number is the mean and standard deviation of 20 trials.

(Fowlkes et al., 2004), and to the random 'landmarks' in dimensionality reduction (Weinberger et al., 2005). Our method is different in that

- It incorporates a generative mixture model, which is a second knowledge source besides the graph;
- The backbone graph is not built on randomly selected points, but on meaningful mixture components;
- When classifying an unseen point x, it does not need graph edges from landmark points to x. This is less demanding on the graph because the burden is transferred to the mixture component models. For example one can now use kNN graphs. In the other works one needs edges between x and the landmarks, which are non-existent or awkward for kNN graphs.

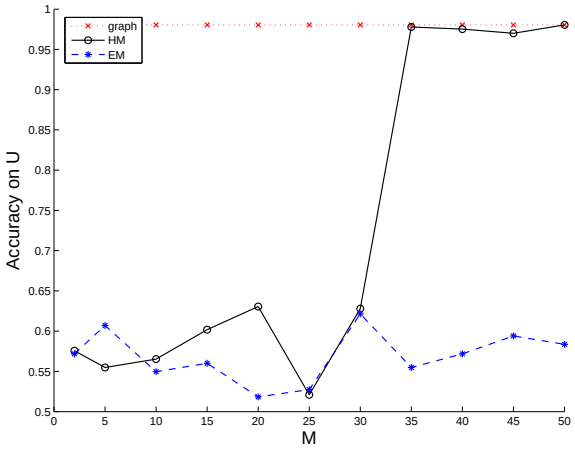
In terms of handling unseen data, our approach is closely related to the regularization framework of (Belkin et al., 2004b; Krishnapuram et al., 2005) as graph regularization on mixture models. However instead of a regularization term we used a discriminative term, which allows for the closed form solution in the special case.

10.6 Discussion

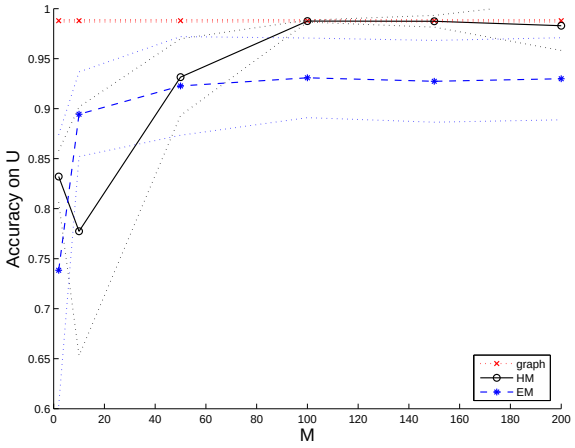
To summarize, the proposed harmonic mixture method reduces the graph problem size, and handles unseen test points. It achieves comparable accuracy as the harmonic function for semi-supervised learning.

There are several questions for further research. First, the component model affects the performance of the harmonic mixtures. For example the Gaussian in the synthetic task and 1 vs. 2 task seem to be more amenable to harmonic mixtures than the multinomial in PC vs. Mac task. How to quantify the influence remains a question. A second question is when $\alpha > 0$ is useful in practice. Finally, we want to find a way to automatically select the appropriate number of mixture components M.

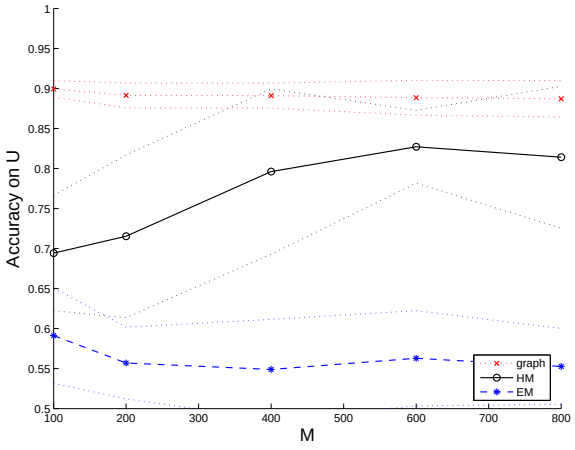
The backbone graph is certainly not the only way to speed up computation. We list some other methods in literature review in Chapter 11. In addition, we also performed an empirical study to compare several iterative methods, including Label Propagation, loopy belief propagation, and conjugate gradient, which all converge to the harmonic function. The study is presented in Appendix F.



(a) synthetic data



(b) 1 vs. 2



(c) PC vs. Mac

Figure 10.3: Sensitivity to M in three datasets. Shown are the classification accuracies on U as M changes. 'graph' is the harmonic function on the complete $L \cup U$ graph; 'HM' is the harmonic mixture, and 'EM' is the standard EM algorithm. The intervals are ± 1 standard deviation with 20 random trials when applicable.

Chapter 11

Literature Review

We review some of the literature on semi-supervised learning. There has been a whole spectrum of interesting ideas on how to learn from both labeled and unlabeled data. The review is by no means comprehensive and the field of semi-supervised learning is evolving rapidly. The author apologizes in advance for any inaccuracies in the descriptions, and welcomes corrections and comments. Please send corrections and suggest papers to zhuxj@cs.cmu.edu. To make the review more useful, we maintain an online version at
`http://www.cs.cmu.edu/~zhuxj/pub/semireview.html`
which will be updated indefinitely.

11.1 Q&A

Q: What is semi-supervised learning?
A: It's a special form of classification. Traditional classifiers need labeled data (feature / label pairs) to train. Labeled instances however are often difficult, expensive, or time consuming to obtain, as they require the efforts of experienced human annotators. Meanwhile unlabeled data may be relatively easy to collect, but there has been few ways to use them. Semi-supervised learning addresses this problem by using large amount of unlabeled data, together with the labeled data, to build better classifiers. Because semi-supervised learning requires less human effort and gives higher accuracy, it is of great interest both in theory and in practice.

Q: Can we really learn anything from unlabeled data? It looks like magic.
A: Yes we can – under certain assumptions. It's not magic, but good matching of problem structure with model assumption.

Q: Does unlabeled data always help?
A: No, there's no free lunch. Bad matching of problem structure with model assumption can lead to degradation in classifier performance. For example, quite a few semi-supervised learning methods assume that the decision boundary should avoid regions with high $p(x)$. These methods include transductive support vector machines (SVMs), information regularization, Gaussian processes with null category noise model, graph-based methods if the graph weights is determined by pairwise distance. Nonetheless if the data is generated from two heavily overlapping Gaussian, the decision boundary would go right through the densest region, and these methods would perform badly. On the other hand EM with generative mixture models, another semi-supervised learning method, would have easily solved the problem. Detecting bad match in advance however is hard and remains an open question.

Q: How many semi-supervised learning methods are there?
A: Many. Some often-used methods include: EM with generative mixture models, self-training, co-training, transductive support vector machines, and graph-based methods. See the following sections for more methods.

Q: Which method should I use / is the best?
A: There is no direct answer to this question. Because labeled data is scarce, semi-supervised learning methods make strong model assumptions. Ideally one should use a method whose assumptions fit the problem structure. This may be difficult in reality. Nonetheless we can try the following checklist: Do the classes produce well clustered data? If yes, EM with generative mixture models may be a good choice; Do the features naturally split into two sets? If yes, co-training may be appropriate; Is it true that two points with similar features tend to be in the same class? If yes, graph-based methods can be used; Already using SVM? Transductive SVM is a natural extension; Is the existing supervised classifier complicated and hard to modify? Self-training is a practical wrapper method.

Q: How do semi-supervised learning methods use unlabeled data?
A: Semi-supervised learning methods use unlabeled data to either modify or reprioritize hypotheses obtained from labeled data alone. Although not all methods are probabilistic, it is easier to look at methods that represent hypotheses by $p(y|x)$, and unlabeled data by $p(x)$. Generative models have common parameters for the joint distribution $p(x, y)$. It is easy to see that $p(x)$ influences $p(y|x)$. Mixture models with EM is in this category, and to some extent self-training. Many other methods are discriminative, including transductive SVM, Gaussian processes, information regularization, and graph-based methods. Original discriminative train-

ing cannot be used for semi-supervised learning, since $p(y|x)$ is estimated ignoring $p(x)$. To solve the problem, $p(x)$ dependent terms are often brought into the objective function, which amounts to assuming $p(y|x)$ and $p(x)$ share parameters.

Q: Where can I learn more?
A: An existing survey can be found in (Seeger, 2001).

11.2 Generative Mixture Models and EM

This is perhaps the oldest semi-supervised learning method. It assumes a generative model $p(x, y) = p(y)p(x|y)$ where $p(x|y)$ is an identifiable mixture distribution, for example Gaussian mixture models. With large amount of unlabeled data, the mixture components can be identified; then ideally we only need one labeled example per component to fully determine the mixture distribution. One can think of the mixture components as 'soft clusters'.

Nigam et al. (2000) apply the EM algorithm on mixture of multinomial for the task of text classification. They showed the resulting classifiers perform better than those trained only from L. Baluja (1998) uses the same algorithm on a face orientation discrimination task.

One has to pay attention to a few things:

11.2.1 Identifiability

The mixture model ideally should be identifiable. In general let $\{p_\theta\}$ be a family of distributions indexed by a parameter vector θ. θ is identifiable if $\theta_1 \neq \theta_2 \Rightarrow p_{\theta_1} \neq p_{\theta_2}$, up to a permutation of mixture components. If the model family is identifiable, in theory with infinite U one can learn θ up to a permutation of component indices.

Here is an example showing the problem with unidentifiable models. The model $p(x|y)$ is uniform for $y \in \{+1, -1\}$. Assuming with large amount of unlabeled data U we know $p(x)$ is uniform in $[0, 1]$. We also have 2 labeled data points $(0.1, +1), (0.9, -1)$. Can we determine the label for $x = 0.5$? No. With our assumptions we cannot distinguish the following two models:

$$p(y = 1) = 0.2,\ p(x|y = 1) = \text{unif}(0, 0.2),\ p(x|y = -1) = \text{unif}(0.2, 1) \quad (11.1)$$

$$p(y = 1) = 0.6,\ p(x|y = 1) = \text{unif}(0, 0.6),\ p(x|y = -1) = \text{unif}(0.6, 1) \quad (11.2)$$

which give opposite labels at $x = 0.5$, see Figure 11.1. It is known that a mixture of Gaussian is identifiable. Mixture of multivariate Bernoulli (McCallum & Nigam, 1998a) is not identifiable. More discussions on identifiability and semi-supervised learning can be found in e.g. (Ratsaby & Venkatesh, 1995) and (Corduneanu & Jaakkola, 2001).

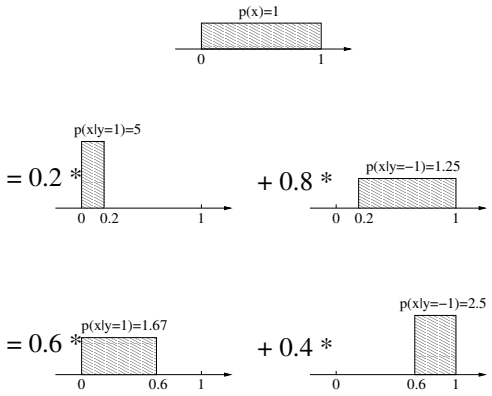
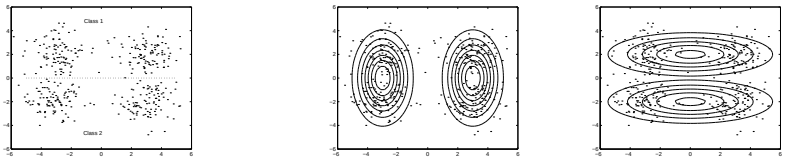


Figure 11.1: An example of unidentifiable models. Even if we known $p(x)$ (top) is a mixture of two uniform distributions, we cannot uniquely identify the two components. For instance, the mixtures on the second and third line give the same $p(x)$, but they classify $x = 0.5$ differently.



(a) Horizontal class separation (b) High probability (c) Low probability

Figure 11.2: If the model is wrong, higher likelihood may lead to lower classification accuracy. For example, **(a)** is clearly not generated from two Gaussian. If we insist that each class is a single Gaussian, **(b)** will have higher probability than **(c)**. But **(b)** has around 50% accuracy, while **(c)**'s is much better.

11.2.2 Model Correctness

If the mixture model assumption is correct, unlabeled data is guaranteed to improve accuracy (Castelli & Cover, 1995) (Castelli & Cover, 1996) (Ratsaby & Venkatesh, 1995). However if the model is wrong, unlabeled data may actually hurt accuracy. Figure 11.2 shows an example. This has been observed by multiple researchers. Cozman et al. (2003) give a formal derivation on how this might happen.

It is thus important to carefully construct the mixture model to reflect reality. For example in text categorization a topic may contain several sub-topics, and will be better modeled by multiple multinomial instead of a single one (Nigam et al., 2000). Some other examples are (Shahshahani & Landgrebe, 1994) (Miller & Uyar, 1997). Another solution is to down-weighing unlabeled data (Corduneanu &

Jaakkola, 2001), which is also used by Nigam et al. (2000), and by Callison-Burch et al. (2004) who estimate word alignment for machine translation.

11.2.3 EM Local Maxima

Even if the mixture model assumption is correct, in practice mixture components are identified by the Expectation-Maximization (EM) algorithm (Dempster et al., 1977). EM is prone to local maxima. If a local maximum is far from the global maximum, unlabeled data may again hurt learning. Remedies include smart choice of starting point by active learning (Nigam, 2001).

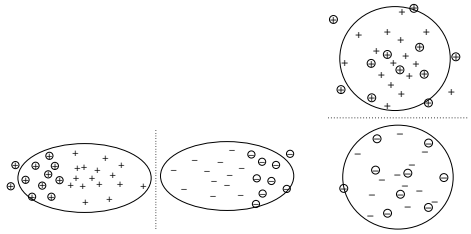
11.2.4 Cluster and Label

We shall also mention that instead of using an probabilistic generative mixture model, some approaches employ various clustering algorithms to cluster the whole dataset, then label each cluster with labeled data, e.g. (Demiriz et al., 1999) (Dara et al., 2000). Although they may perform well if the particular clustering algorithms match the true data distribution, these approaches are hard to analyze due to their algorithmic nature.

11.3 Self-Training

Self-training is a commonly used technique for semi-supervised learning. In self-training a classifier is first trained with the small amount of labeled data. The classifier is then used to classify the unlabeled data. Typically the most confident unlabeled points, together with their predicted labels, are added to the training set. The classifier is re-trained and the procedure repeated. Note the classifier uses its own predictions to teach itself. The procedure is also called self-teaching or bootstrapping (not to be confused with the statistical procedure with the same name). The generative model and EM approach of section 11.2 can be viewed as a special case of 'soft' self-training. One can imagine that a classification mistake can reinforce itself. Some algorithms try to avoid this by 'unlearn' unlabeled points if the prediction confidence drops below a threshold.

Self-training has been applied to several natural language processing tasks. Yarowsky (1995) uses self-training for word sense disambiguation, e.g. deciding whether the word 'plant' means a living organism or a factory in a give context. Riloff et al. (2003) uses it to identify subjective nouns. Maeireizo et al. (2004) classify dialogues as 'emotional' or 'non-emotional' with a procedure involving two classifiers.Self-training has also been applied to parsing and machine translation. Rosenberg et al. (2005) apply self-training to object detection systems from



(a) x^1 view (b) x^2 view

Figure 11.3: Co-Training: Conditional independent assumption on feature split. With this assumption the high confident data points in x^1 view, represented by circled labels, will be randomly scattered in x^2 view. This is advantageous if they are to be used to teach the classifier in x^2 view.

images, and show the semi-supervised technique compares favorably with a state-of-the-art detector.

11.4 Co-Training

Co-training (Blum & Mitchell, 1998) (Mitchell, 1999) assumes that features can be split into two sets; Each sub-feature set is sufficient to train a good classifier; The two sets are conditionally independent given the class. Initially two separate classifiers are trained with the labeled data, on the two sub-feature sets respectively. Each classifier then classifies the unlabeled data, and 'teaches' the other classifier with the few unlabeled examples (and the predicted labels) they feel most confident. Each classifier is retrained with the additional training examples given by the other classifier, and the process repeats.

In co-training, unlabeled data helps by reducing the version space size. In other words, the two classifiers (or hypotheses) must agree on the much larger unlabeled data as well as the labeled data.

We need the assumption that sub-features are sufficiently good, so that we can trust the labels by each learner on U. We need the sub-features to be conditionally independent so that one classifier's high confident data points are *iid* samples for the other classifier. Figure 11.3 visualizes the assumption.

Nigam and Ghani (2000) perform extensive empirical experiments to compare co-training with generative mixture models and EM. Their result shows co-training performs well if the conditional independence assumption indeed holds. In addition, it is better to probabilistically label the entire U, instead of a few most confident data points. They name this paradigm co-EM. Finally, if there is no natural feature split, the authors create artificial split by randomly break the feature set into

two subsets. They show co-training with artificial feature split still helps, though not as much as before. Jones (2005) used co-training, co-EM and other related methods for information extraction from text.

Co-training makes strong assumptions on the splitting of features. One might wonder if these conditions can be relaxed. Goldman and Zhou (2000) use two learners of different type but both takes the whole feature set, and essentially use one learner's high confidence data points, identified with a set of statistical tests, in U to teach the other learning and vice versa. Recently Balcan et al. (2005) relax the conditional independence assumption with a much weaker expansion condition, and justify the iterative co-training procedure.

11.5 Maximizing Separation

11.5.1 Transductive SVM

Discriminative methods work on $p(y|x)$ directly. This brings up the danger of leaving $p(x)$ outside of the parameter estimation loop, if $p(x)$ and $p(y|x)$ do not share parameters. Notice $p(x)$ is usually all we can get from unlabeled data. It is believed that if $p(x)$ and $p(y|x)$ do not share parameters, semi-supervised learning cannot help. This point is emphasized in (Seeger, 2001). Zhang and Oles (2000) give both theoretical and experimental evidence of the same point specifically on transductive support vector machines (TSVM). However this is controversial as empirically TSVMs seem beneficial.

TSVM is an extension of standard support vector machines with unlabeled data. In a standard SVM only the labeled data is used, and the goal is to find a maximum margin linear boundary in the Reproducing Kernel Hilbert Space. In a TSVM the unlabeled data is also used. The goal is to find a labeling of the unlabeled data, so that a linear boundary has the maximum margin on both the original labeled data and the (now labeled) unlabeled data. The decision boundary has the smallest generalization error bound on unlabeled data (Vapnik, 1998). Intuitively, unlabeled data guides the linear boundary away from dense regions. However finding the exact transductive SVM solution is NP-hard. Several approximation algorithms have been proposed and show positive results, see e.g. (Joachims, 1999) (Bennett & Demiriz, 1999) (Demirez & Bennettt, 2000) (Fung & Mangasarian, 1999) (Chapelle & Zien, 2005).

The maximum entropy discrimination approach (Jaakkola et al., 1999) also maximizes the margin, and is able to take into account unlabeled data, with SVM as a special case.

The application of graph kernels (Zhu et al., 2005) to SVMs differs from TSVM. The graph kernels are special semi-supervised kernels applied to a stan-

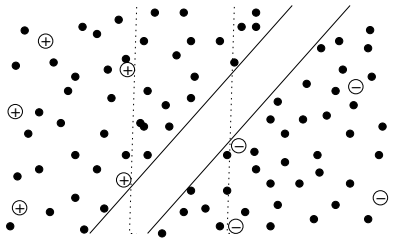


Figure 11.4: In TSVM, U helps to put the decision boundary in sparse regions. With labeled data only, the maximum margin boundary is plotted with dotted lines. With unlabeled data (black dots), the maximum margin boundary would be the one with solid lines.

dard SVM; TSVM is a special optimization criterion regardless of the kernel being used.

11.5.2 Gaussian Processes

Lawrence and Jordan (2005) proposed a Gaussian process approach, which can be viewed as the Gaussian process parallel of TSVM. The key difference to a standard Gaussian process is in the noise model. A 'null category noise model' maps the hidden continuous variable f to three instead of two labels, specifically to the never used label '0' when f is around zero. On top of that, it is restricted that unlabeled data points cannot take the label 0. This pushes the posterior of f away from zero for the unlabeled points. It achieves the similar effect of TSVM where the margin avoids dense unlabeled data region. However nothing special is done on the process model. Therefore all the benefit of unlabeled data comes from the noise model. A very similar noise model is proposed in (Chu & Ghahramani, 2004) for ordinal regression.

This is different from the Gaussian processes in (Zhu et al., 2003c), where we have a semi-supervised Gram matrix, and semi-supervised learning originates from the process model, not the noise model.

11.5.3 Information Regularization

Szummer and Jaakkola (2002) propose the information regularization framework to control the label conditionals $p(y|x)$ by $p(x)$, where $p(x)$ may be estimated from unlabeled data. The idea is that labels shouldn't change too much in regions where $p(x)$ is high. The authors use the mutual information $I(x; y)$ between x and y as a measure of label complexity. $I(x; y)$ is small when the labels are homogeneous,

and large when labels vary. This motives the minimization of the product of $p(x)$ mass in a region with $I(x; y)$ (normalized by a variance term). The minimization is carried out on multiple overlapping regions covering the data space.

The theory is developed further in (Corduneanu & Jaakkola, 2003). Corduneanu and Jaakkola (2005) extend the work by formulating semi-supervised learning as a communication problem. Regularization is expressed as the rate of information, which again discourages complex conditionals $p(y|x)$ in regions with high $p(x)$. The problem becomes finding the unique $p(y|x)$ that minimizes a regularized loss on labeled data. The authors give a local propagation algorithm.

11.5.4 Entropy Minimization

The hyperparameter learning method in section 7.2 uses entropy minimization. Grandvalet and Bengio (2005) used the label entropy on unlabeled data as a regularizer. By minimizing the entropy, the method assumes a prior which prefers minimal class overlap.

11.6 Graph-Based Methods

Graph-based semi-supervised methods define a graph where the nodes are labeled and unlabeled examples in the dataset, and edges (may be weighted) reflect the similarity of examples. These methods usually assume label smoothness over the graph. Graph methods are nonparametric, discriminative, and transductive in nature. This thesis largely focuses on graph-based semi-supervised learning algorithms.

11.6.1 Regularization by Graph

Many graph-based methods can be viewed as estimating a function f on the graph. One wants f to satisfy two things at the same time: 1) it should be close to the given labels y_L on the labeled nodes, and 2) it should be smooth on the whole graph. This can be expressed in a regularization framework where the first term is a loss function, and the second term is a regularizer.

Several graph-based methods listed here are similar to each other. They differ in the particular choice of the loss function and the regularizer. Are these differences crucial? Probably not. We believe it is much more important to construct a good graph than to choose among the methods. However graph construction, as we will see later, is not a well studied area.

Mincut

Blum and Chawla (2001) pose semi-supervised learning as a graph mincut (also known as st-cut) problem. In the binary case, positive labels act as sources and negative labels act as sinks. The objective is to find a minimum set of edges whose removal blocks all flow from the sources to the sinks. The nodes connecting to the sources are then labeled positive, and those to the sinks are labeled negative. Equivalently mincut is the *mode* of a Markov random field with binary labels (Boltzmann machine). The loss function can be viewed as a quadratic loss with infinity weight: $\infty \sum_{i \in L} (y_i - y_{i|L})^2$, so that the values on labeled data are in fact clamped. The labeling y minimizes

$$\frac{1}{2} \sum_{i,j} w_{ij} |y_i - y_j| = \frac{1}{2} \sum_{i,j} w_{ij} (y_i - y_j)^2 \tag{11.3}$$

which can be thought of as a regularizer on binary (0 and 1) labels.

One problem with mincut is that it only gives hard classification without confidence. Blum et al. (2004) perturb the graph by adding random noise to the edge weights. Mincut is applied to multiple perturbed graphs, and the labels are determined by a majority vote. The procedure is similar to bagging, and creates a 'soft' mincut.

Pang and Lee (2004) use mincut to improve the classification of a sentence into either 'objective' or 'subjective', with the assumption that sentences close to each other tend to have the same class.

Gaussian Random Fields and Harmonic Functions

The Gaussian random fields and harmonic function methods in (Zhu et al., 2003a) can be viewed as having a quadratic loss function with infinity weight, so that the labeled data are clamped, and a regularizer based on the graph combinatorial Laplacian Δ:

$$\infty \sum_{i \in L} (f_i - y_i)^2 + 1/2 \sum_{i,j} w_{ij} (f_i - f_j)^2 \tag{11.4}$$

$$= \infty \sum_{i \in L} (f_i - y_i)^2 + f^\top \Delta f \tag{11.5}$$

Recently Grady and Funka-Lea (2004) applied the harmonic function method to medical image segmentation tasks, where a user labels classes (e.g. different organs) with a few strokes. Levin et al. (2004) use essentially harmonic functions for colorization of gray-scale images. Again the user specifies the desired color with

only a few strokes on the image. The rest of the image is used as unlabeled data, and the labels propagation through the image. Niu et al. (2005) applied the label propagation algorithm (which is equivalent to harmonic functions) to word sense disambiguation.

Local and Global Consistency

The local and global consistency method (Zhou et al., 2004a) uses the loss function $\sum_{i=1}^{n}(f_i - y_i)^2$, and the *normalized Laplacian* $D^{-1/2}\Delta D^{-1/2} = I - D^{-1/2}WD^{-1/2}$ in the regularizer,

$$1/2\sum_{i,j} w_{ij}(f_i/\sqrt{D_{ii}} - f_j/\sqrt{D_{jj}})^2 \quad = \quad f^\top D^{-1/2}\Delta D^{-1/2} f \qquad (11.6)$$

Tikhonov Regularization

The Tikhonov regularization algorithm in (Belkin et al., 2004a) uses the loss function and regularizer:

$$1/k\sum_{i}(f_i - y_i)^2 + \gamma f^\top S f \qquad (11.7)$$

where $S = \Delta$ or Δ^p for some integer p.

Graph Kernels

For kernel methods, the regularizer is a (typically monotonically increasing) function of the RKHS norm $||f||_K = f^\top K^{-1} f$ with kernel K. Such kernels are derived from the graph, e.g. the Laplacian.

Chapelle et al. (2002) and Smola and Kondor (2003) both show the spectral transformation of a Laplacian results in kernels suitable for semi-supervised learning. The diffusion kernel (Kondor & Lafferty, 2002) corresponds to a spectrum transform of the Laplacian with

$$r(\lambda) = \exp(-\frac{\sigma^2}{2}\lambda) \qquad (11.8)$$

The regularized Gaussian process kernel $\Delta + I/\sigma^2$ in (Zhu et al., 2003c) corresponds to

$$r(\lambda) = \frac{1}{\lambda + \sigma} \qquad (11.9)$$

Similarly the order constrained graph kernels in (Zhu et al., 2005) are constructed from the spectrum of the Laplacian, with non-parametric convex optimization. Learning the optimal eigenvalues for a graph kernel is in fact a way to

(at least partially) correct an imprecise graph. In this sense it is related to graph construction.

Spectral Graph Transducer

The spectral graph transducer (Joachims, 2003) can be viewed with a loss function and regularizer

$$c(f-\gamma)^\top C(f-\gamma) + f^\top L f \tag{11.10}$$

where $\gamma_i = \sqrt{l_-/l_+}$ for positive labeled data, $-\sqrt{l_+/l_-}$ for negative data, l_- being the number of negative data and so on. L can be the combinatorial or normalized graph Laplacian, with a transformed spectrum.

Tree-Based Bayes

Kemp et al. (2003) define a probabilistic distribution $P(Y|T)$ on discrete (e.g. 0 and 1) labelings Y over an evolutionary tree T. The tree T is constructed with the labeled and unlabeled data being the leaf nodes. The labeled data is clamped. The authors assume a mutation process, where a label at the root propagates down to the leaves. The label mutates with a constant rate as it moves down along the edges. As a result the tree T (its structure and edge lengths) uniquely defines the label prior $P(Y|T)$. Under the prior if two leaf nodes are closer in the tree, they have a higher probability of sharing the same label. One can also integrate over all tree structures.

The tree-based Bayes approach can be viewed as an interesting way to incorporate structure of the domain. Notice the leaf nodes of the tree are the labeled and unlabeled data, while the internal nodes do not correspond to physical data. This is in contrast with other graph-based methods where labeled and unlabeled data are all the nodes.

Some Other Methods

Szummer and Jaakkola (2001) perform a t-step Markov random walk on the graph. The influence of one example to another example is proportional to how easy the random walk goes from one to the other. It has certain resemblance to the diffusion kernel. The parameter t is important.

Chapelle and Zien (2005) use a density-sensitive connectivity distance between nodes i, j (a given path between i, j consists of several segments, one of them is the longest; now consider all paths between i, j and find the shortest 'longest segment'). Exponentiating the negative distance gives a graph kernel.

Bousquet et al. (2004) consider the continuous counterpart of graph-based regularization. They define regularization based on a known $p(x)$ and provide interesting theoretical analysis. However there seem to be problems in applying the theoretical results to higher ($D > 2$) dimensional tasks.

11.6.2 Graph Construction

Although the graph is the heart and soul of graph-based semi-supervised learning methods, its construction has not been studied carefully. The issue has been discussed informally in Chapter 3, and graph hyperparameter learning discussed in Chapter 7. There are relatively few literatures on graph construction. For example Carreira-Perpinan and Zemel (2005) build robust graphs from multiple minimum spanning trees by perturbation and edge removal. It is possible that graph construction is domain specific because it encodes prior knowledge, and has thus far been treated on an individual basis.

11.6.3 Induction

Most graph-based semi-supervised learning algorithms are transductive, i.e. they cannot easily extend to new test points outside of $L \cup U$. Recently induction has received increasing attention. One common practice is to 'freeze' the graph on $L \cup U$. New points do not (although they should) alter the graph structure. This avoids expensive graph computation every time one encounters new points.

Zhu et al. (2003c) propose that new test point be classified by its nearest neighbor in $L \cup U$. This is sensible when U is sufficiently large. In (Chapelle et al., 2002) the authors approximate a new point by a linear combination of labeled and unlabeled points. Similarly in (Delalleau et al., 2005) the authors proposes an induction scheme to classify a new point x by

$$f(x) = \frac{\sum_{i \in L \cup U} w_{xi} f(x_i)}{\sum_{i \in L \cup U} w_{xi}} \tag{11.11}$$

This can be viewed as an application of the Nyström method (Fowlkes et al., 2004).

In the regularization framework of (Belkin et al., 2004b), the function f does not have to be restricted to the graph. The graph is merely used to regularize f which can have a much larger support. It is necessarily a combination of an inductive algorithm and graph regularization. The authors give the graph-regularized version of least squares and SVM. Note such an SVM is different from the graph kernels in standard SVM in (Zhu et al., 2005). The former is inductive with both a graph regularizer and an inductive kernel. The latter is transductive with only the graph regularizer. Following the work, Krishnapuram et al. (2005) use graph

regularization on logistic regression. These methods create inductive learners that naturally handle new test points.

The harmonic mixture model in Chapter 10 naturally handles new points with the help of a mixture model.

11.6.4 Consistency

The consistency of graph-based semi-supervised learning algorithms has not been studied extensively according to the author's knowledge. By consistency we mean whether the classification converges to the right solution as the number of labeled and unlabeled data grows to infinity. Recently von Luxburg et al. (2005) (von Luxburg et al., 2004) study the consistency of spectral clustering methods. The authors find that the normalized Laplacian is better than the unnormalized Laplacian for spectral clustering. The convergence of the eigenvectors of the unnormalized Laplacian is not clear, while the normalized Laplacian always converges under general conditions. There are examples where the top eigenvectors of the unnormalized Laplacian do not yield a sensible clustering. Although these are valuable results, we feel the parallel problems in semi-supervised learning needs further study. One reason is that in semi-supervised learning the whole Laplacian (normalized or not) is often used for regularization, not only the top eigenvectors.

11.6.5 Ranking

Given a large collection of items, and a few 'query' items, ranking orders the items according to their similarity to the queries. It can be formulated as semi-supervised learning with positive data only (Zhou et al., 2004b), with the graph induced similarity measure.

11.6.6 Directed Graphs

Zhou et al. (2005) take a hub/authority approach, and essentially convert a directed graph into an undirected one. Two hub nodes are connected by an undirected edge with appropriate weight if they co-link to authority nodes, and vice versa. Semi-supervised learning then proceeds on the undirected graph.

Lu and Getoor (2003) convert the link structure in a directed graph into per-node features, and combines them with per-node object features in logistic regression. They also use an EM-like iterative algorithm.

11.6.7 Fast Computation

Fast computation with sparse graphs and iterative methods has been briefly discussed in Chapter 10. Recently numerical methods for fast N-body problems have been applied to *dense* graphs in semi-supervised learning, reducing the computational cost from $O(n^3)$ to $O(n)$ (Mahdaviani et al., 2005). This is achieved with Krylov subspace methods and the fast Gauss transform.

11.7 Metric-Based Model Selection

Metric-based model selection (Schuurmans & Southey, 2001) is a method to detect hypotheses inconsistency with unlabeled data. We may have two hypotheses which are consistent on L, for example they all have zero training set error. However they may be inconsistent on the much larger U. If so we should reject at least one of them, e.g. the more complex one if we employ Occam's razor.

The key observation is that a distance metric is defined in the hypothesis space H. One such metric is the number of different classifications two hypotheses make under the data distribution $p(x)$: $d_p(h_1, h_2) = E_p[h_1(x) \neq h_2(x)]$. It is easy to verify that the metric satisfies the three metric properties. Now consider the true classification function h^* and two hypotheses h_1, h_2. Since the metric satisfies the triangle inequality (the third property), we have

$$d_p(h_1, h_2) \leq d_p(h_1, h^*) + d_p(h^*, h_2)$$

Under the premise that labels in L is noiseless, let's assume we can approximate $d_p(h_1, h^*)$ and $d_p(h^*, h_2)$ by h_1 and h_2's training set error rates $d_L(h_1, h^*)$ and $d_L(h_2, h^*)$, and approximate $d_p(h_1, h_2)$ by the difference h_1 and h_2 make on a large amount of unlabeled data U: $d_U(h_1, h_2)$. We get

$$d_U(h_1, h_2) \leq d_L(h_1, h^*) + d_L(h^*, h_2)$$

which can be verified directly. If the inequality does not hold, at least one of the assumptions is wrong. If $|U|$ is large enough and $U \overset{\text{iid}}{\sim} p(x)$, $d_U(h_1, h_2)$ will be a good estimate of $d_p(h_1, h_2)$. This leaves us with the conclusion that at least one of the training errors does not reflect its true error. If both training errors are close to zero, we would know that at least one model is overfitting. An Occam's razor type of argument then can be used to select the model with less complexity. Such use of unlabeled data is very general and can be applied to almost any learning algorithms. However it only selects among hypotheses; it does not generate new hypothesis based on unlabeled data.

The co-validation method (Madani et al., 2005) also uses unlabeled data for model selection and active learning.

11.8 Related Areas

The focus of the thesis is on classification with semi-supervised methods. There are some closely related areas with a rich literature.

11.8.1 Spectral Clustering

Spectral clustering is unsupervised. As such there is no labeled data to guide the process. Instead the clustering depends solely on the graph weights W. On the other hand semi-supervised learning for classification has to maintain a balance between how good the 'clustering' is, and how well the labeled data can be explained by it. Such balance is expressed explicitly in the regularization framework.

As we have seen in section 8.1 and 11.6.4, the top eigenvectors of the graph Laplacian can unfold the data manifold to form meaningful clusters. This is the intuition behind spectral clustering. There are several criteria on what constitutes a good clustering (Weiss, 1999).

The normalized cut (Shi & Malik, 2000) seeks to minimize

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)} \tag{11.12}$$

The *continuous relaxation* of the cluster indicator vector can be derived from the normalized Laplacian. In fact it is derived from the second smallest eigenvector of the normalized Laplacian. The continuous vector is then discretized to obtain the clusters.

The data points are mapped into a new space spanned by the first k eigenvectors of the normalized Laplacian in (Ng et al., 2001a), with special normalization. Clustering is then performed with traditional methods (like k-means) in this new space. This is very similar to kernel PCA.

Fowlkes et al. (2004) use the Nyström method to reduce the computation cost for large spectral clustering problems. This is related to our method in Chapter 10.

Chung (1997) presents the mathematical details of spectral graph theory.

11.8.2 Clustering with Side Information

This is the 'opposite' of semi-supervised classification. The goal is clustering but there are some 'labeled data' in the form of *must-links* (two points must in the same cluster) and *cannot-links* (two points cannot in the same cluster). There is a tension between satisfying these constraints and optimizing the original clustering criterion (e.g. minimizing the sum of squared distances within clusters). Procedurally one can modify the distance metric to try to accommodate the constraints, or one can

bias the search. We refer readers to a recent short survey (Grira et al., 2004) for the literatures.

11.8.3 Nonlinear Dimensionality Reduction

The goal of nonlinear dimensionality reduction is to find a faithful low dimensional mapping of the high dimensional data. As such it belongs to unsupervised learning. However the way it discovers low dimensional manifold within a high dimensional space is closely related to spectral graph semi-supervised learning. Representative methods include Isomap (Tenenbaum et al., 2000), locally linear embedding (LLE) (Roweis & Saul, 2000) (Saul & Roweis, 2003), Hessian LLE (Donoho & Grimes, 2003), Laplacian eigenmaps (Belkin & Niyogi, 2003), and semidefinite embedding (SDE) (Weinberger & Saul, 2004) (Weinberger et al., 2004) (Weinberger et al., 2005).

11.8.4 Learning a Distance Metric

Many learning algorithms depend, either explicitly or implicitly, on a distance metric on X. We use the term metric here loosely to mean a measure of distance or (dis)similarity between two data points. The default distance in the feature space may not be optimal, especially when the data forms a lower dimensional manifold in the feature vector space. With a large amount of U, it is possible to detect such manifold structure and its associated metric. The graph-based methods above are based on this principle. We review some other methods next.

The simplest example in text classification might be Latent Semantic Indexing (LSI, a.k.a. Latent Semantic Analysis LSA, Principal Component Analysis PCA, or sometimes Singular Value Decomposition SVD). This technique defines a linear subspace, such that the variance of the data, when projected to the subspace, is maximumly preserved. LSI is widely used in text classification, where the original space for X is usually tens of thousands dimensional, while people believe meaningful text documents reside in a much lower dimensional space. Zelikovitz and Hirsh (2001) and Cristianini et al. (2001b) both use U, in this case unlabeled documents, to augment the term-by-document matrix of L. LSI is performed on the augmented matrix. This representation induces a new distance metric. By the property of LSI, words that co-occur very often in the same documents are merged into a single dimension of the new space. In the extreme this allows two documents with no common words to be 'close' to each other, via chains of co-occur word pairs in other documents.

Probabilistic Latent Semantic Analysis (PLSA) (Hofmann, 1999) is an important improvement over LSI. Each word in a document is generated by a 'topic' (a

multinomial, i.e. unigram). Different words in the document may be generated by different topics. Each document in turn has a fixed topic proportion (a multinomial on a higher level). However there is no link between the topic proportions in different documents.

Latent Dirichlet Allocation (LDA) (Blei et al., 2003) is one step further. It assumes the topic proportion of each document is drawn from a Dirichlet distribution. With variational approximation, each document is represented by a posterior Dirichlet over the topics. This is a much lower dimensional representation.

Some algorithms derive a metric entirely from the density of U. These are motivated by unsupervised clustering and based on the intuition that data points in the same high density 'clump' should be close in the new metric. For instance, if U is generated from a single Gaussian, then the Mahalanobis distance induced by the covariance matrix is such a metric. Tipping (1999) generalizes the Mahalanobis distance by fitting U with a mixture of Gaussian, and define a Riemannian manifold with metric at x being the weighted average of individual component inverse covariance. The distance between x_1 and x_2 is computed along the straight line (in Euclidean space) between the two points. Rattray (2000) further generalizes the metric so that it only depends on the change in log probabilities of the density, not on a particular Gaussian mixture assumption. And the distance is computed along a curve that minimizes the distance. The new metric is invariate to linear transformation of the features, and connected regions of relatively homogeneous density in U will be close to each other. Such metric is attractive, yet it depends on the homogeneity of the initial Euclidean space. Their application in semi-supervised learning needs further investigation.

We caution the reader that the metrics proposed above are based on unsupervised techniques. They all identify a lower dimensional manifold within which the data reside. However the data manifold may or may not correlate with a particular classification task. For example, in LSI the new metric emphasizes words with prominent count variances, but ignores words with small variances. If the classification task is subtle and depends on a few words with small counts, LSI might wipe out the salient words all together. Therefore the success of these methods is hard to guarantee without putting some restrictions on the kind of classification tasks. It would be interesting to include L into the metric learning process.

In a separate line of work, Baxter (1997) proves that there is a unique optimal metric for classification if we use 1-nearest-neighbor. The metric, named Canonical Distortion Measure (CDM), defines a distance $d(x_1, x_2)$ as the expected loss if we classify x_1 with x_2's label. The distance measure proposed in (Yianilos, 1995) can be viewed as a special case. Yianilos assume a Gaussian mixture model has been learned from U, such that a class correspond to a component, but the correspondence is unknown. In this case CDM $d(x_1, x_2) = p(x_1, x_2 \text{from same component})$

and can be computed analytically. Now that a metric has been learned from U, we can find within L the 1-nearest-neighbor of a new data point x, and classify x with the nearest neighbor's label. It will be interesting to compare this scheme with EM based semi-supervised learning, where L is used to label mixture components.

Weston et al. (2004) propose the neighborhood mismatch kernel and the bagged mismatch kernel. More precisely both are *kernel transformation* that modifies an input kernel. In the neighborhood method, one defines the neighborhood of a point as points close enough according to certain similarity measure (note this is *not* the measure induced by the input kernel). The output kernel between point i, j is the average of pairwise kernel entries between i's neighbors and j's neighbors. In bagged method, if a clustering algorithm thinks they tend to be in the same cluster (note again this is a different measure than the input kernel), the corresponding entry in the input kernel is boosted.

11.8.5 Inferring Label Sampling Mechanisms

Most semi-supervised learning methods assume L and U are both *i.i.d.* from the underlying distribution. However as (Rosset et al., 2005) points out that is not always the case. For example y can be the binary label whether a customer is satisfied, obtained through a survey. It is conceivable survey participation (and thus labeled data) depends on the satisfaction y.

Let s_i be the binary missing indicator for y_i. The authors model $p(s|x, y)$ with a parametric family. The goal is to estimate $p(s|x, y)$ which is the label sampling mechanism. This is done by computing the expectation of an arbitrary function $g(x)$ in two ways: on $L \cup U$ as $1/n \sum_{i=1}^{n} g(x_i)$, and on L only as $1/n \sum_{i \in L} g(x_i)/p(s_i = 1|x_i, y_i)$. By equating the two $p(s|x, y)$ can be estimated. The intuition is that the expectation on L requires weighting the labeled samples inversely proportional to the labeling probability, to compensate for ignoring the unlabeled data.

Chapter 12

Discussions

We have presented a series of semi-supervised learning algorithms, based on a graph representation of the data. Experiments show that they are able to take advantage of the unlabeled data to improve classification. Contributions of the thesis include:

- We proposed a harmonic function and Gaussian field formulations for semi-supervised problems. This is not the first graph-based semi-supervised method. The first one was graph mincut. However our formulation is a continuous relaxation to the discrete labels, resulting in a more benign problem. Several variations of the formulation were proposed independently by different groups shortly after.
- We addressed the problem of graph construction, by setting up parametric edge weights and performing edge hyperparameter learning. Since the graph is the input to all graph-based semi-supervised algorithms, it is important that we construct graphs that best suit the task.
- We combined an active learning scheme that reduces expected error instead of ambiguity, with graph-based semi-supervised learning. We believe that active learning and semi-supervised learning will be used together for practical problems, because limited human annotation resources should be spent wisely.
- We defined optimal semi-supervised kernels by spectral transformation of the graph Laplacian. Such optimal kernels can be found with convex optimization. We can use the kernels with any kernel machine, e.g. support vector machines, for semi-supervised learning. The kernel machines in general can handle noisy labeled data, which is an improvement over the harmonic function solution.

- We kernelized conditional random fields. CRFs were traditionally feature based. We derived the dual problem and presented an algorithm for fast sparse kernel CRF training. With kernel CRFs, it is possible to use a semi-supervised kernel on instances for semi-supervised learning on sequences and other structures.

- We proposed to solve large-scale problems with harmonic mixtures. Harmonic mixtures reduce computation cost significantly by grouping unlabeled data into soft clusters, then carrying out semi-supervised learning on the coarser data representation. Harmonic mixtures also handle new data points naturally, making the semi-supervised learning method inductive.

Semi-supervised learning is a relatively new research area. There are many open questions and research opportunities:

- The graph is the single most important quantity for graph-based semi-supervised learning. Parameterizing graph edge weights, and learning weight hyperparameters, should be the first step of any graph-based semi-supervised learning methods. Current methods in Chapter 7 are not efficient enough. Can we find better ways to learn the graph structure and parameters?

- Real problems can have millions of unlabeled data points. Anecdotal stories and experiments in Appendix F indicate that conjugate gradient with a suitable pre-conditioner is one of the fastest algorithms in solving harmonic functions. Harmonic mixture works along an orthogonal direction by reducing the problem size. How large a dataset can we process if we combine conjugate gradient and harmonic mixture? What can we do to handle even larger datasets?

- Semi-supervised learning on structured data, e.g. sequences and trees, is largely unexplored. We have proposed the use of kernel conditional random fields plus semi-supervised kernels. Much more work is needed in this direction.

- In this thesis we focused on classification problems. The spirit of combining some human effort with large amount of data should be applicable to other problems. Examples include: regression with both labeled and unlabeled data; ranking with ordered pairs and unlabeled data; clustering with cluster membership knowledge. What can we do beyond classification?

- Because labeled data is scarce, semi-supervised learning methods depend more heavily on their assumptions (see e.g. Table 1.1). Can we develop novel semi-supervised learning algorithms with new assumptions?

- Applications of semi-supervised learning are emerging rapidly. These include text categorization, natural language processing, bioinformatics, image processing, and computer vision. Many others are sure to come. Applications are attractive because they solve important practical problems, and provide fertile test bed for new ideas in machine learning. What problems can we apply semi-supervised learning? What applications were too hard but are now feasible with semi-supervised learning?

- The theory of semi-supervised learning is almost absent in both the machine learning literature and the statistics literature. Is graph-based semi-supervised learning consistent? How many labeled and unlabeled points are needed to learn a concept with confidence?

We expect advances in research will address these questions. We hope semi-supervised learning become a fruitful area for both machine learning theory and practical applications.

Appendix A

The Harmonic Function after Knowing One More Label

Construct the graph as usual. We use f to denote the harmonic function. The random walk solution is $f_u = -\Delta_{uu}^{-1}\Delta_{ul}f_l = \Delta_{uu}^{-1}W_{ul}f_l$. There are u unlabeled nodes. We ask the question: what is the solution if we add a node with value f_0 to the graph, and connect the new node to unlabeled node i with weight w_0? The new node is a "dongle" attached to node i. Besides the usage here, dongle nodes can be useful for handling noisy labels where one would put the observed labels on the dongles, and infer the hidden true labels for the nodes attached to dongles. Note that when $w_0 \to \infty$, we effectively assign label f_0 to node i.

Since the dongle is a labeled node in the augmented graph,

$$\begin{aligned} f_u^+ &= {\Delta_{uu}^+}^{-1}W_{ul}^+f_l^+ = (D_{uu}^+ - W_{uu})^{-1}W_{ul}^+f_l^+ \\ &= (w_0ee^\top + D_{uu} - W_{uu})^{-1}(w_0f_0e + W_{ul}f_l) \\ &= (w_0ee^\top + \Delta_{uu})^{-1}(w_0f_0e + W_{ul}f_l) \end{aligned}$$

where e is a column vector of length u with 1 in position i and 0 elsewhere. Note that we can use the matrix inversion lemma here, to obtain

$$\begin{aligned} (w_0ee^\top + \Delta_{uu})^{-1} &= \Delta_{uu}^{-1} - \frac{\Delta_{uu}^{-1}(\sqrt{w_0}e)(\sqrt{w_0}e)^\top\Delta_{uu}^{-1}}{1 + (\sqrt{w_0}e)^\top\Delta_{uu}^{-1}(\sqrt{w_0}e)} \\ &= G - \frac{1}{1 + w_0G_{ii}}w_0G_{|i}G \end{aligned}$$

where we use the shorthand $G = \Delta_{uu}^{-1}$ (the Green's function); G_{ii} is the i-th row, i-th column element in G; $G_{|i}$ is a square matrix with G's i-th column and 0 else-

where. Some calculation gives

$$f_u^+ = f_u + \frac{w_0 f_0 - w_0 f_i}{1 + w_0 G_{ii}} G_{\cdot i}$$

where f_i is the unlabeled node's original solution, and $G_{\cdot i}$ is the i-th column vector in G. If we want to pin down the unlabeled node to value f_0, we can let $w_0 \to \infty$ to obtain

$$f_u^+ = f_u + \frac{f_0 - f_i}{G_{ii}} G_{\cdot i}$$

Appendix B

The Inverse of a Matrix with One Row/Column Removed

Let A be an $n \times n$ non-singular matrix. Given A^{-1}, we would like a fast algorithm to compute $A_{\neg i}^{-1}$, where $A_{\neg i}$ is the $(n-1) \times (n-1)$ matrix obtained by removing the i-th row and column from A.

Let $B = \text{perm}(A, i)$ be the matrix created by moving the i-th row in front of the 1st row, and the i-th column in front of the 1st column of A. Then

$$A_{\neg i}^{-1} \quad = \quad (\text{perm}(A, i)_{\neg 1})^{-1} = (B_{\neg 1})^{-1}$$

Also note $B^{-1} = \text{perm}(A^{-1}, i)$. So we only need to consider the special case of removing the first row/column of a matrix. Write B out as $B = \begin{bmatrix} b_{11} & B_{1*} \\ B_{*1} & B_{\neg 1} \end{bmatrix}$, where $B_{1*} = (b_{12} \dots b_{1n})$ and $B_{*1} = (b_{21} \dots b_{n1})^\top$. We will transform B into a block diagonal form in two steps. First, let $B' = \begin{bmatrix} 1 & 0 \\ B_{*1} & B_{\neg 1} \end{bmatrix} = B + uv^\top$ where $u = (-1, 0, \dots, 0)^\top$ and $v = (b_{11} - 1, B_{1*})^\top$. We are interested in $(B')^{-1}$ which will be used in the next step. By the matrix inversion lemma (Sherman-Morrison-Woodbury formula),

$$(B')^{-1} = (B + uv^\top)^{-1} = B^{-1} - \frac{B^{-1}uv^\top B^{-1}}{1 + v^\top B^{-1}u}$$

Next let $B'' = \begin{bmatrix} 1 & 0 \\ 0 & B_{\neg 1} \end{bmatrix} = B' + wu^\top$ where $w = (0, B_{*1})^\top$. Applying the matrix inversion lemma again,

$$(B'')^{-1} = (B' + wu^\top)^{-1} = (B')^{-1} - \frac{(B')^{-1}wu^\top(B')^{-1}}{1 + u^\top(B')^{-1}w}$$

But since B'' is block diagonal, we know $(B'')^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & (B_{\neg 1})^{-1} \end{bmatrix}$. Therefore $(B_{\neg 1})^{-1} = ((B'')^{-1})_{\neg 1}$.

Appendix C

Laplace Approximation for Gaussian Processes

This derivation largely follows (Herbrich, 2002) (B.7). The Gaussian process model, restricted to the labeled and unlabeled data, is

$$\mathbf{f} \sim \mathcal{N}\left(\mu, \tilde{\Delta}^{-1}\right) \tag{C.1}$$

We will use $G = \tilde{\Delta}^{-1}$ to denote the covariance matrix (i.e. the Gram matrix). Let $\mathbf{y} \in \{-1, +1\}$ be the observed discrete class labels. The hidden variable $\mathbf{f}$ and labels $\mathbf{y}$ are connected via a sigmoid noise model

$$P(y_i|f_i) = \frac{e^{\gamma f_i y_i}}{e^{\gamma f_i y_i} + e^{-\gamma f_i y_i}} = \frac{1}{1 + e^{-2\gamma f_i y_i}} \tag{C.2}$$

where γ is a hyperparameter which controls the steepness of the sigmoid. Given the prior and the noise model, we are interested in the posterior $p(\mathbf{f}_L, \mathbf{f}_U|\mathbf{y}_L)$. By Bayes theorem,

$$p(\mathbf{f}_L, \mathbf{f}_U|\mathbf{y}_L) = \frac{\prod_{i=1}^{l} P(y_i|f_i) p(\mathbf{f}_L, \mathbf{f}_U)}{P(\mathbf{y}_L)} \tag{C.3}$$

Because of the noise model, the posterior is not Gaussian and has no closed form solution. We use the Laplace approximation.

First, we find the mode of the posterior (6.7):

$$(\hat{\mathbf{f}_L}, \hat{\mathbf{f}_U}) = \arg\max_{\mathbf{f}_L, \mathbf{f}_U} \frac{\prod_{i=1}^{l} P(y_i|f_i) p(\mathbf{f}_L, \mathbf{f}_U)}{P(\mathbf{y}_L)} \tag{C.4}$$

$$= \arg\max_{\mathbf{f}_L, \mathbf{f}_U} \sum_{i=1}^{l} \ln P(y_i|f_i) + \ln p(\mathbf{f}_L, \mathbf{f}_U) \tag{C.5}$$

$$= \arg\max_{\mathbf{f}_L, \mathbf{f}_U} Q_1 + Q_2 \tag{C.6}$$

Note $\mathbf{f}_U$ only appears in Q_2, and we can maximize $\hat{\mathbf{f}_U}$ independently given $\hat{\mathbf{f}_L}$. Q_2 is the log likelihood of the Gaussian (C.1). Therefore given $\hat{\mathbf{f}_L}$, $\mathbf{f}_U$ follows the conditional distribution of Gaussian:

$$p(\mathbf{f}_U|\hat{\mathbf{f}_L}) = \mathcal{N}\left(G_{UL}G_{LL}^{-1}\hat{\mathbf{f}_L}, G_{UU} - G_{UL}G_{LL}^{-1}G_{LU}\right) \tag{C.7}$$

Moreover, the mode is the conditional mean

$$\hat{\mathbf{f}_U} = G_{UL}G_{LL}^{-1}\hat{\mathbf{f}_L} \tag{C.8}$$

It's easy to see (C.8) has the same form as the solution for Gaussian Fields (4.11): Recall $G = \tilde{\Delta}^{-1}$. From partitioned matrix inversion theorem,

$$\tilde{\Delta}_{UU} = S_A^{-1}$$

$$\tilde{\Delta}_{UL} = -S_A^{-1}G_{UL}G_{LL}^{-1}$$

where $S_A = G_{UU} - G_{UL}(G_{LL})^{-1}G_{LU}$ is the Schur complement of G_{LL}. This gives us

$$-(\tilde{\Delta}_{UU})^{-1}\tilde{\Delta}_{UL} = S_A S_A^{-1}G_{UL}G_{LL}^{-1} = G_{UL}G_{LL}^{-1}$$

Thus we have

$$\hat{\mathbf{f}_U} = -\tilde{\Delta}_{UU}^{-1}\tilde{\Delta}_{UL}\hat{\mathbf{f}_L} \tag{C.9}$$

$$= \tilde{\Delta}_{UU}^{-1}W_{UL}\hat{\mathbf{f}_L} \tag{C.10}$$

which has the same form as the harmonic energy minimizing function in (Zhu et al., 2003a). In fact the latter is the limiting case when $\sigma^2 \to \infty$ and there is no noise model.

Substitute (C.8) back to Q_2, using partitioned inverse of a matrix, it can be shown that (not surprisingly)

$$Q_2 = -\frac{1}{2}\mathbf{f}_L^\top G_{LL}^{-1}\mathbf{f}_L + c \tag{C.11}$$

Now go back to Q_1. The noise model can be written as

$$P(y_i|f_i) = \frac{e^{\gamma f_i y_i}}{e^{\gamma f_i y_i} + e^{-\gamma f_i y_i}} \tag{C.12}$$

$$= \left(\frac{e^{\gamma f_i}}{e^{\gamma f_i} + e^{-\gamma f_i}}\right)^{\frac{y_i+1}{2}} \left(1 - \frac{e^{\gamma f_i}}{e^{\gamma f_i} + e^{-\gamma f_i}}\right)^{\frac{1-y_i}{2}} \tag{C.13}$$

$$= \pi(f_i)^{\frac{y_i+1}{2}}(1 - \pi(f_i))^{\frac{1-y_i}{2}} \tag{C.14}$$

therefore

$$Q_1 = \sum_{i=1}^{l} \ln P(y_i|f_i) \tag{C.15}$$

$$= \sum_{i=1}^{l} \frac{y_i+1}{2} \ln \pi(f_i) + \frac{1-y_i}{2} \ln(1-\pi(f_i)) \tag{C.16}$$

$$= \gamma(\mathbf{y}_L - \mathbf{1})^\top \mathbf{f}_L - \sum_{i=1}^{l} \ln(1+e^{-2\gamma f_i}) \tag{C.17}$$

Put it together,

$$\hat{\mathbf{f}_L} = \arg\max Q_1 + Q_2 \tag{C.18}$$

$$= \arg\max \gamma(\mathbf{y}_L - \mathbf{1})^\top \mathbf{f}_L - \sum_{i=1}^{l} \ln(1+e^{-2\gamma f_i}) - \frac{1}{2}\mathbf{f}_L^\top G_{LL}^{-1} \mathbf{f}_L \tag{C.19}$$

To find the mode, we take the derivative,

$$\frac{\partial(Q_1+Q_2)}{\partial \mathbf{f}_L} = \gamma(\mathbf{y}_L - \mathbf{1}) + 2\gamma(1-\pi(\mathbf{f}_L)) - G_{LL}^{-1}\mathbf{f}_L \tag{C.20}$$

Because of the term $\pi(\mathbf{f}_L)$ it is not possible to find the root directly. We solve it with Newton-Raphson algorithm,

$$\mathbf{f}_L^{(t+1)} \leftarrow \mathbf{f}_L^{(t)} - H^{-1} \frac{\partial(Q_1+Q_2)}{\partial \mathbf{f}_L}\bigg|_{\mathbf{f}_L{}^{(t)}} \tag{C.21}$$

where H is the Hessian matrix,

$$H = \left[\frac{\partial^2(Q_1+Q_2)}{\partial f_i \partial f_j}\bigg|_{\mathbf{f}_L} \right] \tag{C.22}$$

Note $\frac{d}{df_i}\pi(f_i) = 2\gamma\pi(f_i)(1-\pi(f_i))$, we can write H as

$$H = -G_{LL}^{-1} - P \tag{C.23}$$

where P is a diagonal matrix with elements $P_{ii} = 4\gamma^2\pi(f_i)(1-\pi(f_i))$.

Once Newton-Raphson converges we compute $\hat{\mathbf{f}_U}$ from $\hat{\mathbf{f}_L}$ with (C.8). Classification can be done with $\text{sgn}(\hat{\mathbf{f}_U})$ noting this is the Bayesian classification rule with Gaussian distribution and sigmoid noise model.

To compute the covariance matrix of the Laplace approximation, note by definition the inverse covariance matrix of the Laplace approximation is

$$\Sigma^{-1} = \left[\frac{\partial^2 - \ln p(\mathbf{f}|\mathbf{y})}{\partial f_i \partial f_j} \Big|_{\hat{\mathbf{f}_L}, \hat{\mathbf{f}_U}} \right] \tag{C.24}$$

From (6.7) it is straightforward to confirm

$$\Sigma^{-1} = \begin{bmatrix} P & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} + G^{-1} = \begin{bmatrix} P & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} + \tilde{\Delta} \tag{C.25}$$

Therefore the covariance matrix is

$$\Sigma = \left(\begin{bmatrix} P & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} + \tilde{\Delta} \right)^{-1} \tag{C.26}$$

where P is evaluated at the mode $\hat{\mathbf{f}_L}$.

Appendix D

Hyperparameter Learning by Evidence Maximization

This derivation largely follows (Williams & Barber, 1998). We want to find the MAP hyperparameters Θ which maximize the posterior

$$p(\Theta|\mathbf{y}_L) \propto p(\mathbf{y}_L|\Theta)p(\Theta)$$

The prior $p(\Theta)$ is usually chosen to be simple, and so we focus on the term $p(\mathbf{y}_L|\Theta)$, known as the *evidence*. The definition

$$p(\mathbf{y}_L|\Theta) = \int p(\mathbf{y}_L|\mathbf{f}_L)p(\mathbf{f}_L|\Theta)\ d\mathbf{f}_L$$

is hard to compute analytically. However notice

$$p(\mathbf{y}_L|\Theta) \quad = \quad \frac{p(\mathbf{y}_L|\mathbf{f}_L)p(\mathbf{f}_L|\Theta)}{p(\mathbf{f}_L|\mathbf{y}_L,\Theta)}, \forall \mathbf{f}_L \tag{D.1}$$

Since it holds for all $\mathbf{f}_L$, it holds for the mode of the Laplace approximation $\hat{\mathbf{f}_L}$:

$$p(\mathbf{y}_L|\Theta) = \frac{p(\mathbf{y}_L|\hat{\mathbf{f}_L})p(\hat{\mathbf{f}_L}|\Theta)}{p(\hat{\mathbf{f}_L}|\mathbf{y}_L,\Theta)}$$

The terms on the numerator are straightforward to compute; the denominator is tricky. However we can use the Laplace approximation, i.e. the probability density at the mode: $p(\hat{\mathbf{f}_L}|\mathbf{y}_L,\Theta) \approx \mathcal{N}(\hat{\mathbf{f}_L}|\hat{\mathbf{f}_L},\Sigma_{LL})$. Recall

$$\Sigma = \left(\begin{bmatrix} P & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} + \begin{bmatrix} G_{LL} & G_{LU} \\ G_{UL} & G_{UU} \end{bmatrix}^{-1} \right)^{-1} \tag{D.2}$$

By applying Schur complement in block matrix decomposition twice, we find

$$\Sigma_{LL} = (P + G_{LL}^{-1})^{-1} \tag{D.3}$$

Therefore the evidence is

$$\begin{aligned} p(\mathbf{y}_L|\Theta) &\approx \frac{p(\mathbf{y}_L|\hat{\mathbf{f}_L})p(\hat{\mathbf{f}_L}|\Theta)}{\mathcal{N}(\hat{\mathbf{f}_L}|\hat{\mathbf{f}_L}, \Sigma_{LL})} && \text{(D.4)} \\ &= \frac{p(\mathbf{y}_L|\hat{\mathbf{f}_L})p(\hat{\mathbf{f}_L}|\Theta)}{(2\pi)^{-\frac{n}{2}}|\Sigma_{LL}|^{-\frac{1}{2}}} && \text{(D.5)} \\ &= \frac{p(\mathbf{y}_L|\hat{\mathbf{f}_L})p(\hat{\mathbf{f}_L}|\Theta)}{(2\pi)^{-\frac{n}{2}}|(P + G_{LL}^{-1})^{-1}|^{-\frac{1}{2}}} && \text{(D.6)} \end{aligned}$$

Switching to $\log$ domain, we have

$$\begin{aligned} \log p(\mathbf{y}_L|\Theta) &\approx \Psi(\hat{\mathbf{f}_L}) + \frac{n}{2}\log 2\pi + \frac{1}{2}\log|\Sigma_{LL}| && \text{(D.7)} \\ &= \Psi(\hat{\mathbf{f}_L}) + \frac{n}{2}\log 2\pi - \frac{1}{2}\log|P + G_{LL}^{-1}| && \text{(D.8)} \end{aligned}$$

where $\Psi(\mathbf{f}_L) = \log p(\mathbf{y}_L|\mathbf{f}_L) + \log p(\mathbf{f}_L|\Theta)$. Since $\mathbf{f} \sim \mathcal{N}\left(\mu, \tilde{\Delta}^{-1}\right) = \mathcal{N}(\mu, G)$, we have $\mathbf{f}_L \sim \mathcal{N}(\mu_L, G_{LL})$. Therefore

$$\begin{aligned} \Psi(\hat{\mathbf{f}_L}) &= \log p(\mathbf{y}_L|\hat{\mathbf{f}_L}) + \log p(\hat{\mathbf{f}_L}|\Theta) && \text{(D.9)} \\ &= -\sum_{i=1}^{L}\log(1+\exp(-2\gamma\hat{f}_i y_i)) \\ &\quad -\frac{n}{2}\log 2\pi - \frac{1}{2}\log|G_{LL}| - \frac{1}{2}(\hat{\mathbf{f}_L} - \mu_L)^\top G_{LL}^{-1}(\hat{\mathbf{f}_L} - \mu_L) && \text{(D.10)} \end{aligned}$$

Put it together,

$$\begin{aligned} \log p(\mathbf{y}_L|\Theta) &\approx -\sum_{i=1}^{L}\log(1+\exp(-2\gamma\hat{f}_i y_i)) \\ &\quad -\frac{1}{2}\log|G_{LL}| - \frac{1}{2}(\hat{\mathbf{f}_L} - \mu_L)^\top G_{LL}^{-1}(\hat{\mathbf{f}_L} - \mu_L) - \frac{1}{2}\log|P + G_{LL}^{-1}| \\ &= -\sum_{i=1}^{L}\log(1+\exp(-2\gamma\hat{f}_i y_i)) \\ &\quad -\frac{1}{2}(\hat{\mathbf{f}_L} - \mu_L)^\top G_{LL}^{-1}(\hat{\mathbf{f}_L} - \mu_L) - \frac{1}{2}\log|G_{LL}P + \mathbf{I}| && \text{(D.11)} \end{aligned}$$

This gives us a way to (approximately) compute the evidence.

To find the MAP estimate of Θ (which can have multiple local maxima), we use gradient methods. This involves the derivatives of the evidence $\partial \log p(\mathbf{y}_L|\Theta)/\partial\theta$, where θ is the hyperparameter β, σ, γ or the ones controlling W.

We start from

$$\begin{aligned} \frac{\partial}{\partial\theta}\pi(\hat{f}_i) &= \frac{\partial}{\partial\theta}\frac{1}{1+e^{-2\gamma\hat{f}_i}} && \text{(D.12)} \\ &= 2\pi(\hat{f}_i)(1-\pi(\hat{f}_i))(\hat{f}_i\frac{\partial\gamma}{\partial\theta}+\gamma\frac{\partial\hat{f}_i}{\partial\theta}) && \text{(D.13)} \end{aligned}$$

To compute $\partial\hat{\mathbf{f}_L}/\partial\theta$, note the Laplace approximation mode $\hat{\mathbf{f}_L}$ satisfies

$$\left.\frac{\partial\Psi(\mathbf{f}_L)}{\partial\mathbf{f}_L}\right|_{\hat{\mathbf{f}_L}} = \gamma(\mathbf{y}_L+\mathbf{1}-2\pi(\hat{\mathbf{f}_L}))-G_{LL}^{-1}(\hat{\mathbf{f}_L}-\mu_L)=0 \quad \text{(D.14)}$$

which means

$$\hat{\mathbf{f}_L} = \gamma G_{LL}(\mathbf{y}_L+\mathbf{1}-2\pi(\hat{\mathbf{f}_L}))+\mu_L \quad \text{(D.15)}$$

Taking derivatives on both sides,

$$\begin{aligned} \frac{\partial\hat{\mathbf{f}_L}}{\partial\theta} &= \frac{\partial}{\partial\theta}\gamma G_{LL}(\mathbf{y}_L+\mathbf{1}-2\pi(\hat{\mathbf{f}_L})) && \text{(D.16)} \\ &= \frac{\partial\gamma G_{LL}}{\partial\theta}(\mathbf{y}_L+\mathbf{1}-2\pi(\hat{\mathbf{f}_L}))-2\gamma G_{LL}\frac{\partial\pi(\hat{\mathbf{f}_L})}{\partial\theta} && \text{(D.17)} \\ &= \frac{\partial\gamma G_{LL}}{\partial\theta}(\mathbf{y}_L+\mathbf{1}-2\pi(\hat{\mathbf{f}_L}))-\frac{1}{\gamma}G_{LL}P\hat{\mathbf{f}_L}\frac{\partial\gamma}{\partial\theta}-G_{LL}P\frac{\partial\hat{\mathbf{f}_L}}{\partial\theta} && \text{(D.18)} \end{aligned}$$

which gives

$$\frac{\partial\hat{\mathbf{f}_L}}{\partial\theta} = (\mathbf{I}+G_{LL}P)^{-1}\left[\frac{\partial\gamma G_{LL}}{\partial\theta}(\mathbf{y}_L+\mathbf{1}-2\pi(\hat{\mathbf{f}_L}))-\frac{1}{\gamma}G_{LL}P\hat{\mathbf{f}_L}\frac{\partial\gamma}{\partial\theta}\right] \quad \text{(D.19)}$$

Now it is straightforward to compute the gradient with (D.11):

$$\begin{aligned}
& \frac{\partial}{\partial\theta}\log p(\mathbf{y}_L|\Theta) \\
\approx\ & \frac{\partial}{\partial\theta}\left[-\sum_{i=1}^{L}\log(1+\exp(-2\gamma\hat{f}_i y_i)) - \frac{1}{2}(\hat{\mathbf{f}_L}-\mu_L)^\top G_{LL}^{-1}(\hat{\mathbf{f}_L}-\mu_L) - \frac{1}{2}\log|G_{LL}P+\mathbf{I}|\right] \\
=\ & -\sum_{i=1}^{L}\frac{\exp(-2\gamma\hat{f}_i y_i)(-2y_i)}{1+\exp(-2\gamma\hat{f}_i y_i)}(\hat{f}_i\frac{\partial\gamma}{\partial\theta}+\gamma\frac{\partial\hat{f}_i}{\partial\theta}) \\
& -\frac{1}{2}\left[2(G_{LL}^{-1}(\hat{\mathbf{f}_L}-\mu_L))^\top\frac{\partial\hat{\mathbf{f}_L}}{\partial\theta}+(\hat{\mathbf{f}_L}-\mu_L)^\top\frac{\partial G_{LL}^{-1}}{\partial\theta}(\hat{\mathbf{f}_L}-\mu_L)\right] \\
& -\frac{1}{2}tr\left((G_{LL}P+\mathbf{I})^{-1}\frac{\partial G_{LL}P}{\partial\theta}\right)
\end{aligned} \tag{D.20}$$

where we used the fact

$$\frac{\partial\log|A|}{\partial\theta} = tr\left(A^{-1}\frac{\partial A}{\partial\theta}\right) \tag{D.21}$$

For example, if $\theta=\gamma$, the gradient can be computed by noting $\frac{\partial\gamma G_{LL}}{\partial\gamma}=G_{LL}$, $\frac{\partial\gamma}{\partial\gamma}=1$, $\frac{\partial G_{LL}^{-1}}{\partial\gamma}=0$, and $\frac{\partial G_{LL}P}{\partial\gamma}=G_{LL}\frac{\partial P}{\partial\gamma}$ where $\frac{\partial P_{ii}}{\partial\gamma}=8\gamma\pi(\hat{f}_i)(1-\pi(\hat{f}_i))+4\gamma^2(1-2\pi(\hat{f}_i))\frac{\partial\pi(\hat{f}_i)}{\partial\gamma}$.

For $\theta=\beta$, we have $\frac{\partial\gamma G_{LL}}{\partial\beta}=\gamma(-1/\beta)G_{LL}$, $\frac{\partial\gamma}{\partial\beta}=0$, $\frac{\partial G_{LL}^{-1}}{\partial\beta}=G_{LL}^{-1}/\beta$, and $\frac{\partial G_{LL}P}{\partial\beta}=-G_{LL}P/\beta+G_{LL}\frac{\partial P}{\partial\beta}$ where $\frac{\partial P_{ii}}{\partial\beta}=8\gamma^3\pi(\hat{f}_i)(1-\pi(\hat{f}_i))(1-2\pi(\hat{f}_i))\frac{\partial\hat{f}_i}{\partial\beta}$.

For $\theta=\sigma$, the computation is more intensive because the complex dependency between G and σ. We start from $\frac{\partial G_{LL}}{\partial\sigma}=\left[\frac{\partial G}{\partial\sigma}\right]_{LL}$. Using the fact $\frac{\partial A^{-1}}{\partial\theta}=-A^{-1}\frac{\partial A}{\partial\theta}A^{-1}$ and $G=\tilde{\Delta}^{-1}$, we get $\frac{\partial G}{\partial\sigma}=\beta/\sigma^3G^2$. Note the computation involves the multiplication of the *full* matrix G and is thus more demanding. Once $\frac{\partial G_{LL}}{\partial\sigma}$ is computed the rest is easy.

If we parameterize the weights W in Gaussian Fields with radial basis functions (for simplicity we assume a single length scale parameter α for all dimensions. Extension to multiple length scales is simple),

$$w_{ij}=\exp\left(-\frac{d_{ij}^2}{\alpha^2}\right) \tag{D.22}$$

where d_{ij} is the Euclidean distance between x_i, x_j in the original feature space, we can similarly learn the hyperparameter α. Note $\frac{\partial w_{ij}}{\partial\alpha}=w_{ij}\frac{d_{ij}^2}{\alpha^3}$, $\frac{\partial\Delta}{\partial\alpha}=\frac{\partial D}{\partial\alpha}-\frac{\partial W}{\partial\alpha}$, $\frac{\partial\tilde{\Delta}}{\partial\alpha}=\beta\frac{\partial\Delta}{\partial\alpha}$. The rest is the same as for σ above.

Similarly with a $\tanh()$-weighted weight function $w_{ij} = (\tanh(\alpha_1(d_{ij} - \alpha_2)) + 1)/2$, we have $\frac{\partial w_{ij}}{\partial \alpha_1} = (1 - \tanh^2(\alpha_1(d_{ij} - \alpha_2)))(d_{ij} - \alpha_2)/2$ and $\frac{\partial w_{ij}}{\partial \alpha_2} = -(1 - \tanh^2(\alpha_1(d_{ij} - \alpha_2)))\alpha_1/2$, and the rest follows.

Appendix E

Mean Field Approximation for Kernel CRF Training

In the basic kernel CRF model, each clique c is associated with $|y|^{|c|}$ parameters $\alpha_j^c(\mathbf{y}_c)$. Even if we only consider vertex cliques, there would be hundreds of thousands of parameters for a typical protein dataset. This seriously affects the training efficiency.

To solve the problem, we adopt the notion of "import vector machines" by Zhu and Hastie (2001). That is, we use a subset of the training examples instead of all of them. The subset is constructed by greedily selecting training examples one at a time to minimize the loss function:

$$\arg\min_k R(f_{A\cup\{k\}}, \lambda) - R(f_A, \lambda) \tag{E.1}$$

where

$$f_A(\mathbf{x}, \mathbf{y}) = \sum_{j\in A} \alpha_j(\mathbf{y}) K(\mathbf{x}_j, \mathbf{x}) \tag{E.2}$$

and A is the current *active import vector set*.

(E.1) is hard to compute: we need to update all the parameters for $f_{A\cup\{k\}}$. Even if we keep old parameters in f_A fixed, we still need to use expensive forward-backward algorithm to train the new parameters $\alpha_k(\mathbf{y})$ and compute the loss. Following McCallum (2003), we make a set of speed up approximations.

Approximation 1: Mean field approximation. With the old f_A we have an old distribution $P(\mathbf{y}|\mathbf{x}) = 1/Z \exp(\sum_c f_A^c(\mathbf{x}, \mathbf{y}))$ over a label sequence y. We approximate $P(\mathbf{y}|\mathbf{x})$ by the mean field

$$P_o(\mathbf{y}|\mathbf{x}) = \prod_i P_o(y_i|x_i) \tag{E.3}$$

i.e. the mean field approximation is the independent product of marginal distributions at each position i. It can be computed with the Forward-Backward algorithm on $P(\mathbf{y}|\mathbf{x})$.

Approximation 2: Consider only the vertex kernel. In conjunction with the mean field approximation, we only consider the vertex kernel $K(x_i, x_j)$ and ignore edge or other higher order kernels. The loss function becomes

$$R(f_A, \lambda) = -\sum_{i \in T} \log P_o(y_i|x_i) + \frac{\lambda}{2} \sum_{i,j \in A} \sum_y \alpha_i(y)\alpha_j(y)K(x_i, x_j) \tag{E.4}$$

where $T = \{1, \ldots, M\}$ is the set of training positions on which to evaluate the loss function. Once we add a candidate import vector x_k to the active set, the new model is

$$P_n(y_i|x_i) = \frac{P_o(y_i|x_i) \exp(\alpha_k(y_i)K(x_i, x_k))}{\sum_y P_o(y|x_i) \exp(\alpha_k(y)K(x_i, x_k))} \tag{E.5}$$

The new loss function is

$$R(f_{A\cup\{k\}}, \lambda) = -\sum_{i \in T} \log P_n(y_i|x_i) + \frac{\lambda}{2} \sum_{i,j \in A\cup\{k\}} \sum_y \alpha_i(y)\alpha_j(y)K(x_i, x_j) \tag{E.6}$$

And (E.1) can be written as

$$\begin{aligned} R(f_{A\cup\{k\}}, \lambda) - R(f_A, \lambda) &= -\sum_{i \in T} \alpha_k(y_i)K(x_i, x_k) \\ &\quad + \sum_{i \in T} \log \sum_y P_o(y|x_i) \exp(\alpha_k(y)K(x_i, x_k)) \\ &\quad + \lambda \sum_{j \in A} \sum_y \alpha_j(y)\alpha_k(y)K(x_j, x_k) + \frac{\lambda}{2} \sum_y \alpha_k^2(y)K(x_k, x_k) \end{aligned} \tag{E.7}$$

This change of loss is a convex function of the $|y|$ parameters $\alpha_k(y)$. We can find the best parameters with Newton's method. The first order derivatives are

$$\frac{\partial R(f_{A\cup\{k\}}, \lambda) - R(f_A, \lambda)}{\partial \alpha_k(y)} = -\sum_{i \in T} K(x_i, x_k)\delta(y_i, y) \tag{E.8}$$

$$+ \sum_{i \in T} P_n(y|x_i)K(x_i, x_k) \tag{E.9}$$

$$+ \lambda \sum_{j \in A\cup\{k\}} \alpha_j(y)K(x_j, x_k) \tag{E.10}$$

And the second order derivatives are

$$\frac{\partial^2 R(f_{A\cup\{k\}},\lambda) - R(f_A,\lambda)}{\partial\alpha_k(y)\partial\alpha_k(y')} = \sum_{i\in T}\left[P_n(y|x_i)K^2(x_i,x_k)\delta(y,y') - P_n(y|x_i)K^2(x_i,x_k)P_n(y'|x_i)\right] + \lambda K(x_k,x_k)\delta(y,y') \quad \text{(E.11)}$$

Approximation 1 and 2 allow us to estimate the change in loss function independently for each position in T. This avoids the need of dynamic programming. Although the time complexity to evaluate each candidate x_k is still linear in $|T|$, we save by a (potentially large) constant factor. Further more, they allow a more dramatic approximation as shown next.

Approximation 3: Sparse evaluation of likelihood. A typical protein database has around 500 sequences, with hundreds of amino acid residuals per sequence. Therefore M, the total number of training positions, can easily be around 100,000. Normally $T = \{1,\ldots,M\}$, i.e. we need to sum over all training positions to evaluate the log-likelihood. However we can speed up by reducing T. There are several possibilities:

1. Focus on errors: $T = \{i|y_i \neq \arg\max_y P_o(y|x_i)\}$
2. Focus on low confidence: $T = \{i|P_o(y_i|x_i) < p_0\}$
3. Skip positions: $T = \{ai|ai \leq M; a, i \in N\}$
4. Random sample: $T = \{i|i \sim uniform(1,M)\}$
5. Error/confidence guided sample: errors / low confidence positions have higher probability to be sampled.

We need to scale the log likelihood term to maintain the balance between it and the regularization term:

$$R(f_A,\lambda) = -\frac{M}{|T|}\sum_{i\in T}\log P_o(y_i|x_i) + \frac{\lambda}{2}\sum_{i,j\in A}\sum_y \alpha_i(y)\alpha_j(y)K(x_i,x_j) \quad \text{(E.12)}$$

and scale the derivatives accordingly.

Other approximations: We may want to add more than one candidate import vector to A at a time. However we need to eliminate redundant vectors, possibly by the kernel distance. We may not want to fully train $f_{A\cup\{k\}}$ once we selected k.

Appendix F

An Empirical Comparison of Iterative Algorithms

The single most significant bottleneck in computing the harmonic function is to invert a $u \times u$ matrix, as in $f_u = -\Delta_{uu}^{-1}\Delta_{ul}f_l$. Done naively the cost is close to $O(n^3)$, which is prohibitive for practical problems. For example Matlab inv() function can only handle n in the range of several thousand. Clearly, we need to find ways to avoid the expensive inversion. One can go several directions:

1. One can approximate the inversion of a matrix by its top few eigenvalues and eigenvectors. If a $n \times n$ invertible matrix A has spectrum decomposition $A = \sum_{i=1}^{n} \lambda_i \phi_i \phi_i^\top$, then $A^{-1} = \sum_{i=1}^{n} 1/\lambda_i \phi_i \phi_i^\top \approx \sum_{i=1}^{m} 1/\lambda_i \phi_i \phi_i^\top$. The top $m < n$ eigenvectors ϕ_i with the smallest eigenvalues λ_i is less expensive to compute than inverting the matrix. This has been used in non-parametric transforms of graph kernels for semi-supervised learning in Chapter 8. A similar approximation is used in (Joachims, 2003). We will not pursue it further here.

2. One can reduced the problem size. Instead of using all of the unlabeled data, we can use a subset (or clusters) to construct the graph. The harmonic solution on the remaining data can be approximated with a computationally cheap method. The backbone graph in Chapter 10 is an example.

3. One can use iterative methods. The hope is that each iteration is $O(n)$ and convergence can be reached in relatively few iterations. There is a rich set of iterative methods applicable. We will compare the simple 'label propagation' algorithm, loopy belief propagation and conjugate gradient next.

F.1 Label Propagation

The original label propagation algorithm was proposed in (Zhu & Ghahramani, 2002a). A slightly modified version is presented here. Let $P = D^{-1}W$ be the transition matrix. Let f_l be the vector for labeled set (for multiclass problems it can be an $l \times c$ matrix). The label propagation algorithm consists of two steps:

1. $\begin{pmatrix} f_l^{(t+1)} \\ f_u^{(t+1)} \end{pmatrix} = P \begin{pmatrix} f_l^{(t)} \\ f_u^{(t)} \end{pmatrix}$
2. Clamp the labeled data $f_l^{(t+1)} = f_l$

It can be shown f_u converges to the harmonic solution regardless of initialization. Each iteration needs a matrix-vector multiplication, which can be $O(n)$ for sparse graphs. However the convergence may be slow.

F.2 Conjugate Gradient

The harmonic function is the solution to the linear system

$$\Delta_{uu} f_u = -\Delta_{ul} f_l \tag{F.1}$$

Standard conjugate gradient methods have been shown to perform well (Argyriou, 2004). In particular, the use of Jacobi preconditioner was shown to improve convergence. The Jacobi preconditioner is simply the diagonal of Δ_{uu}, and the preconditioned linear system is

$$\text{diag}(\Delta_{uu})^{-1}\Delta_{uu} f_u = -\text{diag}(\Delta_{uu})^{-1}\Delta_{ul} f_l \tag{F.2}$$

We note this is exactly

$$(I - P_{uu}) f_u = -P_{ul} f_l \tag{F.3}$$

i.e. the alternative definition of harmonic function $f_u = -(I - P_{uu})^{-1} P_{ul} f_l$, where $P = D^{-1}W$ is the transition matrix.

F.3 Loopy belief propagation on Gaussian fields

The harmonic solution

$$f_u = -\Delta_{uu}^{-1}\Delta_{ul} f_l \tag{F.4}$$

computes the mean of the marginals on unlabeled nodes u. Δ is the graph Laplacian. The computation involves inverting a $u \times u$ matrix and is expensive for large

datasets. We hope to use loopy belief propagation instead, as each iteration is $O(n)$ if the graph is sparse, and loopy BP has a reputation of converging fast (Weiss & Freeman, 2001) (Sudderth et al., 2003). It has been proved that if loopy BP converges, the mean values are correct (i.e. the harmonic solution).

The Gaussian field is defined as

$$p(y) \propto \exp(-\frac{1}{2} y \Delta y^{\top}) \tag{F.5}$$

And $f_u = E_p[y_u]$. Note the corresponding pairwise clique representation is

$$\begin{aligned} p(y) &\propto \prod_{i,j} \psi_{ij}(y_i, y_j) && \text{(F.6)} \\ &= \prod_{i,j} \exp\left(-\frac{1}{2} w_{ij}(y_i - y_j)^2\right) && \text{(F.7)} \\ &= \prod_{i,j} \exp\left(-\frac{1}{2} (y_i y_j) \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} y_i \\ y_j \end{pmatrix}\right) && \text{(F.8)} \end{aligned}$$

where $a = d = w_{ij}, b = c = -w_{ij}$, and w_{ij} is the weight of edge ij. Notice in this simple model we don't have n nodes for hidden variables and another n for observed ones; we only have n nodes with some of them observed. In other words, there is no 'noise model'.

The standard belief propagation messages are

$$m_{ij}(y_j) = \alpha \int_{y_i} \psi_{ij}(y_i, y_j) \prod_{k \in N(i) \backslash j} m_{ki}(y_i) dy_i \tag{F.9}$$

where m_{ij} is the message from i to j, $N(i)\backslash j$ is the neighbors of i except j, and α a normalization factor. Initially the messages are arbitrary (e.g. uniform) except for observed nodes $y_l = f_l$, whose messages to their neighbors are

$$m_{lj}(y_j) = \alpha \psi_{ij}(y_l, y_j) \tag{F.10}$$

After the messages converge, the marginals (belief) is computed as

$$b(y_i) = \alpha \prod_{k \in N(i)} m_{ki}(y_i) \tag{F.11}$$

For Gaussian fields with scalar-valued nodes, each message m_{ij} can be parameterized similar to a Gaussian distribution by its mean μ_{ij} and inverse variance (precision) $P_{ij} = 1/\sigma_{ij}^2$ parameters. That is,

$$m_{ij}(x_j) \propto \exp\left(-\frac{1}{2}(x_j - \mu_{ij})^2 P_{ij}\right) \tag{F.12}$$

We derive the belief propagation iterations for this special case next.

$$
\begin{aligned}
& m_{ij}(y_j) \\
= \;& \alpha \int_{y_i} \psi_{ij}(y_i, y_j) \prod_{k \in N(i) \backslash j} m_{ki}(y_i) dy_i \\
= \;& \alpha \int_{y_i} \exp\left(-\frac{1}{2} (y_i y_j) \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} y_i \\ y_j \end{pmatrix} \right) \prod_{k \in N(i) \backslash j} m_{ki}(y_i) dy_i \\
= \;& \alpha_2 \int_{y_i} \exp\left[-\frac{1}{2} \left((y_i y_j) \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} y_i \\ y_j \end{pmatrix} + \sum_{k \in N(i) \backslash j} (x_i - \mu_{ki})^2 P_{ki} \right) \right] dy_i \\
= \;& \alpha_3 \exp\left(-\frac{1}{2} dy_j^2 \right) \\
& \int_{y_i} \exp\left[-\frac{1}{2} \left(\left(a + \sum_{k \in N(i) \backslash j} P_{ki} \right) y_i^2 + 2 \left(b y_j - \sum_{k \in N(i) \backslash j} P_{ki} \mu_{ki} \right) y_i \right) \right] dy_i
\end{aligned}
$$

where we use the fact $b = c$. Let $A = a + \sum_{k \in N(i) \backslash j} P_{ki}$, $B = b y_j - \sum_{k \in N(i) \backslash j} P_{ki} \mu_{ki}$,

$$
\begin{aligned}
& m_{ij}(y_j) \qquad \text{(F.13)} \\
= \;& \alpha_3 \exp\left(-\frac{1}{2} dy_j^2 \right) \int_{y_i} \exp\left[-\frac{1}{2} \left(A y_i^2 + 2 B y_i \right) \right] dy_i \\
= \;& \alpha_3 \exp\left(-\frac{1}{2} dy_j^2 \right) \int_{y_i} \exp\left[-\frac{1}{2} \left((\sqrt{A} y_i + B/\sqrt{A})^2 - B^2/A \right) \right] dy_i \\
= \;& \alpha_3 \exp\left[-\frac{1}{2} \left(dy_j^2 - B^2/A \right) \right] \int_{y_i} \exp\left[-\frac{1}{2} \left((\sqrt{A} y_i + B/\sqrt{A})^2 \right) \right] dy_i
\end{aligned}
$$

Note the integral is Gaussian whose value depends on A, not B. However since A is constant w.r.t. y_j, the integral can be absorbed into the normalization factor,

$$
\begin{aligned}
& m_{ij}(y_j) \qquad \text{(F.14)} \\
= \;& \alpha_4 \exp\left[-\frac{1}{2} \left(dy_j^2 - B^2/A \right) \right] \\
= \;& \alpha_4 \exp\left[-\frac{1}{2} \left(dy_j^2 - \frac{b^2 y_j^2 - 2b \sum_{k \in N(i) \backslash j} P_{ki} \mu_{ki} y_j + (\sum_{k \in N(i) \backslash j} P_{ki} \mu_{ki})^2}{a + \sum_{k \in N(i) \backslash j} P_{ki}} \right) \right] \\
= \;& \alpha_5 \exp\left[-\frac{1}{2} \left(\left(d - \frac{b^2}{a + \sum_{k \in N(i) \backslash j} P_{ki}} \right) y_j^2 + 2 \frac{b \sum_{k \in N(i) \backslash j} P_{ki} \mu_{ki}}{a + \sum_{k \in N(i) \backslash j} P_{ki}} y_j \right) \right]
\end{aligned}
$$

Let $C = d - \frac{b^2}{a+\sum_{k\in N(i)\setminus j} P_{ki}}$, $D = \frac{b\sum_{k\in N(i)\setminus j} P_{ki}\mu_{ki}}{a+\sum_{k\in N(i)\setminus j} P_{ki}}$,

$$
\begin{aligned}
& m_{ij}(y_j) && \text{(F.15)}\\
= \;& \alpha_5 \exp\left[-\frac{1}{2}\left(Cy_j^2 + 2Dy_j\right)\right] && \text{(F.16)}\\
= \;& \alpha_5 \exp\left[-\frac{1}{2}\left(\left(\sqrt{C}y_j + D/\sqrt{C}\right)^2 - D^2/C\right)\right] && \text{(F.17)}\\
= \;& \alpha_6 \exp\left[-\frac{1}{2}\left(\left(\sqrt{C}y_j + D/\sqrt{C}\right)^2\right)\right] && \text{(F.18)}\\
= \;& \alpha_6 \exp\left[-\frac{1}{2}\left(\left(y_j - (-D/C)\right)^2 C\right)\right] && \text{(F.19)}
\end{aligned}
$$

Thus we see the message m_{ij} has the form of a Gaussian density with sufficient statistics

$$
\begin{aligned}
P_{ij} &= C && \text{(F.20)}\\
&= d - \frac{b^2}{a+\sum_{k\in N(i)\setminus j} P_{ki}} && \text{(F.21)}\\
\mu_{ij} &= -D/C && \text{(F.22)}\\
&= -\frac{b\sum_{k\in N(i)\setminus j} P_{ki}\mu_{ki}}{a+\sum_{k\in N(i)\setminus j} P_{ki}} P_{ij}^{-1} && \text{(F.23)}
\end{aligned}
$$

For our special case of $a = d = w_{ij}, b = c = -w_{ij}$, we get

$$
\begin{aligned}
P_{ij} &= w_{ij} - \frac{w_{ij}^2}{w_{ij}+\sum_{k\in N(i)\setminus j} P_{ki}} && \text{(F.24)}\\
\mu_{ij} &= -D/C && \text{(F.25)}\\
&= \frac{w_{ij}\sum_{k\in N(i)\setminus j} P_{ki}\mu_{ki}}{w_{ij}+\sum_{k\in N(i)\setminus j} P_{ki}} P_{ij}^{-1} && \text{(F.26)}
\end{aligned}
$$

For observed nodes $y_l = f_l$, they ignore any messages sent to them, while sending out the following messages to their neighbors j:

$$
\begin{aligned}
\mu_{lj} &= f_l && \text{(F.27)}\\
P_{lj} &= w_{lj} && \text{(F.28)}
\end{aligned}
$$

The belief at node i is

$$
\begin{aligned}
& b_i(y_i) && \text{(F.29)}\\
= \;& \alpha \prod_{k \in N(i)} m_{ki}(y_i) && \text{(F.30)}\\
= \;& \alpha \exp\left[-\frac{1}{2}\left(\sum_{k \in N(i)} (y_i - \mu_{ki})^2 P_{ki}\right)\right] && \text{(F.31)}\\
= \;& \alpha_2 \exp\left[-\frac{1}{2}\left(\sum_{k \in N(i)} P_{ki} y_i^2 - 2 \sum_{k \in N(i)} P_{ki}\mu_{ki} y_i\right)\right] && \text{(F.32)}\\
= \;& \alpha_3 \exp\left[-\frac{1}{2}\left(\left(y_i - \frac{\sum_{k \in N(i)} P_{ki}\mu_{ki}}{\sum_{k \in N(i)} P_{ki}}\right)^2 \cdot \left(\sum_{k \in N(i)} P_{ki}\right)\right)\right] && \text{(F.33)}
\end{aligned}
$$

This is a Gaussian distribution with mean and inverse variance

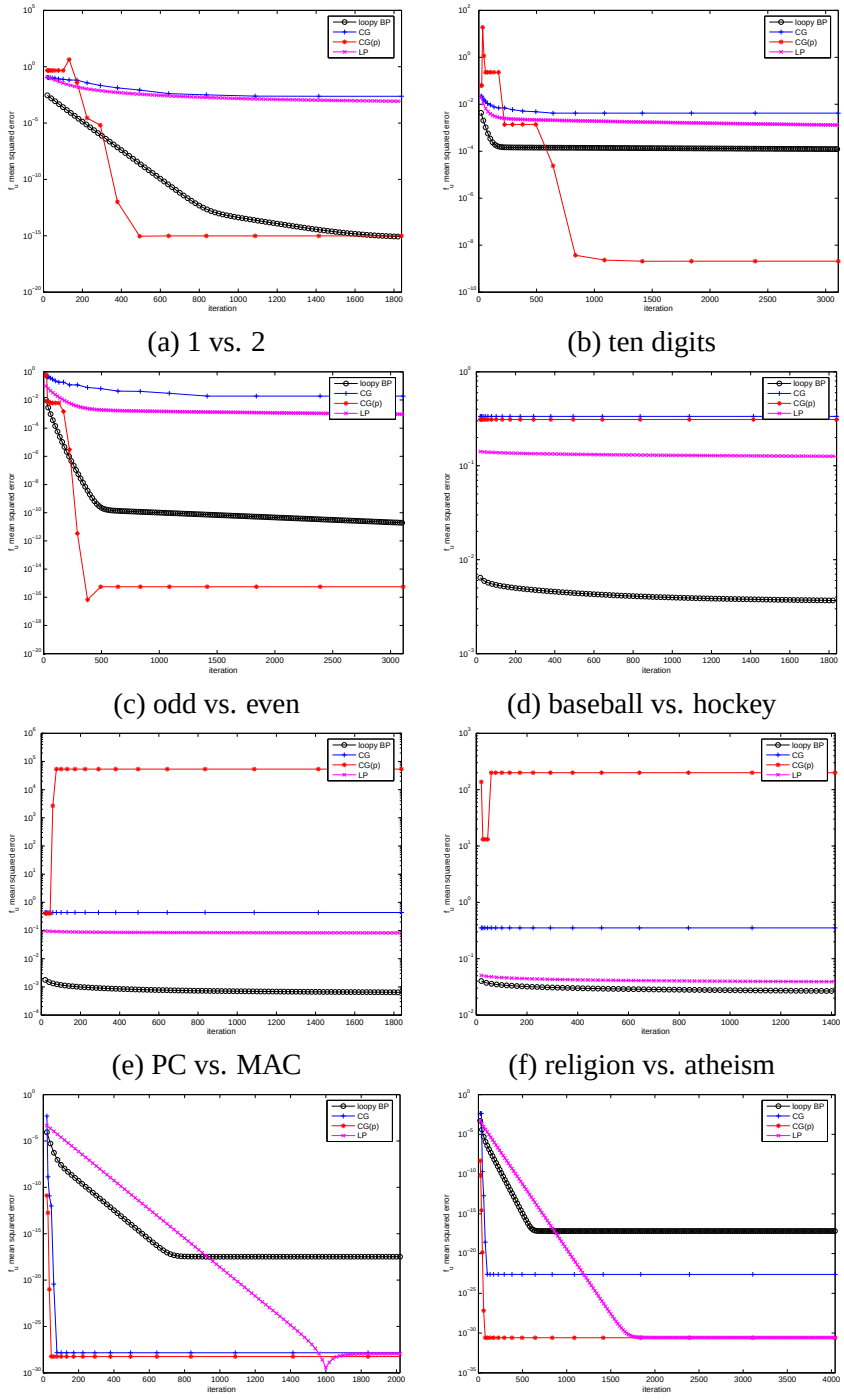
$$
\begin{aligned}
\mu_i &= \frac{\sum_{k \in N(i)} P_{ki}\mu_{ki}}{\sum_{k \in N(i)} P_{ki}} && \text{(F.34)}\\
P_i &= \sum_{k \in N(i)} P_{ki} && \text{(F.35)}
\end{aligned}
$$

F.4 Empirical Results

We compare label propagation (LP), loopy belief propagation (loopy BP), conjugate gradient (CG) and preconditioned conjugate gradient (CG(p)) on eight tasks. The tasks are small because we want to be able to compute the closed form solution f_u with matrix inversion. LP is coded in Matlab with sparse matrix. Loopy BP is implemented in C. CG and CG(p) use Matlab $cgs()$ function.

Figure F.1 compares the mean squared error $\sum_{i \in U} \left(f^{(t)}(i) - f_u(i)\right)^2$ with different methods at iteration t. We assume that with good implementation, the cost per iteration for different methods is similar. For multiclass tasks, it shows the binary sub-task of the first class vs. the rest. Note the y-axis is in log scale. We observe that loopy BP always converges reasonably fast; CG(p) can catch up and come closest to the closed form solution quickly, however sometimes it does not converge (d,e,f); CG is always worse than CG(p); LP converges very slowly.

For classification purpose we do not need to wait for $f_u^{(t)}$ to converge. Another quantity of interest is when does $f_u^{(t)}$ give the same classification as the closed form solution f_u. For the binary case this means $f_u^{(t)}$ and f_u are on the same side



(a) 1 vs. 2

(b) ten digits

(c) odd vs. even

(d) baseball vs. hockey

(e) PC vs. MAC

(f) religion vs. atheism

(g) isolet

(h) freefoodcam

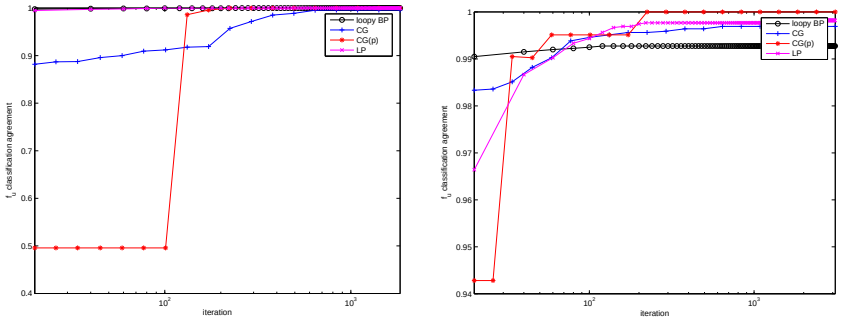
Figure F.1: Mean squared error to the harmonic solution with various iterative methods: loopy belief propagation (loopy BP), conjugate gradient (CG), conjugate gradient with Jacobi preconditioner (CG(p)), and label propagation (LP). Note the log-scale y-axis.

task	nodes	edges	loopy BP	CG	CG(p)	LP	closed form
one vs. two	2200	17000	0.02	0.002	0.001	0.0008	2e+01
odd vs. even	4000	31626	0.03	0.003	0.0007	0.001	1e+02
baseball vs. hockey	1993	13930	0.02	0.001	0.002	0.0007	2e+01
pc vs. mac	1943	14288	0.02	0.002	0.002	0.0007	2e+01
religion vs. atheism	1427	10201	0.01	0.001	0.001	0.0005	7
ten digits	4000	31595	0.03	0.003	0.004	0.008	9e+01
isolet	7797	550297	5	0.0005	0.0003	1	2e+03
freefoodcam	5254	23098	0.02	0.0001	7e-05	0.008	1e+02

Table F.1: Average run time per iteration for loopy belief propagation (loopy BP), conjugate gradient (CG), conjugate gradient with Jacobi preconditioner (CG(p)), and label propagation (LP). Also listed is the run time for closed form solution. Time is in seconds. Loopy BP is implemented in C, others in Matlab.

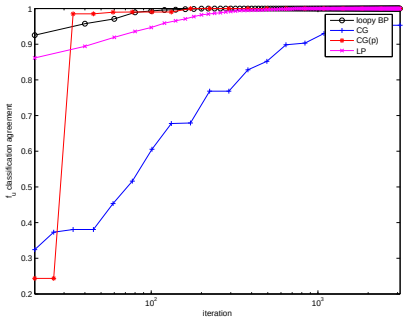
of 0.5, if labels are 0 and 1. We define classification agreement as the percentage of unlabeled data whose $f_u^{(t)}$ and f_u have the same label. Note this is not classification accuracy. Ideally agreement should reach 100% long before $f_u^{(t)}$ converges. Figure F.2 compares the agreement. Note x-axis is in log scale. All methods quickly reach classification agreement with the closed form solution, except CG and CG(p) sometimes do not converge; Task (f) has only 80% agreement.

Since loopy BP code is implemented in C and others in Matlab, their speed may not be directly comparable. Nonetheless we list the average per-iteration run time of different iterative methods in Table F.1. Also listed are the run time of the closed form solution with Matlab $inv()$.

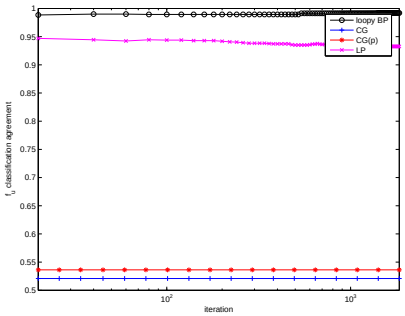


(a) 1 vs. 2

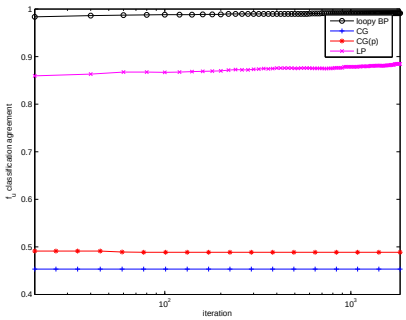
(b) ten digits



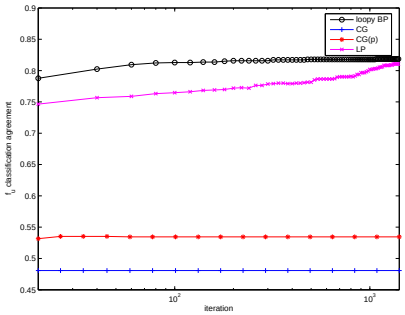
(c) odd vs. even



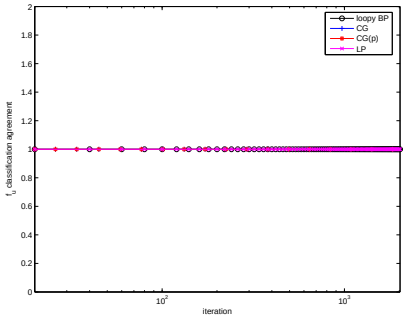
(d) baseball vs. hockey



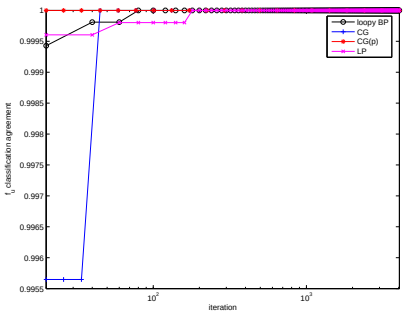
(e) PC vs. MAC



(f) religion vs. atheism



(g) isolet



(h) freefoodcam

Figure F.2: Classification agreement to the closed form harmonic solution with various iterative methods: loopy belief propagation (loopy BP), conjugate gradient (CG), conjugate gradient with Jacobi preconditioner (CG(p)), and label propagation (LP). Note the log-scale x-axis.

Bibliography

Argyriou, A. (2004). Efficient approximation methods for harmonic semi-supervised learning. Master's thesis, University College London.

Balcan, M.-F., Blum, A., & Yang, K. (2005). Co-training and expansion: Towards bridging theory and practice. In L. K. Saul, Y. Weiss and L. Bottou (Eds.), *Advances in neural information processing systems 17*. Cambridge, MA: MIT Press.

Baluja, S. (1998). Probabilistic modeling for face orientation discrimination: Learning from labeled and unlabeled data. *Neural Information Processing Systems*.

Baxter, J. (1997). The canonical distortion measure for vector quantization and function approximation. *Proc. 14th International Conference on Machine Learning* (pp. 39–47). Morgan Kaufmann.

Belkin, M., Matveeva, I., & Niyogi, P. (2004a). Regularization and semi-supervised learning on large graphs. *COLT*.

Belkin, M., & Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation, 15*, 1373–1396.

Belkin, M., Niyogi, P., & Sindhwani, V. (2004b). *Manifold regularization: A geometric framework for learning from examples* (Technical Report TR-2004-06). University of Chicago.

Bennett, K., & Demiriz, A. (1999). Semi-supervised support vector machines. *Advances in Neural Information Processing Systems, 11*, 368–374.

Blake, C., & Merz, C. (1998). UCI repository of machine learning databases.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research, 3*, 993–1022.

Blum, A., & Chawla, S. (2001). Learning from labeled and unlabeled data using graph mincuts. *Proc. 18th International Conf. on Machine Learning*.

Blum, A., Lafferty, J., Rwebangira, M., & Reddy, R. (2004). Semi-supervised learning using randomized mincuts. *ICML-04, 21th International Conference on Machine Learning*.

Blum, A., & Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. *COLT: Proceedings of the Workshop on Computational Learning Theory*.

Bousquet, O., Chapelle, O., & Hein, M. (2004). Measure based regularization. *Advances in Neural Information Processing Systems 16.*.

Boyd, S., & Vandenberge, L. (2004). *Convex optimization*. Cambridge UK: Cambridge University Press.

Callison-Burch, C., Talbot, D., & Osborne, M. (2004). Statistical machine translation with word- and sentence-aligned parallel corpora. *Proceedings of the ACL*.

Carreira-Perpinan, M. A., & Zemel, R. S. (2005). Proximity graphs for clustering and manifold learning. In L. K. Saul, Y. Weiss and L. Bottou (Eds.), *Advances in neural information processing systems 17*. Cambridge, MA: MIT Press.

Castelli, V., & Cover, T. (1995). The exponential value of labeled samples. *Pattern Recognition Letters, 16,* 105–111.

Castelli, V., & Cover, T. (1996). The relative value of labeled and unlabeled samples in pattern recognition with an unknown mixing parameter. *IEEE Transactions on Information Theory, 42,* 2101–2117.

Chaloner, K., & Verdinelli, I. (1995). Bayesian experimental design: A review. *Statistical Science, 10,* 237–304.

Chapelle, O., Weston, J., & Schölkopf, B. (2002). Cluster kernels for semi-supervised learning. *Advances in Neural Information Processing Systems, 15*.

Chapelle, O., & Zien, A. (2005). Semi-supervised classification by low density separation. *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTAT 2005)*.

Chu, W., & Ghahramani, Z. (2004). *Gaussian processes for ordinal regression* (Technical Report). University College London.

Chung, F. R. K. (1997). *Spectral graph theory, regional conference series in mathematics, no. 92*. American Mathematical Society.

Cohn, D. A., Ghahramani, Z., & Jordan, M. I. (1996). Active learning with statistical models. *Journal of Artificial Intelligence Research, 4*, 129–145.

Corduneanu, A., & Jaakkola, T. (2001). *Stable mixing of complete and incomplete information* (Technical Report AIM-2001-030). MIT AI Memo.

Corduneanu, A., & Jaakkola, T. (2003). On information regularization. *Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI03)*.

Corduneanu, A., & Jaakkola, T. S. (2005). Distributed information regularization on graphs. In L. K. Saul, Y. Weiss and L. Bottou (Eds.), *Advances in neural information processing systems 17*. Cambridge, MA: MIT Press.

Cozman, F., Cohen, I., & Cirelo, M. (2003). Semi-supervised learning of mixture models. *ICML-03, 20th International Conference on Machine Learning*.

Cristianini, N., Shawe-Taylor, J., Elisseeff, A., & Kandola, J. (2001a). On kernel-target alignment. *Advances in NIPS*.

Cristianini, N., Shawe-Taylor, J., & Lodhi, H. (2001b). Latent semantic kernels. *Proc. 18th International Conf. on Machine Learning*.

Dara, R., Kremer, S., & Stacey, D. (2000). Clsutering unlabeled data with SOMs improves classification of labeled real-world data. submitted.

Delalleau, O., Bengio, Y., & Roux, N. L. (2005). Efficient non-parametric function induction in semi-supervised learning. *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTAT 2005)*.

Demirez, A., & Bennettt, K. (2000). Optimization approaches to semisupervised learning. In M. Ferris, O. Mangasarian and J. Pang (Eds.), *Applications and algorithms of complementarity*. Boston: Kluwer Academic Publishers.

Demiriz, A., Bennett, K., & Embrechts, M. (1999). Semi-supervised clustering using genetic algorithms. *Proceedings of Artificial Neural Networks in Engineering*.

Dempster, A., Laird, N., & Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*.

Donoho, D. L., & Grimes, C. E. (2003). Hessian eigenmaps: locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Arts and Sciences*, *100*, 5591–5596.

Doyle, P., & Snell, J. (1984). *Random walks and electric networks*. Mathematical Assoc. of America.

Fowlkes, C., Belongie, S., Chung, F., & Malik, J. (2004). Spectral grouping using the Nyström method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *26*, 214–225.

Freund, Y., Seung, H. S., Shamir, E., & Tishby, N. (1997). Selective sampling using the query by committee algorithm. *Machine Learning*, *28*, 133–168.

Fung, G., & Mangasarian, O. (1999). *Semi-supervised support vector machines for unlabeled data classification* (Technical Report 99-05). Data Mining Institute, University of Wisconsin Madison.

Goldman, S., & Zhou, Y. (2000). Enhancing supervised learning with unlabeled data. *Proc. 17th International Conf. on Machine Learning* (pp. 327–334). Morgan Kaufmann, San Francisco, CA.

Grady, L., & Funka-Lea, G. (2004). Multi-label image segmentation for medical applications based on graph-theoretic electrical potentials. *ECCV 2004 workshop*.

Grandvalet, Y., & Bengio, Y. (2005). Semi-supervised learning by entropy minimization. In L. K. Saul, Y. Weiss and L. Bottou (Eds.), *Advances in neural information processing systems 17*. Cambridge, MA: MIT Press.

Grira, N., Crucianu, M., & Boujemaa, N. (2004). Unsupervised and semi-supervised clustering: a brief survey. in 'A Review of Machine Learning Techniques for Processing Multimedia Content', Report of the MUSCLE European Network of Excellence (FP6).

Gunn, S. R. (1997). *Support vector machines for classification and regression* (Technical Report). Image Speech and Intelligent Systems Research Group, University of Southampton.

Herbrich, R. (2002). *Learning kernel classifiers*. The MIT press.

Hofmann, T. (1999). Probabilistic latent semantic analysis. *Proc. of Uncertainty in Artificial Intelligence, UAI'99*. Stockholm.

Hull, J. J. (1994). A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 16*.

Jaakkola, T., Meila, M., & Jebara, T. (1999). Maximum entropy discrimination. *Neural Information Processing Systems, 12, 12*.

Joachims, T. (1999). Transductive inference for text classification using support vector machines. *Proc. 16th International Conf. on Machine Learning* (pp. 200–209). Morgan Kaufmann, San Francisco, CA.

Joachims, T. (2003). Transductive learning via spectral graph partitioning. *Proceedings of ICML-03, 20th International Conference on Machine Learning*.

Jones, R. (2005). *Learning to extract entities from labeled and unlabeled text* (Technical Report CMU-LTI-05-191). Carnegie Mellon University. Doctoral Dissertation.

Kemp, C., Griffiths, T., Stromsten, S., & Tenenbaum, J. (2003). Semi-supervised learning with trees. *Advances in Neural Information Processing System 16*.

Kimeldorf, G., & Wahba, G. (1971). Some results on Tchebychean spline functions. *J. Math. Anal. Applic., 33*, 82–95.

Kondor, R. I., & Lafferty, J. (2002). Diffusion kernels on graphs and other discrete input spaces. *Proc. 19th International Conf. on Machine Learning*.

Krishnapuram, B., Williams, D., Xue, Y., Hartemink, A., Carin, L., & Figueiredo, M. (2005). On semi-supervised classification. In L. K. Saul, Y. Weiss and L. Bottou (Eds.), *Advances in neural information processing systems 17*. Cambridge, MA: MIT Press.

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society* (pp. 48–50).

Lafferty, J., Zhu, X., & Liu, Y. (2004). Kernel conditional random fields: Representation and clique selection. *Proceedings of ICML-04, 21st International Conference on Machine Learning*.

Lanckriet, G., Cristianini, N., Bartlett, P., Ghaoui, L. E., & Jordan, M. (2004). Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research, 5*, 27–72.

Lawrence, N. D., & Jordan, M. I. (2005). Semi-supervised learning via Gaussian processes. In L. K. Saul, Y. Weiss and L. Bottou (Eds.), *Advances in neural information processing systems 17*. Cambridge, MA: MIT Press.

Le Cun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Howard, W., & Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. *Advances in Neural Information Processing Systems, 2*.

Levin, A., Lischinski, D., & Weiss, Y. (2004). Colorization using optimization. *ACM Transactions on Graphics*.

Lu, Q., & Getoor, L. (2003). Link-based classification using labeled and unlabeled data. *ICML 2003 workshop on The Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*.

MacKay, D. J. C. (1998). Introduction to Gaussian processes. In C. M. Bishop (Ed.), *Neural networks and machine learning*, NATO ASI Series, 133–166. Kluwer Academic Press.

MacKay, D. J. C. (2003). *Information theory, inference, and learning algorithms*. Cambridge.

Madani, O., Pennock, D. M., & Flake, G. W. (2005). Co-validation: Using model disagreement to validate classification algorithms. In L. K. Saul, Y. Weiss and L. Bottou (Eds.), *Advances in neural information processing systems 17*. Cambridge, MA: MIT Press.

Maeireizo, B., Litman, D., & Hwa, R. (2004). Co-training for predicting emotions with spoken dialogue data. *The Companion Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*.

Mahdaviani, M., de Freitas, N., Fraser, B., & Hamze, F. (2005). Fast computational methods for visually guided robots. *The 2005 International Conference on Robotics and Automation (ICRA)*.

McCallum, A. (2003). Efficiently inducing features of conditional random fields. *Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI03)*.

McCallum, A., & Nigam, K. (1998a). A comparison of event models for naive bayes text classification. *AAAI-98 Workshop on Learning for Text Categorization*.

McCallum, A. K. (1996). Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. http://www.cs.cmu.edu/ mccallum/bow.

McCallum, A. K., & Nigam, K. (1998b). Employing EM in pool-based active learning for text classification. *Proceedings of ICML-98, 15th International Conference on Machine Learning* (pp. 350–358). Madison, US: Morgan Kaufmann Publishers, San Francisco, US.

Miller, D., & Uyar, H. (1997). A mixture of experts classifier with learning based on both labelled and unlabelled data. *Advances in NIPS 9* (pp. 571–577).

Mitchell, T. (1999). The role of unlabeled data in supervised learning. *Proceedings of the Sixth International Colloquium on Cognitive Science*. San Sebastian, Spain.

Muslea, I., Minton, S., & Knoblock, C. (2002). Active + semi-supervised learning = robust multi-view learning. *Proceedings of ICML-02, 19th International Conference on Machine Learning* (pp. 435–442).

Ng, A., Jordan, M., & Weiss, Y. (2001a). On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems, 14*.

Ng, A. Y., Zheng, A. X., & Jordan, M. I. (2001b). Link analysis, eigenvectors and stability. *International Joint Conference on Artificial Intelligence (IJCAI)*.

Nigam, K. (2001). *Using unlabeled data to improve text classification* (Technical Report CMU-CS-01-126). Carnegie Mellon University. Doctoral Dissertation.

Nigam, K., & Ghani, R. (2000). Analyzing the effectiveness and applicability of co-training. *Ninth International Conference on Information and Knowledge Management* (pp. 86–93).

Nigam, K., McCallum, A. K., Thrun, S., & Mitchell, T. (2000). Text classification from labeled and unlabeled documents using EM. *Machine Learning, 39*, 103–134.

Niu, Z.-Y., Ji, D.-H., & Tan, C.-L. (2005). Word sense disambiguation using label propagation based semi-supervised learning. *Proceedings of the ACL*.

Pang, B., & Lee, L. (2004). A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. *Proceedings of the ACL* (pp. 271–278).

Rabiner, L. (1989). A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE, 77*, 257–285.

Ratsaby, J., & Venkatesh, S. (1995). Learning from a mixture of labeled and unlabeled examples with parametric side information. *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, 412–417.

Rattray, M. (2000). A model-based distance for clustering. *Proc. of International Joint Conference on Neural Networks*.

Riloff, E., Wiebe, J., & Wilson, T. (2003). Learning subjective nouns using extraction pattern bootstrapping. *Proceedings of the Seventh Conference on Natural Language Learning (CoNLL-2003)*.

Rosenberg, C., Hebert, M., & Schneiderman, H. (2005). Semi-supervised self-training of object detection models. *Seventh IEEE Workshop on Applications of Computer Vision*.

Rosset, S., Zhu, J., Zou, H., & Hastie, T. (2005). A method for inferring label sampling mechanisms in semi-supervised learning. In L. K. Saul, Y. Weiss and L. Bottou (Eds.), *Advances in neural information processing systems 17*. Cambridge, MA: MIT Press.

Roweis, S. T., & Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, *290*, 2323–2326.

Roy, N., & McCallum, A. (2001). Toward optimal active learning through sampling estimation of error reduction. *Proc. 18th International Conf. on Machine Learning* (pp. 441–448). Morgan Kaufmann, San Francisco, CA.

Saul, L. K., & Roweis, S. T. (2003). Think globally, fit locally: unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research, 4*, 119–155.

Schneiderman, H. (2004a). Feature-centric evaluation for efficient cascaded object detection. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Schneiderman, H. (2004b). Learning a restricted Bayesian network for object detection. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Schuurmans, D., & Southey, F. (2001). Metric-based methods for adaptive model selection and regularization. *Machine Learning, Special Issue on New Methods for Model Selection and Model Combination*, *48*, 51–84.

Seeger, M. (2001). *Learning with labeled and unlabeled data* (Technical Report). University of Edinburgh.

Shahshahani, B., & Landgrebe, D. (1994). The effect of unlabeled samples in reducing the small sample size problem and mitigating the Hughes phenomenon. *IEEE Trans. On Geoscience and Remote Sensing, 32*, 1087–1095.

Shi, J., & Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 22*, 888–905.

Smola, A., & Kondor, R. (2003). Kernels and regularization on graphs. *Conference on Learning Theory, COLT/KW*.

Sudderth, E., Wainwright, M., & Willsky, A. (2003). *Embedded trees: Estimation of Gaussian processes on graphs with cycles* (Technical Report 2562). MIT LIDS.

Szummer, M., & Jaakkola, T. (2001). Partially labeled classification with Markov random walks. *Advances in Neural Information Processing Systems, 14*.

Szummer, M., & Jaakkola, T. (2002). Information regularization with partially labeled data. *Advances in Neural Information Processing Systems, 15*.

Taskar, B., Guestrin, C., & Koller, D. (2003). Max-margin Markov networks. *NIPS'03*.

Tenenbaum, J. B., de Silva, V., , & Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science, 290*, 2319–2323.

Tipping, M. (1999). Deriving cluster analytic distance functions from Gaussian mixture models.

Tong, S., & Koller, D. (2000). Support vector machine active learning with applications to text classification. *Proceedings of ICML-00, 17th International Conference on Machine Learning* (pp. 999–1006). Stanford, US: Morgan Kaufmann Publishers, San Francisco, US.

Vapnik, V. (1998). *Statistical learning theory*. Springer.

von Luxburg, U., Belkin, M., & Bousquet, O. (2004). *Consistency of spectral clustering* (Technical Report TR-134). Max Planck Institute for Biological Cybernetics.

von Luxburg, U., Bousquet, O., & Belkin, M. (2005). Limits of spectral clustering. In L. K. Saul, Y. Weiss and L. Bottou (Eds.), *Advances in neural information processing systems 17*. Cambridge, MA: MIT Press.

Weinberger, K. Q., Packer, B. D., & Saul, L. K. (2005). Nonlinear dimensionality reduction by semidefinite programming and kernel matrix factorization. *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTAT 2005)*.

Weinberger, K. Q., & Saul, L. K. (2004). Unsupervised learning of image manifolds by semidefinite programming. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 988–995).

Weinberger, K. Q., Sha, F., & Saul, L. K. (2004). Learning a kernel matrix for nonlinear dimensionality reduction. *Proceedings of ICML-05* (pp. 839–846).

Weiss, Y. (1999). Segmentation using eigenvectors: A unifying view. *ICCV (2)* (pp. 975–982).

Weiss, Y., & Freeman, W. T. (2001). Correctness of belief propagation in Gaussian graphical models of arbitrary topology. *Neural Computation, 13*, 2173–2200.

Weston, J., Leslie, C., Zhou, D., Elisseeff, A., & Noble, W. S. (2004). Semi-supervised protein classification using cluster kernels. In S. Thrun, L. Saul and B. Schölkopf (Eds.), *Advances in neural information processing systems 16*. Cambridge, MA: MIT Press.

Williams, C. K. I., & Barber, D. (1998). Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 20*, 1342–1351.

Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics* (pp. 189–196).

Yianilos, P. (1995). *Metric learning via normal mixtures* (Technical Report). NEC Research Institute.

Zelikovitz, S., & Hirsh, H. (2001). Improving text classification with LSI using background knowledge. *IJCAI01 Workshop Notes on Text Learning: Beyond Supervision*.

Zhai, C. (2001). Notes on the Lemur TFIDF model. http://www.cs.cmu.edu/∼lemur/3.1/tfidf.ps.

Zhang, T., & Oles, F. J. (2000). A probability analysis on the value of unlabeled data for classification problems. *Proc. 17th International Conf. on Machine Learning* (pp. 1191–1198). Morgan Kaufmann, San Francisco, CA.

Zhou, D., Bousquet, O., Lal, T., Weston, J., & Schlkopf, B. (2004a). Learning with local and global consistency. *Advances in Neural Information Processing System 16*.

Zhou, D., Schölkopf, B., & Hofmann, T. (2005). Semi-supervised learning on directed graphs. In L. K. Saul, Y. Weiss and L. Bottou (Eds.), *Advances in neural information processing systems 17*. Cambridge, MA: MIT Press.

Zhou, D., Weston, J., Gretton, A., Bousquet, O., & Schlkopf, B. (2004b). Ranking on data manifolds. *Advances in Neural Information Processing System 16*.

Zhu, J., & Hastie, T. (2001). Kernel logistic regression and the import vector machine. *NIPS 2001*.

Zhu, X., & Ghahramani, Z. (2002a). *Learning from labeled and unlabeled data with label propagation* (Technical Report CMU-CALD-02-107). Carnegie Mellon University.

Zhu, X., & Ghahramani, Z. (2002b). *Towards semi-supervised classification with Markov random fields* (Technical Report CMU-CALD-02-106). Carnegie Mellon University.

Zhu, X., Ghahramani, Z., & Lafferty, J. (2003a). Semi-supervised learning using Gaussian fields and harmonic functions. *ICML-03, 20th International Conference on Machine Learning*.

Zhu, X., Kandola, J., Ghahramani, Z., & Lafferty, J. (2005). Nonparametric transforms of graph kernels for semi-supervised learning. In L. K. Saul, Y. Weiss and L. Bottou (Eds.), *Advances in neural information processing systems 17*. Cambridge, MA: MIT Press.

Zhu, X., Lafferty, J., & Ghahramani, Z. (2003b). Combining active learning and semi-supervised learning using Gaussian fields and harmonic functions. *ICML 2003 workshop on The Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*.

Zhu, X., Lafferty, J., & Ghahramani, Z. (2003c). *Semi-supervised learning: From Gaussian fields to Gaussian processes* (Technical Report CMU-CS-03-175). Carnegie Mellon University.

Notation

Δ	combinatorial graph Laplacian
$\tilde{\Delta}$	smoothed Laplacian
α	length scale hyperparameter for edge weights
β	inverse temperature parameter for Gaussian random fields
γ	steepness parameter for the Gaussian process noise model
η	transition probability to the dongle node
θ_m	component class membership $P(y = 1\|m)$ for mixture models
λ	eigenvalues of the Laplacian
μ	optimal spectrum transformation of the Laplacian
σ	smoothing parameter for the graph Laplacian kernel
ϕ	eigenvectors of the Laplacian
D	diagonal degree matrix of a graph
E	energy function on a graph
K	kernel
L	labeled data
$\mathcal{L}$	log likelihood of mixture models
$\mathcal{O}$	combined log likelihood and graph energy objective
P	transition matrix of a graph
$\mathbf{R}$	responsibility of mixture components, $R_{im} = P(m\|i)$
$\mathcal{R}$	risk, the estimated generalization error of the Bayes classifier
U	unlabeled data
W	weight matrix of a graph
f	arbitrary real functions on the graph
$\mathbf{g}_k$	the graph for semi-supervised learning
$\mathbf{g}_s$	the graph encoding sequence structure in KCRFs
h	harmonic function
l	labeled data size
m	length of a sequence
n	total size of labeled and unlabeled data
r	spectral transformation function to turn Laplacian into a kernel
u	unlabeled data size
w	edge weight in a graph
x	Features of a data point
y	Target value. In classification it is the (discrete) class label

Index