HYPER-PARAMETER LEARNING FOR GRAPH BASED SEMI-SUPERVISED LEARNING ALGORITHMS

ZHANG XINHUA

(B. Eng., Shanghai Jiao Tong University, China)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2006

Acknowledgements

First of all, I wish to express my heartfelt gratitude to my supervisor Prof. Lee Wee Sun who guided me into the research of machine learning. When I first walked into his office, I had only limited knowledge about the ad hoc early-time learning algorithms. He introduced me to the state-of-the-art in the current machine learning community, such as graphical models, maximum entropy models and maximum margin models. He is always patient and open to hear my (immature) ideas, obstacles, and then pose corrections, suggestions and/or solutions. He is full of wonderful ideas and energetic with many emails off office hour and even at small hours. I am always impressed by his mathematical rigor, sharp thinking and insight. He gave me a lot of freedom and sustained encouragement to pursue my curiosity, not only in the materials presented in this thesis, but also much other work before it.

I would also like to thank my Graduate Research Paper examiners Prof. Rudy Setiono and Prof. Ng Hwee Tou, who asked pithy questions during my presentation, commented on the work and advised on further study. I also want to express my thanks to Prof. Ng for generously letting me use the Matlab Compiler on his Linux server *twinkle*. This tool significantly boosted the efficiency of implementation and experiments.

Moreover, the graph reading group co-organized by Prof. Lee Wee Sun, Dr. Teh Yee Whye, Prof. Ng Hwee Tou, and Prof. Kan Min Yen has been very enriching for broadening and deepening my knowledge in graphical models. These biweekly discussions bring together faculty and students with shared interest, and offer an opportunity to clarify puzzles and exchange ideas.

Last (almost) but not least, I wish to thank my fellow students and friends, the collaboration with whom made my Master's study life a very memorable experience. Mr. Chieu Hai Leong helped clarify the natural language processing concepts especially on named entity recognition. He also helped me warm-heartedly with a lot of practical problems such as implementation, the usage of computing clusters and MPI. The discussions with Dr. Dell Zhang on graph regularization are also enlightening.

Finally, I owe great gratitude to the School of Computing and Singapore-MIT Alliance which provided latest software, high-performance computing clusters and technical support for research purposes. Without these facilities, the experiments would have been prolonged to a few months.

Table of Contents

Summary

Over the past few years, semi-supervised learning has gained considerable interest and success in both theory and practice. Traditional supervised machine learning algorithms can only make use of labelled data, and reasonable performance is often achieved only when there is a large number of labelled data, which can be expensive, labour and time consuming to collect. However, unlabelled data is usually cheaper and easier to obtain, though it lacks the most important information: label. The strength of semi-supervised learning algorithms lies in its ability to utilize a large quantity of unlabelled data to effectively and efficiently improve learning.

Recently, graph based semi-supervised learning algorithms are being intensively studied, thanks to its convenient local representation, connection with other models like kernel machines, and applications in various tasks like classification, clustering and dimensionality reduction, which naturally incorporates the advantages of unsupervised learning into supervised learning.

Despite the abundance of graph based semi-supervised learning algorithms, the fundamental problem of graph construction, which significantly influences performance, is underdeveloped. In this thesis, we tackle this problem under the task of classification by learning the hyperparameters of a fixed parametric similarity measure, based on the commonly used low leave-one-out error criterion. The main contribution includes an efficient algorithm which significantly reduces the computational complexity, a problem that plagues most leave-one-out style algorithms. We also propose several novel approaches for graph learning regularization, which is so far a less explored field as well. Experimental results show that our graph learning algorithms improve classification accuracy compared with graphs selected by cross validation, and also beat the preliminary graph learning algorithms in literature.

List of Tables

List of Figures

Chapter 1 Introduction and Literature Review

In this chapter, we review some literature on semi-supervised learning. As an introduction, we first take a look at the general picture of machine learning, in order to see the position, role and motivation of semi-supervised learning in the whole spectrum. Then we focus on various algorithms in semi-supervised learning. There is a good review on this topic in (Zhu, 2005). However, we will incorporate some more recently published work, use our own interpretation and organization, and tailor the presentation catering for our original work in this thesis. Careful consideration is paid to the breadth/completeness of the survey and the relevance to our own work.

1.1 Motivation and definition of semi-supervised learning in machine learning

The first question to ask is what is semi-supervised learning and why do we study it. To answer this question, it is useful to take a brief look at the big picture of machine learning and to understand the unsolved challenges. We do not plan to give any rigorous definition of most terminologies (if such definition exists). Instead, we will use a running example, news web page interest learning, to illustrate the different facets of machine learning, including tasks, styles, form and availability of data and label.

Suppose there is a news agency which publishes their news online. The subscribers can log onto their own accounts (not free unfortunately) and read the news they are interested in. On each web page, there are 10 pieces of news, each with a title, a short abstract and a link

to the full text. Once the reader gets interested in a certain piece of news after reading the abstract, he/she will click into the full story. So the web site can know whether the reader is interested in a piece of news by measuring the clicks. So by "*news*", we only refer to the abstract in the form of *plain text* for simplicity.

To promote subscription, the news agency would like to find out the interest of the readers in order to streamline the collection, and composition of the news pool to satisfy common interests. Knowing the readers' interest will also allow the agency to present the news catering for each individual's preference, e.g., in what order should the news be presented, whether some links to background information or related news need to be inserted and what they are. Unfortunately, it is normally not advisable to ask readers to fill up a questionnaire. Firstly, most readers feel it far easier to say whether a specific piece of news is interesting, than to generalize and precisely describe their interest. Secondly, readers' interest may change from time to time and it is impossible to ask readers to update the personal interest information regularly, because besides inconvenience, the change may be too subtle to be noticed by readers themselves. Thirdly, the less we ask our subscribers to do, the better. Thus, it is highly desirable to automatically *learn* the interest of the readers from their past read news. As the simplest case adopted in the following section 1.1.1, we only care about the crisp classification: interested or not. To motivate the use of semi-supervised learning, we first study a variety of different scenarios of learning, according to the availability of data (text) and label (interest).

1.1.1 Different learning scenarios: classified by availability of data and label

In this section we divide the learning scenarios based on the availability of data and label in different phases of experiment and the different querying/sampling mechanism.

1. Supervised learning

When a subscriber first logs onto the web site, the best we can do is just to list some commonly interesting news. Then we record which news articles attracted the reader and which do not. Finally we obtain a set of tuples $<x_i, y_i>$, where x_i stands for the news i (precisely, title and abstract) and y_i stands for interested in x_i or not. Then we learn a *function* which can predict whether the reader will be interested in an *unseen* piece of news x. At the second time the user logs on, the system can organize the news catering for the user's interest learned from the first time. Note the function can be in any form: rule based, real-valued function with a threshold like a neural network, or a segment of program learned by genetic algorithm, etc., as long as it is computable.

2. Unsupervised learning

In this setting, the readers are not involved. We can imagine it as an offline process on the new agency's web server. The goal is to learn an inner structure of the news articles, which includes clustering similar articles into groups, reducing the dimensionality of the news pool by mapping news into a smaller set of synopsis on certain facets such as: who, when, where, category (sports/entertainment/politics), length, etc. In theory, these facets (or more rigorously called dimensions in lower dimensional spaces), are not necessarily ex-

plicitly defined and their distance on a learned manifold is of more interest.

3. Semi-supervised learning

So far so good, at least in task specification. However, in supervised learning, it is usually necessary have a large set of labelled data before reasonable prediction accuracy can be achieved. For example, in the UseNet news experiment, (Lang, 1995) found that after a person read and hand-labelled about 1000 articles, a learned classifier achieved precision of about 50% when making predictions for only the top 10% of documents about which it was most confident. Obviously, no reader or news agency will accept such a low learning rate. A dozen articles may be the upper limit that they can tolerate. Other examples include[1]:

- **Satellite surveillance**. In geographical information systems (GIS), remote imaging and sensing are widely used to detect various natural structures and objects. However, manually identifying objects in an image costs huge amount of time and effort, because it usually involves field trip on a regular basis. On the other hand, modern sensors can easily obtain a huge amount of unlabelled signals.
- **Word sense disambiguation**. It is labour consuming to determine the meaning of a word in a given context. (Mihalcea & Chklovski, 2003) estimated that 80 person-year is need to create manually labelled training data for about 20,000 words in a common English dictionary. But documents and sentences are readily available in large amount.
- **Predicting compounds' ability to bind** to a target site on a receptor for drug discovery. It usually costs considerable expert human resources and financial expense to test the

[1] Note for some cases, the labelling is easy but the data is complex to describe. See direction 1 in Section 1.1.2.

binding activity. But the compound database is established and accessible.

- **Handwritten character or speech recognition**. With computers being more and more user friendly, new input devices such as speech input and handwritten input are becoming popular. However, it is extremely time consuming to annotate the speech utterance at either phonetic level or word level. For some languages like Chinese which is not based on a small alphabet and spelling, the task of labelling characters is complicated.

On the other hand, in all of these cases (and many other cases) data without labels is usually readily and cheaply available in large quantity. Therefore incorporating these unlabelled data to learn a classifier will be very desirable if it does improve the performance. The following two settings 3.1 and 3.2 are two concrete scenarios.

3.1. Transductive learning

This setting is the same as supervised learning, except that the pool of today's news which has not been displayed to the reader is also incorporated in the process of learning the classifier. However the important feature of transductive learning is that the testing examples, or the candidate set of news being evaluated of their appeal to the reader, are also known at the learning phase. The learning process just labels the candidate news without outputting a decision function, which was emphasized in the discussion of supervised learning in section 1.1.1. So when new unseen news is composed, the whole system must be learned again.

3.2. Inductive semi-supervised learning.

This is also similar to supervised learning. It is different from transductive learning in that it does learn a function using labelled and unlabelled data, so it is easy to classify an unseen example by just feeding it to the function. Normally this function does not change after classifying the new unseen example.

4. Online learning

The crux of online learning is to update the classifier immediately after a testing example is given, while its label can be given immediately or not. In our web news interest example, when the reader chooses to view the next screen of 10 abstracts, the classifier will be re-trained by making use of the reader's interest in the news of the last screen. The updated classifier will be used to choose another 10 articles for the next screen. The key factor here is efficiency, because no reader will be willing to wait for long if such re-training will take more than a few seconds. So most online learning algorithms are based on incremental and/or local update (Kohonen, 2000). In contrast to online learning, offline learning means that the classifier is learned on the server when the reader is logged off. Therefore the efficiency may not be the bottleneck in offline learning.

5. Reinforcement learning

We consider an interesting situation, making one additional assumption that there is an identifiable group of subscribers who have the same interest pattern (called side information). A reader of the group is a very busy CEO, and he only had time to skim the web pages,

reading the abstracts only and seldom clicks into a full text. So the web site cannot get any response to its classification performance. However, after one month, the reader suddenly decided to unsubscribe. Only at that moment can the system realize that it had been doing an unsatisfactory job, and it can still learn from it to serve other subscribers in that group of similar interest better. The most difficult challenge in reinforcement learning is to find a good trade-off between *exploitation*: the reader did not unsubscribe so we just stick to the current classifier; and *exploration*: why not try offering some news which is now regarded as uninteresting by the *im*perfect classifier in case it would make the reader more satisfied and recommend subscription for his friends or relatives, though at the risk of selecting even less interesting news which finally results in unsubscription.

6. Active learning

This form of learning is very similar to reinforcement learning. Now suppose the user offers to, or we are sure that the user is willing to, collaborate with the learner and answer whether he is interested in an article *selected by the learning system*. Since this opportunity is rare and valuable (say, when the reader just paid next month's subscription fee), we should be very careful in picking the most informative unlabelled articles, i.e., knowing the label of which will most significantly improve the classifier. One early and popular approach is called query-by-committee (QBC) (Freund et al., 1997; Seung et al., 1992), whereby an ensemble of classifiers are built and the examples with the highest classification variance are selected. (Freund et al., 1997) showed theoretically that if there is no error in labelling, QBC can exponentially reduce the number of labelled examples needed for learning. As for approaches

based on a single classifier instead of a committee, (Lewis & Gale, 1994; Lewis, 1995) examined pool-based *uncertainty sampling* and *relevance sampling*. (Schohn & Cohn, 2000) used an approach for SVM, selecting the example that is closest to the linear decision boundary given by the classifier. (Tong & Koller, 2001) used SVM as well, but did selection to maximally reduce the size of the version space of good hypotheses.

1.1.2 Learning tasks benefiting from semi-supervised learning

With semi-supervised learning well motivated, we are interested in what kind of learning tasks it can help. Can we benefit from unlabelled data in tasks other than classification? The answer is yes. Almost all classical machine learning tasks have been shown to *be able to* benefit from unlabelled data points, though hurting of performance is also frequently reported. We mention a few tasks which are related to the thesis. As semi-supervised learning stands between supervised and unsupervised learning, it is expected that unsupervised learning techniques will also be helpful and are thus studied.

- **Clustering and dimensionality reduction**

We have discussed clustering and dimensionality reduction in section 1.1 under unsupervised learning. We just want to emphasize that these tasks can also be carried out in a supervised or semi-supervised setting, because it is usually beneficial to let the labels guide the process (Grira et al., 2004).

- **Regression**

In this task, we are more ambitious and want to learn the extent to which the reader is interested. This can be any real number or bounded in [0, 1] up to definition. For example, readers can spend a long time (normalized by word number) reading one article or can manually score an article by 1 star to 5 stars.

- **Ranking**

Suppose that each screen can display at most 10 pieces of news. Then we want to pick out the 10 articles which are of top interest to the reader. This task is different from classification in that it only concerns the ranking, and performance is not measured on the non-top-10 news.

Up to now, we have been making a simplified assumption that the data is just a plain-text article and the label is just a Boolean value. Extensions can be made in two directions relating to this thesis:

1. Feature engineering. The article can contain photos, video and audio, etc. So how to integrate the heterogeneous forms of the data into a well organized ensemble of useful features is a problem. For example, in bioinformatics a gene can have: DNA, mRNA and amino acid sequences, expression profile in microarray experiments, location in a partially known genetic network, location in a protein-protein interaction network, gene ontology classification, and so on (De Bie, 2005).
2. Structured, especially sequential data. One example of this second direction of extension is to look at the news in sequel, rather than in separation. There is often a series

of reports in newspapers, and the appeal of each article is dependent on the articles in the previous issues and on what it promises to cover in the forthcoming issues.

In this thesis, we only focus on the simple data representation, assuming that each data point is just a real vector with a Boolean label. Realizing the motivation of semi-supervised learning, the first few natural questions to ask are whether it really helps, why it helps and how it helps.

1.2 Why do unlabelled data help: an intuitive insight first

In this section, we just give some intuitive explanations of the questions, together with examples. Detailed algorithms will be reviewed from the next section. At first glance, it might seem that nothing is to be gained from unlabelled data. After all, an unlabelled document does not contain the most important piece of information ─ its label. But thinking in another way may reveal the value of unlabelled data. In the field of information retrieval, it is well known that words in natural language occur in strong co-occurrence patterns (van Rijsbergen, 1977). Some words are likely to occur together in one document, others are not. For example, when asking search engine *Google* about all web pages containing the words *sugar* and *sauce*, it returns 1,390,000 results. When asking for the documents with the words *sugar* and *math*, we get only 191,000 results, though *math* is a more popular word on the web than *sauce*. Suppose we are interested in recognizing web pages about cuisine. We are given just a few known cuisine and non-cuisine web pages, along with a large number of web pages that are unlabelled. By looking at the labelled data only, we

determine that pages containing the word *sugar* tend to be about cuisine. If we use this fact to estimate the class of many unlabelled web pages, we might find that the word *sauce* occurs frequently in the unlabelled examples that are now believed to be about cuisine. This co-occurrence of the words *sugar* and *sauce* over the large set of unlabelled training data can provide useful information to construct a more accurate classifier that considers both *sugar* and *sauce* as indicators of positive examples.

In this thesis, we show how unlabelled data can be used to improve classification accuracy, especially when labelled data are scarce. Formally, unlabelled data only informs us of the distribution of examples in input space. In the most general case, distributional knowledge will not provide helpful information to supervised learning. Consider classifying uniformly distributed instances based on conjunctions of literals. Here there is no relationship between the uniform instance distribution and the space of possible classification tasks, thus unlabelled data obviously cannot help.

No matter how justified it is, the co-occurrence is just an assumption we make. We need to introduce appropriate assumptions/biases into our learner about some dependence between the instance distribution and the classification task. Even standard supervised learning with labelled data must introduce bias in order to learn. In a well-known and often-proven result in (Watanabe, 1969), the theorem of the Ugly Duckling shows that without bias all pairs of training examples look equally similar, and generalization into classes is impossible. This result was foreshadowed long ago by both William of Ockham's philosophy of radical

nominalism in the 1300's and by David Hume in the 1700's. Somewhat more recently, (Zhang & Oles, 2000) formalized and proved that supervised learning with unlabelled examples must assume a dependence between the instance distribution and the classification task. In this thesis, we will look at several assumptions of such dependence.

We now go on to the survey of semi-supervised learning algorithms, restricted to classification. We adopt the normal criterion to divide them into generative and discriminative model. In the section after these general reviews, we will focus on graph based semi-supervised learning algorithms, extending the focus from classification to clustering, which are the main topics of the thesis. After these high level algorithms are introduced, we *summarize* and review the optimization and inference techniques used therein. On top of general semi-supervised learning algorithms, wrapping algorithms will be surveyed and finally we briefly mention some investigation into the theoretical value of unlabelled data.

1.3 Generative models for semi-supervised learning

Perhaps the first kind of semi-supervised learning models proposed in machine learning research are the generative models. Generative learning models the probability of generating a data example for each class. Given a data example, the posterior probability of its belonging to each class is then calculated using Bayes rule. Mathematically, it assumes

$$p(x,y) = p(y)\,p(x \mid y) \quad \text{and} \quad p(x) = \sum_y p(y)\,p(x \mid y).$$

The main assumptions include two parts:

1. How many different discrete y values to take on, also called components, generators,

clusters, topics, or mixtures. Using too few components will fail to capture the underlying structure while too many clusters will cause overfitting.

2. The parametric form of $p(x|y)$ (e.g., Gaussian distribution). One crucial requirement on this choice is that the model $p(x, y)$ (not $p(x|y)$) must be identifiable, i.e., for a family of distributions $\{p_\theta(x, y)\}$ parameterized by θ, $p_{\theta_1}(x,y) = p_{\theta_2}(x,y)$ implies $\theta_1 = \theta_2$ up to a permutation of mixture components. Otherwise, one can contrive an example $p_{\theta_1}(x,y)$, $p_{\theta_2}(x,y)$ and some observations, where the total likelihood of the observations is the same for $p_{\theta_1}(x,y)$ and $p_{\theta_2}(x,y)$. However, there exists a point x_0 such that $p_{\theta_1}(y|x_0) > p_{\theta_1}(y'|x_0)$ and $p_{\theta_2}(y|x_0) < p_{\theta_2}(y'|x_0)$, i.e., contradictory decisions are made. Mixture of Gaussian is identifiable, and mixture of Bernoulli or uniform is not identifiable (Corduneanu & Jaakkola, 2001; McCallum & Nigam, 1998; Ratsaby & Venkatesh, 1995).

Then what to be estimated are the values of $p(y)$, and the parameters of $p(x|y)$ (e.g., mean, covariance), typically estimated by maximizing likelihood: $\prod_i p(x_i, y_i)$ for supervised learning, by using the expectation maximization algorithm (see section 1.7). The main reason why generative models are attractive for semi-supervised learning and thus historically earlier used to this end is that it can naturally incorporate unlabelled data into the model and evidence by:

$$\prod_{i \in labeled} p(x_i, y_i) \cdot \prod_{i \in unlabeled} p(x_i) = \prod_{i \in labeled} p(x_i | y_i) p(y_i) \cdot \prod_{i \in unlabeled} \sum_y p(x_i | y) p(y).$$

If we model by $p(y|x)$, then

$$\prod_{i \in labeled} p(x_i, y_i) \cdot \prod_{i \in unlabeled} p(x_i) = \prod_{i \in labeled} p(y_i | x_i) p(x_i) \cdot \prod_{i \in unlabeled} p(x_i) \sum_y p(y | x_i)$$

where $\sum_y p(y|x_i)$ is trivially equal to 1, and the unlabelled data cannot be utilized straightforwardly. (Miller & Uyar, 1996; Nigam et al., 1998; Baluja, 1999) started using maximum likelihood of mixture models to combine labelled and unlabelled data for classification. (Ghahramani & Jordan, 1994) used EM to fill in missing feature values of examples when learning from incomplete data by assuming a mixture model. Generative mixture models have also been used for unsupervised clustering, whose parameters have been estimated with EM (Cheeseman et al., 1988; Cheeseman & Stutz, 1996; Hofmann & Puzicha, 1998).

When model assumptions are correct, the generative models can usually learn fast because their assumptions are quite strong and thus the model is quite restrictive, especially with a fixed parametric form $p(x|y)$. Moreover, as desired, unlabelled data will guarantee the improvement of performance (Castelli & Cover, 1996; Ratsaby & Venkatesh, 1995; Castelli & Cover, 1995). However, high benefit is usually at high risk. If the model assumption deviates from the reality, then it will not be able to learn to satisfactory performance and unlabelled data may hurt accuracy. Putting it another way for generative models specifically, the classification accuracy and model likelihood should be correlated in order to make unlabelled data useful. The details of the undesired role of unlabelled data will be discussed in section 1.8.

To make the assumptions more realistic, a number of methods have been proposed. (Jordan & Jacobs, 1994) proposed hierarchical mixture-of-experts that are similar to mixture models,

and their parameters are also typically set with EM. (Nigam, 2001; Nigam et al., 1999) proposed introducing hierarchical class relationship: sub-topics under one relatively broad topic (i.e., using multiple multinomials), or inversely, super-topics to generalize similar topics. The first comprehensive work of semi-supervised generative model for text classification is (Nigam et al., 1999b; Nigam, 2001). For more examples, see (Miller & Uyar, 1996; Shahshahani & Landgrebe, 1994).

Finally, as a historical note, though the idea of generative models has been introduced to the machine learning community for semi-supervised learning for about 10 years, it is not new in the statistics community. At least as early as 1968, it was suggested that labelled and unlabelled data could be combined for building classifiers with likelihood maximization by testing all possible class assignments (Hartley & Rao, 1968; Day, 1969). (Day, 1969) presented an iterative EM-like approach for parameters of a mixture of two Gaussian distributions with known covariances from unlabelled data alone. Similar iterative algorithms for building maximum likelihood classifiers from labelled and unlabelled data are primarily based on mixtures of normal distributions (McLachlan, 1975; Titterington, 1976).

1.4 Discriminative models for semi-supervised learning

Instead of learning $p(x|y)$ and then calculating $p(y|x)$ by Bayes rule as in generative models, the discriminative models directly learn $p(y|x)$. For a lot of real world problems, we can not model the input distribution with sufficient accuracy or the number of parameters to estimate for generative model is too large compared with the training data at hand. Then a

practical approach is to build classifiers by directly calculating its probability of belonging to each class, $p(y|x)$. However, this makes it less straightforward to incorporate unlabelled data, whose only informative contribution lies in $p(x)$. (Seeger, 2000) pointed out that if $p(x)$ and $p(y|x)$ do not share parameters (coupled), then $p(x)$ will be left outside of the parameter estimation process, leading to unlabelled data being ignored.

Using this interpretation, different discriminative semi-supervised learning algorithms can be divided according to their assumption over the relationship between $p(x)$ and $p(y|x)$, i.e., how they are coupled. As commonly used in literature especially for two class classifications, $p(y|x)$ is modelled by a real-valued soft label $f(x)$, e.g.,

$$p(y=1|x,f) \triangleq \frac{\exp(\alpha f(x))}{1+\exp(\alpha f(x))}, \quad p(y=0|x,f) \triangleq \frac{1}{1+\exp(\alpha f(x))} \quad (\alpha > 0), \quad (1.1)$$

and the hard label of x (positive/negative) is determined by applying some decision rules to f (e.g., sign function or thresholding). So now we look at $p(x)$ and $f(x)$. Besides some general requirements on $f(x)$ as in supervised learning, $f(x)$ is assumed to satisfy some additional conditions in semi-supervised learning. We now use these various assumptions to divide the wide range of discriminative models. Note that some assumptions are just restatement of some others, and we present these assumptions from as many different perspectives as possible, because in practice, different expressions can result in different levels of difficulty in model design or mathematical formulation.

1. $f(x)$ yields a confidently correct hard label on labelled data

This first requirement is usually a problem of choosing the loss function, with typical

choices as L_1, L_2 loss , ε-insensitive loss (Smola & Schölkopf, 2003; Vapnik, 1999; Vapnik, 1998), or even incorporating a noise model, e.g., Gaussian noise, sigmoid noise (Zhu et al., 2003b), flipping noise (Kapoor et al., 2005), etc. Generally speaking, this loss alone will not make use of the unlabelled data.

2. $f(x)$ yields a confident labelling of unlabelled data by entropy minimization

(Zhu et al., 2003a) proposed learning the hyperparameters by minimizing the average label entropy of unlabelled data. Lower entropy means more confident labelling. In (Grandvalet & Bengio, 2004), entropy minimization was introduced as a regularizer or prior. Unfortunately, entropy minimization does not have unique solutions because for a single random variable x, $p(x=1) \rightarrow 1$ and $p(x=0) \rightarrow 1$ will both have small entropy. This assumption also implies that different classes overlap at a low level, which may not be realistic in some cases. For example, in two heavily overlapping Gaussians, the separator goes exactly across the most ambiguous region and the unlabelled data there should have high entropy. This constitutes a counter example for most of the following assumptions. However, it can be easily learned by generative models like two mixing Gaussians.

3. Both labelled and unlabelled data should have a large distance from the decision boundary of $f(x)$ (maximum margin)

This criterion is very similar to the previous one as large distance from decision boundary usually means confident labelling. A simplification of $f(x)$ is as a linear function $w^T x+b$ and its decision boundary is a hyperplane. Now this assumption means that the linear

separator should maximize the margin over both the labelled and unlabelled examples (and of course place the labelled examples on the correct side). A transductive support vector machine (TSVM) (Vapnik, 1998) is designed by using this philosophy. *The contribution of unlabelled data is to expel the separating hyperplane away from them.* However, as in the assumption 2, the solution to such a maximum margin separator is not unique, because an unlabelled point can lie on two different sides of the hyperplane with the same distance.

Moreover, finding the optimal hyperplane is NP-hard, so one must fall back on approximate algorithms. (Joachims, 1999) designed an iterative relaxation algorithm and demonstrated the efficacy of this approach for several text classification tasks. (Bennett & Demiriz, 1998) proposed a computationally easier variant of TSVM and found small improvements on some datasets. Other works include (Chapelle & Zien, 2005; Fung & Mangasarian, 1999). However, (Zhang & Oles, 2000) argues that TSVMs are asymptotically unlikely to be helpful for classification in general, both experimentally and theoretically (via Fisher's information matrices) because its assumption may be violated. (Seeger, 2000) also cast the similar doubt.

4. $f(x)$ should not change quickly in regions where $p(x)$ is high

This idea has been widely used in supervised learning, if we deem a "quickly changing" f as a "complex" one. Balancing hypothesis space generality with predictive power is one of the central tasks in inductive learning. The difficulties that arise in seeking an appropriate tradeoff go by a variety of names: overfitting, data snooping, memorization, bias/variance

tradeoff, etc. They lead to a number of known solution techniques or philosophies, including regularization, minimum description length, model complexity penalization (e.g., BIC, AIC), Ockham's razor (e.g., decision tree), training with noise, ensemble methods (e.g., boosting), structural risk minimization (e.g., SVMs), cross validation, holdout validation, etc. *Unlabelled data can provide a more accurate sampling of the function.*

The biggest challenge is how to define a proper measure of the function's complexity, which is the synonym of "change quickly". To proceed, we should define the meaning of "change" and "quickly". In semi-supervised learning, there are at least three different definitions which lead to different models.

Firstly, the most natural definition is the norm of gradient $\|\partial f/\partial x\|$ or its associated $\left|\partial p(y\,|\,x,f)/\partial x\right|$. One example is TSVM. Using the $p(y\,|\,x,f)$ defined in (1.1), $\left|\partial p(y\,|\,x,f)/\partial x\right|$ reaches its maximum (for both $y = 1$ and $y = 0$) when f is 0. So this assumption pushes the final learned separator ($f = 0$) to low density regions (hopefully the space between the clusters of the two classes), which the unlabelled data helps to locate.

Secondly, the Laplace operator: $\frac{\partial^2 f}{\partial x_1^2}+\frac{\partial^2 f}{\partial x_2^2}+\cdots+\frac{\partial^2 f}{\partial x_m^2}$. Strictly speaking, this is the same as $\|\partial f/\partial x\|^2$ in continuous compact space. It is usually difficult to calculate exactly and (Belkin & Nigam, 2004; Belkin et al., 2004b) tried to approximate it discretely using graph Laplacian. We will review the details in the next section.

Thirdly, mutual information $I(x:y)$ between x and y (now we use hard labels directly). Noticing that $I(x:y)$ describes how homogeneous the soft labels are, (Szummer & Jaakkola, 2002) proposed information regularization, which minimizes, on multiple overlapping regions, the product of $p(x)$ and the mutual information $I(x:y)$ normalized by variance. Furthermore, after some additional development in (Corduneanu & Jaakkola, 2003), (Corduneanu & Jaakkola, 2004) formulated regularization as rate of information, discouraging quickly varying $p(y|x)$ when $p(x)$ is large. Then it uses a local propagation scheme to minimize the regularized loss on labelled data.

More measure based regularization is available in (Bousquet et al., 2003), assuming known $p(x)$, though there are open problems in applying the results to high dimensional tasks.

5. Similar/closer examples should have similar $f(x)$

k nearest neighbour (*kNN*) is the clearest representative of this assumption. In this assumption, *the contribution of unlabelled data is to provide additional neighbouring information, which helps to propagate the labels among unlabelled points.* (Zhu et al., 2003b) used *kNN* for induction on unseen data. Most graph based semi-supervised learning algorithms to be discussed in the next section are also making this assumption explicitly or implicitly. One extension is from direct neighbour to a path connecting two points. Combining with the above assumption 4, one can make a so-called cluster assumption: two points are likely to have the same class if there is a path connecting them passing through regions of high density only. (Szummer & Jaakkola, 2001) assumed that the influence of one example A on

another example B is proportional to the probability of walking from A to B in t-step Markov random walk. However, the parameter t is crucial and requires special care to select.

6. Entropy maximization

This seems to be contradictory to assumption 2, but in fact the entropy here is of classifier parameters. (Jaakkola et al., 1999) proposed a maximum entropy discrimination model based on maximum margin. It maximizes the entropy distribution over classifier parameters, with constraints that the distance between each labelled example and decision boundary (margin) should be no less than a fixed value, or incurring soft penalty as in SVM. Interestingly, unlabelled data are also covered by margin constraints in order to commit to a certain class during parameter estimation by entropy maximization. *The contribution of unlabelled data is to provide a more accurate distribution over the unknown class labels.* Classification is then performed in a Bayesian manner, combining the expected value of the example's class over the learned parameter distribution. As for optimization method, an iterative relaxation algorithm is proposed that converges to local minima, as the problem is not convex. The experimental results for predicting DNA splice points using unlabelled data is encouraging.

7. $f(x)$ has a small norm in associated reproducing kernel Hilbert space

Some algorithms assume that the hypothesis space is a reproducing kernel Hilbert space (RKHS). They regularize the f by its norm in this RKHS. SVM and TSVM can both be viewed as examples of this assumption (Schölkopf & Smola, 2001). *The unlabelled data is*

used to define a richer RKHS and to give a more accurate regularization. It can be associated with the graph Laplacian as discussed in the next section by matrix pseudo-inversion.

Finally, we point out that there are some work which combines the generative models and discriminative models. See (Zhu & Lafferty, 2005) as an example.

1.5 Graph based semi-supervised learning

In this section, we review the graph based semi-supervised learning algorithms. These algorithms start with building a graph whose nodes correspond to the labelled and unlabelled examples, and whose edges (non-negatively weighted or unweighted) correspond to the similarity between the examples. They are discriminative, transductive and nonparametric in nature. Intuitively, graph based methods are motivated by learning globally based on local similarity.

A starting point for this study is the question of what constitutes a good graph. (Li & McCallum, 2004) argued that having a good distance metric is a key requirement for success in semi-supervised learning. As the final performance is not defined directly on graphs, but rather on higher level learning algorithms which utilize graphs, we postpone this question until the graph based learners are reviewed, and we first assume that a proper graph has been given.

1.5.1 Graph based semi-supervised learning algorithms

Graphs can be used in a number of ways going far beyond the scope of this thesis. We just review a few of them related to the project, namely: smooth labelling, regularization and kernels, spectral clustering, and manifold embedding for dimensionality reduction.

1.5.1.1 Smooth labelling

In this category of classification methods, the soft labels of the set of labelled examples (L) are clamped (or pushed) to the given labels, i.e., noiseless or zero training error. The task is to find a labelling of the set of unlabelled examples (U), which is smooth on the graph. Formally, it minimizes

$$\sum_{i,j} w_{ij} \left(y_i - y_j \right)^2 ,$$

where $y_i = 1$ if x_i is labelled positive and $y_i = 0$ if x_i is labelled negative.

Different algorithms have been proposed based on different constraints on y_i for $i \in U$. If we restrict $y_i \in \{0, 1\}$, then we arrive at the st-mincut algorithm (Blum & Chawla, 2001). The intuition behind this algorithm is to output a classification corresponding to partitioning the graph in a way which minimizes the number of similar pairs of examples that are given different labels. The idea originated from computer vision (Yu et al., 2002; Boykov et al., 1998; Greig et al., 1989). (Blum & Chawla, 2001) applied it to semi-supervised learning. The formulation is illustrated in Figure 1.1.

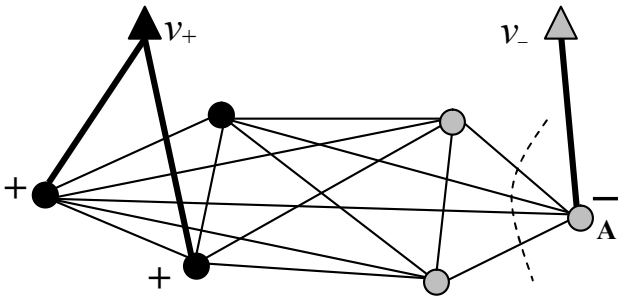


Figure 1.1 Illustration of st-mincut algorithm

The three labelled examples are denoted with + and –. They are connected to the respective classification nodes (denoted by triangles) with bold lines. Other three nodes are unlabelled and finally the left one is classified as + and right two nodes are classified as –.

v_+ and v_- are artificial vertices called classification nodes. They are connected to all positive examples and negative examples respectively, with infinite weight represented by bold lines. Now we determine a minimum (v_+, v_-) cut for the graph, i.e., find set of edges with minimum total weight whose removal disconnects v_+ and v_-. The problem can be solved using max-flow algorithm, in which v_+ is the source, v_- is the sink and the edge weights are treated as capacities (e.g.,(Cormen et al., 2001)). Removing the edges in the cut partitions the graph into two sets of vertices which we call V_+ and V_-, with $v_+ \in V_+$, $v_- \in V_-$. We assign positive label to all unlabelled examples in V_+ and negative label to all examples in V_-. (Blum & Chawla, 2001) also showed several theoretical proofs for the equivalence between st-mincut and leave-one-out error minimization under various 1NN or kNN settings. (Kleinberg & Tardos, 1999) considered this idea in multi-way cuts setting.

But an obvious problem of st-mincut is that it may lead to degenerative cuts. Suppose A is the only negative labelled example in Figure 1.1. Then the partition given by st-mincut may be the dashed curve. Obviously this is not desired. The remedies in (Blum & Chawla, 2001), such as carefully adjusting edge weights, do not work across all problems they study. To fix this problem, these algorithms usually require a priori fraction of positive examples in the test set, which is hard to estimate in practice when the training sets are small. (Blum et al., 2004) used the idea of bagging and perturbed the graph by adding random noise to the edge weights. On multiple perturbed graphs, st-mincut is applied and the labels are determined by a majority vote, yielding a soft st-mincut. Other algorithms like TSVM and co-training also suffer from similar degeneracy.

A simple analysis reveals the cause of the problem. The sum of the edge weights on the cut is related to the number of edges being cut, which in turn depends directly on the size of the two cut sets. In particular, the number of edges being cut with $|V_+|$ vertices on one side and $|V_-|$ vertices on the other side can potentially be $|V_+| \cdot |V_-|$. The st-mincut objective is inappropriate, since it does not account for the dependency on the cut size. A natural way proposed by (Joachims, 2003) is to normalize by dividing the objective with $|V_+| \cdot |V_-|$: $\frac{cut(V_+, V_-)}{|V_+| \cdot |V_-|}$. This problem is related to ratiocut (Hagen & Kahng, 1992) in the unsupervised setting, whose solution is NP-hard (Shi & Malik, 2000). So (Hagen & Kahng, 1992) proposed good approximations to the solution based on the spectrum of the graph.

The idea of approximation motivated the relaxation of binary constraints $y_i \in \{0,1\}$. The

first step of relaxation is to allow *y* to assume real-valued soft labels. Using the notation in section 1.4, we replace *y* by *f*, though the *f* value on labelled data is still fixed to 1 or 0. (Zhu et al., 2003a) proposed minimizing $\sum_{i,j} w_{ij}\left(f_i - f_j\right)^2$ over the *f* value of the unlabelled data in continuous space. The solution is actually a minimal norm interpolation and it turns out to be unique which can be simply calculated by solving a linear equation. This model constitutes the basic framework of this thesis. It has been applied to medical image segmentation (Grady & Funka-Lea, 2004), colorization of gray-level images (Levin et al., 2004), and word sense disambiguation (Niu et al., 2005).

(Joachims, 2003) goes one step further by making the soft labels of labelled data unfixed, and proposed a similar spectral graph transducer (SGT), which optimizes: $c\left(f-\gamma\right)^T C\left(f-\gamma\right)+f^T\Delta f$, under the constraint that $f^T\vec{1}_n = 0$ and $f^T f = n$. Here $\gamma_i \triangleq \sqrt{l_-/l_+}$ for positively labelled data, $\gamma_i \triangleq -\sqrt{l_-/l_+}$ for negatively labelled data, where l_+ and l_- stand for the number of positive and negative labelled examples respectively. $\gamma_i = 0$ for unlabelled data. $\Delta_{ij} \triangleq \delta\left(i=j\right)\sum_j w_{ij} - w_{ij}$ (graph Laplacian). In this way, the soft labels *f* on labelled data will be no longer fixed to 1 or 0, and the global optimal solution can be found by spectral methods efficiently.

As a final analysis of semi-supervised smooth graphical labelling, (Joachims, 2003) posed three postulates for building a good transductive learner:

1. It achieves low training error;
2. The corresponding inductive learner is highly self-consistent (e.g., low leave-one-out

error);

3. Averages over examples (e.g., average margin, pos/neg ratio) should have the same expected value in the training and in the test set.

St-mincut does not satisfy the third criterion, and the later algorithms patched it up.

1.5.1.2 Regularization and kernels

All the above mentioned algorithms of smooth labelling can be viewed as based on regularization: $\sum_{i,j} w_{ij}\left(f_i - f_j\right)^2$. Define a matrix Δ, with $\Delta_{ij} \triangleq \delta\left(i=j\right)\sum_j w_{ij} - w_{ij}$, this regularizer can be written as: $f^T \Delta f$. Δ is usually called graph Laplacian. Since $\sum_j w_{ij}$ can be wildly different for different *i*, there have been various ways proposed to normalize Δ. One most commonly used one is $\tilde{\Delta} = D^{-1/2}\Delta D^{-1/2}$, where D is a diagonal matrix with $D_{ii} = \sum_j w_{ij}$. Using these concepts, we can simply re-write all the smoothing algorithms above and there are some other examples in literature. (Zhou et al., 2003) minimized $\sum_{i\in L}\left(f_i - y_i\right)^2 + f^T\tilde{\Delta}f$. (Belkin et al., 2004a) used the same objective function except replacing $\tilde{\Delta}$ by Δ^p, where p is a natural number, which is essentially Tikhonov regularization (Tikhonov, 1963). Refer to the relationship of approximation between graph Laplacian and Laplace operator in section 2.6.1.

The real power of graph in regularization stems primarily from the spectral transformation on its associated graph Laplacian and the induced kernels. (Smola & Kondor, 2003; Chapelle et al., 2002) motivated the use of the graph as a regularizer by spectral graph the-

ory (Chung, 1997). According to this theory, the eigenvectors of Δ correspond to the local minima of the function $f^T \Delta f$. For eigenvector ϕ_i with eigenvalue λ_i, $f^T \Delta f\big|_{f=\phi_i} = \lambda_i$. Since the eigenvectors constitute a basis of the vector space, for any soft label vector $f = \sum_i \alpha_i \phi_i$, $f^T \Delta f = \sum_i \alpha_i^2 \lambda_i$. Therefore we wish to penalize more seriously those α_i associated with larger eigenvalues λ_i. How serious the penalty is as a function of α_i and λ_i has motivated many different spectral transformation functions, because a simplest way to describe this penalty relationship is by mapping λ_i to $r(\lambda_i)$, where $r(\cdot)$ is a non-negative monotonically *increasing* function (larger penalty for larger λ_i). So $f^T \Delta f = \sum_i r(\lambda_i) \alpha_i^2$. Commonly used $r(\lambda)$ include:

$r(\lambda) = 1 + \sigma^2 \lambda$ (regularized Laplacian), $r(\lambda) = \exp(\sigma^2 / 2\lambda)$ (diffusion kernel)

$r(\lambda) = (a - \lambda)^{-p}$ ($p \geq 1$, p-step random walk), $r(\lambda) = (\cos \lambda \pi / 4)^{-1}$ (Inverse cosine).

The form $f^T \Delta f$ has naturally elicited the connection between Δ and kernels, because the norm of any function f in a reproducing kernel Hilbert space induced by kernel K is $f^T K^{-1} f$. Based on this analogy, (Smola & Kondor, 2003) established and proved the relationship $K = \Delta^{\dagger}$, where † stands for pseudo-inverse. Therefore, we now only need to apply the reciprocal of the above $r(\cdot)$ on the spectrum of Δ in order to derive a kernel. (Kondor & Lafferty, 2002) proposed diffusion kernels by effectively using $r^{-1}(\lambda) = \exp(-\sigma^2 \lambda / 2)$. $r^{-1}(\lambda) = (a - \lambda)^t$ and $r^{-1}(\lambda) = (\lambda + \sigma)^{-1}$ were used in (Szummer & Jaakkola, 2001) for t-step random walk and in (Zhu et al., 2003b) for Gaussian process respectively.

This kernel view offers the leverage of applying the large volume of tools in kernel theory, such as the representer theorem (Schölkopf & Smola, 2001) to facilitate optimization (Belkin et al., 2005). Moreover, since the new kernel is $\sum_i r^{-1}(\lambda_i)\phi_i\phi_i^T$, this view has also opened window to *learning* the kernel by maximizing the kernel alignment (Cristianini et al., 2001) or low rank mapping to feature space (Kwok & Tsang, 2003), with respect to those hyperparameters σ, a, t. However, though it is more restrictive than the overly free semi-definite programming algorithms (Lanckriet et al., 2004) which only require positive semi-definiteness, these methods are too restrictive given the fixed parametric form. Considering the monotonically decreasing property of spectral transformation for kernels, (Zhu et al., 2004) proposed a tradeoff by maximizing alignment with empirical data from a family of kernels $\sum_i \mu_i\phi_i\phi_i^T$, with nonparametric constraints that $\mu_i \geq \mu_j$ if and only if $\lambda_i \leq \lambda_j$. Recently, (Argyriou et al., 2005) learns a convex combination of an ensemble of kernels derived from different graphs.

Finally, for completeness, there are also some kernel based algorithms for generative models. As an extension of *Fisher kernel* (Jaakkola & Haussler, 1998), a marginalized kernel for mixture of Gaussians (μ_k, Σ_k) is proposed: $K(x,y) = \sum_{k=1}^{q} p(k \mid x)p(k \mid y)x^T\Sigma_k^{-1}y$ (Tsuda et al., 2002). But in addition to common problem of generative models, this method requires building a generative model: finding parameters μ_k and Σ_k using unsupervised learning.

1.5.1.3 Spectral clustering

Why is clustering related to classification in a semi-supervised setting? One ideal situation

when clustering helps is: the whole set of data points are well clustered, with each cluster being represented by a labelled data point. Then we can simply label the rest data in that cluster by the same label. Generally, we are not that lucky because the clustering process is blind to the labels, and its policy may be significantly different from the labelling mechanism. However, studying the clustering algorithms does help us learn a graph, because no matter how the data points are labelled, it is desirable that data from different classes are well clustered, i.e., there *exists* some view based on which the result of clustering tallies with the labelling mechanism. So studies in this area can inform us what makes a good graph, hopefully in the form of an objective function to be optimized, though it is not the original task of clustering.

A most natural model for clustering is generative mixture models. However, its parametric density model with simplified assumptions (such as Gaussian) and the local minima in maximum evidence estimation poses a lot of inconvenience. A promising alternative family of discriminative algorithms formulate objective functions which can be solved by spectral analysis (top eigenvectors), similar to the role of eigensystem mentioned in section 1.5.1.2 on kernel and regularization. A representative algorithm is normalized cut (Shi & Malik, 2000), which minimizes: $Ncut(A,B) \triangleq \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)}$, where V is the whole node set, A and B are the two disjoint sets V is to be partitioned into, and $assoc(A,V) = \sum_{u \in A, t \in V} w_{ut}$ (the total weight of edges from nodes in A to all nodes in the graph). The normalized cut is minimized in two steps: first find a soft cluster indicator vector, which is the eigenvector with the second smallest eigenvalue (sometimes directly

called second smallest eigenvector). The underlying math tool is the Rayleigh quotient (Golub & van Loan, 1996). Secondly, discretize the soft indicator vector to get a hard clustering. To find *k* clusters, the process is applied recursively (Spielman & Teng, 1996). (Ng et al., 2001a) mapped the original data points to a new space spanned by the *k* smallest eigenvectors of the normalized graph Laplacian (well, we present in a slightly different but equivalent way of the original paper). Then *k*-means or other traditional algorithms are performed in the new space to derive the *k*-way clustering directly. However there is still disagreement on exactly which eigenvectors to use and how to discretize, i.e., derive clusters from eigenvectors (Weiss, 1999). For large spectral clustering problems, this family of algorithms are too computationally expensive, so (Fowlkes et al., 2004) proposed Nyström method, which extrapolates the complete clustering solution using only a small number of examples.

The soft indicators (actually the second smallest eigenvector) in normalized cut tell how well the graph is naturally clustered. If the elements of the vector are numerically well grouped within each class and well separated between classes (e.g., many close to 1, many close to –1 and few near 0), then the discretization step will confidently classify them into two parts. Otherwise, if we assume that the clustering algorithm is well designed and reliable, then we can conclude that the graph itself is not well constructed, i.e., heavy overlapping between classes and large variance within each class. (Ng et al., 2001a) used eigengap by matrix perturbation theory (Stewart & Sun, 1990) to present some desired property of a good graph for clustering. The central idea is: if a graph has *k* well clustered groups, then

the clustering result is more stable (i.e., the smallest k eigenvectors are more stable to small perturbations to graph edge weights) when the difference between the k^{th} and $(k+1)^{th}$ smallest eigenvalues of the normalized graph Laplacian is larger. See more details of eigengap in section 4.2.2.

1.5.1.4 Manifold embedding and dimensionality reduction

These are essentially techniques for unsupervised learning. The central idea of manifold is that classification functions are naturally defined only on the sub-manifold in question rather than the total ambient space. So transforming the representation of data points into a reasonable model of manifold is effectively performing feature selection, noise reduction, and data compression. These will hopefully improve classification. As this process does not depend on examples' label, unlabelled data can be naturally utilized by refining the approximation of the manifold by the weighted discrete graph.

For example, handwritten digit **0** can be fairly accurately represented as an ellipse, which is completely determined by the coordinates of its foci and the sum of distances from the foci to any point. Thus the space of ellipses is a five-dimensional manifold. Although a real handwritten **0** may require more parameters, the dimensionality is absolutely less than ambient representation space, which is the number of pixels. In text data, documents are typically represented by long vectors whereas researchers are convinced by experiments that its space is a manifold, with complicated intrinsic structure occupying only a tiny portion of the original space. The main problem is how to choose a reasonable model of manifold, or

more concretely speaking, how to find a proper set of basis of the manifold space.

(Belkin & Nigam, 2004; Belkin et al., 2004b) used the Laplace-Beltrami operator $\Delta = \sum_i \partial^2 / \partial x_i^2$, which is positive semidefinite self-adjoint on twice differentiable functions. When $\mathcal{M}$ is a compact manifold, Δ has a discrete spectrum and its eigen-functions provide an orthogonal basis for the Hilbert space $\mathcal{L}^2(\mathcal{M})$. Suppose we are given *n* points $x_1, \ldots, x_n \in \mathbb{R}^m$ and the first $l < n$ points have labels $c_i \in \{-1, 1\}$. First construct an adjacency graph with *n* nearest and reverse nearest neighbours, with distance defined as standard Euclidean distance in $\mathbb{R}^m$, or other distance like angle/cosine. Then define adjacency matrix $W_{n \times n}$. $w_{ij} = 1$ if points x_i and x_j are close (in some sense). Otherwise, $w_{ij} = 0$. Compute *p* eigenvectors corresponding to the smallest *p* eigenvalues for the graph Laplacian. They are proved to unfold the data manifold to form meaningful clusters. Now constitute a matrix $E_{p \times n}$ by stacking the transposition of the *p* eigenvectors in *rows* and the column order corresponds to the original index of x_i. Denote the left $p \times l$ sub-matrix of $E_{p \times n}$ as E_L and calculate $\vec{a} = (E_L^T E_L)^{-1} E_L^T \vec{c}$. Finally, for x_i ($i > l$) the rule for classification is: $c_i = 1$ if $\sum_{j=1}^{p} e_{ij} a_j \geq 0$ and $c_i = -1$ otherwise. In essence, this is similar to spectral clustering and there are many variants of such a PCA-style data representation (Ng et al., 2001a; Weiss, 1999).

Other graph based dimensionality reduction algorithms include locally linear embedding (LLE) (Roweis & Saul, 2000), Isomap (de Silva & Tenenbaum, 2003; Tenenbaum et al., 2000), multidimensional scaling (MDS) (Cox & Cox, 2001), semidefinite embedding

(Weinberger et al., 2005; Weinberger et al., 2004; Weinberger & Saul, 2004), etc. (Ham et al., 2004) shed the insight that the first three are all kernel PCA on specially constructed Gram matrices.

The contribution to classification by dimensionality reduction or manifold embedding is similar to the role of feature selection, helping to find a lower dimensional (simpler) structure in the data which is expected to be related to the class labels of the data. However, it is likely that the manifold is uncorrelated with the labels and this unsupervised technique will then hurt the classification performance. (Weinberger et al., 2004) gives an example.

1.5.2 Graph construction

It is interesting that the question of how to construct a good graph has not received intensive attention, albeit its fundamental role in graph based semi-supervised learning algorithms. In the applications considered here, the graphs are closest to real data and interfaces to the learning algorithm as an intermediate form of representation. It is conjectured that this problem is very domain specific and depends mainly on prior expert knowledge. Their utility is supposed to depend, to some extent, on the algorithms that they will ultimately be used for.

From an unsupervised view, the graph should represent the underlying manifolds well. Intuitively, it should avoid shortcuts that travel outside a manifold, avoid gaps that erroneously disconnect regions of a manifold, and be dense within the manifold and clusters.

Also, even if the algorithms differ with respect to connectedness, (clustering wants the graph to be disconnected, while for manifold learning the graph should be connected), they both want at least the inside of the clusters, or dense areas of the manifold, to be enhanced relative to the between-cluster, or sparse manifold connections.

In (Zhu, 2005; Carreira-Perpiňán & Zemel, 2004) , the following graph construction approaches are summarized. Note the edge weights w_{ij} are all based on a *given* distance d_{ij} between all node pairs (i, j), and w_{ij} should be a monotonically decreasing function of d_{ij}.

1. Fully connected graphs, which are often used for spectral clustering and multidimensional scaling.
2. Fixed grid. Connect each point to some small fixed number of neighbours in a pre-defined grid, generally used in image segmentation.
3. Unweighted *kNN* graph. Connect node i and j by an edge with weight 1, if node i is one of node j's k nearest neighbours or vice versa. The desirable property of this construction is the spatial adaptive scaling, i.e., in low density regions the distance between k nearest neighbours can be farther than in high density regions.
4. Unweighted *ε-ball* graph. Connect node i and j by an edge with weight 1, if $d_{ij} \le \varepsilon$.
5. tanh-weighted graph. Define $w_{ij} \triangleq (\tanh(\alpha_1(d_{ij} - \alpha_2) - 1)) + 1)/2$. This function will create a soft cutoff around length α_2 and the hyperbolic tangent function simulates *ε-ball*, with α_1 and α_2 controlling the slope and cutoff value respectively. It has a notable advantage that the graph is continuous with respect to d_{ij}, so gradient based learning methods are applicable.

6. RBF weighted graph: $w_{ij} \triangleq \exp\left(-d_{ij}^2 / \sigma^2\right)$. σ controls the decay rate, and σ is also called bandwidth. This graph is continuous with respect to d_{ij}.

The hyperparameters ε, k draw a considerable impact on the performance of graph algorithm. And because they are at the lowest layer, the normal cross validation algorithms have to construct graphs for a range of hyperparameter values, redo the graph based learning algorithm and then pick the best settings. In order to circumvent these trouble, (Carreira-Perpiňán & Zemel, 2004) proposed building robust graphs by applying random perturbation and edge removal from an ensemble of minimum spanning trees.

However, these methods still depend on a given distance d_{ij}. In real practice, rarely are we simply given a correct distance metric. Even restricted to Euclidean distance, the square distance along feature s, t between x_i and x_j may be attached with different importance for the learning task. In this thesis, we restrict our attention to Gaussian RBF weighted graphs: $w_{ij} \triangleq \exp\left(-\sum_k \left(x_{i,k} - x_{j,k}\right)^2 / \sigma_k^2\right)$ and learn different bandwidth σ_k for different dimensions k. All procedures apply to tanh-weighted graph with no change, except rewriting the gradient formula.

1.6 Wrapping algorithms

Just as there are ensemble methods in supervised learning such as boosting and bagging which can be generically applied on top of almost all algorithms to boost the performance, so are there such counterparts in semi-supervised learning. We call them wrapping algo-

rithms. Here we briefly overview some well known wrappers: self-training, co-training, co-boosting, co-validation, co-testing, and down-weighting.

1. Self-training

The earliest and simplest wrapping algorithm is self-training. Its basic idea is:

1. The initial labelled set L and unlabelled set U are the originally labelled and unlabelled data, respectively.
2. Train the classifier by L, and classify the data in U.
3. Pick the most confidently classified examples and move them from U to L with the predicted labels.
4. Go to step 2 and retrain until U is empty.

So the algorithm uses its own predictions to supervise itself, and one can expect that mistakes can be propagated and magnified through iteration. There are some unlearning or anti-learning algorithms to early detect and stop the propagation of errors. Applications of self-training are reported in (Yarowsky, 1995) for word sense disambiguation, (Riloff et al., 2003) for subjective noun identification, (Maeireizo et al., 2004) for emotional and non-emotional dialogue classification, and (Weiss, 1999) for object detection in images, etc.

2. Co-training

The idea of co-training is: suppose the feature of each data point is composed of two sets, which are separately used to train two classifiers. Then each classifier will provide the la-

bel of their most confident unlabelled data to the other classifier, which is in turn used for retraining. The process iterates. Co-training is based on two assumptions namely the two sets of features are conditionally independent given the class, and each set is enough to train a reasonably performing classifier. The role played by the unlabelled data is to restrict the version/hypothesis space such that the two classifiers must tally on unlabelled dataset which is much larger than the labelled dataset.

In the original papers (Mitchell, 1999; Blum & Mitchell, 1998), a case in point mentioned is web page classification, where each example has words occurring on a web page, and also anchor texts attached to hyperlinks pointing *to* the web page. They also proved in a PAC-style framework that under certain theoretical assumptions, any weak hypothesis can be boosted from unlabelled data. (Nigam & Ghani, 2000) argue that algorithms explicitly leveraging a natural independent split of the features outperform algorithms that do not. When a natural split does not exist, co-training algorithms that manufacture a feature split may outperform algorithms not using a split. These arguments help explain why co-training algorithms are both discriminative in nature and robust to the assumptions of their embedded classifiers. (Goldman & Zhou, 2000) went one step further, showing that co-training can even succeed on datasets without separate views, by means of carefully selecting the underlying classifiers. (Balcan et al., 2004) proposed so-called expansion condition to relax the conditional independence assumption.

3. Co-boosting

Also in the setting of co-training, (Collins & Singer, 1999) presented a boosting algorithm, called co-boosting. It builds a number of classifiers using different views of the data, and minimizes their difference of classification on the unlabelled data. Some other bootstrapping techniques can learn from nearly no labelled data and iteratively develop a concept of interest. (Riloff & Jones, 1999) requires only unannotated training texts and a handful of seed words for a category as input. They use a mutual bootstrapping technique to alternately select the best extraction pattern for the category and bootstrap its extractions into the semantic lexicon, which is the basis for selecting the next extraction pattern. This wrapping algorithm is very similar to self-training, except that it is uses different views of the data similar to co-training.

4. Co-validation

(Madani et al., 2004) proposed co-validation to estimate error expectation (lower bounds) and variance, through training two independent functions that in a sense validate (or invalidate) one another by examining their mutual rate of disagreement across a set of unlabelled data, which simultaneously reflects notions of algorithm stability, model capacity, and problem complexity. One advantage of this wrapper is that it does not rely on the independent identically distributed (iid) sampling assumption.

5. Co-testing

Co-testing is basically an active learning algorithm, focusing on selective sampling for domains with redundant multiple views. It is based on the idea of learning from mistakes.

More precisely, it queries examples on which the views predict a different label: if two views disagree, one of them is guaranteed to make a mistake. In a variety of real-world domains mentioned in (Muslea et al., 2000), from information extraction to text classification and discourse tree parsing, co-testing outperforms existing active learners.

6. Down-weighting unlabelled data

It is completely improper to call down-weighting as a wrapping algorithm as it is not built upon any underlying algorithm. However, it can be almost seamlessly and effortlessly incorporated to all semi-supervised learning algorithms with little modification. Its idea is to imagine that the labelled data are super-sampled by β times, on the ground that labelled data are more reliable and more informative than unlabelled data. Suppose originally we have examples $\underbrace{x_1^l, x_2^l, \ldots, x_{n_1}^l}_{n_1 \text{ labeled data}}, \underbrace{x_1^u, x_2^u, \ldots, x_{n_2}^u}_{n_2 \text{ unlabeled data}}$, then now we have $\underbrace{\overbrace{x_1^l, \ldots x_1^l}^{\beta \text{ copies of } x_1^l}, \ldots, \overbrace{x_{n_1}^l, \ldots x_{n_1}^l}^{\beta \text{ copies of } x_{n_1}^l}}_{\beta n_1 \text{ labeled data}}, \underbrace{x_1^u, x_2^u, \ldots, x_{n_2}^u}_{n_2 \text{ unlabeled data}}$.

The central idea is to change the $p(x)$ for labelled data from $\frac{1}{n_1 + n_2}$ to $\frac{\beta}{\beta n_1 + n_2}$ and the $p(x)$ for unlabelled data has been changed from $\frac{1}{n_1 + n_2}$ to $\frac{1}{\beta n_1 + n_2}$. Down-weighting has been used in (Blum & Chawla, 2001) to re-weight the edges, and in (Lee & Liu, 2003).

As a final note for completeness, we point out that another *somewhat* equivalent perspective to interpret that above co-*X* (where *X* = training, testing, …) is that unlabelled data can be used to reduce overfitting. If we have two candidate classifiers or regressors, then overfitting is believed to occur (approximately) when the number of different classification on the

unlabelled data is larger than the number of their errors on labelled data. (Schuurmans, 1997) used this observation to select the best complexity of a polynomial for regression and (Schuurmans & Southey, 2000) applied it for pruning decision trees. (Cataltepe & Magdon-Ismail, 1998) extended the minimization criterion of mean squared error with terms based on unlabelled (and testing) data to reduce overfitting in linear regression.

1.7 Optimization and inference techniques

In the previous sections, we have mentioned some important optimization techniques in machine learning. Here, we just summarize them and review some other important optimization and inference algorithms, some of which are on an approximate basis.

The most commonly used optimization algorithms for machine learning includes (but not limited to): expectation maximization (EM) or more generally variational methods, linear programming (LP), quadratic programming (QP), sequential minimal optimization (SMO), semi-definite programming (SDP), eigen-decomposition, and boosting.

The seminal paper by (Dempster et al., 1977) presented the theory of the Expectation-Maximization (EM) framework, bringing together and formalizing many of the commonalities of previously suggested iterative techniques for likelihood maximization with missing data (or latent variables). It was immediately recognized that EM is applicable to estimating maximum likelihood (ML) or maximum a posteriori (MAP) parameters for mixture models from labelled and unlabelled data (Murray & Titterington, 1978) and then using this

for classification (Little, 1977). Since then, this approach continues to be used and studied (McLachlan & Ganesalingam, 1982; Ganesalingam, 1989; Shahshahani & Landgrebe, 1994). EM and its application to mixture modeling enjoy a splendid history, summarized in (McLachlan & Basford, 1988; McLachlan & Krishnan, 1997; McLachlan & Peel, 2000).

There are two main problems with EM. Firstly, it can only find a local maximum by variational methods. Therefore re-starts with different initial conditions are necessary to find a more reliable solution. Secondly it involves calculating the probability density distribution in each iteration, which can be very computationally costly. Sometimes even calculating it once can be prohibitive especially on some real valued Bayes nets, so approximate algorithms and sampling algorithms are proposed, e.g. Markov Chain Monto Carlo (Neal, 1993; Mackay), (loopy/generalized) belief propagation (Yedidia et al., 2005; Murphy et al., 1999; McEliece et al., 1998), expectation propagation (Minka, 2001a; Minka, 2001b), variational methods (Wainwright & Jordan, 2005; Wainwright & Jordan, 2003; Jordan et al., 1999).

SMO is another important optimization algorithm, which is used for solving large scale QP, especially in training SVM and TSVM (Platt, 1998). "SMO breaks a large QP problem into a series of smallest possible QP problems. These small QP problems are solved analytically, which avoids using a time-consuming numerical QP optimization as an inner loop. The amount of memory required for SMO is linear in the training set size, which allows SMO to handle very large training sets. Without kernel caching, SMO scales somewhere between linear and quadratic in the training set size for various test problems, while a standard projected conjugate gra-

dient (PCG) chunking algorithm scales somewhere between linear and cubic in the training set size. SMO's computation time is dominated by SVM evaluation, hence SMO is fastest for linear SVMs and sparse data sets."

Eigen-decomposition is usually applied in spectral graph methods, where the problems are ultimately reduced to a Rayleigh quotient. Normally what is needed is the eigenvectors associated with the largest/smallest eigenvalues. Even though there is normally no need of calculating the whole eigensystem, it can be computationally prohibitive for large datasets. So lots of work in matrix theory is introduced.

Semi-definite programming is so far mainly used for learning kernel, because kernels are positive semi-definite by definition. It is an optimization technique to deal with the positive semi-definiteness constraints together with other linear constraints (Lanckriet et al., 2004; Boyd & Vandenberghe, 2004).

1.8 Theoretical value of unlabelled data

In this thesis, we generally discuss the value of unlabelled data in an empirical sense, but looking at the existing research on their theoretical value helps to understand what the limit is. (Ganesalingam & McLachlan, 1978) examined the simplest uni-variate normal distribution with variances known and equal. They calculated the asymptotic relative value of labelled and unlabelled data to first-order approximation. (O'Neill, 1978) calculated the same value further, but for multivariate normals with equivalent and known covariance matrices. (Ratsaby & Venkatesh, 1995) used PAC framework to perform a similar analysis. (Chen,

1995) worked on a class of mixture distributions (including normals) where the number of mixture components is bounded but not known. He bounded the rate of convergence of parameter estimates from unlabelled examples but nothing was said about classification error. All above mentioned results assume that the global ML parameterization is found instead of a local maximum, and the data were actually generated by the model used. For the more general and challenging cases beyond Gaussians, there are few known results. (Cozman & Cohen, 2002) argued that unlabelled data can degrade the performance of a classifier when there are incorrect model assumptions (e.g., set of independence relations among variables or fixed number of labels). (Castelli & Cover, 1995) showed that labelled data reduce error exponentially fast with an infinite amount of unlabelled data, assuming the component distributions are known, but the class-to-component correspondence is not known. Further, (Castelli & Cover, 1996) obtained an important result that for class probability parameter estimation, labelled examples are exponentially more valuable than unlabelled examples, assuming the underlying component distributions are known and correct. (Zhang & Oles, 2000) examine the value of unlabelled data for discriminative classifiers such as TSVMs and for active learning. As mentioned above, they cast doubt on the generality of the helpfulness of TSVMs.

Chapter 2 Basic Framework and Interpretation

In this chapter, we review the details of the basic framework by (Zhu et al., 2003a) upon which this thesis' main contribution is made. We will also interpret this model in many different ways in order to introduce some original extensions based on these different views.

2.1 Preliminaries, notations and assumptions

We suppose that the size of the dataset is n, including l labelled data points and u unlabelled data points ($n = l + u$). The l labelled data points are denoted as (x_1, y_1), (x_2, y_2), …, (x_l, y_l), where x_i represents a data point (e.g., feature vector, tree, graph) and the labels y_i are binary: $y_i \in \{0, 1\}$. The u unlabelled data points are denoted as $x_{l+1}, x_{l+2}, \ldots, x_{l+u}$. In the typical settings of semi-supervised learning, we assume $l \ll u$. We consider a completely connected graph $G = \langle V, E \rangle$, which is undirected and weighted. The node set V is composed of two disjoint sets: $L = \{1,...,l\}$ and $U = \{l+1,...,l+u\}$, which correspond to the set of labelled points and unlabelled points respectively. So the following three statements are equivalent: x is a labelled point, $x \in L$, and $x \in \{1...,l\}$. The weight of the edges in set E depends on an important assumption of the model, namely the availability of a similarity measure $sim(x_i, x_j)$, which indicates how similar two points x_i and x_j are. For simplicity as a start, we assume that the x_i are represented by a fixed m-dimensional vector $x \in \mathbb{R}^m$ and

$$sim(x_i, x_j) \triangleq \exp\left(-\sum_{d=1}^{m} \frac{(x_{id} - x_{jd})^2}{\sigma_d^2} \right) \tag{2.1}$$

where x_{id} denotes the d^{th} component of x_i. σ_d is the length scale or bandwidth for dimension *d*, now assumed to be a given hyperparameter. The main topic of the thesis is how to learn these hyperparameters automatically, but at present, we assume they are given. As $sim(x_i, x_j) = sim(x_j, x_i)$, we use $sim(x_i, x_j)$ as the weight of the edge between x_i and x_j (w_{ij}) and the graph is undirected. So if x_i and x_j are similar, the weight of their link should be *large*, and the weight w_{ij} is bounded in (0, 1].

Given the labels $\{y_1, ..., y_l\}$, our task is to predict the labels $y_i \in \{0, 1\}$ ($i \in U$). In normal machine learning, especially in classification, we base the classification of a point on a real-valued soft label function $f : V \to R$, and then apply thresholding $y_i = \begin{cases} 1 & \text{if } f_i \geq t \\ 0 & \text{if } f_i < t \end{cases}$ or other labelling technique. In this model we define that $f_i \triangleq y_i$ for $i \in L$. So the task becomes to find a good algorithm that predicts f_i for $i \in U$ and then to find a reasonable translation rule from f to labels y.

2.2 Motivation and formulation

The most fundamental assumption and motivation are that the data points with higher similarity should have more similar labels. (Zhu et al., 2003a) proposed that we should minimize:

$$E(f) \triangleq \sum_{i,j=1}^{n} w_{ij} (f_i - f_j)^2 . \tag{2.2}$$

Remember that $f_L \triangleq \{f_1, ..., f_l\}$ are clamped to the given labels, so the minimization is with respect to the $f_U \triangleq \{f_{l+1}, ..., f_{l+u}\}$. For larger w_{ij}, f_i and f_j are more encouraged to be similar.

Now we are assuming that $w_{ij} \in (0,1]$. So $E(f)$ must be positive semi-definite and there must exist an optimal solution. However, if we allow w_{ij} to be negative then the optimal solution may be obtained when some f_i tends to infinity, thus bound constraints or other constraints must be imposed in order to keep the problem well defined.

To solve this minimization problem (2.2), we use the notation of combinatorial graph Laplacian, defined in matrix form as $\Delta \triangleq D - W$, where $W \triangleq \left(w_{ij}\right)$, $D \triangleq diag\left(d_i\right)$ is a diagonal matrix with entries $d_i \triangleq \sum_{j=1}^{n} w_{ij}$. So

$$E\left(f\right) = f^T \Delta f = f_U^T \Delta_{UU} f_U + 2 f_U^T \Delta_{UL} f_L + f_L^T \Delta_{LL} f_L , \tag{2.3}$$

where $f \triangleq \left(f_L^T, f_U^T\right)^T$ and Δ_{UU}, Δ_{UL}, Δ_{LL} denote the sub-matrices whose meaning is clear:

$$W = \begin{pmatrix} W_{LL} & W_{LU} \\ W_{UL} & W_{UU} \end{pmatrix}, \qquad \Delta = \begin{pmatrix} \Delta_{LL} & \Delta_{LU} \\ \Delta_{UL} & \Delta_{UU} \end{pmatrix}.$$

It can be proven that once we define w_{ij} in the form of (2.1), Δ_{UU} must be positive definite and thus invertible. See (Chung, 1997) for detailed proof. So the minimizer of (2.3) is

$$\begin{aligned} f_U^{opt} &= -\Delta_{UU}^{-1} \Delta_{UL} f_L = \left(D_{UU} - W_{UU}\right)^{-1} W_{UL} f_L \\ &= \left(D_{UU} - W_{UU}\right)^{-1} D_{UU} D_{UU}^{-1} W_{UL} f_L = \left(I - P_{UU}\right)^{-1} P_{UL} f_L \end{aligned} \tag{2.4}$$

where $P \triangleq D^{-1} W$, i.e., normalizing each row of W to sum up to 1. (2.5)

For convenience, we just call f_U instead of f_U^{opt}. The solution employs the interesting harmonic property, namely

$$\Delta \cdot f = \begin{pmatrix} \Delta_{LL} f_L + \Delta_{LU} f_U \\ \Delta_{UL} f_L + \Delta_{UU} f_U \end{pmatrix} = \begin{pmatrix} \Delta_{LL} f_L + \Delta_{LU} f_U \\ \Delta_{UL} f_L - \Delta_{UL} f_L \end{pmatrix} = \begin{pmatrix} \Delta_{LL} f_L + \Delta_{LU} f_U \\ 0 \end{pmatrix}.$$

Note Zhu's original paper (Zhu et al., 2003a) was incorrect in claiming that the upper

sub-vector $\Delta_{LL} f_L + \Delta_{LU} f_U$ is always equal to f_L. This does not generally hold. The harmonic property means that the f value at each unlabelled data point is the average of f at its neighbouring points:

$$f_i = \sum_{j=1:j\neq i}^{n} w_{ij} f_j \Big/ \sum_{j=1:j\neq i}^{n} w_{ij}, \quad \text{for all } j \in U. \tag{2.6}$$

The original paper (Zhu et al., 2003a) made a mistake in *its* Eq. (3) by dividing by $\sum_{j=1}^{n} w_{ij}$. Fortunately, from (2.2), it can be seen that the value of w_{ii} has no influence on $E(f)$, so we can safely fix $w_{ii} = 0$, though $w_{ii} \triangleq 1$ by (2.1). We will henceforth fix $w_{ii} = p_{ii} = 0$. Now (2.6) can also be expressed as $f = Pf$ (P defined in (2.5)). By virtue of the maximum principle of harmonic functions, f is unique and is either constant or it satisfies $0 < f_i < 1$ for all $i \in U$ (Doyle & Snell, 1984).

Now the only remaining task is to design a classification rule which translates f_U to hard labels y_U. Since we clamp the f_L to 1 for positive points and 0 for negative points, and f_U are between 0 and 1, a natural choice is thresholding by 0.5. That is, classify x_i to positive if $f_i \geq 0.5$ and to negative if $f_i < 0.5$. Our interpretation in the following sections will also support this rule, and it works well especially when the classes are well separated, in which case the elements in f_U are either close to 1 or close to 0, representing strong confidence in the prediction. However, in real datasets when points from different classes are not well separated, this heuristic rule can produce inaccurate predictions, particularly for unbalanced classifications.

To fix this problem, one can make use of the assumption that the class priors, i.e., the ratio

between positive and negative points in the test set should be an important measure to guide the classification. It can be given by an "oracle", or be estimated from that ratio in the training set, according to the *iid* sampling assumption. In (Zhu et al., 2003a), a simple post-processing method called class mass normalization (CMN) was proposed. The idea is similar to classifying a given proportion of points with the largest f_i to be positive. Suppose that the desirable ratio between positive and negative classes' size in unlabelled dataset is q : $1 - q$. Then for each unlabelled data point i with f_i, we first normalize f_i and $1 - f_i$ by

$$\tilde{f}_i^+ \triangleq q\frac{f_i}{\sum_{j=l+1}^{l+u} f_j}, \qquad \tilde{f}_i^- \triangleq (1-q)\frac{1-f_i}{\sum_{j=l+1}^{l+u}(1-f_i)}. \tag{2.7}$$

Then we compare $\tilde{f}_i^+$ and $\tilde{f}_i^-$. If $\tilde{f}_i^+ \geq \tilde{f}_i^-$, then classify point i as positive, otherwise negative. This method extends naturally to multi-class cases by similar definition as (2.7).

2.3 Interpreting HEM 1: Markov random walk

Interestingly, this model can be interpreted in at least 5 ways as pointed out by (Zhu et al., 2003a), namely Markov random walk on graph, electric network, harmonic mean, graph/ heat kernels, and Laplace equations. Among them, the random walk and electric networks views will be further extended in this thesis for our graph learning algorithm. It was also shown in (Zhu et al., 2003a) that the algorithm is closely connected to spectral clustering and graph st-mincuts.

Suppose a particle is walking along a graph with n nodes. Starting from an unlabelled node x_i, it moves to a node x_j with probability p_{ij} after one step. $\sum_{j=1}^{n} p_{ij} = 1$, $p_{ij} \geq 0$ for all

$i, j = 1...n$. p_{ij} can be defined as the normalized weight: $p_{ij} = w_{ij} \Big/ \sum_{j=1}^{n} w_{ij}$. Remember we have fixed $w_{ii} = p_{ii} = 0$. The walk continues until the particle hits a labelled node, viewed as absorbing boundary. In other words, random walk is allowed only on unlabelled nodes and once it reaches a labelled node, it must stop.

Denote as f_i^t the probability that a particle, starting from node i ($i \in U$) hits a *positively* labelled node within t steps (including t steps). We assume that f_i^t is well defined as long as t is large enough, which is equivalent to assuming that from any unlabelled node, there is a path linking to a *certain* labelled data point within t steps. Then f_i^t can be expressed as:

$$\begin{aligned} f_i^t = \; & P(\text{reaching a labelled positive node in 1 step}) \\ & + P(\text{reaching an unlabelled node next and then walk to a positive node in } t-1 \text{ steps}) \end{aligned}$$

$$= \begin{cases} \sum_{j=1:y_j=1}^{l} p_{ij} + \sum_{j=l+1}^{n} p_{ij} f_j^{t-1} & t \geq 2 \\ \sum_{j=1:y_j=1}^{l} p_{ij} & t = 1 \end{cases} = \begin{cases} \sum_{j=1}^{l} p_{ij} f_j + \sum_{j=l+1}^{n} p_{ij} f_j^{t-1} & t \geq 2 \\ \sum_{j=1}^{l} p_{ij} f_j & t = 1 \end{cases}$$

If we allow t to tend to infinity and denote $f_i \triangleq \lim_{t \to +\infty} f_i^t$, then we have

$$f_i = \sum_{j=1}^{l} p_{ij} f_j + \sum_{j=l+1}^{n} p_{ij} f_j, \quad \text{for all } i \in U. \tag{2.8}$$

Putting $i = l+1, ..., n$ together from (2.8) and using matrix notation, it gives

$$\left(I - P_{UU}\right) f_U = P_{UL} f_L,$$

where $f_L = \left(f_1, ..., f_l\right)^T$, $f_U = \left(f_{l+1}, ..., f_n\right)^T$,

$$P_{UU} = \begin{pmatrix} 0 & p_{l+1,l+2} & \cdots & p_{l+1,n} \\ p_{l+2,l+1} & 0 & \cdots & p_{l+2,n} \\ \vdots & \vdots & \vdots & \vdots \\ p_{n,l+1} & p_{n,l+2} & \cdots & 0 \end{pmatrix}, \text{ and } P_{UL} = \begin{pmatrix} p_{l+1,1} & p_{l+1,2} & \cdots & p_{l+1,l} \\ p_{l+2,1} & p_{l+1,2} & \cdots & p_{l+2,l} \\ \vdots & \vdots & \vdots & \vdots \\ p_{n,1} & p_{n,2} & \cdots & p_{n,l} \end{pmatrix}.$$

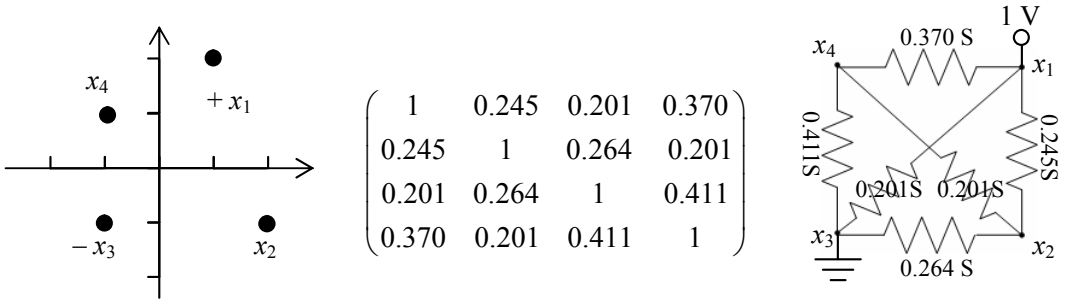
This finally gives $f_U = \left(I - P_{UU}\right)^{-1} P_{UL} f_L$, which is the same as (2.4). In this sense, the original model is basing the classification on the probability of hitting a positively labelled node in the random walk.

One important insight we make from this interpretation is that the classification is on a local basis. That is, due to the hit-and-stop rule on labelled data points, the labelled data point in the vicinity of an unlabelled point *x* will affect the prediction of *x* by far more significantly than those faraway points. This means that the graph can be very disconnected and in each small subgraph (cluster), the labelled point there will dominate the labelling of all unlabelled points in that cluster. This property is not clear when we review the harmonic property (2.6) which is in the form of global average, though the weights are small for far distant pairs of points. Moreover, the right-hand side of (2.6) also involves unlabelled data. Thus compared with this random walk interpretation, it tells us less about the relationship between a *single* unlabelled data and the set of labelled data points.

2.4 Interpreting HEM 2: Electric circuit

According to (Doyle & Snell, 1984), we can imagine that the edges of *G* are resistors with conductance w_{ij} (not p_{ij}). Then we connect all positively labelled nodes to a +1V voltage

source and ground all negatively labelled nodes. The nodes corresponding to unlabelled points are not clamped to any voltage source, and their voltage at electric equilibrium just corresponds to the optimal f_U^{opt} in the model. The following is an example with x_1 labelled as positive and x_3 labelled as negative, and x_2, x_4 are unlabelled. $\sigma = 1.5$.



original position of points similarity/weight matrix corresponding circuit

Figure 2.1 Electric network interpretation of HEM

Then the total power consumed on these resistors is:

$$\sum_{i,j=1}^{n} g_{ij}\left(V_i - V_j\right)^2 = \sum_{i,j=1}^{n} w_{ij}\left(f_i - f_j\right)^2 ,$$

which is the same as (2.2). As we know from physics, a pure resistor electric network always stays at the unique equilibrium state where the power dissipated from the resistors is minimized. So the minimiser f_U given by (2.4) can be interpreted as the node voltage, which minimizes the network energy consumption. Besides, according to Kirchoff's and Ohm's laws, the solution automatically satisfies the harmonic property (2.6).

This interpretation has suggested to us some other formulations based on physics, with the same idea of fixing some bound conditions (like grounding or voltage source) and then deriving the state of free nodes/variables by considering the mutual influence among the ob-

jects in the whole system. We can replace the resistors with capacitors or inductors which will lead to a time-sensitive exponential charging curve, similar to (Kondor & Lafferty, 2002). Furthermore, some electromagnetic induction models can be exploited as well. However, this kind of models seems less well motivated in terms of machine learning theory. So we put the next physics model into the next section, showing how the physics laws can be used to motivate machine learning intuitions.

2.5 Interpreting HEM 3: Harmonic mean

The harmonic mean principle (2.6) is very appealing. It says that the value of a function f at a point x should be equal to or close to the average value of f at other points in the proximity of x, weighted by their similarity to x. This principle has been justified by the physical model of electric circuit. Now we show another interesting natural model which also supports this principle. Suppose we introduce to the space some point charges of $+Q$ or $-Q$ Coulomb. Each $+Q$ or $-Q$ point charge corresponds to a positively or negatively labelled instance. They are insulated from everything else. Then we add into the space some other grounded small metal spheres, which are also viewed as point charges. These correspond to the unlabelled data. We know the mutual distance between all pairs of point charges, which can be related to the weights of the graph G. As unlabelled point charges are grounded, there will be some induced charges on them at static electric equilibrium. Then whether these spheres have positive net charge or negative net charge will indicate whether the data point is positive or negative.

As a simple example, we see how a single positive point charge elicits the induced charge on a grounded point charge. Suppose the left insulated metal sphere has +Q. The right grounded metal sphere has radius r and the distance between the centres of the two spheres is d. Assuming that $d \gg r$, we can view the spheres as point charges.

Figure 2.2 Static induction model for semi-supervised learning

The left +Q's contribution to the right point's potential is Q/d. Suppose the right (unlabelled) grounded point has q coulomb induced charge, then the fact that it is grounded entails $Q/d + q/r = 0$. Now suppose there are l insulated point charges each with charge $q_i \in \{+Q, -Q\}$ (i = 1, 2, …, l), and other u (unlabelled) grounded charges q_{l+1}, …, q_{l+u}. Let the radius of all points be r and the distance between point i and point j be d_{ij} ($d_{ij} \gg r$). Then by the fact that the voltage of the unlabelled charges is 0, we have

$$\frac{q_i}{r} = -\sum_{j=1:j\neq i}^{n} \frac{q_j}{d_{ij}} \quad \text{for all} \quad i = l+1,...,l+u .$$

These u equations are very similar to the harmonic properties in (2.6), by replacing the exponential decaying with inverse linear decaying. The only caveat is the negative sign, which people may attribute to the fact that like charges repel each other while opposite charges attract. However, even if we manually change the rule to its opposite, the negative sign will still be there, i.e., $Q/d + q/r = 0$ still holds for Figure 2.2. We omit the physics details here. This model can be regarded as another natural instance of harmonic mean.

2.6 Interpreting HEM 4: Graph kernels and heat/diffusion kernels

In section 2.2, we defined the graph Laplacian and used it just as a notation. We now study the real role of this notion and then understand this model in a more profound way.

2.6.1 Preliminaries of graph Laplacian

Assume there is an undirected and unweighted graph $G = \langle V, E \rangle$, with vertex set $V = \{1...n\}$, and edge set E. We write $i \sim j$ ($i \neq j$) to denote that i and j are connected, i.e., $(i, j) \in E$, $(j, i) \in E$. First define the adjacency matrix W: $w_{ij} = 1$ if $i \sim j$, and 0 otherwise. $w_{ii} = 0$. So W is symmetric with diagonal values being 0. In a more general setting, we can extend w_{ij} to $[0, +\infty)$. Denote $d_i \triangleq \sum_{j=1}^{n} w_{ij}$ for all i and assume that $d_i > 0$, $\forall i$. All the results below, mainly from (Smola & Kondor, 2003), apply to this general definition of adjacency matrix.

Now we define D as an $n \times n$ diagonal matrix with $D_{ii} \triangleq d_i$. The transitional probability matrix P of G, the combinatorial Laplacian Δ, and normalized Laplacian $\tilde{\Delta}$ are defined as

$$P = D^{-1}W, \quad \Delta \triangleq D - W = D(I - P), \quad \tilde{\Delta} \triangleq D^{-\frac{1}{2}}\Delta D^{-\frac{1}{2}} = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}} \tag{2.9}$$

respectively. Then we list some important properties of Δ and $\tilde{\Delta}$ with brief proof.

1. (Chung, 1997) Δ and $\tilde{\Delta}$ are both symmetric and positive semi-definite. The eigenvalues of $\tilde{\Delta}$ are all real and in $[0, 2]$ and the number of 0 eigenvalues is equal to the number of disjoint components in G.

These bounds are straightforward from Gerschgorin's Theorem (Chung, 1997).

2. (Smola & Kondor, 2003) If d_i is constant throughout all i (so denoted as d), then the relationship between the eigenvalues and eigenvectors of Δ, $\tilde{\Delta}$, and P is:

	W	P	Δ	$\tilde{\Delta}$
Eigenvalue	λ_i	λ_i / d	$d - \lambda_i$	$1 - \lambda_i / d$
Eigenvector	v_i	v_i	v_i	v_i

Table 2.1 Relationship of eigen-system for graph matrices

If $\sum_{j=1}^{n} w_{ij}$ is not constant throughout all i, then there is no simple relationship. The only surviving property is that P's eigenvector v_i corresponds to $\tilde{\Delta}$'s eigenvectors $D^{\frac{1}{2}} v_i$, and P's eigenvalue λ_i corresponds to $1 - \lambda_i$. This is because if $Pv_i = \lambda_i v_i$, then

$$\tilde{\Delta} D^{\frac{1}{2}} v_i = \left(I - D^{\frac{1}{2}} P D^{-\frac{1}{2}} \right) D^{\frac{1}{2}} v_i = D^{\frac{1}{2}} v_i - D^{\frac{1}{2}} P v_i = \left(1 - \lambda_i\right) D^{\frac{1}{2}} v_i .$$

This result is useful because it is normally easier to calculate P than $\tilde{\Delta}$, particularly when we need to take derivatives of them. As most important conclusions are expressed in terms of the eigen-system of normalized $\tilde{\Delta}$, it will be convenient to study the theoretical properties in $\tilde{\Delta}$ and then implement/calculate by using P in practice.

3. (Smola & Kondor, 2003) Δ and $\tilde{\Delta}$ can also be viewed as linear operators on functions $\boldsymbol{f} : V \mapsto \mathbb{R}$. Let $\boldsymbol{f} \triangleq \left(f_1, f_2, ..., f_n\right)^T$, then we can endow the functional space of $\boldsymbol{f}$ with a *semi*-norm defined as the inner product between $\boldsymbol{f}$ and $\Delta \boldsymbol{f}$:

$$\left\| \boldsymbol{f} \right\|_{\Delta}^2 \triangleq \left\langle \boldsymbol{f}, \Delta \boldsymbol{f} \right\rangle = \boldsymbol{f}^T \Delta \boldsymbol{f} = \tfrac{1}{2} \sum_{i \sim j} w_{ij} \left(f_i - f_j \right)^2 \quad \text{for all } \boldsymbol{f} \in \mathbb{R}^n . \tag{2.10}$$

This semi-norm (since $\left\| \boldsymbol{f} \right\|_{\Delta}^2 = 0$ iff $\boldsymbol{f}$ is a constant vector) depicts the weighted smoothness of $\boldsymbol{f}$ on the n vertices.

4. (Chung, 1997) Δ is a discrete approximation on the graph manifold of the Laplacian operator $\nabla^2 = \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \cdots + \frac{\partial^2}{\partial x_m^2}$ in continuous spaces.

First, the approximation relationship can be demonstrated by looking at an *m*-dimensional lattice. Suppose the orthonormal bases are $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_m$. $\boldsymbol{x} = x_1\boldsymbol{e}_1 + \ldots + x_m\boldsymbol{e}_m$. where x_i are integers. Let $f(x) = \boldsymbol{f}_{x_1,\ldots,x_m}$. Then

$$\nabla^2 f(x) = \sum_{i=1}^{m} \frac{\partial}{\partial x_i^2} f(x) \approx \sum_{i=1}^{m} \frac{\frac{\partial}{\partial x_i} f\left(x + \frac{1}{2}\boldsymbol{e}_i\right) - \frac{\partial}{\partial x_i} f\left(x - \frac{1}{2}\boldsymbol{e}_i\right)}{\delta}$$

$$\approx \sum_{i=1}^{m} \frac{f(x+\boldsymbol{e}_i) + f(x-\boldsymbol{e}_i) - 2f(x)}{\delta^2} = -\frac{1}{\delta^2}\left[\Delta \boldsymbol{f}\right]_{x_1,\ldots,x_m}.$$

Second, we show that Laplacian operator ∇^2 also depicts the smoothness of *f*, but in a continuous compact space Ω:

$$\|f\|_\Delta^2 = \langle f, \Delta f \rangle = \int_\Omega f(\Delta f)\, d\omega = \int_\Omega (\nabla f) \cdot (\nabla f)\, d\omega = \int_\Omega \|\nabla f\|^2\, d\omega.$$

where a simple integral by part is applied. The meaning of smoothness by integrating $\|\nabla f\|^2$ is very clear because $\|\nabla f\|^2$ depicts the rate of local variance of *f*.

5. (Original work) If $w_{ij} > 0$ for all *i*, *j*, and *S* is a proper subset of *V*, then the eigenvalues of $\tilde{\Delta}_{SS}$ are in (0, 2), where $\tilde{\Delta}_{SS}$ stands for the sub-matrix of $\tilde{\Delta}$ restricted to *S*.

This property means the eigenvalues of $I - \tilde{\Delta}_{SS}$ are all in (–1, 1) and this is important to prove the convergence of some processes to be discussed later. To prove it, we see now the $(i, i)^{th}$ diagonal elements of Δ_{SS} (not $\tilde{\Delta}_{SS}$) is larger than the negative sum of all other elements in the i^{th} row of Δ_{SS}, which we denote as $d_{S,i}$. Denote $\Delta_{SS} \triangleq D_{SS} + \Delta'_{SS}$, where D_{ss} is a diagonal matrix and Δ'_{SS} has the same off-diagonal elements as Δ_{SS}, and its $(i, i)^{th}$ diagonal element is $d_{S,i}$. Therefore, D_{ss} has a straight positive diagonal and it is positive

definite. By above property 1, we already know that Δ'_{SS} is positive semi-definite. So Δ_{SS} is positive definite and its eigenvalues are all positive. By definition (2.9), $\tilde{\Delta}_{SS}$ is also positive semi-definite.

To prove that the eigenvalues of $\tilde{\Delta}_{SS}$ are all less than 2, we only need to show that $2I - \tilde{\Delta}_{SS}$ or $(2D - \Delta)_{SS}$ is positive definite, where $(2D - \Delta)_{SS}$ stands for the submatrix of $2D - \Delta$ restricted to *S*. The proof is similar to the above one, except that we now use the fact that $2D - \Delta = D + W$ is positive semi-definite. We restrict on *S* which results in weakening of the off-diagonal, yielding a positive definite matrix in consequence. □

We will relax the condition of $w_{ij} > 0$ into the assumption that there exists a *path* from *every* node in *S* to a *certain* node in $V \setminus S$ in section 4.4.1.

2.6.2 Graph and kernel interpretation 1: discrete time soft label summation

(Original work) We propose considering the following process. Suppose at the beginning $t = 0$, the *u* unlabelled points have soft labels f_U^0. Then at time *t*, the average of other points' soft labels weighted by their similarity to x_i is: $\sum_{j=1:j\neq i}^{u} w_{ij} f_j^t \Big/ \sum_{j=1:j\neq i}^{u} w_{ij}$. So we update the f_i^t by:

$$f_i^{t+1} - f_i^t = \delta_i^t \triangleq \sum_{j=1:j\neq i}^{u} w_{ij} f_j^t \Big/ \sum_{j=1:j\neq i}^{u} w_{ij} - f_i^t , \tag{2.11}$$

which bears clear resemblance to some basic forms of neural network update with δ_i^t being the difference between the weighted neighbours' average and f_i^t itself. Sometimes,

$f_i^{t+1} - f_i^t = \eta \delta_i^t$ is used where $\eta \in (0, 1]$. We simplify (2.11) into:

$$f_i^{t+1} \leftarrow \sum_{j=1:j\neq i}^{u} w_{ij} f_j^t \Big/ \sum_{j=1:j\neq i}^{u} w_{ij} \qquad \text{for } t = 0, 1, \ldots \tag{2.12}$$

Now we have $f_U^{t+1} = P_{UU} f_U^t$, where $p_{ij} = w_{ij} \Big/ \sum_{j=1}^{n} w_{ij}$ ($w_{ii} = 0$ by definition). We define the final determinant soft label as the sum of the soft labels over all t:

$$f_U \triangleq \sum_{t=0}^{+\infty} f_U^t = \sum_{t=0}^{+\infty} \left(P_{UU}^t f_U^0 \right) = \left(\sum_{t=0}^{+\infty} P_{UU}^t \right) f_U^0 . \tag{2.13}$$

We first need to show that this definition converges. Recall the property 5 in section 2.6.1. We know that the eigenvalues of $I - \tilde{\Delta}_{UU}$ are in (–1, 1). So $\sum_{t=0}^{+\infty} \left(I - \tilde{\Delta}_{UU} \right)^t$ converges. Moreover, as $\tilde{\Delta} = I - D^{\frac{1}{2}} P D^{-\frac{1}{2}}$ and D is diagonal, we have

$$I - \tilde{\Delta}_{UU} = D_{UU}^{1/2} P_{UU} D_{UU}^{-1/2}, \tag{2.14}$$

and $\left(I - \tilde{\Delta}_{UU} \right)^t = D_{UU}^{1/2} \left(P_{UU} \right)^t D_{UU}^{-1/2}$, $f_U^t = D_{UU}^{-1/2} \left(I - \tilde{\Delta}_{UU} \right)^t D_{UU}^{1/2} f_U^0$ for all $t = 0, 1, \ldots$.

Thus $\sum_{t=0}^{+\infty} P_{UU}^t = D_{UU}^{-1/2} \left(\sum_{t=0}^{+\infty} \left(I - \tilde{\Delta}_{UU} \right)^t \right) D_{UU}^{1/2}$ (then eigen-decompose $I - \tilde{\Delta}_{UU}$)

$$= D_{UU}^{-1/2} \left(\sum_{t=0}^{+\infty} \sum_{i=1}^{u} \lambda_i^t \phi_i \phi_i^T \right) D_{UU}^{1/2} \qquad (|\lambda_i| < 1, \text{ so summation is interchangeable})$$

$$= D_{UU}^{-1/2} \left(\sum_{i=1}^{u} \sum_{t=0}^{+\infty} \lambda_i^t \phi_i \phi_i^T \right) D_{UU}^{1/2} = D_{UU}^{-1/2} \left(\sum_{i=1}^{u} \left(1 - \lambda_i \right)^{-1} \phi_i \phi_i^T \right) D_{UU}^{1/2}$$

$$= D_{UU}^{-1/2} \left(I - \left(I - \tilde{\Delta}_{UU} \right) \right)^{-1} D_{UU}^{1/2} = D_{UU}^{-1/2} \tilde{\Delta}_{UU}^{-1} D_{UU}^{1/2}$$

$$= D_{UU}^{-1/2} \left(I - D_{UU}^{1/2} P_{UU} D_{UU}^{-1/2} \right)^{-1} D_{UU}^{1/2} = \left(I - P_{UU} \right)^{-1} . \tag{2.15}$$

So, f_U is well defined by (2.13) and the exact value is $f_U = \left(I - P_{UU} \right)^{-1} f_U^0$. (2.16)

Note although the final result of (2.15) is not surprising, we can not derive it directly because there is no guarantee that the infinite summation will converge. We can also make use of the property 2 in section 2.6.1 to derive the result. Comparing with (2.4), we have $f_U^0 = P_{UL} f_L$, which can be simply modelled by propagating the labels of f_L to the u unlabelled

points through probability P_{UL}.

2.6.3 Graph and kernel interpretation 2: continuous time soft label integral

(Original work) As a straightforward extension to continuous time soft label integral, we modify (2.11) and define the dynamic system as:

$$\frac{\partial}{\partial t} f_U^t = -\left(I - P_{UU}\right) f_U^t = -D_{UU}^{-1/2} \tilde{\Delta}_{UU} D_{UU}^{1/2} f_U^t . \tag{2.17}$$

Let $g_U^t \triangleq D_{UU}^{1/2} f_U^t$, then (2.17) becomes

$$\frac{\partial}{\partial t} g_U^t = -\tilde{\Delta}_{UU} g_U^t . \tag{2.18}$$

This is the typical heat diffusion equation, whose solution uses the exponentiated matrix:

$$g_U^t = \exp\left(-t\tilde{\Delta}_{UU}\right) g_U^0 . \tag{2.19}$$

The matrix $\exp\left(-t\tilde{\Delta}_{UU}\right)$ is exactly the *diffusion kernel* proposed by (Kondor & Lafferty, 2002). Its $(i, j)^{th}$ element can be visualized as the quantity of some substance that would accumulate at vertex x_j after a given amount of time t if we injected the substance at vertex x_i at time 0 and let it diffuse through the graph along the edges according to the conductivity defined by $\tilde{\Delta}_{UU}$.

Now the final soft label is defined by

$$\begin{aligned} f_U &\triangleq \int_{t=0}^{+\infty} f_U^t dt = D_{UU}^{-1/2} \int_{t=0}^{+\infty} g_U^t dt = D_{UU}^{-1/2} \int_{t=0}^{+\infty} \exp\left(-t\tilde{\Delta}_{UU}\right) dt \cdot g_U^0 = D_{UU}^{-1/2} \tilde{\Delta}_{UU}^{-1} g_U^0 \\ &= D_{UU}^{-1/2} D_{UU}^{1/2} \left(I - P_{UU}\right)^{-1} D_{UU}^{-1/2} D_{UU}^{1/2} f_U^0 = \left(I - P_{UU}\right)^{-1} f_U^0 \end{aligned} \tag{2.20}$$

which is the same as (2.16).

(Zhu et al., 2003a) pointed out that the HEM model is different from many time-dependent

models such as diffusion kernels in (Kondor & Lafferty, 2002) and fixed step Markov random walks in (Szummer & Jaakkola, 2001). Now we find that the difference is just because HEM summed/integrated out the time *t* for the soft labels.

2.7 Interpreting HEM 5: Laplacian equation with Dirichlet Green's functions

(Zhu et al., 2003a) pointed out the relationship of the harmonic energy minimization model with the discrete Laplace equation with Dirichlet boundary conditions, solved by using Green's function (Chung & Yau, 2000). The Laplace equation is defined as:

$$\Delta f(x) = \sum_{y=1}^{u} \left(f(x) - f(y) \right) p_{xy} = g(x) \qquad \text{for all } x \in U,$$

and the Dirichlet boundary condition is defined as

$$f(x) = \sigma(x) = f_x \qquad \text{for all } x \in L.$$

Here $f(x)$ is interpreted as the expected *hard* label of the first hit labelled point. So $g(x) \triangleq \sum_{y \in L} w_{xy} f_y$ for all $x \in U$. According to the Theorems 1 and 2 in (Chung & Yau, 2000), the solution can be written as:

$$f(z) = \sum_{i=1}^{u} \left(\frac{1}{\lambda_i} \sum_{\substack{x \in U, y \in L \\ p_{xy} \neq 0}} d_x^{-1/2} \phi_i(x) f_y \right) d_z^{-1/2} \phi_i + [Gg]_z \quad \text{for all } z \in U,$$

where λ_i and ϕ_i are the eigenvalues and eigenvectors of $\tilde{\Delta}_{UU}$, and G is the Green's function of $I - P_{UU}$. After calculation, we find $f_U = G P_{UL} f_L = \left(I - P_{UU} \right)^{-1} P_{UL} f_L$.

2.8 Applying matrix inversion lemma

In this section, we see how matrix inversion lemma can be utilized for active learning, in-

ductive learning and leave-one-out cross validation model selection. Active learning is efficiently combined with semi-supervised learning in the framework of HEM through the application of matrix inversion lemma and marginal Gaussian distribution formulation (Zhu et al., 2003c). We mention these algorithms because they rely on the matrix inversion lemma, an indispensable tool to make our later hyperparameter learning algorithm tractable. Only the first section active learning is by (Zhu et al., 2003c) and the rest are original work.

2.8.1 Active learning

First of all, the true risk $\mathcal{R}(f)$ of harmonic function f is defined as:

$$\mathcal{R}(f) \triangleq \sum_{i=1}^{n} \sum_{y_i=0,1} \delta\left(\operatorname{sgn}(f_i) \neq y_i\right) p^*(y_i \mid L),$$

where sgn(f_i) is the decision rule, e.g., sgn(f_i) = 1 if f_i > 0.5 and 0 otherwise. $p^*(y_i \mid L)$ is the unknown and incomputable true posterior label distribution at node i given L. Assume $p^*(y_i = 1 \mid L) \approx f_i$, which is very similar to self-training assumptions, then the estimated risk becomes $\hat{\mathcal{R}}(f) \triangleq \sum_{i=1}^{n} \delta\left(\operatorname{sgn}(f_i) \neq 0\right)(1-f_i) + \delta\left(\operatorname{sgn}(f_i) \neq 1\right) f_i = \sum_{i=1}^{n} \min(f_i, 1-f_i)$.

Now suppose we query the unlabelled node x_k and expect to receive answer y_k, then after re-training with (x_k, y_k), there will be a new harmonic function $f^{+(x_k, y_k)}$, whose estimated risk will be: $\hat{\mathcal{R}}\left(f^{+(x_k, y_k)}\right) = \sum_{i=1}^{n} \min\left(f_i^{+(x_k, y_k)},\ 1 - f_i^{+(x_k, y_k)}\right)$. However we still do not know y_k so we assume that $f_k \approx p^*(y_k = 1 \mid L)$ and calculate the expected estimated risk:

$$\hat{\mathcal{R}}\left(f^{+x_k}\right) = (1-f_k)\hat{R}\left(f^{+(x_k,0)}\right) + f_k \hat{R}\left(f^{+(x_k,1)}\right),$$

And finally the query k^{opt} chosen is: $k^{opt} = \arg\min_k \hat{R}\left(f^{+x_k}\right)$.

There are two computationally intensive steps and the approaches proposed to tackle them

are interesting. Firstly, we need to calculate $f^{+(x_k,y_k)}$:

$$f^{+(x_k,y_k)} = -\Delta^{-1}_{U\backslash k,U\backslash k}\Delta_{U\backslash k,L\cup k}f_{L\cup k},$$

where $\Delta^{-1}_{U\backslash k,U\backslash k}$ is the matrix after removing the k^{th} row and k^{th} column from Δ_{UU}. Naïvely inverting the matrix and multiplying matrix and vector for all $k = 1\ldots u$ will be very costly. The crucial observation is that f_U (its original meaning as a random variable noted immediately after (2.5), while most of the time we used it as a shorthand for f_U^{opt}) conforms to a Gaussian distribution with mean $-\Delta^{-1}_{UU}\Delta_{UL}f_L$ and covariance Δ^{-1}_{UU}. $f^{+(x_k,y_k)}$ is conditioning this Gaussian distribution on $f_k = y_k$, and calculating the posterior mean. There is a direct formula proved using Schur complement in (Jordan, preprint):

$$f^{+(x_k,y_k)} = f_U + (y_k - f_k)(\Delta^{-1}_{UU})_{Uk} \big/ (\Delta^{-1}_{UU})_{kk} \tag{2.21}$$

where $(\Delta^{-1}_{UU})_{Uk}$ is the k^{th} column of Δ^{-1}_{UU} and $(\Delta^{-1}_{UU})_{kk}$ is the $(k, k)^{th}$ element of Δ^{-1}_{UU}. So $f^{+(x_k,y_k)}$ can be calculated in only $O(u)$ time.

Secondly, if we want to select one more query after querying x_k and getting the answer y_k, then it is necessary to calculate the inverse of $\Delta_{U\backslash k,U\backslash k}$, a matrix removing the k^{th} column and k^{th} row from Δ_{UU}. Consider $k = 1$ for instance. To calculate $\Delta^{-1}_{U\backslash 1,U\backslash 1}$, we only need to calculate the inverse of $\Delta'_{UU} = \begin{pmatrix} \Delta_{11} & 0 \\ 0 & \Delta_{U\backslash 1,U\backslash 1} \end{pmatrix}$, since $\left(\Delta'_{UU}\right)^{-1} = \begin{pmatrix} \Delta_{11}^{-1} & 0 \\ 0 & \Delta^{-1}_{U\backslash 1,U\backslash 1} \end{pmatrix}$.

But Δ'_{UU} is different from Δ_{UU} only by the 1^{st} row and 1^{st} column, so there exist two vectors α_k and β_k, such that $\Delta'_{UU} = \Delta_{UU} + e_u\alpha^T + \beta e_u^T$, where $e_u = (1, 0, \ldots, 0)^T \in \mathbb{R}^u$. The matrix inversion lemma in its most primitive form says for any two vectors s and t,

$$\left(A + st^T\right)^{-1} = A^{-1} - \frac{A^{-1}s \cdot t^T A^{-1}}{1 + s^T At} \quad \text{(as long as } A + st^T \text{ is invertible, i.e., } 1 + s^T At \neq 0\text{)}.$$

Therefore we can first invert $\Delta_{UU} + e_u\alpha^T$ on top of Δ_{UU} by viewing e_u as s and α as t.

Then we invert $\Delta_{UU} + e_u\alpha^T + \beta e_u^T$ on top of $\Delta_{UU} + e_u\alpha^T$ by viewing β as s and e_u as t. In sum, the computational cost is $O(u^2)$.

2.8.2 Inductive learning

In contrast to active learning where the major computational cost stems from inverting a matrix after *removing* one row and one column, the bottleneck of inductive learning is to invert a matrix after *adding* one row and one column. Still, the key tool is matrix inversion lemma.

Suppose the learner is given an unseen data x_{n+1}. Then we just need to calculate the new f_U:

$$f_U^{+x_{n+1}} = -\Delta_{U\cup(n+1),U\cup(n+1)}^{-1}\Delta_{U\cup(n+1),L} f_L .$$

The matrix $\Delta_{U\cup(n+1),U\cup(n+1)}$ is just adding one row and one column to Δ_{UU}:

$$\Delta_{U\cup(n+1),U\cup(n+1)} = \begin{pmatrix} \Delta_{UU} & \Delta_{U,n+1} \\ \Delta_{n+1,U} & \Delta_{n+1,n+1} \end{pmatrix}. \tag{2.22}$$

We can invert $\Delta_{U\cup(n+1),U\cup(n+1)}$ efficiently because we can apply matrix inversion for two times like in active learning on top of matrix $\begin{pmatrix} \Delta_{UU} & 0 \\ 0 & \Delta_{n+1,n+1} \end{pmatrix}$, whose inverse is $\begin{pmatrix} \Delta_{UU}^{-1} & 0 \\ 0 & \Delta_{n+1,n+1}^{-1} \end{pmatrix}$. So in sum, the cost for classifying an unseen data is $O(u^2+ul) = O(nu)$.

2.8.3 Online learning

For inductive learning, the unseen data is forgotten once it is classified and the prediction for the upcoming unseen data is not affected by the previously (seen) unseen data. However, the online learning in this model can make use of these data. We consider two settings.

1. The classifier is given the correct label of an unseen data immediately after it makes the prediction. So now the Δ_{UU} is fixed while Δ_{UL} and f_L grow. The cost for computing $-\Delta_{UU}^{-1}\Delta_{UL}f_L$ increases linearly to the number of unseen data s at $O(u^2+u(l+s))$.
2. The classifier does not know the correct label of an unseen data even after it outputs the prediction. Now we can update Δ_{UU} in a run, so that it grows larger and larger with more and more unseen data. Since the computational cost for updating Δ_{UU} and computing $-\Delta_{UU}^{-1}\Delta_{UL}f_L$ is $O(nu)$, it is acceptable in many cases. If there are s unseen data, then the total cost is $O(u^2+(u+1)^2+\ldots+(u+s)^2+ul+(u+1)l+\ldots+(u+s)l) = O(nsu+ns^2+s^3)$.

2.8.4 Leave-one-out cross validation

Leave-one-out cross validation (LOOCV) is usually an effective way for model selection. In the semi-supervised setting, this means for each $i \in L$, we view x_i as an unlabelled point and do transductive learning together with the pool of unlabelled data to get the soft label of x_i. Then we pick the model which makes the soft label maximally tally with the correct class (i.e., maximally close to 1 for positive data and maximally close to 0 for negative data), averaged over all $i \in L$.

Although the scarcity of labelled data assumed in semi-supervised learning makes LOOCV less justified by common theory, it is still an advisable way for model selection. The most prominent difficulty lies in computational cost, as the model must be retrained for l times. Since each training takes $O(u^3+ul)$, the total cost will be high $O(lu^3+ul^2)$. By virtue of matrix inversion lemma, we can reduce the total cost to $O(nu^2+ul^3)$, which is more managable

given $l \ll u$.

Again, the key idea is to look at the bottleneck of computation: matrix inversion. When x_i is held out from L, the Δ_{UU} is augmented by adding one row and one column, whose values correspond to x_i. Similar to what we have already discussed for (2.22), the new augmented matrix $\Delta_{U\cup i,U\cup i}$ can be inverted efficiently.

2.8.5 Two serious cautions

The simplicity of (2.21) offers not only delight, but also a serious caution: an algorithm may be unexpectedly *not* using unlabelled data at all, though the optimization is formulated as if it were semi-supervised learning.

The idea in (2.21) is that for a multivariate Gaussian distribution, if we condition on one variable, then the mean of the other variables can be simply calculated by scaling the corresponding columns in the covariance matrix. Putting it on a more concrete footing, suppose there is a Gaussian distributed random vector $\left(x_L^T, x_U^T\right)^T$ with mean $\left(\mu_L^T, \mu_U^T\right)^T$ and covariance matrix $\Sigma = \begin{pmatrix} \Sigma_{LL} & \Sigma_{LU} \\ \Sigma_{UL} & \Sigma_{UU} \end{pmatrix}$. Then the conditional probability $p\left(x_U \mid x_L\right)$ is a Gaussian distribution with mean $\mu_U + \Sigma_{UL}\Sigma_{LL}^{-1}\left(x_L - \mu_L\right)$, and covariance $\Sigma_{UU} - \Sigma_{UL}\Sigma_{LL}^{-1}\Sigma_{LU}$.

Now suppose we parameterize the covariance matrix directly, with each $(i, j)^{th}$ element determined only by x_i and x_j only. Then each element in the mean vector $\mu_U + \Sigma_{UL}\Sigma_{LL}^{-1}\left(x_L - \mu_L\right)$ will just depend on the corresponding row in Σ_{UL}. This is equiva-

lent to classifying the unlabelled data one by one, without exploiting their mutual relationship or mutual help. The HEM avoided this trap by parameterizing the *inverse* covariance matrix. In a more general picture, this phenomenon can also occur when regularizing by a directly parameterized kernel K (e.g., $y^T K^{-1} y$), where K is playing a similar role as covariance matrix as illustrated in Figure 2.3.

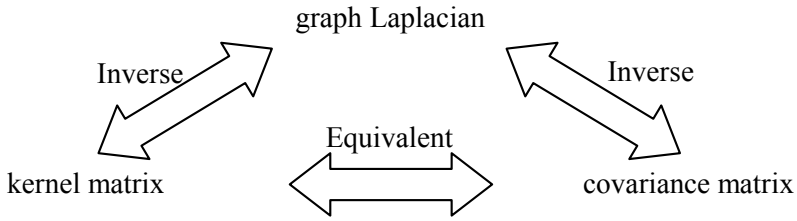


Figure 2.3 Relationship between graph Laplacian, kernel, and covariance matrix

The second caution is about overfitting in LOOCV. It seems surprising because LOOCV is designed just to avoid it. However unwise and intensive use of LOOCV can overfit. How? In http://www.autonlab.org/tutorials/overfit10.pdf, Andrew Moore used an example for illustration. Imagine there is a dataset with 50 records and 1000 attributes. We try 1000 linear regression models, each one using one of the attributes. The best of those 1000 looks good. But we realize it would have looked good even if the output had been purely random! The solution proposed therein is: hold out an additional test set before doing any model selection. Check the best model that performs well even on the additional test set. Or, use randomization testing.

Chapter 3 Graph Hyperparameter Learning

So far, all the algorithms are assuming that the graph of pairwise similarity measure is given. However, it is usually difficult to find a good graph. Using the kernel view of the HEM, constructing the graph is of the same importance as choosing a kernel function, which is also still under current popular research. Interestingly, however, there have been few graph learning algorithms published hitherto which interface directly to the raw data. As we pointed out in section 1.5.2, there do exist some algorithms based on a given distance/similarity measure, i.e., d_{ij} between example i and j. Nonetheless, now our interest is in learning the d_{ij}. If we use the Gaussian radial basis function as similarity measure, then learning the bandwidth is very similar to feature selection.

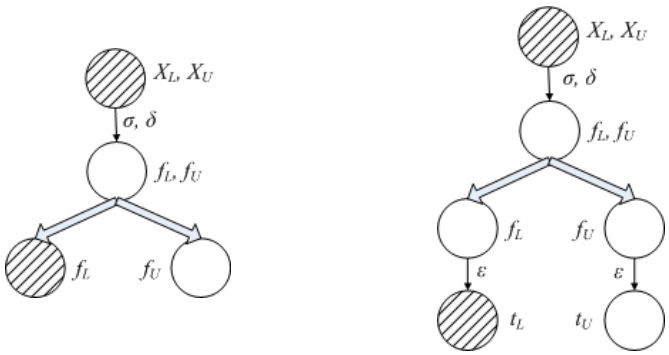
In this chapter, we will first review the preliminary graph learning algorithm proposed in (Zhu et al., 2003a), and a latest work by (Kapoor et al., 2005). The main part of this chapter will be proposing a simple graph learning principle based on the idea of leave-one-out cross validation. We call it leave-one-out hyperparameter learning (LOOHL). The primary contribution of the work is an efficient gradient calculation algorithm based on the matrix inversion lemma. It significantly reduced the computational complexity, with which most leave-one-out style learning algorithms are plagued.

3.1 Review of existing graph learning algorithms

In section 1.4, we reviewed some principles such as maximum margin and maximizing the

likelihood on labelled and unlabelled points. In (Kapoor et al., 2005), the problem is formulated in a Bayes net framework and the graph is learned by evidence/likelihood maximization, at the cost of approximate inference and expensive EM optimization. In (Zhu et al., 2003a), the spirit of maximum margin is adopted in a tailored form, called entropy minimization, to fit his HEM. We review these two approaches in this section before proposing our graph learning algorithm in the next section.

3.1.1 Bayes network through evidence maximization



(a) (b)

Figure 3.1 Bayes network of hyperparameter learning

First of all, the HEM is formulated in the Bayes inference framework, as in Figure 3.1 (a). We use double arrow from node (f_L, f_U) to (f_L) and (f_U) to represent that the first node *fully* determines the second and third nodes, which are actually just a subset of the first node. Bayes network does not have an efficient mechanism to express that one variable determines the joint distribution of two other variables, and this is the only choice. The f_L are fixed to the given labels. (X_L, X_U) and (f_L) are both observed, and we want to infer the distribution of, and mean or MAP of f_U. The only probabilistic edge is from (X_L, X_U) to (f_L, f_U), param-

eterized by the similarity matrix and graph Laplacian Δ with coefficients σ (e.g., RBF bandwidth), and possible graph smoothing (e.g., eigen-transform $r(\Delta)$ on graph Laplacian as reviewed in section 1.5.1.2) with coefficient δ, which is not used in original HEM (i.e., $r(\Delta)=\Delta$). This conditional probability is defined as:

$$p(f \mid X) \propto \exp\left(-\tfrac{1}{2} f^T \Delta(\sigma) f\right) = N\left(0, \Delta(\sigma)^{-1}\right) \tag{3.1}$$

where $f \triangleq \left(f_L^T, f_U^T\right)^T$. As the mean and MAP of a Gaussian distribution collapse, the resulting soft label f_U is just the same as the minimizer of (2.3):

$$f^T \Delta(\sigma) f = f_U^T \Delta_{UU}(\sigma) f_U + 2 f_U^T \Delta_{UL}(\sigma) f_L + f_L^T \Delta_{LL}(\sigma) f_L . \tag{3.2}$$

The important difference between transductive classification and graph learning is that the former minimizes (3.2) with respect to (wrt) f_U by fixing σ, while the latter minimizes wrt both f_U *and* σ. The latter can be optimized by first minimizing wrt f_U, and then σ.

$f_U^{opt}(\sigma) = -\Delta_{UU}^{-1}(\sigma)\Delta_{UL}(\sigma) f_L$, which gives

$$f^T \Delta(\sigma) f = f_L^T \left(\Delta_{LL}(\sigma) - \Delta_{UL}^T(\sigma)\Delta_{UU}^{-1}(\sigma)\Delta_{UL}(\sigma)\right) f_L .$$

Unfortunately, this will trivially push all $\sigma \to 0$, $w_{ij} \to 0$ and $E(f) = \sum_{i,j=1}^{n} w_{ij}\left(f_i - f_j\right)^2 \to 0$. So the normalized graph Laplacian is necessary to make it work.

Like HEM, the above formulation also assumes that the labels are noiseless. To incorporate the noise, (Kapoor et al., 2005) suggested using the Bayes net in Figure 3.1 (b). The difference from Figure 3.1 (a) is that the soft label *f* is no longer fixed to the observed hard label *t*. The assumption (3.1) is still used, but there is a new noise assumption earlier used

for Gaussian process classification by (Kim & Ghahramani, 2004; Opper & Winther, 1998):

$$p(t_i \mid f_i) = \begin{cases} 1-\varepsilon & f_i t_i \geq 0 \\ \varepsilon & f_i t_i < 0 \end{cases}, \quad \text{where } \varepsilon \in [0,1] \text{ stands for the noise level.}$$

Note $p(t_i \mid f_i)$ depends on the soft label f only via its sign. This flipping likelihood is different from other linear or quadratic slack in likelihood, and it is particularly desirable for cases where label errors are far from the decision boundary (Kim & Ghahramani, 2004).

Now $\{(X_L, X_U), (t_L)\}$ are observed and $\{(f_L, f_U), (t_U)\}$ are hidden. The parameters are $\{\sigma, \delta, \varepsilon\}$, and they are learned by maximizing the evidence likelihood of $P(X_L, X_U, t_L)$ through EM algorithm. Note the variable t_U is not the ancestor of any observed variable, so it is marginalized out and thus has no influence on $P(X_L, X_U, t_L)$. In the E-step, the posterior $q(f_L, f_U)$ is inferred by evidence propagation algorithm, while in the M-step the parameters are optimized by (approximate) gradient descent. Finally, to infer t_U, we only need to integrate out (f_L, f_U).

The performance of this model and actually many other Gaussian process models are highly dependent on the noise model. The learning and inference, which are usually intractable in the exact form, are normally carried out in an approximate way. Difficulty also arises from optimization over the graph smoothing coefficient δ (e.g., eigen-transform $r(\Delta)$ on graph Laplacian). This can usually transform the eigenvalues λ to $\exp(\sigma^2/2\lambda)$, $(aI - \lambda)^{-1}$, or even more complicated $(\cos \lambda\pi/4)^{-1}$, etc. It is either impossible to calculate the exact gradient wrt the parameters, or calculating the spectral transformed $r(\Delta)$ will be too expensive in every iteration. That is why (Kapoor et al., 2005) restricted the transform to the

simplest way: $r(\lambda) = \lambda + \delta$, which means transforming $\tilde{\Delta}$ to $\tilde{\Delta} + \delta I$.

3.1.2 Entropy minimization

An alternative graph learning approach is based on the maximum margin principle, in a form tailored for HEM. The idea is to make the classification on *unlabelled* points as confident as possible, with f_L clamped to 0/1. In other words, the f value of an unlabelled point x_i should be either close to 1 or close to 0, instead of around 0.5. Since f is bounded by (0, 1), this objective can be equivalently formulated as minimizing the entropy of f_i:

$$H(f_i) = -f_i \log f_i - (1 - f_i)\log(1 - f_i)$$

The entropy is minimized when f_i is either near 0 or 1 and maximized when f_i is 0.5. So the final objective function to *minimize* is:

$$H(f) \triangleq \frac{1}{u}\sum_{i=l+1}^{n} H(f_i) = -\frac{1}{u}\sum_{i=l+1}^{n}\left(f_i \log f_i + (1 - f_i)\log(1 - f_i)\right) \tag{3.3}$$

This idea is also used successfully in (Grandvalet & Bengio, 2004) and (Niu et al., 2005). It is different from normal entropy maximization formulations, where there is a unique optimal solution. There can be an exponentially large number of entropy minimizing solutions which suggests that the criterion may not work consistently. However, (Zhu et al., 2003a) argued that since the f_i are fixed on labelled points, the unlabelled points do not have much freedom in assuming so many possible values, i.e., most of the low entropy labellings are inconsistent with the labelled points. Unfortunately, they did not elaborate on this claim and our experimental results show that this criterion often fails to learn a good graph.

This criterion also suffers from the risk of overfitting. First, the number of features may be far larger than the number of labelled points and we can have considerable freedom to construct the graph into any one we want. Second, even if the number of parameters is small, there is still a trivial solution which yields entropy 0, when $\sigma_d \to 0$. The bandwidth is so small that $w_{ij} \gg w_{ij'}\left(j' \neq j\right)$, where x_j is the nearest neighbour of x_i. So p_{ij} is close to 1 and $p_{ij'} \approx 0$ for all $j' \neq j$ and $j' \neq i$. This results in the following procedure:

1. For each point in the labelled point set *x*, label its nearest unlabelled point with the same label as *x*;
2. Add that unlabelled point to the labelled point set, and repeat from 1.

This procedure does give a minimum entropy labelling, but it is obviously undesired unless the classes are perfectly separated.

The solution proposed in (Zhu et al., 2003a) in to add a uniform distribution regularizer, inspired by the work in PageRank algorithm (Ng et al., 2001b). They simply replace *P* with

$$\tilde{P} \triangleq \varepsilon \mathcal{U} + \left(1-\varepsilon\right) P \tag{3.4}$$

where $\mathcal{U}$ is the uniform matrix $\mathcal{U}_{ij} = n^{-1}$. This ensures that the graph is always fully connected. Numerically, this smoothing is also necessary to make the optimization proceed properly because in practice the transition matrix *I* – *P* quickly becomes close to singular if we do not use this smoothing. However, (Zhu et al., 2003a) did not say whether (3.4) is also used in classification, or just used in learning the graph. In experiments, we find that graph *P* performs consistently better than $\tilde{P}$, so we stick to *P* when doing final classifica-

tion.

Since this is not a convex optimization problem, we just apply gradient based optimization algorithms. The gradient is:

$$\frac{\partial H}{\partial \sigma_d} = \frac{1}{u}\sum_{i=l+1}^{n} \log\left(\frac{1-f_i}{f_i}\right)\frac{\partial f_i}{\partial \sigma_d}$$

where $\frac{\partial f_U}{\partial \sigma_d} = \left(I - \tilde{P}_{UU}\right)^{-1}\left(\frac{\partial \tilde{P}_{UU}}{\partial \sigma_d} f_U + \frac{\partial \tilde{P}_{UL}}{\partial \sigma_d} f_L\right)$, $\frac{\partial \tilde{P}_{UU}}{\partial \sigma_d} = (1-\varepsilon)\frac{\partial P_{UU}}{\partial \sigma_d}$, $\frac{\partial \tilde{P}_{UL}}{\partial \sigma_d} = (1-\varepsilon)\frac{\partial P_{UL}}{\partial \sigma_d}$,

by applying the fact that $d\boldsymbol{X}^{-1} = -\boldsymbol{X}^{-1}(d\boldsymbol{X})\boldsymbol{X}^{-1}$.

Since $p_{ij} = w_{ij} \Big/ \sum_{j=1}^{n} w_{ij}$, we have $\frac{\partial p_{ij}}{\partial \sigma_d} = \frac{\frac{\partial w_{ij}}{\partial \sigma_d} - p_{ij}\sum_{k=1}^{l+u}\frac{\partial w_{ik}}{\partial \sigma_d}}{\sum_{k=1}^{l+u} w_{ik}}$,

where $\frac{\partial w_{ij}}{\partial \sigma_d} = 2w_{ij}\left(x_{id} - x_{jd}\right)^2 \Big/ \sigma_d^3$ because $w_{ij} = \exp\left(-\sum_{d=1}^{m}\frac{\left(x_{i,d} - x_{j,d}\right)^2}{\sigma_d^2}\right)$.

Other transforms like CMN can be incorporated in a similar manner.

3.2 Leave-one-out hyperparameter learning: motivation and formulation

Graph learning algorithms mentioned above are relatively ad hoc. To design our own graph learning algorithm in a more principled manner, the first question we need to answer is what the desired properties of a graph are. In nature, the graph is just a parameter, and the most typical model selection method is *k*-fold cross validation, or leave-one-out cross validation. Motivated by this idea, we propose minimizing the leave-one-out (LOO) error on *labelled* data points. Suppose we hold out a labelled point t ($t \in L$), and we train by using the x_U, x_L, $y_L \backslash y_t$ and test on the point t. If $y_t = 1$, then we hope that the soft label f_t is

as close to 1 as possible and if $y_t = 0$, then we hope that the f_t is as close to 0 as possible.

Note this rule can be easily made independent of the classification algorithms. Using the random walk view, this is equivalent to the motivation that a particle leaving from a positively labelled point is expected to first hit (and absorbed by) another positively labelled point with as high probability as possible. Then we want to maximize the sum or product of these leave-one-out posterior probabilities.

Now we present the detailed formulation of leave-one-out hyper-parameter learning. In this section, as the math equations are pretty complex, we write in a slightly verbose style to help understanding by controlled redundancy. Suppose we examine the case when the t^{th} labelled data point is held out. Denote

$$P_{UU}^t \triangleq \begin{pmatrix} p_{tt} & p_{tU} \\ p_{Ut} & P_{UU} \end{pmatrix}, \quad (t \in L)$$

where $p_{Ut} \triangleq \left(p_{l+1,t}, \ldots, p_{n,t}\right)^T$, $p_{tU} \triangleq \left(p_{t,l+1}, \ldots, p_{t,n}\right)$,

$$P_{UL}^t \triangleq \begin{pmatrix} p_{t1} & \cdots & p_{t,t-1} & p_{t,t+1} & \cdots & p_{tl} \\ p_{l+1,1} & \cdots & p_{l+1,t-1} & p_{l+1,t+1} & \cdots & p_{l+1,l} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ p_{n,1} & \cdots & p_{n,t-1} & p_{n,t+1} & \cdots & p_{n,l} \end{pmatrix},$$

$$f_L^t \triangleq \left(f_1, \ldots, f_{t-1}, f_{t+1}, \ldots, f_l\right)^T,$$

$$f_U^t \triangleq \left(f_t, f_{l+1}, \ldots, f_n\right)^T = \left(I - \tilde{P}_{UU}^t\right)^{-1} \tilde{P}_{UL}^t f_L^t,$$

where $\tilde{P}$ is just applying (3.4) to P.

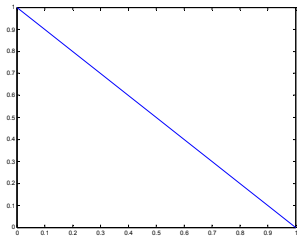
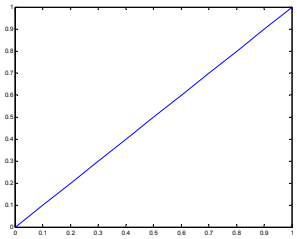
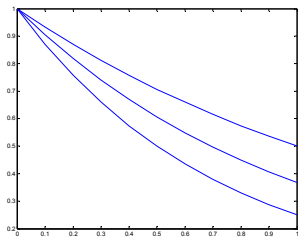
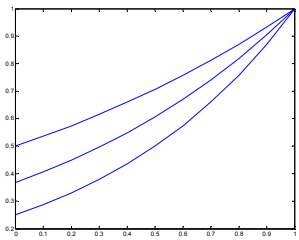
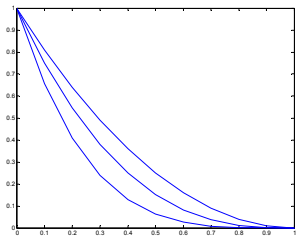
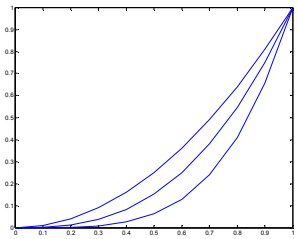
The LOO prediction on the held out t^{th} labelled point is the first component of f_U^t, so the

objective function to be **maximized** is:

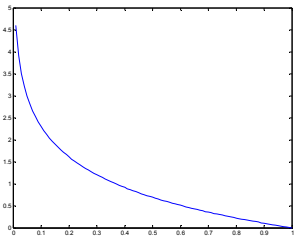
$$Q = \sum_{t=1}^{l} c_t f_{U,1}^t = \sum_{t=1}^{l} c_t s^T \left(I - \tilde{P}_{UU}^t \right)^{-1} \tilde{P}_{UL}^t f_L^t \,, \tag{3.5}$$

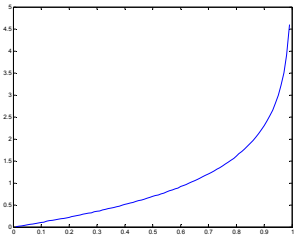
where $s = (1,0,...,0)^T \in \mathbb{R}^{u+1}$, and c_t is the class of the t^{th} example taking value in –1/+1 (not 0/1) corresponding to negative/positive respectively.

In addition to linear sum of $c_t f_{U,1}^t$, there can be other loss functions for each labelled point based on $f_{U,1}^t$. For example:

	positively labelled	negatively labelled
linear:		
	$loss = 1 - f$	$loss = f$
exponential:		
	$loss = a^{-f} \quad (a > 1)$	$loss = a^{f-1} \quad (a > 1)$
polynomial:		
	$loss = (1 - f)^a \quad (a > 1)$	$loss = f^a \quad (a > 1)$

logarithm:





$loss = -\log(f)$ $loss = -\log(1-f)$

Figure 3.2 Output transformation functions for leave-one-out loss

The common log likelihood criterion prompts minimizing the log loss function:

$$Q = -\sum_{t=1:c_t=1}^{l} \log\left(f_{U,1}^t\right) - \sum_{t=1:c_t=-1}^{l} \log\left(1-f_{U,1}^t\right).$$

The disadvantage is that the log value is unbounded near the boundaries, which will cause numerical problems in optimization. The main motivation of using non-linear loss function is that we want to make the loss change slowly near the desired limit and change quickly when far from it. In this way, we can push the f values to the desired limit more effectively. For example, suppose there are two positive points. In one case, their f values are 0.9, 0.4, while in another case, their f values are 0.6, 0.6. Using linear loss, we prefer the former case (1 – 0.9 + 1 – 0.4 = 0.7) to the latter one (1 – 0.6 + 1 – 0.6 = 0.8). However, the 0.4 is very far from 1 (and even misclassified if we use 0.5 threshold), so we may prefer the latter. Using the polynomial loss x^3, the former case's loss is $(1–0.9)^3+(1–0.4)^3 = 0.2170$, while the latter case's loss $(1–0.6)^3+(1–0.6)^3 = 0.1280 < 0.2170$. So the polynomial loss is intuitively more satisfactory than the linear loss. This phenomenon is often observed in experiments, when one f is being pushed to very close to 1 whereas some other f's are still below 0.5.

To make use of non-linear optimization packages, we need to calculate the partial derivatives of Q with respect to the hyperparameters. Assuming

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

the partial derivatives are:

$$\frac{\partial Q}{\partial \sigma_d} = \sum_{t=1}^{l} c_t s^T \left(\left(I - \tilde{P}_{UU}^t \right)^{-1} \frac{\partial \tilde{P}_{UU}^t}{\partial \sigma_d} \left(I - \tilde{P}_{UU}^t \right)^{-1} \tilde{P}_{UL}^t f_L^t + \left(I - \tilde{P}_{UU}^t \right)^{-1} \frac{\partial \tilde{P}_{UL}^t}{\partial \sigma_d} f_L^t \right)$$

$$= (1-\varepsilon) \sum_{t=1}^{l} c_t s^T \left(I - \tilde{P}_{UU}^t \right)^{-1} \left(\frac{\partial P_{UU}^t}{\partial \sigma_d} f_U^t + \frac{\partial P_{UL}^t}{\partial \sigma_d} f_L^t \right)$$

where the fact that $d\boldsymbol{X}^{-1} = -\boldsymbol{X}^{-1} (d\boldsymbol{X}) \boldsymbol{X}^{-1}$ is applied as in the section 3.1.2. Since in the original matrix P, $p_{ij} = w_{ij} \Big/ \sum_{k=1}^{n} w_{ik}$, we have

$$\frac{\partial p_{ij}}{\partial \sigma_d} = \frac{\frac{\partial w_{ij}}{\partial \sigma_d} - p_{ij} \sum_{k=1}^{l+u} \frac{\partial w_{ik}}{\partial \sigma_d}}{\sum_{k=1}^{l+u} w_{ik}}.$$

Finally, $\frac{\partial w_{ij}}{\partial \sigma_d} = 2 w_{ij} \left(x_{id} - x_{jd} \right)^2 \Big/ \sigma_d^3$. (3.6)

Note the equation (3.6) also indicates the generality of our algorithm in terms of the parametric form of similarity measures, as long as the measure is non-negative and continuous. We may safely replace the Gaussian RBF form with polynomial form or tanh, and the only change is the partial derivative in (3.6).

For convenience, we assume $\varepsilon = 0$ in the rest part of the chapter, and then $\tilde{P} = P$. It is easy to extend the following techniques to $\varepsilon > 0$.

The pseudo-code below calculates the gradient for the leave-one-out hyperparameter learning in a very naïve way.

1. Function value: $F = 0$; Gradient: $g = (0, \ldots, 0)^T \in \mathbb{R}^m$.

2. For each $t = 1...l$ (leave-one-out loop)

$$P_{UU}^t \leftarrow \left(\begin{array}{c|c} p_{t,t} & p_{tU} \\ \hline p_{Ut} & P_{UU} \end{array}\right), \quad p_{Ut} = \left(p_{l+1,t},...,p_{n,t}\right)^T \text{ and } p_{tU} = \left(p_{t,l+1},...,p_{t,n}\right), \tag{3.7}$$

$$P_{UL}^t \leftarrow \left(\begin{array}{ccc|ccc} p_{t1} & \cdots & p_{t,t-1} & p_{t,t+1} & \cdots & p_{tl} \\ \hline p_{l+1,1} & \cdots & p_{l+1,t-1} & p_{l+1,t+1} & \cdots & p_{l+1,l} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ p_{n,1} & \cdots & p_{n,t-1} & p_{n,t+1} & \cdots & p_{n,l} \end{array}\right), \tag{3.8}$$

$$f_U^t \leftarrow \left(I - P_{UU}^t\right)^{-1} P_{UL}^t f_L^t, \tag{3.9}$$

$$F \leftarrow F + c_t s^T \left(I - P_{UU}^t\right)^{-1} P_{UL}^t f_L^t. \tag{3.10}$$

Let W_{UU}^t and W_{UL}^t be the weight matrix with the same index corresponding to P_{UU}^t and P_{UL}^t. We will denote the $(i,j)^{th}$ element of W_{UU}^t as $w_{UU}^t(i,j)$. We use

$$\sum_{k=1}^n \frac{\partial w_{UN}^t(i,k)}{\partial \sigma_d} \quad \text{and} \quad \sum_{k=1}^n w_{UN}^t(i,k)$$

to denote the sum of the elements in the i^{th} row of $\left(\frac{\partial W_{UU}^t}{\partial \sigma_d} \quad \frac{\partial W_{UL}^t}{\partial \sigma_d}\right)$ and $\left(W_{UU}^t \quad W_{UL}^t\right)$ respectively. In fact, by (3.7) and (3.8), regardless of the value of t, the 2^{nd} row up to the $(u+1)^{th}$ row of W_{UU}^t and W_{UL}^t always corresponds to the u unlabelled points. The first row (i=1) always corresponds to the labelled data point currently held out. So for $i = 1$, $\sum_{k=1}^n \frac{\partial w_{UN}^t(i,k)}{\partial \sigma_d}$ and $\sum_{k=1}^n w_{UN}^t(i,k)$ are equal to the sum of the elements in the t^{th} row of $\frac{\partial W}{\partial \sigma_d}$ and W respectively. For $i = 2,...,u+1$, $\sum_{k=1}^n \frac{\partial w_{UN}^t(i,k)}{\partial \sigma_d}$ and

$\sum_{k=1}^{n} w_{UN}^{t}(i,k)$ are equal to the sum of the elements in the $(i + l - 1)^{th}$ row of $\frac{\partial W}{\partial \sigma_d}$ and W respectively. This is an important property which we will utilize later.

3. For each $d = 1 \ldots m$ (for all features inside the leave-one-out loop)

$$\frac{\partial P_{UU}^{t}(i,j)}{\partial \sigma_d} \leftarrow \frac{1}{\sum_{k=1}^{n} w_{UN}^{t}(i,k)} \left(\frac{\partial w_{UU}^{t}(i,j)}{\partial \sigma_d} - p_{UU}^{t}(i,j) \sum_{k=1}^{n} \frac{\partial w_{UN}^{t}(i,k)}{\partial \sigma_d} \right) \qquad (3.11)$$

$$i = 1 \ldots u+1, j = 1 \ldots u+1$$

$$\frac{\partial P_{UL}^{t}(i,j)}{\partial \sigma_d} \leftarrow \frac{1}{\sum_{k=1}^{n} w_{UN}^{t}(i,k)} \left(\frac{\partial w_{UL}^{t}(i,j)}{\partial \sigma_d} - p_{UL}^{t}(i,j) \sum_{k=1}^{n} \frac{\partial w_{UN}^{t}(i,k)}{\partial \sigma_d} \right)$$

(3.12)

$$i = 1 \ldots u+1, j = 1 \ldots l-1$$

where $\frac{\partial w_{UL}^{t}(i,j)}{\partial \sigma_d}$, if corresponds to the p^{th} and q^{th} example (indices of the original W), equals $2 w_{pq} \left(x_{p,d} - x_{q,d} \right)^2 / \sigma_d^3$.

$$g_d \leftarrow g_d + c_t s^T \left(I - P_{UU}^{t} \right)^{-1} \left(\frac{\partial P_{UU}^{t}}{\partial \sigma_d} f_U^t + \frac{\partial P_{UL}^{t}}{\partial \sigma_d} f_L^t \right) \qquad (3.13)$$

End

End

The naïve algorithm is very computationally expensive: $O\left(lu\left(mn + u^2\right)\right)$, just for calculating the gradient once. In detail, the calculation of W and P in (3.7) and (3.8) costs $O\left(mn^2\right)$. (3.9) and (3.10) cost $O\left(l\left(lu + u^2 + u^3\right)\right) = O\left(lnu + lu^3\right)$ (matrix inversion costs $O\left(u^3\right)$). (3.11) and (3.12) cost $O\left(lmnu\right)$. (3.13) costs $O\left(lmnu\right)$. To be more precise, matrix inversion is known to have the same difficulty as matrix multiplication, which costs

$O\left(u^{\log_2 7}\right) \approx O\left(u^{2.81}\right)$ by Strassen's algorithm and $O\left(u^{2.37}\right)$ by Winograd & Coppersmith's algorithm. See (Cormen et al., 2001) for details. So finally the complexity can be reduced to $O\left(lu\left(mn + u^{1.37}\right)\right)$.

This high complexity is due to the following drawbacks:

1. The expensive matrix inversion is performed for l times, though the matrices to be inverted are only slightly different by the first row and first column.
2. It ignores the fact that for each example x_i, there may be only a small number of features whose value is nonzero (active features). This is particularly true for text or image datasets.
3. In the leave-one-out loop, a lot of intermediate terms such as $\frac{\partial P_{UU}^t(i,j)}{\partial \sigma_d}$ and $\frac{\partial P_{UL}^t(i,j)}{\partial \sigma_d}$ are repeatedly calculated which constitutes a considerable waste. A careful pre-computation of common terms is desirable to reduce the cost.

3.3 An efficient implementation

By paying attention to the above three problems, we now propose an efficient implementation which helps to reduce the complexity to $O\left(\tilde{m}n^2 + lnu + u^{2.37}\right)$ for calculating the gradient, where $\tilde{m}$ is the average of mutually active features:

$$\tilde{m} \triangleq \frac{2}{n(n-1)} \sum_{1 \le i < j \le n} \left| \left\{ d \in 1...m \,\middle|\, x_i \text{ or } x_j \text{ is not zero on feature } d \right\} \right|.$$

This is already close to the lower bound because it is inevitable to calculate

$\frac{\partial w_{ij}}{\partial \sigma_d} = 2w_{ij}\left(x_{id} - x_{jd}\right)^2 / \sigma_d^3$ (for $i, j = 1...u$, $d = 1...\tilde{m}$), particularly $\left(x_{id} - x_{jd}\right)^2$ which can not be further factorized. Although $\left(x_{id} - x_{jd}\right)^2$ is constant throughout all iterations, we can not pre-compute it because the cost for storing the result is $O(u^2\tilde{m})$, usually beyond the memory capacity. The following describes the factorization procedure which yields the computational cost $O\left(\tilde{m}n^2 + lnu + u^{2.37}\right)$ for gradient calculation.

The first expensive step is matrix inversion, which costs $O\left(lu^3\right)$. To start with, we use matrix inversion lemma to reduce its cost to a one-time full matrix inversion and then some $O\left(u^2\right)$ operations for $l - 1$ times. In (3.7) and (3.9), the matrix to be inverted is $I - P_{UU}^t$. For different t, only the first row and first column of P_{UU}^t are different. In essence, we only need to efficiently invert a $u \times u$ matrix $B = A + e\alpha^T + \beta e^T$, where A^{-1} is known. Here α and β are two vectors and $e = (1, 0, \ldots, 0)^T \in \mathbb{R}^{u+1}$. By matrix inversion lemma,

$$\left(A + \alpha\beta^T\right)^{-1} = A^{-1} - \frac{A^{-1}\alpha \cdot \beta^T A^{-1}}{1 + \alpha^T A\beta}$$

we can invert $A + e\alpha^T$ and then $A + e\alpha^T + \beta e^T$ in $O\left(u^2\right)$ time.

However, we still cannot avoid inverting a full matrix for $t = 1$. Therefore, the cost of (3.9) has been reduced to $O\left(lu^2 + u^{2.37}\right)$, while (3.10) costs $O\left(lnu\right)$. So (3.9) and (3.10) cost $O\left(lnu + u^{2.37}\right)$ in sum.

The second expensive step is in (3.11), (3.12) and (3.13), particularly by the partial derivatives:

$$\frac{\partial P_{UU}^t(i,j)}{\partial \sigma_d} = \frac{1}{\sum_{k=1}^n w_{UN}^t(i,k)} \left(\frac{\partial w_{UU}^t(i,j)}{\partial \sigma_d} - p_{UU}^t(i,j) \sum_{k=1}^n \frac{\partial w_{UN}^t(i,k)}{\partial \sigma_d} \right)$$

$$\frac{\partial P_{UL}^t(i,j)}{\partial \sigma_d} = \frac{1}{\sum_{k=1}^n w_{UN}^t(i,k)} \left(\frac{\partial w_{UL}^t(i,j)}{\partial \sigma_d} - p_{UL}^t(i,j) \sum_{k=1}^n \frac{\partial w_{UN}^t(i,k)}{\partial \sigma_d} \right)$$

$$G_d \leftarrow G_d + c_t s^T \left(I - P_{UU}^t \right)^{-1} \left(\frac{\partial P_{UU}^t}{\partial \sigma_d} f_U^t + \frac{\partial P_{UL}^t}{\partial \sigma_d} f_L^t \right)$$

Examining them carefully,

$$\begin{aligned} \frac{\partial P_{UU}^t(i,\cdot)}{\partial \sigma_d} f_U^t &= \frac{1}{\sum_{k=1}^n w_{UN}^t(i,k)} \left(\frac{\partial w_{UU}^t(i,\cdot)}{\partial \sigma_d} - p_{UU}^t(i,\cdot) \sum_{k=1}^n \frac{\partial w_{UU}^t(i,k)}{\partial \sigma_d} \right) f_U^t \\ &= \frac{1}{\sum_{k=1}^n w_{UN}^t(i,k)} \left(\frac{\partial w_{UU}^t(i,\cdot)}{\partial \sigma_d} f_U^t - p_{UU}^t(i,\cdot) f_U^t \cdot \sum_{k=1}^n \frac{\partial w_{UN}^t(i,k)}{\partial \sigma_d} \right) \end{aligned} \quad (3.14)$$

where $\frac{\partial w_{UU}^t(i,\cdot)}{\partial \sigma_d}$ and $p_{UU}^t(i,\cdot)$ are interpreted as row vectors covering all proper indices for the "$\cdot$", restricted by P_{UU}^t. Here we have lifted out $\sum_{k=1}^n \frac{\partial w_{UN}^t(i,k)}{\partial \sigma_d}$ as common factor by making use of the fact that this term in the definition of $\frac{\partial P_{UU}^t(i,j)}{\partial \sigma_d}$ is independent of *j*. Note P_{UL}^t also has such property which will be utilized as well:

$$\begin{aligned} \frac{\partial P_{UL}^t(i,\cdot)}{\partial \sigma_d} f_L^t &= \frac{1}{\sum_{k=1}^n w_{UN}^t(i,k)} \left(\frac{\partial w_{UL}^t(i,j)}{\partial \sigma_d} - p_{UL}^t(i,j) \sum_{k=1}^n \frac{\partial w_{UN}^t(i,k)}{\partial \sigma_d} \right) f_L^t \\ &= \frac{1}{\sum_{k=1}^n w_{UN}^t(i,k)} \left(\frac{\partial w_{UL}^t(i,\cdot)}{\partial \sigma_d} f_L^t - p_{UL}^t(i,\cdot) f_L^t \cdot \sum_{k=1}^n \frac{\partial w_{UN}^t(i,k)}{\partial \sigma_d} \right). \end{aligned} \quad (3.15)$$

Now we look at the computational cost of each term in (3.14).

1. *W* and *P*. It costs $O\left(\tilde{m}n^2\right)$ to calculate *W* and *P*.
2. $\sum_{k=1}^n w_{UN}^t(i,k)$ costs $O(n)$ for a given *i*, $O\left(nu\right)$ to cover all *i*. But it costs only $O\left(n^2\right)$ to cover all *t*, because for different *t*, the 2^{nd} row up to the $(u+1)^{th}$ rows are unchanged. Formally, for all $i = 2...u+1$, $j = 2...u+1$, and any $t_1 \neq t_2$:

$$w_{UU}^{t_1}(i,j) = w_{UU}^{t_2}(i,j) \quad \text{and} \quad \sum_{k=1}^{u+1} w_{UU}^{t_1}(i,k) + \sum_{k=1}^{l-1} w_{UL}^{t_1}(i,k) = \sum_{k=1}^{u+1} w_{UU}^{t_2}(i,k) + \sum_{k=1}^{l-1} w_{UL}^{t_2}(i,k).$$

3. $\sum_{k=1}^{n} \frac{\partial w_{UN}^{t}(i,k)}{\partial \sigma_d}$ costs $O(\tilde{m}n)$ for a given (i, t) (with all the *nonzero* features in either x_i or x_k already covered), and $O(\tilde{m}nu)$ to cover all i. Since only the first row and first column of W_{UU}^{t} change for different t, it costs $O(\tilde{m}n(u+l)) = O(\tilde{m}n^2)$ to cover all t.

4. $p_{UU}^{t}(i,\cdot) f_U^t$ costs $O(u)$ for fixed i and t, so it costs $O(u^2)$ to cover all i. Although the second row up to the $(u + 1)^{th}$ row of p_{UU}^{t} are unchanged for different t, the f_U^t changes completely with t. So it still costs $O(lu^2)$ to cover t.

5. $\frac{\partial w_{UU}^{t}(i,\cdot)}{\partial \sigma_d} f_U^t$ costs $O(\tilde{m}u)$, with all the *nonzero* features in x_i already covered. So it costs $O(\tilde{m}u^2)$ to cover all i and $O(l\tilde{m}u^2)$ to cover all t. This will be too expensive. Fortunately, with t changing, $\frac{\partial w_{UU}^{t}(i,\cdot)}{\partial \sigma_d}$ is basically constant: only the first row and first column change with t. So we just need to record the f_U^t for all $t \in L$ and then traverse all items in $\frac{\partial w_{UU}^{t}(i,\cdot)}{\partial \sigma_d}$ only once (in contrast to l times). In this way, the cost to cover all i and t is $O(\tilde{m}nu)$.

We summarize the costs below and underline the bottleneck computational terms.

$$\frac{\partial P_{UU}^{t}(i,\cdot)}{\partial \sigma_d} f_U^t = \frac{1}{\sum_{k=1}^{n} w_{UN}^{t}(i,k)} \left(\frac{\partial w_{UU}^{t}(i,\cdot)}{\partial \sigma_d} f_U^t - p_{UU}^{t}(i,\cdot) f_U^t \cdot \sum_{k=1}^{n} \frac{\partial w_{UN}^{t}(i,k)}{\partial \sigma_d} \right) \quad (3.14)$$

	with i, t fixed	to cover all i	to cover all t
W, P	$O(\tilde{m}n^2)$	$O(\tilde{m}n^2)$	$\underline{O(\tilde{m}n^2)}$

	with i, t fixed	to cover all i	to cover all t
$\sum_{k=1}^{n} w_{UN}^{t}(i,k)$	$O(n)$	$O(nu)$	$O(n^2)$
$\sum_{k=1}^{n} \frac{\partial w_{UN}^{t}(i,k)}{\partial \sigma_d}$	$O(\tilde{m}n)$ (with all m features, indexed by d, covered)	$O(\tilde{m}nu)$	$\underline{O(\tilde{m}n^2)}$
$p_{UU}^{t}(i,\cdot) f_U^t$	$O(u)$	$O(u^2)$	$\underline{O(lu^2)}$
$\frac{\partial w_{UU}^{t}(i,\cdot)}{\partial \sigma_d} f_U^t$	$O(\tilde{m}u)$ (with all m features, indexed by d, covered)	$O(\tilde{m}u^2)$	$O(\tilde{m}nu)$

Table 3.1 Computational cost for term 1

Similarly,

$$\frac{\partial P_{UL}^{t}(i,\cdot)}{\partial \sigma_d} f_L^t = \frac{1}{\sum_{k=1}^{n} w_{UN}^{t}(i,k)} \left(\frac{\partial w_{UL}^{t}(i,j)}{\partial \sigma_d} - p_{UL}^{t}(i,j) \sum_{k=1}^{n} \frac{\partial w_{UN}^{t}(i,k)}{\partial \sigma_d} \right) f_L^t$$

$$= \frac{1}{\sum_{k=1}^{n} w_{UN}^{t}(i,k)} \left(\frac{\partial w_{UL}^{t}(i,\cdot)}{\partial \sigma_d} f_L^t - p_{UL}^{t}(i,\cdot) f_L^t \cdot \sum_{k=1}^{n} \frac{\partial w_{UN}^{t}(i,k)}{\partial \sigma_d} \right). \qquad (3.15)$$

$\sum_{k=1}^{n} w_{UN}^{t}(i,k)$ costs $O(nu)$ to cover all i and $O(n^2)$ to cover all t. $\sum_{k=1}^{n} \frac{\partial w_{UN}^{t}(i,k)}{\partial \sigma_d}$ is exactly the same as in calculating (3.14). $p_{UL}^{t}(i,\cdot) f_L^t$ costs $O(l)$, so $O(lu)$ to cover all i and $O(ln)$ to cover all t. $\frac{\partial w_{UL}^{t}(i,\cdot)}{\partial \sigma_d} f_L^t$ costs $O(l\tilde{m})$, so $O(l\tilde{m}u)$ to cover all i and $O(l^2\tilde{m}u)$ to cover all t. Similar to calculating (3.14), we make use of the fact that different t only results in different first row and two different columns for w_{UL}^{t}. In this way, the cost to cover all t is reduced to $O(l\tilde{m}n)$.

In sum, the costs are:

$$\frac{\partial P_{UL}^t(i,\cdot)}{\partial \sigma_d} f_L^t = \frac{1}{\sum_{k=1}^n w_{UN}^t(i,k)} \left(\frac{\partial w_{UL}^t(i,\cdot)}{\partial \sigma_d} f_L^t - p_{UL}^t(i,\cdot) f_L^t \cdot \sum_{k=1}^n \frac{\partial w_{UN}^t(i,k)}{\partial \sigma_d} \right) \tag{3.15}$$

	with i, t fixed	to cover all i	to cover all t
W, P	$O(\tilde{m}n^2)$	$O(\tilde{m}n^2)$	$\underline{O(\tilde{m}n^2)}$
$\sum_{k=1}^n w_{UN}^t(i,k)$	$O(n)$	$O(nu)$	$O(n^2)$
$\sum_{k=1}^n \frac{\partial w_{UN}^t(i,k)}{\partial \sigma_d}$	$O(\tilde{m}n)$ (with all m features, indexed by d, covered)	$O(\tilde{m}nu)$	$\underline{O(\tilde{m}n^2)}$
$p_{UL}^t(i,\cdot) f_L^t$	$O(l)$	$O(lu)$	$O(ln)$
$\frac{\partial w_{UL}^t(i,\cdot)}{\partial \sigma_d} f_L^t$	$O(l\tilde{m})$ (with all F_m features already covered)	$O(l\tilde{m}u)$	$O(l\tilde{m}n)$

Table 3.2 Computational cost for term 2

So the overall computational cost is

$$O\left(\tilde{m}n^2 + lnu + u^{2.37}\right). \tag{3.16}$$

This is already close to the lower bound, due to the inevitable cost of calculating W and P.

The memory cost is not the bottleneck and it depends on implementation. The lower bound is $O(n^2)$, just to store W and P. In our implementation, we only used a constant number of $n \times m$ matrices, for each index (i, d) of $\sum_{k=1}^n \frac{\partial w_{UN}^t(i,k)}{\partial \sigma_d}$.

3.4 A mathematical clarification of the algorithm

The above analysis is rather intuitive and complicated. We wish to give a neat mathematical form which clearly shows the computational cost. Note this mathematical formulation will not further reduce the cost, because the algorithm's complexity has already reached its lower bound. However, the artifice we adopt in the following math equations will also be applied in the next chapter to calculate the gradient of the regularizers.

Combining (3.11), (3.14) and (3.15), we find that what we really need to compute is:

$$Q = \sum_{t=1}^{l} c_t s^T \left(I - P_{UU}^t\right)^{-1} \left(\frac{\partial P_{UU}^t}{\partial \sigma_d} f_U^t + \frac{\partial P_{UL}^t}{\partial \sigma_d} f_L^t \right)$$

where $$\frac{\partial P_{UU}^t(i,\cdot)}{\partial \sigma_d} f_U^t = \frac{1}{\sum_{k=1}^{n} w_{UN}^t(i,k)} \left(\frac{\partial w_{UU}^t(i,\cdot)}{\partial \sigma_d} f_U^t - p_{UU}^t(i,\cdot) f_U^t \cdot \sum_{k=1}^{n} \frac{\partial w_{UN}^t(i,k)}{\partial \sigma_d} \right)$$

$$\frac{\partial P_{UL}^t(i,\cdot)}{\partial \sigma_d} f_L^t = \frac{1}{\sum_{k=1}^{n} w_{UN}^t(i,k)} \left(\frac{\partial w_{UL}^t(i,\cdot)}{\partial \sigma_d} f_L^t - p_{UL}^t(i,\cdot) f_L^t \cdot \sum_{k=1}^{n} \frac{\partial w_{UN}^t(i,k)}{\partial \sigma_d} \right).$$

Let $\beta^t \triangleq c_t \left(s^T \left(I - P_{UU}^t\right)^{-1} \right)^T$. Expanding the equations above, we have

$$\begin{aligned} Q = \sum_{t=1}^{l} \sum_{i=1}^{u+1} \frac{\beta_i^t}{\sum_{k=1}^{n} w_{UN}^t(i,k)} \Bigg(& \sum_{j=1}^{u+1} \frac{\partial w_{UU}^t(i,j)}{\partial \sigma_d} f_{U,j}^t + \sum_{j=1}^{l-1} \frac{\partial w_{UL}^t(i,j)}{\partial \sigma_d} f_{L,j}^t \\ & - \sum_{k=1}^{n} \frac{\partial w_{UN}^t(i,k)}{\partial \sigma_d} \left(\sum_{j=1}^{u+1} p_{UU}^t(i,j) f_{U,j}^t + \sum_{j=1}^{l-1} p_{UL}^t(i,j) f_{L,j}^t \right) \Bigg) \end{aligned} \tag{3.17}$$

$$\triangleq \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_{ij} \frac{\partial w_{ij}}{\partial \sigma_d}.$$

The crucial idea is to pre-compute α_{ij}, which is independent of the feature *d*. Then we perform the following operations at computational cost $O\left(\tilde{m}n^2\right)$:

For $i, j = 1 \ldots n$

For all feature *d* which is nonzero in either x_i or x_j

$$g_d = g_d + \alpha_{ij} \frac{\partial w_{ij}}{\partial \sigma_d} \tag{3.18}$$

End

End

Figure 3.3 Pseudo-code for the framework of the efficient implementation

All terms in $\frac{\partial w_{ij}}{\partial \sigma_d}$ which are independent of feature index *d* can be absorbed by α_{ij}. For example, using exponential similarity (3.6): $\frac{\partial w_{ij}}{\partial \sigma_d} = 2 w_{ij} \left(x_{id} - x_{jd} \right)^2 \Big/ \sigma_d^3$, the w_{ij} can be absorbed into α_{ij}. Also, in (3.18) we do not need to divide by σ_d^3 for each *i*, *j*. We can do that after the whole loop of (*i*, *j*, *d*) is finished, and then divide g_d by σ_d^3 for one time only.

Now the only remaining problem is how to pre-calculate α_{ij}. After careful observation and comparison (details omitted), we derive the following result. First of all, $\alpha_{ii} = 0$ for all $i = 1, ..., n$, because w_{ii} are clamped to 0. Even if we consider $\varepsilon \neq 0$ in (3.4), we still have $\frac{\partial p_{ii}}{\partial \sigma_d} = 0$ as $p_{ii} \equiv \frac{\varepsilon}{n}$. So we only consider α_{ij} ($i \neq j$) below. Letting

$$bf_{ij} \triangleq \sum_{t=1}^{l} \beta_{i-l+1}^t f_{U,j-l+1}^t \quad \text{for} \quad i, j \in [l+1, n], \qquad pbf_i \triangleq \sum_{k=l+1}^{n} p_{ik} bf_{ik} \quad \text{for} \quad i \in [l+1, n],$$

$$pf_i^L \triangleq \sum_{k=1}^{l} p_{ik} f_k \quad \text{for} \quad i \in [1, n], \qquad pf_i^U \triangleq \sum_{k=l+1}^{n} p_{ik} f_{U,k-l+1}^i \quad \text{for} \quad i \in [1, l],$$

$$bpf_i \triangleq \sum_{t=1}^{l} \beta_{i-l+1}^t p_{it} \left(f_t - f_{U,1}^t \right) \quad \text{for} \quad i \in [l+1, n], \qquad sb_i \triangleq \sum_{t=1}^{l} \beta_{i-l+1}^t \quad \text{for} \quad i \in [l+1, n],$$

$$sw_i \triangleq \sum_{k=1}^{n} w_{ik} \quad \text{for} \quad i \in [1, n],$$

we have:

1. If $i > l$ and $j > l$, then

$$\alpha_{ij} = \sum_{t=1}^{l} \frac{\beta_{i-l+1}^{t}}{\sum_{k=1}^{n} w_{ik}} \left(f_{U,j-l+1}^{t} - \sum_{k=1}^{u+1} p_{UU}^{t}\left(i-l+1,k\right) f_{U,k}^{t} - \sum_{k=1}^{l-1} p_{UL}^{t}\left(i-l+1,k\right) f_{L,k}^{t} \right)$$

$$= \frac{1}{\sum_{k=1}^{n} w_{ik}} \sum_{t=1}^{l} \beta_{i-l+1}^{t} \left(f_{U,j-l+1}^{t} - \sum_{k=l+1}^{n} p_{ik} f_{U,k-l+1}^{t} - p_{it} f_{U,1}^{t} - \sum_{k=1:k\neq t}^{l} p_{ik} f_{k} \right)$$

$$= \frac{1}{sw_i} \left(\sum_{t=1}^{l} \beta_{i-l+1}^{t} f_{U,j-l+1}^{t} - \sum_{t=1}^{l} \beta_{i-l+1}^{t} \sum_{k=l+1}^{n} p_{ik} f_{U,k-l+1}^{t} \right.$$

$$\left. - \sum_{t=1}^{l} \beta_{i-l+1}^{t} p_{it} f_{U,1}^{t} - \sum_{t=1}^{l} \beta_{i-l+1}^{t} \sum_{k=1}^{l} p_{ik} f_{k} + \sum_{t=1}^{l} \beta_{i-l+1}^{t} p_{it} f_{t} \right)$$

$$= \frac{1}{sw_i} \left(\sum_{t=1}^{l} \beta_{i-l+1}^{t} f_{U,j-l+1}^{t} - \sum_{k=l+1}^{n} p_{ik} \sum_{t=1}^{l} \beta_{i-l+1}^{t} f_{U,k-l+1}^{t} + \sum_{t=1}^{l} \beta_{i-l+1}^{t} p_{it} \left(f_{t} - f_{U,1}^{t} \right) - \sum_{t=1}^{l} \beta_{i-l+1}^{t} \sum_{k=1}^{l} p_{ik} f_{k} \right)$$

$$= \frac{1}{sw_i} \left(bf_{ij} - \sum_{k=l+1}^{n} p_{ik} bf_{ik} + bpf_{i} - sb_{i} \cdot pf_{i}^{L} \right)$$

$$= \frac{1}{sw_i} \left(bf_{ij} - pbf_{i} + bpf_{i} - sb_{i} \cdot pf_{i}^{L} \right).$$

2. If $i > l$ and $j \le l$, then

$$\alpha_{ij} = \frac{1}{\sum_{k=1}^{n} w_{ik}} \sum_{t=1}^{l} \beta_{i-l+1}^{t} \left(f_{U,1}^{t} \delta\left(t=j\right) + f_{j} \delta\left(t\neq j\right) - \sum_{k=l+1}^{n} p_{ik} f_{U,k-l+1}^{t} - p_{it} f_{U,1}^{t} - \sum_{k=1:k\neq t}^{l} p_{ik} f_{k} \right)$$

$$= \frac{1}{sw_i} \left(\beta_{i-l+1}^{t} f_{U,1}^{j} + \left(sb_{i} - \beta_{i-l+1}^{j}\right) f_{j} - \sum_{t=1}^{l} \beta_{i-l+1}^{t} \sum_{k=l+1}^{n} p_{ik} f_{U,k-l+1}^{t} - \sum_{t=1}^{l} \beta_{i-l+1}^{t} p_{it} f_{U,1}^{t} \right.$$

$$\left. - \sum_{t=1}^{l} \beta_{i-l+1}^{t} \sum_{k=1}^{l} p_{ik} f_{k} + \sum_{t=1}^{l} \beta_{i-l+1}^{t} p_{it} f_{t} \right)$$

$$= \frac{1}{sw_i} \left(sb_{i} f_{j} + \beta_{i-l+1}^{j} \left(f_{U,1}^{j} - f_{j} \right) - pbf_{i} + bpf_{i} - sb_{i} \cdot pf_{i}^{L} \right).$$

where $\delta\left(condition\right)$ is 1 when the *condition* is met and is 0 otherwise.

3. If $i \le l$ and $j > l$, then only when $t = i$ will (3.17) make any contribution to α_{ij}.

$$\alpha_{ij} = \frac{\beta_{1}^{i}}{\sum_{k=1}^{n} w_{ik}} \left(f_{U,j-l+1}^{i} - \sum_{k=1}^{u+1} p_{UU}^{i}\left(1,k\right) f_{U,k}^{i} - \sum_{k=1}^{l-1} p_{UL}^{i}\left(1,k\right) f_{L,k}^{i} \right)$$

$$= \frac{\beta_{1}^{i}}{\sum_{k=1}^{n} w_{ik}} \left(f_{U,j-l+1}^{i} - p_{ii} f_{U,1}^{i} - \sum_{k=l+1}^{n} p_{ik} f_{U,k-l+1}^{i} - \sum_{k=1:k\neq i}^{l} p_{ik} f_{k} \right)$$

$$= \frac{\beta_1^i}{\sum_{k=1}^{n} w_{ik}} \left(f_{U,j-l+1}^i - \sum_{k=l+1}^{n} p_{ik} f_{U,k-l+1}^i - \sum_{k=1}^{l} p_{ik} f_k \right) \quad \text{(note } p_{ii} = 0\text{)}$$

$$= \frac{\beta_1^i}{sw_i} \left(f_{U,j-l+1}^i - pf_i^U - pf_i^L \right).$$

4. If $i \le l$ and $j \le l$, then

$$\alpha_{ij} = \frac{\beta_1^i}{\sum_{k=1}^{n} w_{ik}} \left(f_{U,1}^i \delta(i=j) + f_j \delta(i \ne j) - \sum_{k=l+1}^{n} p_{ik} f_{U,k-l+1}^i - \sum_{k=1}^{l} p_{ik} f_k \right)$$

$$= \frac{\beta_1^i}{sw_i} \left(f_j - pf_i^U - pf_i^L \right) \text{ (for } i \ne j \text{)}.$$

All bf_{ij}, pbf_i, pf_i^L, pf_i^U, bpf_i, sb_i, sw_i can be computed within $O(lu^2)$ time, so all these α_{ij} can be pre-computed in $O(lu^2)$ time, with memory cost $O(n^2)$. The overall computational cost is $O(lnu + n^2\tilde{m} + u^{2.37})$, including matrix inversion at $O(u^{2.37})$, f_U^t at $O(lnu)$, α_{ij} at $O(lu^2)$, and (3.18) at $O(\tilde{m}n^2)$. This is our final result and it is the same as the conclusion in (3.16).

As a final note, we can apply this mathematical artifice to the implementation of entropy minimization in section 3.1.2. Using (3.3),

$$H = \frac{-1}{u} \sum_{i=l+1}^{n} \left(f_i \log f_i + (1-f_i) \log(1-f_i) \right).$$

$$\frac{\partial H}{\partial \sigma_d} = \frac{1}{u} \sum_{i=l+1}^{n} \log\left(\frac{1-f_i}{f_i} \right) \frac{\partial f_i}{\partial \sigma_d} = s^T \frac{\partial f_U}{\partial \sigma_d},$$

where $s \triangleq \frac{1}{u} \left(\log\left(\frac{1-f_{l+1}}{f_{l+1}} \right), ..., \log\left(\frac{1-f_n}{f_n} \right) \right)^T$.

$$\frac{\partial f_u}{\partial \sigma_d} = (I - P_{UU})^{-1} \left(\frac{\partial P_{UU}}{\partial \sigma_d} f_U + \frac{\partial P_{UL}}{\partial \sigma_d} f_L \right). \quad \text{Let } r = \left(I - P_{UU}^T \right)^{-1} s \text{, then}$$

$$\frac{\partial H}{\partial \sigma_d} = \sum_{i,j=l+1}^{n} r_{i-l} \frac{\partial P_{UU}(i,j)}{\partial \sigma_d} f_j + \sum_{i=l+1}^{n} r_{i-l} \sum_{j=1}^{l} \frac{\partial P_{UL}(i,j)}{\partial \sigma_d} f_j$$

$$= \sum_{i=l+1}^{n} r_{i-l} \sum_{j=l+1}^{n} f_j \frac{1}{\sum_{k=1}^{n} w_{ik}} \left(\frac{\partial w_{ij}}{\partial \sigma_d} - p_{ij} \sum_{k=1}^{n} \frac{\partial w_{ik}}{\partial \sigma_d} \right) + \sum_{i=l+1}^{n} r_{i-l} \sum_{j=1}^{l} f_j \frac{1}{\sum_{k=1}^{n} w_{ik}} \left(\frac{\partial w_{ij}}{\partial \sigma_d} - p_{ij} \sum_{k=1}^{n} \frac{\partial w_{ik}}{\partial \sigma_d} \right)$$

$$= \sum_{i=l+1}^{n} \frac{r_{i-l}}{\sum_{k=1}^{n} w_{ik}} \sum_{j=1}^{n} f_j \left(\frac{\partial w_{ij}}{\partial \sigma_d} - p_{ij} \sum_{k=1}^{n} \frac{\partial w_{ik}}{\partial \sigma_d} \right)$$

$$= \sum_{i=l+1}^{n} \sum_{j=1}^{n} \frac{\partial w_{ij}}{\partial \sigma_d} \cdot \frac{r_{i-l}}{\sum_{k=1}^{n} w_{ik}} \left(f_j - \sum_{k=1}^{n} p_{ik} f_k \right)$$

So the term to be multiplied with $\frac{\partial w_{ij}}{\partial \sigma_d}$ is $\frac{r_{i-l}}{\sum_{k=1}^{n} w_{ik}} \left(f_j - \sum_{k=1}^{n} p_{ik} f_k \right)$ for $i \in [l+1, n]$, $j \in [1, n]$,

which can be pre-computed efficiently, and easily embedded into the framework in Figure 3.3.

3.5 Utilizing parallel processing

A simple way to apply parallel processing is through dividing features. Since the expensive calculations (underlined in Table 3.1 and Table 3.2, and the inner loop in Figure 3.3) involve $\tilde{m}$, it improves performance to distribute the partial derivatives to different CPUs. The parallelism for calculating *W* and *P* is needed only for calculating the exponent in (2.1). Each processor calculates the contribution of their features to the exponent of edge weight between x_i and x_j for all i, j. Finally the master processor assembles those $n(n+1)/2$-sized (upper triangle) tables to calculate *W*. Using parallelism for calculating the partial derivatives according to Figure 3.3 is similar. In all, parallel processing can reduce complexity to

$O(n^2\tilde{m}/r)$, where *r* is the number of processors available.

Furthermore, how to divide the features into several sets is also a problem, if we want the load on different processors to be balanced. Though this is an NP-hard problem, we only need an approximate solution. Suppose we have a list of length *m*, each element recording $a_d = \sum_i \sum_j \text{active}(x_i, x_j, d)$, where $\text{active}(x_i, x_j, d)$ is 0 if neither x_i nor x_j is nonzero on feature *d*. Otherwise, it is set to 1. So the problem is: given *n* examples, *m* features, and *r* processors, find a partition of $C = \{1, 2, ..., m\}$ into A_1, …, A_r ($\bigcup_{k=1}^{r} A_k = C$, $A_i \bigcap A_j = \phi$ for $\forall i \neq j$), such that $\max_{k=1...r} load(k)$ is *minimized* where $load(k) = \sum_{d \in A_k} a_d = \sum_{d \in A_k} \sum_i \sum_j active(x_i, x_j, d)$. For simplicity, we only need to consider unlabelled x_i and x_j, because the leave-one-out loop treats only one labelled data as unlabelled. Therefore, this sub-optimal partitioning needs to be computed only once given the set of labelled and unlabelled examples.

Chapter 4 Regularization in Learning Graphs

We intentionally ignored the problem of overfitting in the previous chapter. When the number of labelled points is small and there are a large number of features, we can expect that by tuning the bandwidth of all these features, one will get almost all possible graphs he wants. In the following, we first give two simple overfitting examples. Then we give a short review of existing related algorithms. Finally, we propose a few ways to regularize the graph tailored for our leave-one-out learning framework.

4.1 Motivation of regularization

We use two examples to illustrate that without proper regularization, a degenerative graph can be learned by LOOHL with very poor generalization performance.

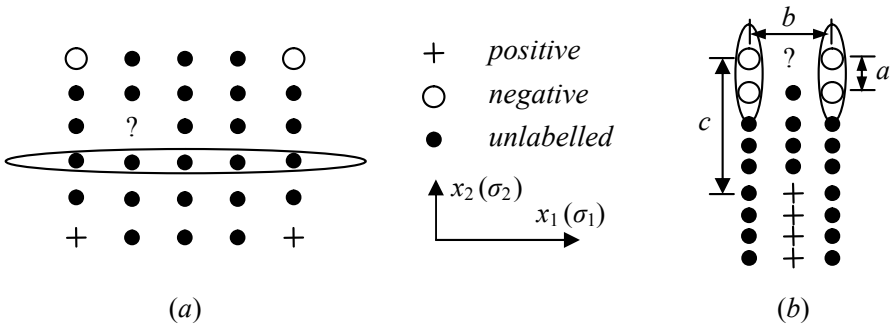


Figure 4.1 Examples of degenerative leave-one-out hyperparameter labelling

The plus signs, hollow circles and solid circles stand for positive, negative and unlabelled data points, respectively. We want to label the point denoted by the question mark. The horizontal and vertical directions are called dimension x_1 and x_2 respectively. Recall the random walk interpretation. Learning the bandwidth for the two dimensions is equivalent

to learning the tendency to walk horizontally or vertically, if equidistance. Suppose there are three points $W(a, b)$, $X(a+1, b)$, $Y(a, b+1)$. The particle is now at W. Then if bandwidth σ_1 is larger than σ_2, it means that a point is more likely to walk horizontally (to X) than vertically (to Y). When $\sigma_1 \to +\infty$, points with the same x_2 co-ordinate are effectively collapsing into one point.

Now in example Figure 4.1 (a), the points of the same classes all have the same x_2 coordinate. For each labelled point, there is another labelled point from the opposite class which has the same x_1 coordinate. So the leave-one-out hyperparameter learning will push σ_2 to positive infinity, i.e., all points can transfer only horizontally. Therefore the graph will be effectively divided into 6 groups, each with the same x_2 coordinate. We showed one group in Figure 4.1 (a). So this causes a problem. First the desired gradual change of labelling from positive to negative along dimension x_2 cannot appear. And as the point at question mark can only transfer horizontally, it cannot hit any labelled point and cannot be classified. In practice, any small numerical perturbation will cause it to be classified to an unforeseeable class.

In Figure 4.1 (b), we assume $a \ll b \ll c$. Although the negative points will encourage both horizontal and vertical walk, horizontal walk will make the leave-one-out soft label error large on positive points. So the learned σ_2 will be far larger than σ_1, i.e., strongly encourage walking vertically. As a result, the point at the question mark will be labelled as positive, although by nearest neighbour intuition, it should be labelled as negative. Besides, we notice that the four negative points will also be partitioned into two groups as shown in

the figure.

How to avoid these undesired behaviours will motivate different regularizers. We mention some ideas below and we will develop them in the following sections. The first remedy is through directly controlling σ, while the rest control σ indirectly by considering the property of the resulting graphs.

1. Add square loss $\sum_i \sigma_i^2$. However, the bandwidths obviously do not have a zero mean Gaussian prior, so we need to modify it into $\sum_i (\sigma_i - \hat{\sigma}_i)^2$, where $\hat{\sigma}_i$ stands for the prior mean, which can be assigned by cross validation under the original HEM algorithm.
2. Modify transfer probability from p_{ij} to $\varepsilon / n + (1-\varepsilon) p_{ij}$. This method will help connect all nodes together and was used in (Zhu et al., 2003a).
3. Maximize the entropy of row transfer probability on unlabelled data points. Note this entropy is not the posterior soft label entropy as used in section 3.1.2. It will prefer more balanced (uniform) transfer probability distribution to other nodes, and it connects each node to as many nodes as possible so that large clusters are formed. This will encourage the unlabelled data in Figure 4.1 (a) and (b) to connect to more nodes in vertical direction and horizontal direction respectively.
4. Minimize $E[t]$, the expected number of steps needed for a given unlabelled data to hit a labelled point. In Figure 4.1 (a), $E[t]$ is approaching infinity for all unlabelled nodes not in the top or lowest x_2 coordinate. In Figure 4.1 (b), $E[t]$ for nodes near the negative data can be reduced if the horizontal transfer is more encouraged.

5. Other means to encourage larger clusters, i.e., smaller number of clusters. We will use spectral methods in section 4.3. In both examples in Figure 4.1, we can see the existence of small and separate clusters may harm the performance.

In section 4.2, we give a brief survey of the literature related to regularizing the complexity of a graph. Due to the scarcity of existing work, such a relationship may seem far-fetched, though one regularizer will be later motivated from clustering. After that we propose several ad hoc regularizers based on above intuitively desirable properties of a graph.

4.2 How to regularize? A brief survey of related literature

To the best of our knowledge, there has been hardly any existing work that explicitly regularizes the learning of the graphs. We call this task *graph learning regularization* instead of *graph regularization* in order to avoid confusion with the graph Laplacian used for regularizing the labelling (see section 1.5.1.2). Despite the lack of directly related literature, we try to relate the graph learning problem to kernel learning and the desired properties in spectral clustering. The latter view motivated one graph learning regularizer which will be presented in section 4.3.

4.2.1 Regularization from a kernel view

The first source of graph learning regularizer we have found is through the graph's association with kernels. Firstly, such an association is straightforward as described in section 1.5.1.2. Secondly, the theory of kernels is relatively mature and it offers a lot of conven-

ience to consider in the reproducing kernel Hilbert space (RKHS). Unfortunately, further investigation reveals a disappointing fact that regularizers for learning kernels are scarce by itself, though the theory of using kernels themselves to do regularization is abundant. The most promising method is kernel learning, which is very popular recently. See for example, (Lanckriet et al., 2004; Weinberger et al., 2004; Zhu et al., 2004; Kwok & Tsang, 2003; Crammer et al., 2002; Cristianini et al., 2001). However, they all implicitly restrict the parametric form of the kernel, without explicit terms that penalize the complexity of the kernels. This may cause problems in general. For example in (Lanckriet et al., 2004), learning kernels with only positive semi-definiteness constraints yields consistently inferior empirical performance to learning from a positive combination of several fixed kernels, for which semi-definite programming actually degenerates to quadratic programming. So how to restrict the capacity of the candidate kernel set, or regularize the kernel learning, is still an important and open problem even in the field of kernel learning.

(Ong et al., 2005; Ong & Smola, 2003) proposed an interesting measure of kernel space capacity called hyperkernels. In addition to the normal cost function terms such as the loss function on labelled data, and the norm of the classification function in the RKHS induced by the kernel K, it also includes a term which depicts the *norm of that kernel K* in the space of candidate kernels, which is also a RKHS induced by a so-called hyperkernel. In that way, there will be an additional penalty to choosing a complex kernel. This is exactly what we want in our graph learning algorithms: how to measure and penalize the complexity of a graph.

However, there are still three obstacles to using this method. Firstly, the optimization technique, semi-definite programming, is still not efficient enough for large scale datasets. Secondly, in graph algorithms, we parameterize the graph Laplacian, which is the pseudo-inverse of the associated kernel (see Figure 2.3). So it is indirect and inconvenient to formulate, implement, and optimize (e.g., calculating gradient) the regularizer. Thirdly, the proper forms of hyperkernel still leave much to be further studied.

4.2.2 Regularization from a spectral clustering view

We introduced the idea of spectral graphical clustering in section 1.5.1.3. We now develop it in detail. From section 1.5.1.3, we know the (normalized) graph Laplacian contains useful information of the graph's clustering structure. The eigenvalues of the Laplacian and normalized Laplacian play an important role. So if we want to design a regularizer relating to the graph clustering structure, the eigenvalues will be a good choice of study.

Suppose the symmetric affinity matrix is W. The Laplacian is defined as $\Delta = D - W$, where $D = diag(d_1,...,d_n)$ with $d_i = \sum_{j=1}^{n} w_{ij} > 0$. The normalized Laplacian is $\tilde{\Delta} = D^{-1/2}(D-W)D^{-1/2}$. By property 2 in section 2.6.1, the eigenvalues of $\tilde{\Delta}$ are equal to $\hat{P} = D^{-1}(D-W)$. So although the elements in $\tilde{\Delta}$ are difficult to handle (square roots), we only need to investigate $\hat{P}$, which has the same eigenvalues as $\tilde{\Delta}$. By property 1 in section 2.6.1, the eigenvalues of $\hat{P}$ lie in $[0, 2]$. In the following, we will only consider $\tilde{P} = \frac{1}{2}D^{-1}(D+W)$ so that the eigenvalues lie in $[0, 1]$. The next question is what property do we expect from the eigenvalues?

The following figures extracted from (Meilă & Shi, 2001) show two instances of eigenvalues' distribution. The upper 20*20 images are the affinity (weight) matrices W and the lower are the eigenvalues of $\tilde{P}$. The *darker* the pixel is, the *smaller* is the weight (ranging in $[0, 1]$). The range of y-axis in the lower two images is $[0, 1]$.

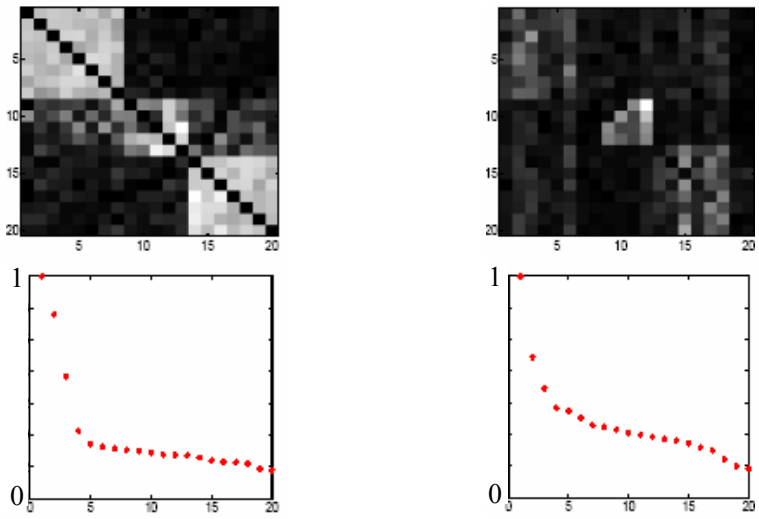


Figure 4.2 Eigenvalue distribution example of two images

$\tilde{P}$ has a trivial eigenvalue 1, with eigenvector $\vec{1}_n$. Theoretically, if there are k mutually disconnected components, then 1 should be an eigenvalue of $\tilde{P}$ with multiplicity k. The average of the rest eigenvalues is $(n/2-k)/(n-k)$, because the trace of $\tilde{P}$ (the sum of all eigenvalues) is $n/2$ if the diagonal of W is straight 0. $(n/2-k)/(n-k)$ approaches 1/2 when n is large, and most of the eigenvalues not equal to 1 lie in $[0, 0.5]$. In the left figure, there are roughly three clusters, and we can see three large eigenvalues which are far away from the rest. In the right figure, some eigenvalues other than the largest three eigenvalues are rather close to them (decreasing slowly), which corresponds to the more ambiguous clustering structure in the original W. So given that our objective is to encourage

the graph to be well clustered (reducing the inter-cluster similarity and increasing the intra-cluster similarity), we wish to have some eigenvalues close to 1 and the rest eigenvalues close to 0. The gap between these two groups of eigenvalues is expected to be as large as possible. This is also in line with the concept of eigengap maximization in matrix perturbation theory (Ng et al., 2001a; Stewart & Sun, 1990). Of course, we wish to restrict the number of 1-valued eigenvalues as well, in order to avoid many small clusters.

In addition, we can also analyse through kernel PCA. $\tilde{\Delta}$ can be viewed as the pseudo-inverse covariance matrix, or the pseudo-inverse of the kernel. An eigenvalue $\lambda \in [0, 2]$ of $\tilde{\Delta}$ is mapped to the eigenvalue $1-\lambda/2$ of $\tilde{P}$. So the above requirements on $\tilde{P}$'s eigenvalues can be translated as: the variance in the non-leading principal directions should be small, or the difference of variance between leading and non-leading principal directions should be large. This is also well known as an ideal property for the principal components in the kernel space for denoising.

4.3 Graph learning regularizer 1: approximate eigengap maximization

Summarizing the ideas in section 4.2.2, the penalty function over eigenvalues may look like Figure 4.3. The shape can be approximated by polynomials like $x^{16}(1-x)$ or other $x^r(1-x)$ for large *r*'s. So the objective function is $\sum_{i=1}^{n} \lambda_i^r (1-\lambda_i) = \sum_{i=1}^{n} \lambda_i^r - \sum_{i=1}^{n} \lambda_i^{r+1}$. As $\sum_{i=1}^{n} \lambda_i^r$ is equal to the trace of $\tilde{P}^r$, it is equivalent to minimizing the trace of $\tilde{P}^r - \tilde{P}^{r+1}$. Note the left tail covers 0.6 or so, because most non-principal eigenvalues average 0.5 and we do not want to penalize over these eigenvalues. Of course, the exponent *r* can be adjusted

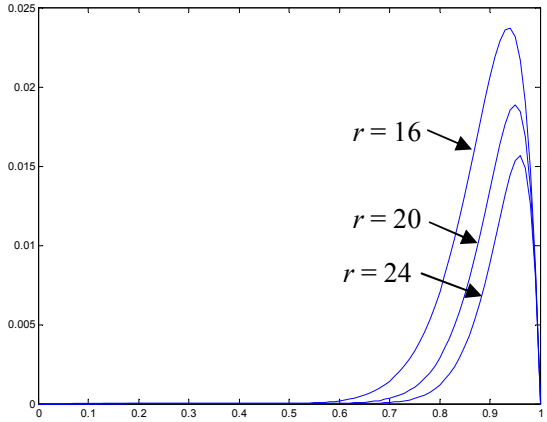


Figure 4.3 Example of penalty function over eigenvalues

so that the rising point can be shifted horizontally (see the three curves in Figure 4.3). The only disadvantage is that the sharp decrease of penalty function near $\lambda = 1$ may not be the best choice. Fortunately, if we start from a graph which is not too disconnected, then most eigenvalues are on the left tail of the curve. Minimizing the penalty by gradient descent will just push them to the left, rather than climbing over the peak and decline in the right tail. For more simplicity, we can just adopt penalty functions like

$$\lambda^r \quad (r \geq 2), \tag{4.1}$$

which is finally used in our experiments.

Calculating $\tilde{P}^r$ only costs $\lceil \log r \rceil$ times of matrix multiplication. Another computational advantage is that the partial derivatives are easy to calculate due to the special property of trace: $\frac{\partial}{\partial X} tr\left(\boldsymbol{X}^r\right) = r\left(\boldsymbol{X}^{r-1}\right)^T$, where $\boldsymbol{X}$ is a $n \times n$ matrix. So $\frac{\partial}{\partial \sigma_d} tr\left(\boldsymbol{X}^r\right) = r \sum_{i,j=1}^{n} x_{ji}^{r-1} \frac{\partial x_{ij}}{\partial \sigma_d}$, where x_{ji}^{r-1} stands for the $(j, i)^{th}$ element of $\boldsymbol{X}^{r-1}$.

Now we present the math formulae for calculating the objective function and its gradient. Let $P = D^{-1}W$, then $tr\left(\tilde{P}^r\right) = tr\left(\left((I+P)/2\right)^r\right)$. Specifically, when $r=2$, $tr\left(\tilde{P}^2\right) = \frac{1}{4}\left(\sum_{i,j=1}^{n} p_{ij}p_{ji} + 2\sum_{i=1}^{n} p_{ii} + n\right)$, which can be calculated in $O\left(n^2\right)$ time. For $r>2$, the Matlab function *mpower* (mlfMpower in Matlab C Math Library) can be used to compute the power by repeated squaring. Recall $\frac{\partial p_{ij}}{\partial \sigma_d} = \left(\frac{\partial w_{ij}}{\partial \sigma_d} - p_{ij}\sum_{k=1}^{n}\frac{\partial w_{ik}}{\partial \sigma_d}\right)\Big/\sum_{k=1}^{n} w_{ik}$, then we have

$$\begin{aligned}
\frac{\partial}{\partial \sigma_d} tr\left(\tilde{P}^r\right) &= r\sum_{i,j=1}^{n} \tilde{p}_{ji}^{r-1}\frac{\partial \tilde{p}_{ij}}{\partial \sigma_d} = \frac{1}{2}r\sum_{i,j=1}^{n} \tilde{p}_{ji}^{r-1}\frac{\partial p_{ij}}{\partial \sigma_d} \\
&= \frac{1}{2}r\sum_{i,j=1}^{n} \tilde{p}_{ji}^{r-1}\left(\frac{\partial w_{ij}}{\partial \sigma_d} - p_{ij}\sum_{k=1}^{n}\frac{\partial w_{ik}}{\partial \sigma_d}\right)\Big/\sum_{k=1}^{n} w_{ik} \\
&= \frac{1}{2}r\sum_{i=1}^{n}\frac{1}{\sum_{k=1}^{n} w_{ik}}\sum_{j=1}^{n}\left(\frac{\partial w_{ij}}{\partial \sigma_d}\tilde{p}_{ji}^{r-1} - p_{ij}\tilde{p}_{ji}^{r-1}\sum_{k=1}^{n}\frac{\partial w_{ik}}{\partial \sigma_d}\right) \\
&= \frac{1}{2}r\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial w_{ij}}{\partial \sigma_d}\frac{1}{\sum_{k=1}^{n} w_{ik}}\left(\tilde{p}_{ji}^{r-1} - \sum_{k=1}^{n} p_{ik}\tilde{p}_{ki}^{r-1}\right)
\end{aligned}$$

So the factor to be multiplied with $\frac{\partial w_{ij}}{\partial \sigma_d}$ is $\frac{1}{2}r\left(\tilde{p}_{ji}^{r-1} - \sum_{k=1}^{n} p_{ik}\tilde{p}_{ki}^{r-1}\right)\Big/\sum_{k=1}^{n} w_{ik}$, which can be pre-computed efficiently. Note here that *i* and *j* vary from 1 to *n*, i.e., the whole graph rather than restricted to unlabelled data. Finally, we just apply the framework in Figure 3.3.

4.4 Graph learning regularizer 2: first-hit time minimization

In this formulation, we consider the random walk interpretation under unsupervised setting. For a particle starting to walk from an unlabelled data point *i*, it will take some steps to hit a

labelled data point, upon which it stops walking. Since the walk is random, we are concerned about the average (expected) number of steps needed to hit a labelled point, denoted as a_i. The objective is to minimize the total average steps $\sum_{i=l+1}^{n} a_i$ defined on unlabelled data only. To proceed, we first need to derive a mathematical form of $\sum_{i=l+1}^{n} a_i$.

We have shown in section 2.3 that for $i \in U = \{l+1,...,n\}$, f_i^{opt} (defined in (2.4)) can be interpreted as the probability that a particle starting from node *i* hits a positively labelled data first. Now we go to more details and define f_i^t as the probability that a particle starting from node *i* hits a labelled data point within *t* steps (including *t* steps). It does not matter whether the labelled data is positive or negative. Then

$$\sum_{i=l+1}^{n} a_i = \sum_{i=l+1}^{n} \sum_{t=1}^{+\infty} t\left(f_i^t - f_i^{t-1}\right) = \sum_{t=1}^{+\infty} t \cdot \sum_{i=l+1}^{n} \left(f_i^t - f_i^{t-1}\right) \tag{4.2}$$

and the objective function to be minimized is:

$$\sum_{i=l+1}^{n} a_i = \sum_{i=l+1}^{n} \sum_{t=1}^{+\infty} t\left(f_i^t - f_i^{t-1}\right). \tag{4.3}$$

We can calculate the f_i^t ($i \in U$) recursively with ease:

$$f_i^t = \begin{cases} \sum_{j=1}^{l} p_{ij} + \sum_{j=l+1}^{n} p_{ij} f_j^{t-1} & t \geq 1 \\ 0 & t = 0 \end{cases} \tag{4.4}$$

where p_{ij} is the transition probability from *i* to *j*. The first question that should be answered is whether the definition of a_i in (4.2) is sound, i.e., does the summation converge? The answer is *yes*, as long as some preconditions are met. Once it is proved to be theoretically well defined, how to compute its value and gradient efficiently? The next sub-sections will deal with the two questions respectively.

4.4.1 Theoretical proof and condition of convergence

***Proposition* 1.**

$\sum_{i=l+1}^{n} a_i$ **converges if for each unlabelled point, there exists an *edge* to a certain labelled data point, i.e.,** $\sum_{j=1}^{l} p_{ij} > 0$ **for all** $i \in l+1,...,n$**.**

Proof. If this precondition is met, then we have

$$M \triangleq \max_{i \in U} \sum_{j=l+1}^{n} p_{ij} < 1 . \tag{4.5}$$

Let $f_M \triangleq \max_{i \in l+1,...,n} f_i^1$, and $\vec{1}_s \triangleq (1,1,...,1)^T \in \mathbb{R}^s$ for any natural number *s*.

$$f_i^t - f_i^{t-1} = \left(\sum_{j=1}^{l} p_{ij} + \sum_{j=l+1}^{n} p_{ij} f_j^{t-1} \right) - \left(\sum_{j=1}^{l} p_{ij} + \sum_{j=l+1}^{n} p_{ij} f_j^{t-2} \right) = \sum_{j=l+1}^{n} p_{ij} \left(f_j^{t-1} - f_j^{t-2} \right)$$

$$= \sum_{j=l+1}^{n} p_{ij} \sum_{k=l+1}^{n} p_{jk} \left(f_k^{t-2} - f_k^{t-3} \right) = \cdots\cdots$$

$$= \sum_{i_1, i_2, \cdots, i_{t-1} \in U} p_{i,i_1} p_{i_1,i_2} \cdots p_{i_{t-2},i_{t-1}} f_{i_{t-1}}^1$$

So $$\sum_{i=l+1}^{n} \left(f_i^t - f_i^{t-1} \right) = \sum_{i_0, i_1, \cdots, i_{t-1} \in l+1,...,n} p_{i_0,i_1} p_{i_1,i_2} \cdots p_{i_{t-2},i_{t-1}} f_{i_{t-1}}^1$$

$$= \sum_{i_0, i_1, \cdots, i_{t-1} \in l+1,...,n} f_{i_0}^1 p_{i_0,i_1} p_{i_1,i_2} \cdots p_{i_{t-2},i_{t-1}}$$

$$= \vec{1}_u^T \cdot \left(P_{UU} \right)^{t-1} \cdot f_U^1 \tag{4.6}$$

where $f_U^1 \triangleq \left(f_{l+1}^1, f_{l+2}^1, \cdots, f_n^1 \right)^T$. Thus to prove that $\sum_{t=1}^{+\infty} t \cdot \sum_{i=l+1}^{n} \left(f_i^t - f_i^{t-1} \right)$ converges, it suffices to show that $\left(P_{UU} \right)^{t-1}$ [1] decays sufficiently faster (e.g., exponentially) than the linear increase rate of *t*. This is guaranteed by the proposition's precondition. By definition of (4.5), each element in P_{UU} is less than or equal to M^1 (here, 1 is exponent, not super-

[1] Note here *t* – 1 is exponent. To avoid confusion of superscript and exponent, we will not write P_{UU}^{t-1}, but always use parenthesis for power in this chapter.

script). Suppose in $\left(P_{UU}\right)^{t-1}$, each element p_{ij}^{t-1} ($t-1$ is superscript) is less than or equal to M^{t-1}. Then the elements in $\left(P_{UU}\right)^{t}$ are $p_{ij}^{t}=\sum_{k=l+1}^{n}p_{ik}p_{kj}^{t-1}\le\sum_{k=l+1}^{n}p_{ik}M^{t-1}=M^{t-1}\sum_{k=l+1}^{n}p_{ik}$ $\le M^{t}$. So by induction, we have proved that all elements in $\left(P_{UU}\right)^{t}$ are smaller or equal to M^{t} for all $t\ge 1$. Continuing (4.6), we have

$$\sum_{i=l+1}^{n}\left(f_i^t-f_i^{t-1}\right)\le f_M U^2 M^{t-1} \tag{4.7}$$

which decays exponentially with respect to t and thus $\sum_{t=1}^{+\infty}t\cdot\sum_{i=l+1}^{n}\left(f_i^t-f_i^{t-1}\right)$ converges. □

The proposition's precondition is met under our RBF weight model, at least theoretically because exp(·) is always positive.

The equation (4.6) has also provided a starting point of calculating $\sum_{i=l+1}^{n}a_i$. Continuing it,

$$\sum_{i=l+1}^{n}a_i=\sum_{t=1}^{+\infty}t\cdot\sum_{i=l+1}^{n}\left(f_i^t-f_i^{t-1}\right)=\sum_{t=1}^{+\infty}t\cdot\vec{1}_U^T\cdot\left(P_{UU}\right)^{t-1}\cdot f_U^1=\vec{1}_U^T\cdot\left(\sum_{t=1}^{+\infty}t\cdot\left(P_{UU}\right)^{t-1}\right)\cdot f_U^1$$

(4.8)

So we only need to calculate $\sum_{t=1}^{+\infty}t\cdot\left(P_{UU}\right)^{t-1}$. For scalar valued x, we have

If $f(x)\triangleq\sum_{t=1}^{+\infty}tx^{t-1}$ ($|x|<1$), then $f(x)=\sum_{t=1}^{+\infty}\frac{d}{dx}x^t=\frac{d}{dx}\sum_{t=1}^{+\infty}x^t=\frac{d}{dx}\left(\frac{x}{1-x}\right)=\left(1-x\right)^{-2}$.

Another proof is by comparing the power series terms' coefficients in $f(x)$ and $(1+x+x^2+x^3+\cdots)^2$, and then apply $\left(1-x\right)^{-1}=1+x+x^2+x^3+\cdots$.

For matrices, since proposition 1 has already guaranteed the convergence of the objective function, it is also true that $\sum_{t=1}^{+\infty}t\cdot\left(P_{UU}\right)^{t-1}=\left(I-P_{UU}\right)^{-2}$. Firstly, for matrix, it is well

known that $(I-A)^{-1}=\sum_{t=0}^{+\infty}A^t$. Then applying the coefficient comparison proof as in the scalar case, we safely and concisely prove the matrix case. Considering $f_U^1=P_{UL}\vec{1}_l$, and $P_{UL}\cdot\vec{1}_l=(I-P_{UU})\vec{1}_u$ (due to $\sum_{j=1}^{n}p_{ij}=1$), our regularizer finally becomes

$$F=\vec{1}_u^T\cdot(I-P_{UU})^{-2}\cdot P_{UL}\cdot\vec{1}_l=\vec{1}_u^T\cdot(I-P_{UU})^{-1}\cdot\vec{1}_u. \tag{4.9}$$

F is pretty concise and easy to calculate, especially after $(I-P_{UU})^{-1}$ has already been calculated for entropy minimization regularizer, or calculated to test the accuracy in each iteration of optimization. Gradient is also simple to calculate.

The following Proposition 2 weakens the precondition of proposition 1 by relieving the requirement of *direct edge* to a labelled node. It only requires a directed *path*. By definition of a_i, this is already the weakest possible requirement to make a_i well defined.

***Proposition* 2.**

$\sum_{i=l+1}^{n}a_i$ converges if and only if for each unlabelled point, there exists a directed *path* to a certain labelled data point. Formally, for all $i\in l+1,...,n$, there exist $l(i)\in 1...l$, $s_i\geq 1$ and unlabelled nodes $t(1)=i,t(2),...,t(s_i)\in l+1,...,n$ such that

$$\left(\prod_{k=1}^{s_i-1}p_{t(k),t(k+1)}\right)\cdot p_{t(s_i),l(i)}>0. \tag{4.10}$$

Before proving this proposition, we first briefly introduce a lemma, which says that the maximum of elements in $(P_{UU})^t$ will not increase with increasing t.

Lemma 1.

Denote the sum of the i^{th} row of $(P_{UU})^t$ as $M_i^t\triangleq\sum_{j=l+1}^{n}p_{ij}^t$, where p_{ij}^t stands for the $(i,j)^{th}$ element of $(P_{UU})^t$. Then M_i^t is monotonically non-increasing with respect to t, i.e.,

$M_i^{t+1} \le M_i^t$ for all $t \ge 1$.

Proof. $M_i^{t+1} = \sum_{j=l+1}^{n} p_{ij}^{t+1} = \sum_{j=l+1}^{n} \sum_{k=l+1}^{n} p_{ik}^t p_{kj} = \sum_{j=l+1}^{n} p_{ij}^t \sum_{k=l+1}^{n} p_{jk} \le \sum_{j=l+1}^{n} p_{ij}^t = M_i^t$. So $M_i^{t+1} \le M_i^t$. □

Now we prove Proposition 2. The condition's necessity is by a_i's definition. We now prove its sufficiency.

Proof. By Lemma 1, to prove that the elements of $(P_{UU})^t$ decay exponentially fast, it is sufficient to find a subsequence of M_i^t (i fixed) such that this subsequence can be shown to decay exponentially fast wrt t. Although there are many (or infinite) choices of $l(i)$ and s_i which satisfy (4.10), we will randomly choose one and refer only to that choice henceforth.

If the labelled nodes are not absorbing boundaries, then the $(i,j)^{th}$ element of P^t will be the probability of walking from node i to node j with exactly t steps. Now in the special case of absorbing boundary, the effective transmission matrix P is $\begin{pmatrix} I_{LL} & 0_{LU} \\ P_{UL} & P_{UU} \end{pmatrix}$. So $P^t = \begin{pmatrix} I_{LL} & 0_{LU} \\ P_{UL}^t & (P_{UU})^t \end{pmatrix}$, where the t in P_{UL}^t is superscript. $(P_{UU})^t$ still represents the probability of walking from unlabelled node i to unlabelled node j with exactly t steps. Consider an unlabelled node $r \in U$. By the precondition of proposition 2, there is a directed path with length s_r from node r to a labelled node $l(r)$. In other words, there is a nonzero element among the $(r, 1)^{th}$, $(r, 2)^{th}$, …, $(r, l)^{th}$ element of P^{s_r}. Therefore the sum of the $(r, l+1)^{th}$, $(r, l+2)^{th}$, …, $(r, n)^{th}$ elements of $(P_{UU})^{s_r}$ (denoted as $q_r \triangleq \sum_{j=l+1}^{n} p_{rj}^{s_r}$) is smaller than 1. Denote $q \triangleq \max_{r \in U} q_r < 1$, $s \triangleq \max_{r \in U} s_r \ge 1$, and $M \triangleq \max_{i,j \in U} p_{ij}$. So all elements in the r^{th} row of

$\left(P_{UU}\right)^{s_r+1} = \left(P_{UU}\right)^{s_r} \cdot P_{UU}$ will be $p_{rj}^{s_r+1} = \sum_{k=l+1}^{n} p_{rk}^{s_r} p_{kj} \le \sum_{k=l+1}^{n} p_{rk}^{s_r} M = Mq_r \le Mq$ for all $r, j \in U$. Applying it recursively, we have $p_{rj}^{ns_r+1} \le Mq^n$ and thus $M_r^{ns_r+1} \le uMq^n$. By Lemma 1, we have for all $t \in \left[ns_r+1, (n+1)s_r+1\right)$, $p_{rj}^t \le M_r^t \le uMq^n$, i.e., $p_{rj}^t \le uMq^{\left[(t-1)/s_r\right]} \le uMq^{\left[(t-1)/s\right]}$. Since $q<1$ and *M*, *u*, *s* are all constant, we have proved that all elements in $\left(P_{UU}\right)^t$ decay exponentially with respect to *t*, and thus $\sum_{i=l+1}^{n} a_i$ converges. □

4.4.2 Efficient computation of function value and gradient

Since there is no leave-one-out computation for this method, the computation cost is relatively mild. We use the same artifice as in section 3.4:

$$F = \vec{1}_u^T \cdot \left(I - P_{UU}\right)^{-1} \cdot \vec{1}_u .$$

$$\frac{\partial F}{\partial \sigma_d} = \vec{1}_U^T \left(I - P_{UU}\right)^{-1} \frac{\partial P_{UU}}{\partial \sigma_d} \left(I - P_{UU}\right)^{-1} \vec{1}_u .$$

Let $\vec{l} = \left(I - P_{UU}^T\right)^{-1} \vec{1}_u$, $\vec{r} = \left(I - P_{UU}\right)^{-1} \vec{1}_u$. Recall $\frac{\partial p_{ij}}{\partial \sigma_d} = \left(\frac{\partial w_{ij}}{\partial \sigma_d} - p_{ij} \sum_{k=1}^{n} \frac{\partial w_{ik}}{\partial \sigma_d} \right) \Big/ \sum_{k=1}^{n} w_{ik}$, then

$$\frac{\partial F}{\partial \sigma_d} = \sum_{i,j=l+1}^{n} \frac{\partial P_{UU}(i,j)}{\partial \sigma_d} l_i r_j = n \sum_{i,j=l+1}^{n} \frac{1}{\sum_{k=1}^{n} w_{ik}} \left(\frac{\partial w_{ij}}{\partial \sigma_d} - p_{ij} \sum_{k=1}^{n} \frac{\partial w_{ik}}{\partial \sigma_d} \right) l_{i-l} r_{j-l}$$

$$= \sum_{i=l+1}^{n} \frac{l_{i-l}}{\sum_{k=1}^{n} w_{ik}} \left(\sum_{j=l+1}^{n} \frac{\partial w_{ij}}{\partial \sigma_d} r_{j-l} - \sum_{j=l+1}^{n} p_{ij} r_{j-l} \cdot \sum_{k=1}^{n} \frac{\partial w_{ik}}{\partial \sigma_d} \right)$$

$$= \sum_{i=l+1}^{n} \sum_{j=l+1}^{n} \frac{\partial w_{ij}}{\partial \sigma_d} \cdot \frac{l_{i-l}}{\sum_{k=1}^{n} w_{ik}} \left(r_{j-l} - \sum_{k=l+1}^{n} p_{ik} r_{k-l} \right) - \sum_{i=l+1}^{n} \sum_{j=1}^{l} \frac{\partial w_{ij}}{\partial \sigma_d} \cdot \frac{l_{i-l}}{\sum_{k=1}^{n} w_{ik}} \sum_{k=l+1}^{n} p_{ik} r_{k-l}$$

So the factor to be multiplied to $\frac{\partial w_{ij}}{\partial \sigma_d}$ is

$$\begin{cases} \dfrac{l_{i-l}}{\sum_{k=1}^{n} w_{ik}} \left(r_{j-l} - \sum_{k=l+1}^{n} p_{ik} r_{k-l} \right) & for\ \ i, j \geq l+1 \\ \dfrac{-l_{i-l}}{\sum_{k=1}^{n} w_{ik}} \sum_{k=l+1}^{n} p_{ik} r_{k-l} & for\ \ i \geq l+1,\ j \leq l \end{cases},$$

which can be pre-computed efficiently. Finally just apply the framework in Figure 3.3.

4.5 Graph learning regularizer 3: row entropy maximization

This regularizer stems from the intuition that we encourage more balanced (uniform) transfer probability distribution from each unlabelled node to other labelled and unlabelled nodes, so that each node will connect to as many nodes as possible and large clusters are formed. Mathematically it maximizes:

$$F = -\sum_{i=l+1}^{n} \sum_{j=1}^{n} p_{ij} \log p_{ij}$$

$$\begin{aligned} \frac{\partial}{\partial \sigma_d} F &= -\sum_{i=l+1}^{n} \sum_{j=1}^{n} \left(1 + \frac{\partial p_{ij}}{\partial \sigma_d} \log p_{ij} \right) \\ &= -\sum_{i=l+1}^{n} \sum_{j=1}^{n} \left(\log p_{ij} \cdot \left(\frac{\partial w_{ij}}{\partial \sigma_d} - p_{ij} \sum_{k=1}^{n} \frac{\partial w_{ik}}{\partial \sigma_d} \right) \Big/ \sum_{k=1}^{n} w_{ik} \right) - nu \\ &= \sum_{i=l+1}^{n} \sum_{j=1}^{n} \frac{\partial w_{ij}}{\partial \sigma_d} \frac{1}{\sum_{k=1}^{n} w_{ik}} \left(\sum_{k=1}^{n} p_{ik} \log p_{ik} - \log p_{ij} \right) - nu \end{aligned}$$

So the factor to be multiplied to $\dfrac{\partial w_{ij}}{\partial \sigma_d}$ is

$$\frac{1}{\sum_{k=1}^{n} w_{ik}} \left(\sum_{k=1}^{n} p_{ik} \log p_{ik} - \log p_{ij} \right) \quad \text{for} \quad i = l+1, ..., n, \quad j = 1...n,$$

which can be pre-computed efficiently. Finally just apply the framework in Figure 3.3.

4.6 Graph learning regularizer 4: electric circuit conductance maximization

Finally, we propose a regularizer based on the electric circuit interpretation in section 2.4. This formulation exploits the circuit interpretation of (Zhu et al., 2003a). Suppose we ground all labelled points. Then we only connect one unlabelled node i ($i \in U$) to +1V and leave other unlabelled nodes unclamped. Now we can calculate the conductance of node i relative to the ground as c_i, where c_i is the current that flows into the circuit through node i. The objective is to maximize the total conductance $\sum_{i=l+1}^{n} c_i$. This will encourage the unlabelled nodes to be connected to labelled nodes locally.

Figure 4.4 illustrates this idea.

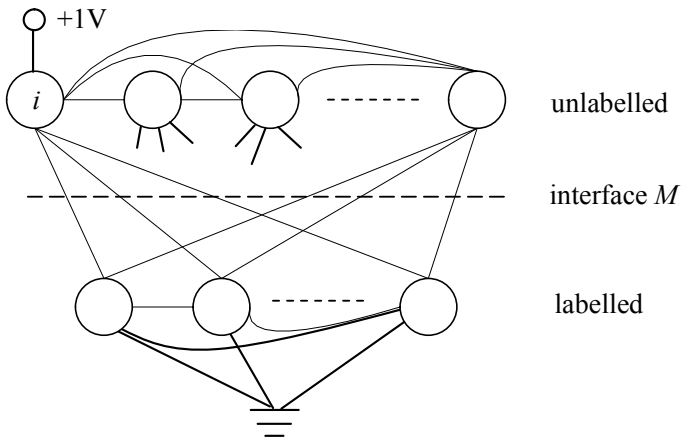


Figure 4.4 Circuit regularizer

Suppose the unlabelled node i is clamped to +1V. We wish to calculate the potential of all other unlabelled nodes. Then the conductance of node i relative to the ground is the sum of currents that flow from the unlabelled nodes to the labelled nodes (across the interface M). For explanation, we introduce some notations. Define:

$g_L = (0,0,...,0,1)^T \in \mathbb{R}^{l+1}$, meaning all labelled nodes are grounded and one +1V node.

$$Q_{UL}^i \triangleq \begin{pmatrix} w_{l+1,1} & \cdots & w_{l+1,l} & w_{l+1,i} \\ \vdots & \vdots & \vdots & \vdots \\ w_{i-1,1} & \cdots & w_{i-1,l} & w_{i-1,i} \\ w_{i+1,1} & \cdots & w_{i+1,l} & w_{i+1,i} \\ \vdots & \vdots & \vdots & \vdots \\ w_{n,1} & \cdots & w_{n,l} & w_{n,i} \end{pmatrix}, \quad Q_{UU}^i \triangleq \begin{pmatrix} w_{l+1,l+1} & \cdots & w_{l+1,i-1} & w_{l+1,i+1} & \cdots & w_{l+1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{i-1,l+1} & \cdots & w_{i-1,i-1} & w_{i-1,i+1} & \cdots & w_{i-1,n} \\ w_{i+1,l+1} & \cdots & w_{i+1,i-1} & w_{i+1,i+1} & \cdots & w_{i+1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{n,l+1} & \cdots & w_{n,i-1} & w_{n,i+1} & \cdots & w_{n,n} \end{pmatrix},$$

$$Q_{LU} \triangleq \begin{pmatrix} w_{1,l+1} & \cdots & w_{1,n} \\ \vdots & \vdots & \vdots \\ w_{l,l+1} & \cdots & w_{l,n} \end{pmatrix}, \qquad Q_{LU}^i \triangleq \begin{pmatrix} w_{1,l+1} & \cdots & w_{1,i-1} & w_{1,i+1} & \cdots & w_{1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{l,l+1} & \cdots & w_{l,i-1} & w_{l,i+1} & \cdots & w_{l,n} \end{pmatrix}.$$

$g_U^i \triangleq (g_{l+1},\cdots,g_{i-1},g_{i+1},\cdots,g_n)^T$, which are the potentials of the unlabelled nodes except *i*.

$\tilde{g}_U^i \triangleq (g_{l+1},\cdots,g_{i-1},1,g_{i+1},\cdots,g_n)^T$, now includes the potential of node *i*.

Then we have $g_U^i = \left(I - Q_{UU}^i\right)^{-1} Q_{UL}^i g_L$.

The conductance of node *i* relative to the ground is: $\vec{1}_l^T Q_{LU} \tilde{g}_U^i = \vec{1}_l^T Q_{LU}^i g_U^i + \sum_{j=1}^{l} w_{ji}$. The final overall objective function to be maximized is: $\sum_{i=l+1}^{n} \vec{1}_l^T Q_{LU}^i g_U^i + \sum_{i=1}^{l} \sum_{j=l+1}^{n} w_{ij}$.

It will take considerable effort to design an efficient algorithm to calculate the gradient. This regularizer is also in leave-one-out form, but leaving one *unlabelled* example out!

Chapter 5 Experiments

In this chapter, we present a comprehensive empirical comparison among a number of graph based semi-supervised learning algorithms on a common set of datasets under the comparable experimental settings. The main design concern is the tradeoff between fairness and efficiency. Though all experiments are random in nature, *we have very carefully ensured them to be reproducible*, following Kevin Murphy's suggestion (http://www.cs.ubc.ca/~murphyk /Teaching/guideForStudents.html). The code for both learning algorithm implementation and cross validation testing using the parallel Sun Grid Engine is also downloadable from http://www.comp.nus.edu.sg/~zhangxi2/thesis.html, and will be made easily found by search engines after the author's current account expires. A detailed analysis of the comparison results is presented in this chapter.

5.1 Algorithms compared

The 12 algorithms compared are:

1. Harmonic energy minimization (HEM) as described by Equation (2.4) in section 2.2;
2. Entropy minimization graph learning (MinEnt) as described in section 3.1.2;
3. Leave-one-out hyperparameter learning (LOOHL) with square loss regularizer (Sqr) as described in point 1 of section 4.1;
4. Leave-one-out hyperparameter learning with eigengap maximization regularizer (Eigen) as described in section 4.3;
5. Leave-one-out hyperparameter learning with first-hit time minimization regularizer

(Step) as described in section 4.4;

6. Leave-one-out hyperparameter learning with row entropy maximization regularizer (REnt) as described in section 4.5.

Each algorithm above employs 2 different ways to transform soft labels to hard labels: thresholding (Thrd) by 0.5, and class mass normalization (CMN) as described in section 2.2. We refer to these algorithms as HEM+Thrd, LOOHL+Sqr+CMN, etc. When we just say HEM, it means both HEM+Thrd and HEM+CMN.

We briefly compare the complexity of the above algorithms in Table 5.1. The rough time cost is based on a 4-vs-9 dataset with 475 256-level discrete features and 2000 data points including 30 labelled points. We use SVM as unit. As MinEnt and LOOHL use gradient descent whose total number of iteration before convergence is not fixed, we only give the theoretical complexity and rough time cost for each gradient evaluation. We also include the complexity and model coefficients of other four commonly used semi-supervised algorithms just for reference as baseline, namely SVM, transductive SVM (TSVM), st-mincut, and spectral graphical transducer (SGT). We do not compare accuracy with these algorithms in this thesis because our focus is on graph learning for graph based semi-supervised algorithms, particularly for HEM. So the comparison between classification algorithms is not of our main interest.

	model coefficients	dominating computational cost	time
HEM	distance model (RBF, polynomial), and its coefficients (RBF bandwidth)	invert a $n \times n$ matrix: $O(n^3)$	7
MinEnt	all coefficients in HEM, plus smoothing factor ε (uniform dist'n).	Gradient descent, each iteration: invert a $n \times n$ matrix: $O(n^3)$	10
LOOHL + Sqr / + Step / +REnt	all coefficients in MinEnt, plus LOO loss/regularizer tradeoff factor	Gradient descent, in each iteration: $O(n^3 + n^2\tilde{m})$, see section 4.1, 4.4, and 4.5 for Sqr, Step, and REnt respectively.	50
LOOHL + Eigen	all coefficients in above LOOHL, power of eigenvalues.	Gradient descent, each iteration: $O(n^3 + n^2\tilde{m})$, see section 4.3.	50
SVM	kernel type (RBF, polynomial, linear), kernel parameter (RBF bandwidth), loss/margin tradeoff factor.	solve a *QP*, number of variables = number of labelled examples. with bound constraints. Fastest!	1
TSVM	all coefficients in SVM, fraction of unlabelled examples to be classified as positive.	non-convex! solving the same *QP* as in SVM at each iteration, but slowly increment the cost factors	10
SGT	all coefficients in HEM, nearest neighbour number *k*, number of smallest eigenvectors kept.	graph construction $O(n^2\tilde{m})$, eigen-decomposition $O(kn^2)$.	10

st-mincut	all coefficients in HEM, threshold of edge weight filtering δ, downscaling factor of edges between two unlabelled data points.	graph construction: $O(n^2\tilde{m})$, max flow: $O(n^2E)$ or $O(n^3)$. (E is the number of edges)	5

Table 5.1 Comparison of semi-supervised learning algorithms' complexity

All the 12 algorithms were implemented by the author in C++, mixing two toolboxes: Matlab C++ Math library for four matrix operations, and Toolkit for Advanced Optimization (TAO) by (Benson et al., 2004) for gradient descent optimization. See Appendix B.1 for details of the two toolboxes.

For LOOHL+Regularizers, we re-weighted leave-one-out loss and normalized the regularizers. Suppose there are r_+ positive labelled examples and r_- negative labelled examples. Then we normalized the leave-one-out loss (*Loo_loss*) as

$$Loo\_loss \triangleq \frac{1}{2r_+}\sum_{y_i \text{ is positive}} Loo\_loss(x_i, y_i) + \frac{1}{2r_-}\sum_{y_i \text{ is negative}} Loo\_loss(x_i, y_i). \quad (5.1)$$

and the final objective function is: $C_1 \times Loo\_loss + C_2 \times regularizer$.

The regularizers Sqr, Eigen, Step, and REnt were normalized by dividing by m, n, u^2, and nu respectively, where m stands for the number of features, u stands for the number of unlabelled examples and n stands for the number of total examples.

5.2 Datasets chosen

We compare the above mentioned 12 semi-supervised learning algorithms on 5 datasets, covering five different domains of real world applications. Each of the first four datasets has two forms, namely the original form and the processed form (called probe form), where certain noisy and useless features called probes are added to test performance of the algorithms in the presence of irrelevant features. So there are 9 datasets in all. The 5 datasets in their original form are:

1. Handwritten digits discrimination: 4 vs 9;
2. Distinguish cancer versus normal patterns from mass-spectrometric data;
3. Reuters text categorization: about "corporate acquisitions" or not;
4. Predict whether a compound binds to thrombin;
5. Ionosphere from UCI repository (Newman et al., 1998).

The main purpose of introducing the probe versions of the data is to see the effect of useless features as a supplementary result. The probe version of the first four datasets was used in the NIPS 2003 workshop on feature selection, which does NOT provide any source code describing how the original features were processed. Though we attempted to reproduce the pre-processing of NIPS workshop according to its design write-up, we find that the resulting accuracy is far different from the directly downloaded dataset. So we downloaded the probe version dataset from the workshop web site and used it in experiments. We also downloaded the raw dataset from various sources and applied preliminary pre-processing by

ourselves. Therefore, special care must be paid when comparing the performance between original form and probe form (see section 5.4.1.2). The details of pre-processing and the properties of the five datasets are available in Appendix A. We summarize the properties and statistics of the five datasets in Table 5.2. All tasks are binary classification (positive/negative). We define *sparsity* of a dataset as:

$$\frac{\sum_i number\ of\ zero\ valued\ features\ in\ example\ i}{(total\ number\ of\ features\ in\ whole\ dataset)\times(number\ of\ examples)}.$$

	feature property: continuous or discrete (how many levels), sparsity	number of features	*dataset* size (#positive: #negative: #unlabelled)[1]
handwritten digits (4 vs 9)	Original: 256 levels, 79.7 % Probe: 1000 levels, 85.8 %	Original: 475 Probe: 4525	1000 : 1000 : 0
cancer vs normal	Original: continuous, 50.3 % Probe: 1000 levels, 45.4 %	Original: 18543 Probe: 9961[2]	88 : 112 : 0
Reuters text categorization	Original: continuous, 99.0 % Probe: 1000 levels, 99.3 %	Original: 4608 Probe: 13459	300 : 300 : 1400
compounds bind to thrombin	Original: binary, 99.43 % Probe: binary, 98.55 %	Original: 31976 Probe: 58417	112 : 1038 : 0
Ionosphere	continuous, 38.3 %	33	225 : 126: 0

Table 5.2 Summary of the five datasets' property

[1] This is common to both original and probe forms.
[2] There are more features in original form than in probe form, because the feature selection process in the NIPS workshop is beyond our domain knowledge and thus only the very original dataset was used.

We call the whole available set of {positive examples, negative examples} as *labelled set*, and call its union with {unlabelled examples} as *dataset*. The numbers in the last column are based on the *dataset*. In experiment, only a few labelled data will be sampled from the labelled set, and they are called *labelled examples*. The rest labelled data in the labelled set are used as testing data, which are effectively unlabelled for transductive learning. When computational cost is manageable, unlabelled examples (called unlabelled *set*), if available, are added to the dataset, as in Reuters text categorization, just for learning and not for testing. Thus, all examples in the dataset will be involved in the learning process. See details of dataset selection in Appendix B.2.

5.3 Detailed procedure of cross validation with transductive learning

To make the comparison fair, we used the *k*-fold cross validation in its most standard form to do model selection, which is complicated under the settings of semi-supervised learning and is different from normal settings for supervised learning. We will describe it in the framework of self-repartitioning test.

To begin with, we introduce a shorthand as the atomic step in cross validation:

$$Score(X^l, Y^l, X_{te}, Y_{te}, X^u, A, \lambda)$$

which represents the following two steps:

1. Train by algorithm A with coefficient λ and output a learned model M, using the labelled training data (X^l, Y^l), the unlabelled data X^u, and the test example set X_{te}. Since data points used for training can also be unlabelled, and labelled data points can

only be used for training, we will henceforth just call X^l *labelled examples* instead of training points.

2. Now feed the testing points X_{te} to M and get the predictions, which are compared against the correct labels Y_{te} to produce a performance score, e.g., accuracy. This process may also make use of the $\left(X^l, Y^l\right)$ or X^u, depending on different algorithms used.

Sometimes, especially in the transductive settings, these two steps may not be separable. By using the notation *Score*($X^l, Y^l, X_{te}, Y_{te}, X^u, A, \lambda$), we can treat transductive and inductive semi-supervised learning algorithms in the same way, without worrying about whether there is a intermediate (inductive) model M. See examples of $Score(\cdot)$ in Appendix B.3.

Secondly, to evaluate the performance of an algorithm A on l labelled examples with a given coefficient λ (ref. Table 5.1), we adopt the self-repartitioning test as follows. Suppose we have a labelled set (X, Y) with size n ($n > l$) and an unlabelled set X^u. Then we do the following steps:

1	For $i = 1 \ldots r$ (r trials of test)
2	Randomly pick l examples from (X, Y), denoted as $\left(X_i^l, Y_i^l\right)$. Denote the rest $n - l$ examples as $\left(X_i^t, Y_i^t\right)$.
3	Calculate s_i = *Score*($X_i^l, Y_i^l, X_i^t, Y_i^t, X^u, A, \lambda$).
	End

4	Calculate the average of $s_1 \ldots s_r$. This is the self-repartitioning test score of A under λ for l training examples, $n - l$ testing examples, and X^u.

Table 5.3 Self-partitioning diagram for transductive performance evaluation

When r is large, this self-repartitioning test score approaches the expected test error of A under λ, for l training examples, $n - l$ testing examples, and X^u.

Finally, we describe in Table 5.4 the process of k-fold cross validation, using the same notation of algorithm A, labelled set (X, Y), and unlabelled set X^u as in Table 5.3. However, now the coefficient λ is not given, but picked by cross validation.

1	For $i = 1 \ldots r$ (r trials of test)
2	Randomly pick l examples from (X, Y), denoted as $\left(X_i^l, Y_i^l\right)$. Denote the rest $n - l$ examples as $\left(X_i^{te}, Y_i^{te}\right)$.[1]
3	Randomly partition $\left(X_i^l, Y_i^l\right)$ into k equally sized subsets: $\left(X_i^l(t), Y_i^l(t)\right)_{t=1}^k$.[2] Denote $X_i^l(-t) \triangleq \bigcup_{s \neq t} X_i^l(s)$, $Y_i^l(-t) \triangleq \bigcup_{s \neq t} Y_i^l(s)$.
4	For all possible coefficient configurations λ_j ($j = 1 \ldots q$).
5	Calculate $s_j^i \triangleq \frac{1}{k} \sum_{t=1}^{k} Score\left(X_i^l(-t), Y_i^l(-t), X_i^l(t), Y_i^l(t), X_i^{te} \cup X^u, A, \lambda_j\right)$
	End
6	Pick $\lambda_i^{opt}(l) \triangleq \arg\max_{j \in 1 \ldots q} s_j^i$.

[1] We require that the number of examples from each class be no less than k.
[2] We require that in each subset, the number of examples from each class is larger than zero.

7 Calculate $s_i = Score\left(X_i^l, Y_i^l, X_i^{te}, Y_i^{te}, X^u, A, \lambda_i^{opt}(l)\right)$.

End

8 Calculate the average of $s_1 \ldots s_r$: $S(A,l) \triangleq \frac{1}{r}\sum_{i=1}^{r} s_i$.

Table 5.4 Pseudo-code for *k*-fold semi-supervised cross validation

Despite the fairness offered by *k*-fold cross validation, its expensive computational complexity is a big problem. So in practice, we only applied the above *k*-fold cross validation to HEM, and used some fixed rules or fixed parameters for LOOHL across all datasets to achieve both fairness and efficiency. Refer to Appendix B.3 for a detailed analysis of the complexity of semi-supervised cross validation both theoretically and empirically.

5.4 Experimental results: comparison and analysis

In this section, we present the results centring on two comparisons, which answer two questions:

1. Does LOOHL improve accuracy compared with HEM and MinEnt, in both Thrd and CMN? We use LOOHL+Sqr as the representative of LOOHL due to its simplicity.
2. How do the four regularizers behave comparatively?

The details of the experimental settings are available in Appendix B.4. We will just highlight some important points of the settings in the following two sections.

5.4.1 Comparing LOOHL+Sqr with HEM and MinEnt, under threshold and CMN

We first compare the accuracy of LOOHL+Sqr, HEM, and MinEnt, each with both threshold form and CMN form, on the same $4 \times 2 + 1 = 9$ datasets. Since the Sqr regularizer is the simplest, we use it as a representative for LOOHL.

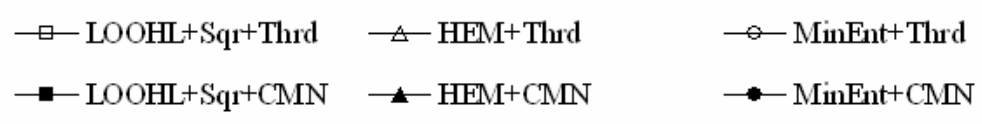
For each dataset, we normalized all input feature vectors to have length 1. We tested on three different numbers of labelled points, l_1, l_2, l_3, whose exact value will be shown in the figures. For each l_i, we randomly picked the labelled points for 10 times, and report the average result of the 10 trials. We used 5 fold cross validation to select the model for HEM. For MinEnt, we tested the accuracy on *all* the discretized coefficients and then report the highest accuracy without cross validation, which is an unfair advantage for MinEnt. For LOOHL+Sqr, we chose the prior mean of the bandwidth in the regularizer as the bandwidth selected by HEM+Thrd via cross validation. A *fixed* rule to adjust the bandwidth according to optimization behavior without peeking into the test labels was used and all other coefficients were fixed across all datasets. Details are available in Appendix B.4.

5.4.1.1 Comparison on original forms

We first present the results on the original forms of the 5 datasets in Figure 5.1 to Figure 5.5. From the results in these figures, we can make the following observations and conclusions:

1. LOOHL+Sqr generally outperforms HEM and MinEnt. Both LOOHL+Sqr+Thrd and LOOHL+Sqr+CMN outperform HEM and MinEnt (regardless of Thrd or CMN) on all

datasets except thrombin and ionosphere, where the better of LOOHL+Sqr+CMN and LOOHL+Sqr +Thrd finally performs best.



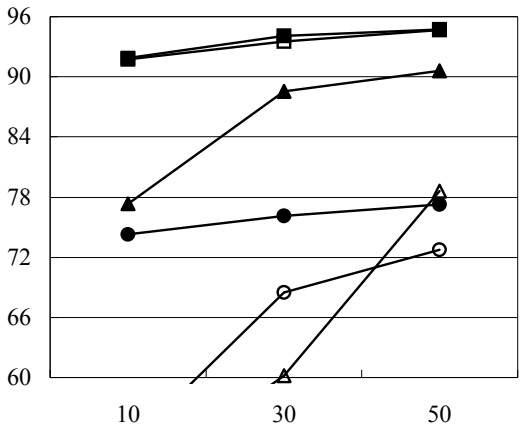


Figure 5.1 Accuracy of LOOHL+Sqr, HEM and MinEnt on 4vs9 (original)

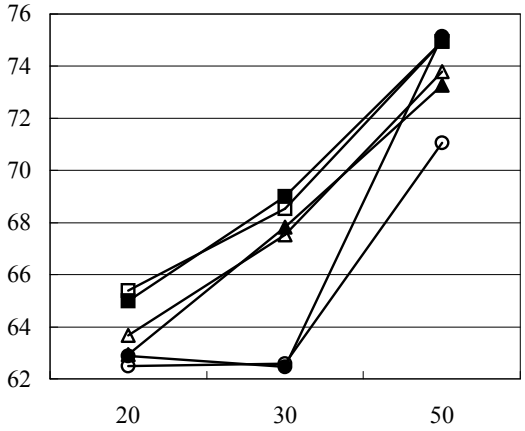


Figure 5.2 Accuracy of LOOHL+Sqr, HEM and MinEnt on cancer (original)

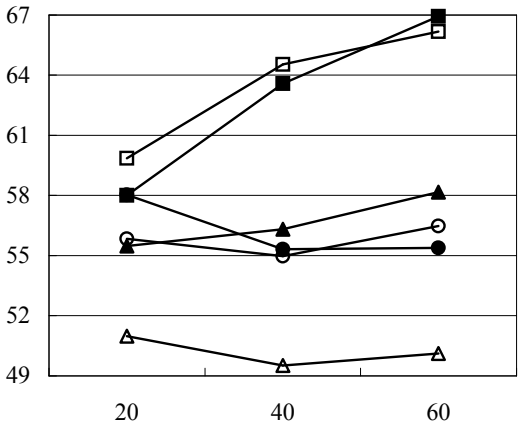


Figure 5.3 Accuracy of LOOHL+Sqr, HEM and MinEnt on text (original)

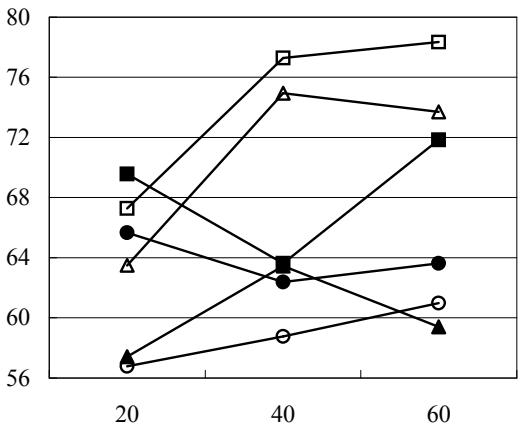


Figure 5.4 Accuracy of LOOHL+Sqr, HEM and MinEnt on thrombin (original)

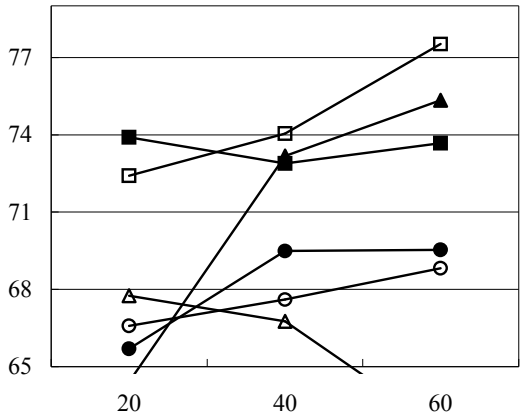


Figure 5.5 Accuracy of LOOHL+Sqr, HEM and MinEnt on ionosphere

2. CMN is almost always better than thresholding, except on the thrombin dataset, where CMN hurts both HEM and LOOHL+Sqr. In (Zhu et al., 2003a), it is claimed that although the theory of HEM is sound, CMN is still necessary to achieve reasonable performance because the underlying graph is often poorly estimated and may not reflect the classification goal, i.e., one should not rely exclusively on the graph. Now that our LOOHL+Sqr is aimed at learning a good graph, the ideal case is that the graph learned is suitable for our classification such that the improvements by CMN will *not* be large. In other words, the difference between LOOHL+Sqr+CMN and LOOHL+Sqr+Thrd, compared with the difference between HEM+CMN and HEM+Thrd, can be viewed as an *approximate* indicator of how well the graph is learned by LOOHL+Sqr. Of course the absolute accuracy value is also important. If the accuracy for CMN and Thrd are equally low, then we may not conclude the graph is well learned, or the classification algorithm is bad.

The efficacy of LOOHL+Sqr can be clearly observed in datasets 4vs9, cancer, and text. In these cases, we see that LOOHL+Sqr+Thrd is already achieving high accuracy and LOOHL+Sqr+CMN does not offer much improvement then. However, we can see that HEM+CMN does yield significant improvement on top of HEM+Thrd for these datasets, which means that the graph chosen by cross validation is still not good, and it is thus desirable to learn the bandwidth for each dimension of the feature vector.

On the ionosphere dataset, LOOHL+Sqr+CMN significantly outperforms LOOHL+Sqr +Thrd, which indicates that the graph learned by LOOHL+Sqr is still not good enough. This conclusion is further supported by the fact that HEM+CMN outperforms LOOHL+Sqr+Thrd. To analyse why this happens, we notice that the class distribution for ionosphere is *slightly* unbalanced at 1:2. As the labelled set we have sampled for training is rather small, we observed in the experiment that the class distribution in the labelled data is deviated from the correct ratio. The more unbalanced the dataset is, the more serious is the deviation. Since we are using the balanced leave-one-out loss defined in (5.1) for graph learning, we conjecture that the inaccurate sampling of the class distribution in the labelled points can make the balanced leave-one-out loss inaccurate, and therefore the graph learned may not be suitable for the test set. However, as the class distribution is not too unbalanced, the CMN can still be helpful at the classification phase, although the efficacy of graph learning may not survive. In other words, inaccurate sampling of unbalanced class distribution is believed to hurt more on graph learning than on CMN classification.

On the other hand, CMN may sometimes hurt accuracy. This occurred in the thrombin dataset, where HEM+Thrd and LOOHL+Sqr+Thrd outperform HEM+CMN and LOOHL+Sqr+CMN respectively. One possible explanation is that the class distribution in thrombin is *highly* unbalanced (about 1:10). Therefore the estimation of class distribution in the test dataset by using labelled points can be very inaccurate, which makes CMN behave poorly.

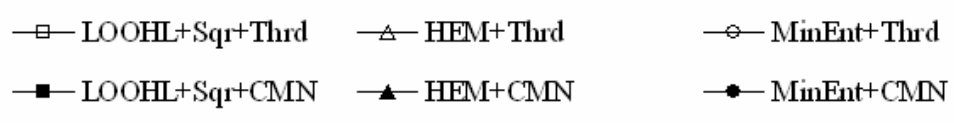
3. The performance of MinEnt is generally inferior to HEM and LOOHL+Sqr. We have already reported the highest accuracy by exhaustive search on the coefficient grid, rather than selecting models by cross validation. MinEnt+Thrd has the equal chance of outperforming or losing to HEM+Thred, while HEM+CMN is almost always better than MinEnt+CMN. Most of the time, MinEnt+CMN performs significantly better than MinEnt+Thrd, so we can conclude that MinEnt fails to learn a good graph. This may be due to converging to a poor local minimum, or the idea of minimizing the entropy on unlabelled data is by itself unjustified.

5.4.1.2 Comparison on probe forms

Secondly, we present the result on the probe forms of the 4 datasets in Figure 5.6 to Figure 5.9.

As a supplementary result, the main purpose of studying the accuracy after introducing the

probe versions of the data is to see the effect of useless features. First of all, by just looking at the results of probe version, we draw similar conclusions as in the original form. LOOHL+Sqr is performing consistently better than HEM and MinEnt. MinEnt works consistently poorly. CMN almost always improves performance on top of thresholding for all the three algorithms, and for all the four datasets.



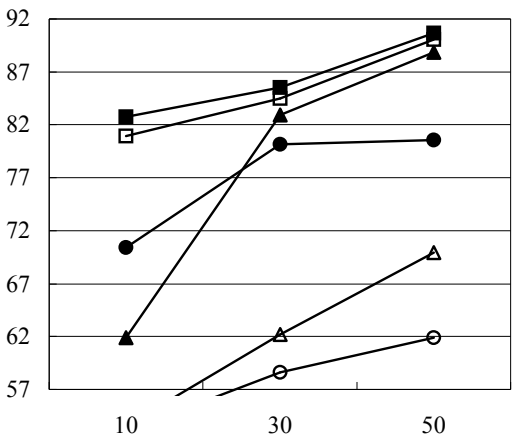


Figure 5.6 Accuracy of LOOHL+Sqr, HEM and MinEnt on 4vs9 (probe)

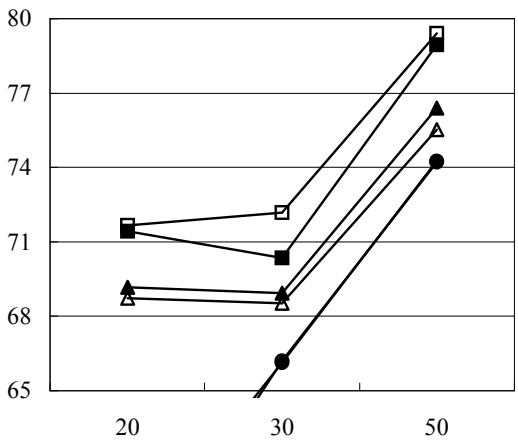


Figure 5.7 Accuracy of LOOHL+Sqr, HEM and MinEnt on cancer (probe)

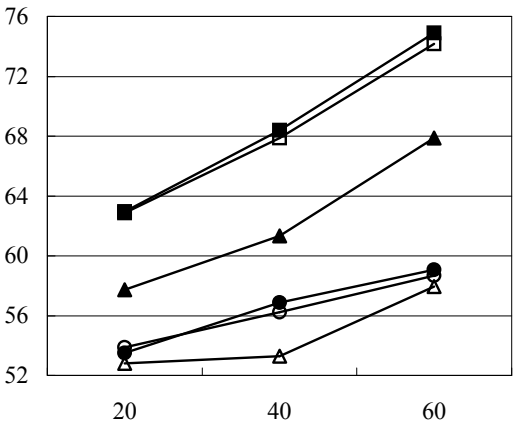


Figure 5.8 Accuracy of LOOHL+Sqr, HEM and MinEnt on text (probe)

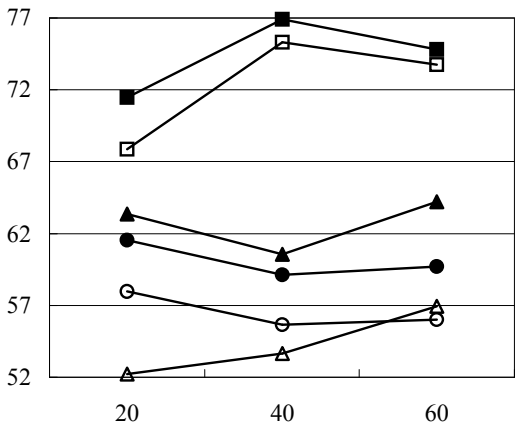


Figure 5.9 Accuracy of LOOHL+Sqr, HEM and MinEnt on thrombin (probe)

Next, we compare the results on the original version and on the probe version. For cancer and text dataset probe version outperforms original version, while for 4vs9 and thrombin, probe version actually underperforms. Since probe version is both performing feature selection by discarding less informative (probably noisy) features and adding new random features, the real effect of this pre-processing is not very clear. Besides, we downloaded the probe version directly from NIPS workshop web page, while the original version was downloaded from other sources. Since the NIPS workshop did not offer the source code and its description is not in enough detail, it is expected that some fancy pre-processing may actually improve/hurt the performance of probe version. To be more careful, we pre-processed the 4vs9 dataset by strictly following the descriptions in NIPS workshop write-up. However, our probe version dataset yielded significantly different performance from the probe version directly downloaded. So we are very confident that there must be some processes omitted in that write-up. Our inability to reproduce the pre-processing makes the results less conclusive, and the comparison between the performance on original version and on probe version should be viewed just as reference.

We paid special attention to the probe version of thrombin, and surprisingly CMN does give improvements there for both LOOHL+Sqr and HEM. This phenomenon does not tally with our previous argument that the highly unbalanced class distribution should make CMN behave poorly. We believe that it is not due to our LOOHL+Sqr because HEM is also behaving in the same way in these two versions. So we may attribute it to the unknown

pre-processing in the probe version of thrombin, which made CMN immune to the bad estimation of class distribution.

5.4.2 Comparing four regularizers of LOOHL, under threshold and CMN

In this section, we compare the four regularizers of LOOHL: square loss minimization (Sqr), eigengap maximization (Eigen), first-hit time minimization regularizer (Step), and row entropy maximization (REnt). The same 9 datasets are used, except the probe form of thrombin which is too spatially costly to run in this large scale experiment.

For Eigen, Step and REnt, we enumerated the performance on a grid of the coefficients. The two coefficients are the initial values of bandwidth σ and the weight ratio between leave-one-out loss and regularization C_1:C_2. Since we want to use the result of Sqr in the previous section as a baseline, we just copied the results in the previous section 5.4 to this section and did not run on different model coefficients settings. To be fair, we still ran 10 trials using the same labelled examples randomly chosen in section 5.4. In this way, the Sqr result copied from section 5.4 is comparable with the result of the other regularizers.

The five sub-plots from left to right in the following 8 figures correspond to different values of σ, common to all datasets. We report here the highest accuracy among all C_1:C_2, because reporting the result for each C_1:C_2 will have to take 24 figures (C_1:C_2 is discretized to 3 levels). Some points are missing because optimization always failed at the corresponding coefficient setting. Again, see section B.4 for the details of experimental settings.

From the results in Figure 5.10 to Figure 5.17 and the experience of experiment, especially about C_1:C_2 which we present in Appendix C, we can draw the following conclusions:

1. The Sqr regularizer with our fixed coefficients generally performs reliably well among all regularizers. Since the results of Sqr are based on a fixed coefficients and the results of other regularizers are enumeration of the performance by a number of models instead of using cross validation model selection, we are quite confident that Sqr is a reasonable regularizer.
2. The Eigen regularizer yields more promising and more stable accuracy than REnt and Step. It performs competitively to Sqr regularizer particularly on probe versions. Besides, it is also less prone to numerical problems in optimization.
3. Step regularizer must be used by carefully tuning C_1:C_2, because unlike Eigen and Rent, its value is unbounded. It is the regularizer most susceptible to optimization breakdown. As for accuracy, it is not as good as Eigen.
4. The REnt regularizer almost always yields poor accuracy, though its optimization is the smoothest (i.e., least susceptible to numerical problems) among all the three regularizers.
5. σ has a more profound influence on both optimization and accuracy than C_1:C_2. With larger σ, the optimization can usually proceed more smoothly to the end.
6. When C_1:C_2 is set in an improper range, the performance can be very poor and numerical problems are frequently encountered. But when it is set to the proper range, the

accuracy is less sensitive to C_1:C_2 than to σ.

7. CMN is useful for most regularizers, offering improvement on top of thresholding.

8. There still leaves much space to be desired in the direction of regularization. Though the Eigen, Step, and REnt regularizers seem more theoretically sound than the Sqr regularizer, their empirical performance shows that further research needs to be done into it.

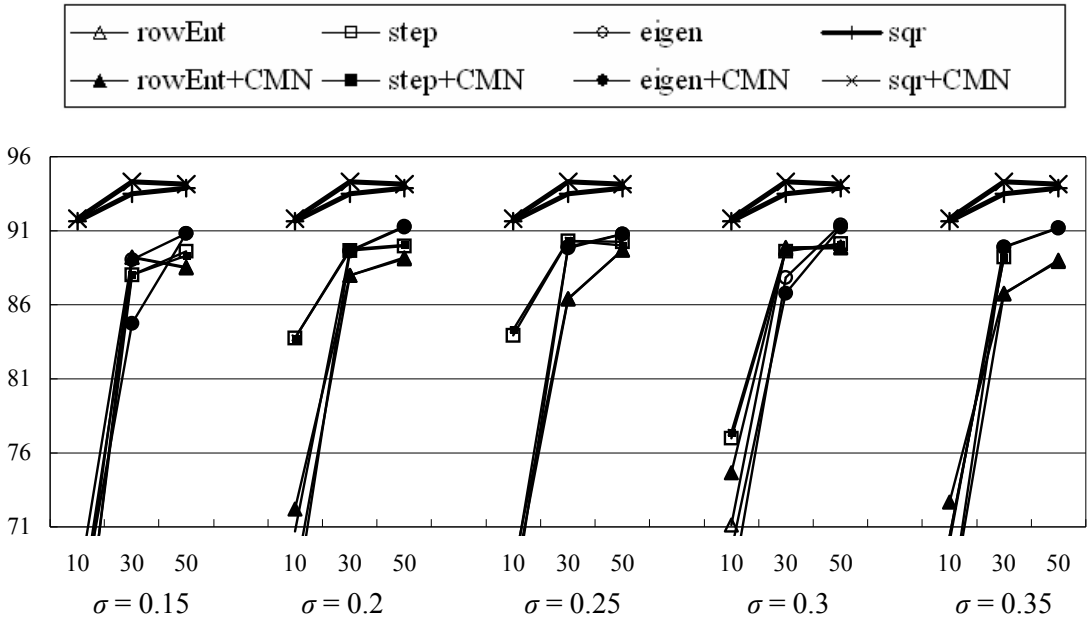


Figure 5.10 Accuracy of four regularizers on 4vs9 (original)

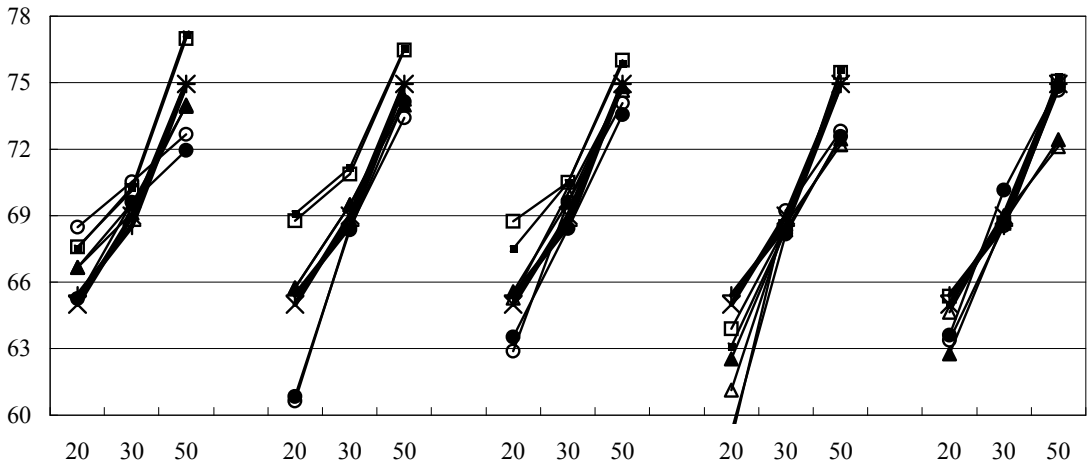


Figure 5.11 Accuracy of four regularizers on cancer (original)

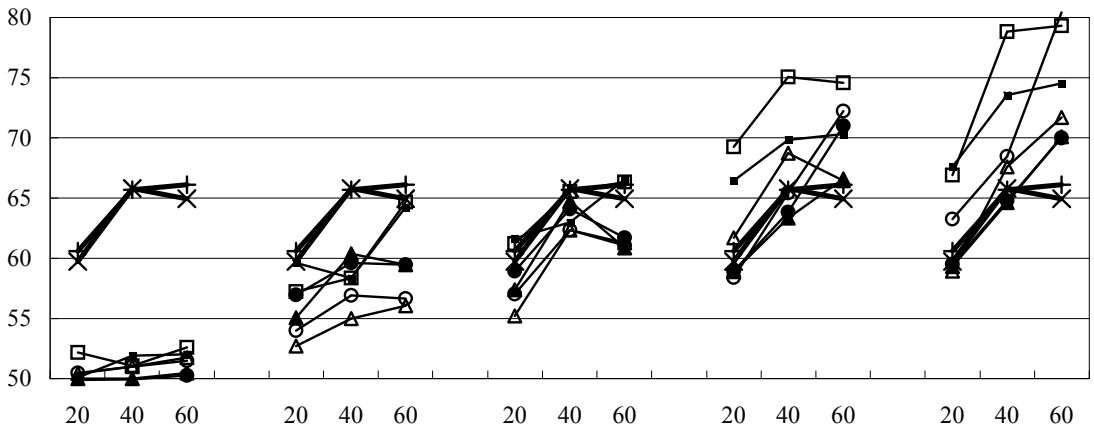


Figure 5.12 Accuracy of four regularizers on text (original)

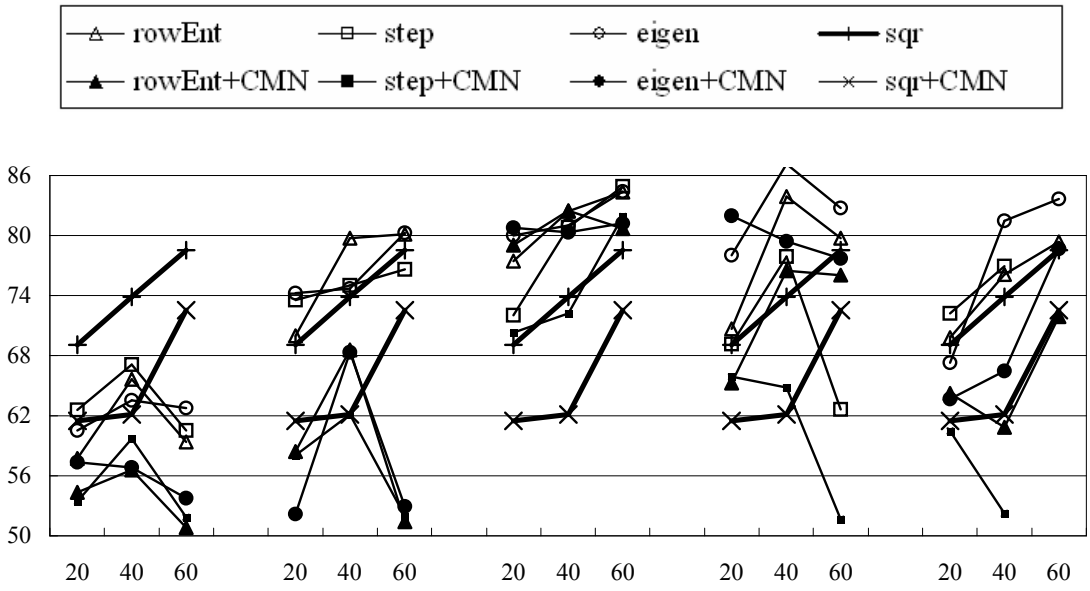


Figure 5.13 Accuracy of four regularizers on thrombin (original)

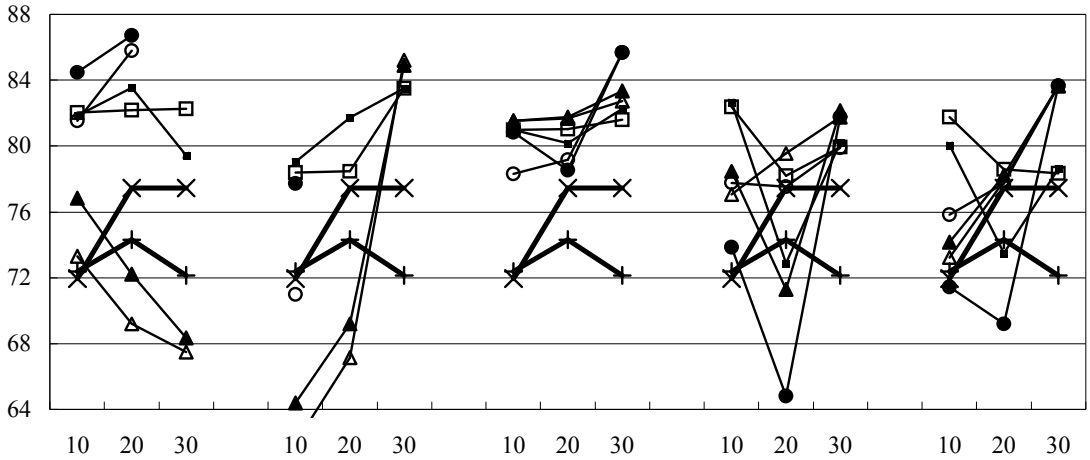


Figure 5.14 Accuracy of four regularizers on ionosphere

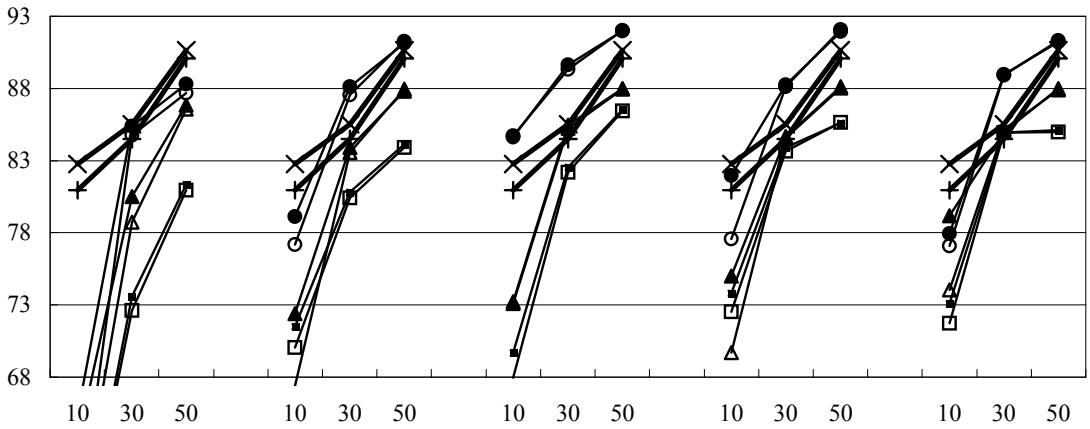


Figure 5.15 Accuracy of four regularizers on 4vs9 (probe)

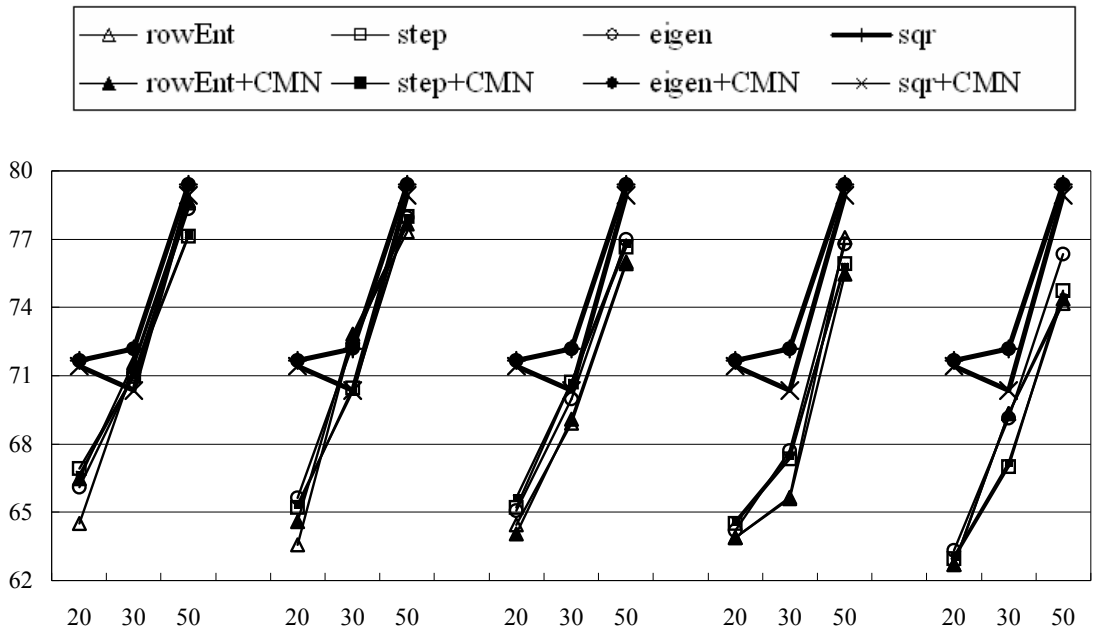


Figure 5.16 Accuracy of four regularizers on cancer (probe)

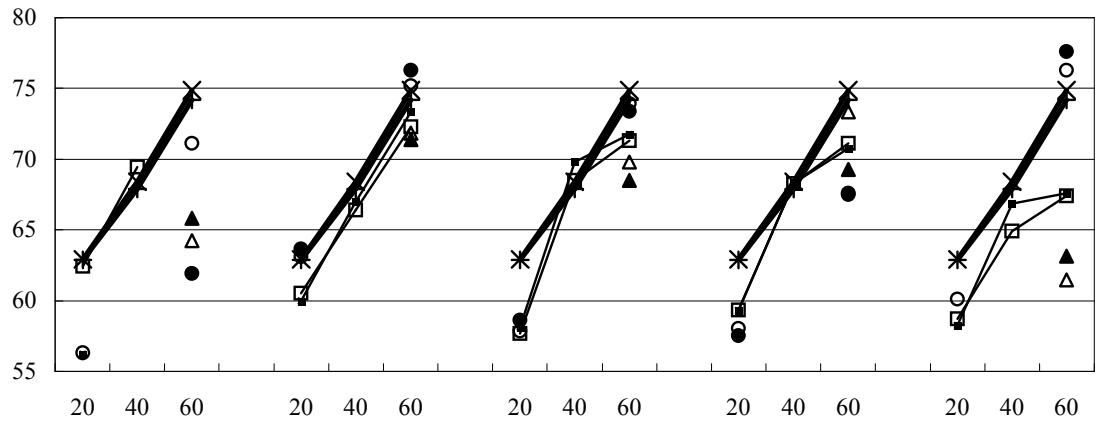


Figure 5.17 Accuracy of four regularizers on text (probe)

Chapter 6 Conclusions and Future Work

In many practical applications of data classification and data mining, one finds a wealth of easily available unlabelled examples, while collecting labelled examples can be costly and time-consuming. In order to make use of the unlabelled data to improve learning performance, a wide spectrum of semi-supervised learning algorithms have been proposed with published success both in theory and in practice. One class of semi-supervised learning algorithm, which is based on graph with nodes representing labelled and unlabelled data and edges encoding the similarity between data points, has gained particular popularity due to its natural bridging between supervised learning and unsupervised learning. However, how to build the underlying graph still has not been well studied, though it is shown to exert substantial influence on the performance of the learner built upon it.

In this thesis, we presented a graph hyperparameter learning method for graph based semi-supervised learning. The approach is based on the minimal leave-one-out loss criterion, and an efficient implementation algorithm is designed to overcome the problem of computational complexity, suffered by most leave-one-out style algorithms. Furthermore, we investigated the field of graph learning regularization, which is still an unexplored area. Various regularizers based on graph property, random walk, and spectral clustering are proposed. They help to overcome the problem of learning a degenerative graph by the leave-one-out loss minimization.

However, there are still a lot of open questions and challenges in semi-supervised learning, and in graph learning in particular. We mention a few below:

1. How to make the optimization more tractable? Although the computational cost is our central focus and it has already almost reached its lower bound, the optimization process can still be very expensive, especially when a large number of iterations are required or the size of the dataset is large (beyond a few thousand). It might help if an optimization algorithm is specially designed for this task, rather than using generic optimization algorithms (e.g., gradient descent) and packages.

2. How to formulate the problem as a convex optimization problem. Now local minima are still a problem. However, the matrix inversion operation and the fact that optimization variables appear in an exponential function make it almost impossible to design a convex objective function. Is there any other way to define the similarity measure whose resulting leave-one-out loss function is convex? Or is the/a good graph intrinsically not unique?

3. How to deal with structured data, e.g., sequences, trees with all variants? Very recent work has been done in (Altun et al., 2005) by using maximum margin, graph Laplacian regularizer and its associated kernel concept in RKHS. Our proposed hyperparameter learning algorithm assumes that the data points are represented by a numerical vector. Without such assumption, what is a good parametric form that suits the structured data,

and most importantly, keeping the key techniques still applicable which make the leave-one-out loss minimization efficient and tractable?

4. Is one graph learning algorithm good for all graph based semi-supervised algorithms? Do different classification algorithm benefit from different graphs? Or do different tasks, such as clustering, regression, ranking call for graphs designed with different philosophy? Or even for the classification algorithm, does a multi-class task need different graphs in applying one-again-rest and one-against-one heuristic?

5. How to incorporate the ideas in kernel learning to graph learning, and vice versa. We know that there is abundant work on learning the kernels. Since it has been pointed out that graph Laplacian is closely related to kernels, is it possible to borrow the kernel learning algorithms to learn a graph, in a form beyond merely learning the spectral transform?

6. How to design a better graph learning regularizer by making use of reproducing kernel Hilbert space theory and vice versa. Now even the literature for regularizing learning the kernels is scarce. However, as we saw in Chapter 4, graph learning provides many intuitive views on this problem. It will be desirable if such graph properties can be used to regularize the kernel learning, or in the reverse direction, use the reproducing kernel Hilbert space theory to design graph learning regularizers.

The author hopes that his future doctoral study will lead to the solution of some or most of these challenges with success.

Bibliography

Altun, Yasemin, McAlleste, David and Belkin, Mikhail (2005). Maximum Margin Semi-Supervised Learning for Structured Variables. *Neural Information Processing Systems*.

Argyriou, Andreas, Herbster, Mark and Pontil, Massimiliano (2005). Combining Graph Laplacians for Semi-Supervised Learning. *Neural Information Processing Systems*.

Balcan, Maria-Florina, Blum, Avrim and Yang, Ke (2004). Co-Training and Expansion: Towards Bridging Theory and Practice. *Neural Information Processing Systems*.

Belkin, Mikhail, Matveeva, Irina and Niyogi, Partha (2004a). Regularization and Semi-supervised Learning on Large Graphs. *Conference on Computational Learning Theory*.

Belkin, Mikhail and Nigam, Kamal (2004). Semi-Supervised Learning on Riemannian Manifolds. *Machine Learning* 56: 209-239.

Belkin, Mikhail, Niyogi, Partha and Sindhwani, Vikas (2004b). Manifold Regularization: A Geometric Framework for Learning from Examples. Technical Report.

Belkin, Mikhail, Niyogi, Partha and Sindhwani, Vikas (2005). On Manifold Regularization. *AI & Statistics*.

Bennett, Kristin P. and Demiriz, Ayhan (1998). Semi-Supervised Support Vector Machines. *Neural Information Processing Systems*.

Benson, Steven J., McInnes, Lois Curfman, Moré, Jorge and Sarich, Jason (2004). TAO User Manual. Manual.

Blum, Avrim and Mitchell, Tom (1998). Combining Labeled and Unlabeled Data with Co-Training. *Conference on Computational Learning Theory*.

Blum, Avrim and Chawla, Shuchi (2001). Learning from Labeled and Unlabeled Data using Graph Mincuts. *International Conference on Machine Learning*.

Blum, Avrim, Lafferty, John, Rwebangira, Mugizi Robert and Reddy, Rajashekar (2004). Semi-supervised Learning Using Randomized Mincuts. *International Conference on Machine Learning*.

Bousquet, Olivier, Chapelle, Olivier and Hein, Matthias (2003). Measure Based Regularization. *Neural Information Processing Systems*.

Boyd, Stephen and Vandenberghe, Lieven (2004). Convex Optimization. Cambridge University Press.

Boykov, Yuri, Veksler, Olga and Zabih, Ramin (1998). Markov Random Fields with Efficient Approximations. *IEEE Computer Society Conference on Computer Vision and Pat-*

tern Recognition.

Carreira-Perpiňán, Miguel Á. and Zemel, Richard S. (2004). Proximity Graphs for Clustering and Manifold Learning. *Neural Information Processing Systems*.

Castelli, Vittorio and Cover, Thomas M. (1995). On the Exponential Value of Labeled Samples. *Pattern Recognition Letters* 16(1): 105-111.

Castelli, Vittorio and Cover, Thomas M. (1996). The Relative Value of Labeled and Unlabeled Samples in Pattern Recognition with an Unknown Mixing Parameter. *IEEE Transactions on Information Theory* 42(6): 2102-2117.

Cataltepe, Zehra and Magdon-Ismail, Malik (1998). Incorporating Test Inputs into Learning. *Neural Information Processing Systems*.

Chapelle, Olivier, Weston, Jason and Schölkopf, Bernhard (2002). Cluster Kernels for Semi-Supervised Learning. *Neural Information Processing Systems*.

Chapelle, Olivier and Zien, Alexander (2005). Semi-Supervised Classification by Low Density Separation. *International Workshop on Artificial Intelligence and Statistics*.

Chung, Fan R. K. (1997). Spectral Graph Theory. American Mathematical Society.

Chung, Fan R. K. and Yau, S. T. (2000). Discrete Green's functions. *Journal of Combinatorial Theory (A)* 91(1-2): 191-214.

Collins, Michael and Singer, Yoram (1999). Unsupervised Models for Named Entity Classification. *Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*.

Corduneanu, Adrian and Jaakkola, Tommi S. (2001). Stable Mixing of Complete and Incomplete Information. Technical Report.

Corduneanu, Adrian and Jaakkola, Tommi (2003). On Information Regularization. *Conference on Uncertainty in AI*.

Corduneanu, Adrian and Jaakkola, Tommi (2004). Distributed Information Regularization on Graphs. *Neural Information Processing Systems*.

Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L. and Stein, Clifford (2001). Introduction to Algorithms.

Cox, Trevor F. and Cox, Michael A. A. (2001). Multidimensional Scaling. London: Chapman and Hall.

Crammer, Koby, Keshet, Joseph and Singer, Yoram (2002). Kernel Design Using Boosting. *Neural Information Processing Systems*.

Cristianini, Nello, Shawe-Taylor, John, Andre, Elisseeff and Kandola, Jaz (2001). On Kernel-Target Alignment. *Neural Information Processing Systems*.

De Bie, Tijl (2005). Semi-supervised Learning Based on Kernel Methods and Graph Cut Algorithms. PhD Thesis.

de Silva, Vin and Tenenbaum, Joshua B. (2003). Global versus Local Methods in Nonlinear Dimensionality Reduction. *Neural Information Processing Systems*.

Doyle, Peter G. and Snell, J. Laurie (1984). Random Walks and Electric Networks. *Mathematical Association of America*.

Fowlkes, Charless, Belongie, Serge, Chung, Fan and Malik, Jitendra (2004). Spectral Grouping Using the Nyström Method. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26(2): 214-225.

Freund, Yoav, Seung, H. Sebastian, Shamir, Eli and Tishby, Naftali (1997). Selective Sampling Using the Query by Committee Algorithm. *Machine Learning* 28(2-3): 133-168.

Fung, Glenn and Mangasarian, O. L. (1999). Semi-Supervised Support Vector Machines for Unlabeled Data Classification. Technical Report.

Goldman, Sally and Zhou, Yan (2000). Enhancing Supervised Learning with Unlabeled Data. *International Conference on Machine Learning*.

Golub, Gene H. and van Loan, Charles F. (1996). Matrix Computations. The Johns Hopkins University Press.

Grady, Leo and Funka-Lea, Gareth (2004). Multi-Label Image Segmentation for Medical Applications Based on Graph-Theoretic Electrical Potentials. *European Conference on Computer Vision 2004 Workshop CVAMIA and MMBIA*.

Grandvalet, Yves and Bengio, Yoshua (2004). Semi-supervised Learning by Entropy Minimization. *Neural Information Processing Systems*.

Greig, D. M., Porteous, B. and Seheult, A. (1989). Exact Maximum a Posteriori Estimation for Binary Images. *Journal of Royal Statistical Society, Series B* 51: 271-279.

Grira, Nizar, Crucianu, Michel and Boujemaa, Nozha (2004). Unsupervised and Semi-supervised Clustering: a Brief Survey. Technical Report.

Hagen, Lars and Kahng, Andrew B. (1992). New Spectral Methods for Ratio Cut Partitioning and Clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 11(9): 1074-1085.

Ham, Jihun, Lee, Daniel D., Mika, Sebastian and Schölkopf, Bernhard (2004). A Kernel View of the Dimensionality Reduction of Manifolds. *International Conference on Machine Learning*.

Jaakkola, Tommi, Meila, Marina and Jebara, Tony (1999). Maximum Entropy Discrimination. *Neural Information Processing Systems*.

Joachims, Thorsten (1999). Transductive Inference for Text Classification using Support Vector Machines. *International Conference on Machine Learning*.

Joachims, Thorsten (2003). Transductive Learning via Spectral Graph Partitioning. *International Conference on Machine Learning*.

Jordan, Michael I., Ghahramani, Zoubin, Jaakkola, Tommi S. and Saul, Lawrence K. (1999). An Introduction to Variational Methods for Graphical Models. *Machine Learning* 37: 183-233.

Jordan, Michael I. (preprint). An Introduction to Probabilistic Graphical Models.

Kapoor, Ashish, Qi, Yuan (Alan), Ahn, Hyungil and Picard, Rosalind W. (2005). Hyperparameter and Kernel Learning for Graph Based Semi-Supervised Classification. *Neural Information Processing Systems*.

Kim, Hyun-Chul and Ghahramani, Zoubin (2004). The EM-EP Algorithm for Gaussian Process Classification. *European Conference on Machine Learning*.

Kleinberg, Jon and Tardos, Eva (1999). Approximation Algorithms for Classification Problems with Pairwise Relationships: Metric Labeling and Markov Random Fields. *Symposium on Foundations of Computer Science*.

Kohonen, Teuvo (2000). Self-Organizing Maps. Springer.

Kondor, Risi Imre and Lafferty, John (2002). Diffusion Kernels on Graphs and Other Discrete Input Spaces. *International Conference on Machine Learning*.

Kwok, James T. and Tsang, Ivor W. (2003). Learning with Idealized Kernels. *International Conference on Machine Learning*.

Lanckriet, Gert R. G., Cristianini, Nello, Bartlett, Peter L., Ghaoui, Laurent El and Jordan, Michael I. (2004). Learning the Kernel Matrix with Semidefinite Programming. *Journal of Machine Learning Research* 5: 27-72.

Lang, Kevin J. (1995). NewsWeeder: Learning to Filter Netnews. *International Conference on Machine Learning*.

Lee, Wee Sun and Liu, Bing (2003). Learning from Positive and Unlabeled Examples Using Weighted Logistic Regression. *International Conference on Machine Learning*.

Levin, Anat, Lischinski, Dani and Weiss, Yair (2004). Colorization using Optimization. *ACM Transactions on Graphics* 23 (3): 689-694.

Li, Wei and McCallum, Andrew (2004). A Note on Semi-Supervised Learning using Markov Random Fields. Technical Report.

Mackay, David J.C. Introduction to Monte Carlo Methods. Tutorial.

Madani, Omid, Pennock, David M. and Flake, Gary W. (2004). Co-Validation: Using Model Disagreement to Validate Classification Algorithms. *Neural Information Processing Systems*.

Maeireizo, Beatriz, Litman, Diane and Hwa, Rebecca (2004). Co-training for Predicting Emotions with Spoken Dialogue Data. *Annual Meeting of the Association for Computational Linguistics*.

McCallum, Andrew and Nigam, Kamal Paul (1998). A Comparison of Event Models for

Naive Bayes Text Classification. *AAAI-98 Workshop on Learning for Text Categorization*.

McEliece, Robert J., MacKay, David J. C. and Cheng, Jung-Fu (1998). Turbo Decoding as an Instance of Pearl's `Belief Propagation' Algorithm. *IEEE Journal on Selected Areas in Communication* 16(2): 140-152.

Meilă, Marina and Shi, Jianbo (2001). A Random Walks View of Spectral Segmentation. *AI and Statistics*.

Mihalcea, Rada and Chklovski, Timothy (2003). OPEN MIND WORD EXPERT: Creating Large Annotated Data Collections with Web Users' Help. *EACL 2003 Workshop on Linguistically Annotated Corpora*.

Miller, David J. and Uyar, Hasan S. (1996). A Mixture of Experts Classifier with Learning Based on Both Labelled and Unlabelled Data. *Neural Information Processing Systems*.

Minka, Thomas P. (2001a). A Family of Algorithms for Approximate Bayesian Inference. PhD Thesis, MIT.

Minka, Thomas P. (2001b). Expectation Propagation for Approximate Bayesian Inference. *Conference on Uncertainty in AI*.

Mitchell, Tom M (1999). The Role of Unlabeled Data in Supervised Learning. *International Colloquium on Cognitive Science*.

Murphy, Kevin P., Weiss, Yair and Jordan, Michael I. (1999). Loopy Belief Propagation for Approximate Inference: an Empirical Study. *Conference on Uncertainty in AI*.

Muslea, Ion, Minton, Steve and Knoblock, Craig (2000). Selective Sampling with Redundant Views. *National Conference on Artificial Intelligence*.

Neal, Radford M. (1993). Probabilistic Inference Using Markov Chain Monte Carlo Methods. Tutorial.

Newman, D.J., Hettich, S., Blake, C.L. and Merz, C.J. (1998). UCI Repository of Machine Learning Databases. http://www.ics.uci.edu/~mlearn/MLRepository.html Thesis, University of California, Irvine.

Ng, Andrew Y., Jordan, Michael I. and Weiss, Yair (2001a). On Spectral Clustering: Analysis and an Algorithm. *Neural Information Processing Systems*.

Ng, Andrew Y., Zheng, Alice X. and Jordan, Michael I. (2001b). Link Analysis, Eigenvectors and Stability. *International Joint Conference on Artificial Intelligence*.

Nigam, Kamal, McCallum, Andrew, Thrun, Sebastian and Mitchell, Tom (1999). Text Classification from Labeled and Unlabeled Documents using EM. *Machine Learning* 39(2/3): 103-134.

Nigam, Kamal Paul and Ghani, Rayid (2000). Analyzing the Effectiveness and Applicability of Co-training. *International Conference on Information and Knowledge Management*.

Nigam, Kamal Paul (2001). Using Unlabeled Data to Improve Text Classification. PhD The-

sis, CMU.

Niu, Zheng Yu, Ji, Dong Hong and Tan, Chew Lim (2005). Word Sense Disambiguation Using Label Propagation Based Semi-Supervised Learning. *Annual Meeting of the Association for Computational Linguistics*.

Ong, Cheng Soon and Smola, Alex (2003). Machine Learning using Hyperkernels. *International Conference on Machine Learning*.

Ong, Cheng Soon, Smola, Alexander J. and Williamson, Robert C. (2005). Learning the Kernel with Hyperkernels. *Journal of Machine Learning Research* 6: 1043-1071.

Opper, Manfred and Winther, Ole (1998). Mean field Methods for Classification with Gaussian Processes. *Neural Information Processing Systems*.

Platt, John (1998). Fast Training of Support Vector Machines using Sequential Minimal Optimization. Advances in Kernel Methods - Support Vector Learning. B. Schölkopf, C. Burges and A. J. Smola. MIT Press.

Ratsaby, Joel and Venkatesh, Santosh, S. (1995). Learning From a Mixture of Labeled and Unlabeled Examples with Parametric Side Information. *Conference on Computational Learning Theory*.

Riloff, Ellen and Jones, Rosie (1999). Learning Dictionaries for Information Extraction by Multi-Level Bootstrapping. *National Conference on Artificial Intelligence*.

Riloff, Ellen, Wiebe, Janyce and Wilson, Theresa (2003). Learning Subjective Nouns using Extraction Pattern Bootstrapping. *Conference on Natural Language Learning*.

Roweis, Sam and Saul, Lawrence K. (2000). Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science* 290(5500): 2323-2326.

Schölkopf, Bernhard and Smola, Alexander J. (2001). Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. The MIT Press.

Schuurmans, Dale (1997). A New Metric-based Approach to Model Selection. *National Conference on Artificial Intelligence*.

Schuurmans, Dale and Southey, Finnegan (2000). An Adaptive Regularization Criterion for Supervised Learning. *International Conference on Machine Learning*.

Seeger, Matthias (2000). Learning with Labeled and Unlabeled Data. Technical Report.

Seung, H., Opper, M. and Sompolinsky, H. (1992). Query by Committee. *Annual Workshop on Computational Learning Theory*.

Shahshahani, Behzad M. and Landgrebe, David A. (1994). The Effect of Unlabeled Samples in Reducing the Small Sample Size Problem and Mitigating the Hughes Phenomenon. *IEEE Transactions on Geoscience and Remote Sensing* 18(7): 763-767.

Shi, Jianbo and Malik, Jitendra (2000). Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22(8): 888-905.

Smola, Alex and Schölkopf, Bernhard (2003). A Tutorial on Support Vector Regression. Tutorial.

Smola, Alexander J. and Kondor, Risi Imre (2003). Kernels and Regularization on Graphs. *Conference on Computational Learning Theory*.

Spielman, Daniel A. and Teng, Shang-Hua (1996). Spectral Partitioning Works: Planar Graphs and Finite Element Meshes. *IEEE Symposium on Foundations of Computer Science*.

Stewart, G. W. and Sun, Ji-guang (1990). Matrix Perturbation Theory. Academic Press.

Szummer, Martin and Jaakkola, Tommi (2001). Partially Labeled Classification with Markov Random Walks. *Neural Information Processing Systems*.

Szummer, Martin and Jaakkola, Tommi (2002). Information Regularization with Partially Labeled Data. *Neural Information Processing Systems*.

Tenenbaum, Joshua B., de Silva, Vin and Langford, John C. (2000). A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science* 290(5500): 2319-2323.

Tikhonov, A. N. (1963). On Solving Incorrectly Posed Problems and Method of Regularization. *Doklady Akademmi Nauk USSR* 151: 501-504.

van Rijsbergen, Keith C. J. (1977). A Theoretical Basis for the Use of Co-occurrence Data in Information Retrieval. *Journal of Documentation* 33(2): 106-119.

Vapnik, Vladimir N. (1998). Statistical Learning Theory. Wiley-Interscience.

Vapnik, Vladimir N. (1999). The Nature of Statistical Learning Theory. Springer.

Wainwright, Martin J. and Jordan, Michael I. (2003). Graphical models, exponential families, and variational inference. Technical Report.

Wainwright, Martin J. and Jordan, Michael I. (2005). A Variational Principle for Graphical Models. New Directions in Statistical Signal Processing. S. Haykin, J. Principe, T. Sejnowski and J. McWhirter. MIT Press. 21-93.

Watanabe, Satoshi (1969). Knowing and Guessing; a Quantitative Study of Inference and Information. New York, John Wiley and Sons.

Weinberger, Kilian Q. and Saul, Lawrence K. (2004). Unsupervised Learning of Image Manifolds by Semidefinite Programming. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Weinberger, Kilian Q., Sha, Fei and Saul, Lawrence K. (2004). Learning a Kernel Matrix for Nonlinear Dimensionality Reduction. *International conference on Machine learning*.

Weinberger, Kilian Q., Packer, Benjamin D. and Saul, Lawrence K. (2005). Nonlinear Dimensionality Reduction by Semidefinite Programming and Kernel Matrix Factorization. *AI & Statistics*.

Weiss, Yair (1999). Segmentation using Eigenvectors: a Unifying View. *International Con-*

ference on Computer Vision.

Weston, Jason, Perez-Cruz, Fernando, Bousquet, Olivier, Chapelle, Olivier, Elisseeff, Andre and Schölkopf, Bernhard (2002). Feature Selection and Transduction for Prediction of Molecular Bioactivity for Drug Design.

Yarowsky, David (1995). Unsupervised Word Sense Disambiguation Rivaling Supervised methods. *Annual Meeting of the Association for Computational Linguistics*.

Yedidia, Jonathan S., Freeman, William T. and Weiss, Yair (2005). Approximations and Generalized Belief Propagation Algorithms. *IEEE Transactions on Information Theory* 51(7): 2282-2312.

Yu, Stella X., Gross, Ralph and Shi, Jianbo (2002). Concurrent Object Recognition and Segmentation by Graph Partitioning. *Neural Information Processing Systems*.

Zhang, Tong and Oles, Frank J. (2000). A Probability Analysis on the Value of Unlabeled Data for Classification Problems. *International Conference on Machine Learning*.

Zhou, Dengyong, Bousquet, Olivier, Lal, Thomas Navin, Weston, Jason and Schölkopf, Bernhard (2003). Learning with Local and Global Consistency. *Neural Information Processing Systems*.

Zhu, Xiaojin, Ghahramani, Zoubin and Lafferty, John (2003a). Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. *International Conference on Machine Learning*.

Zhu, Xiaojin, Lafferty, John and Ghahramani, Zoubin (2003b). Semi-Supervised Learning: From Gaussian Fields to Gaussian Processes. Technical Report.

Zhu, Xiaojin, Lafferty, John and Ghahramani, Zoubin (2003c). Combining Active Learning and Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. *International Conference on Machine Learning Workshop*.

Zhu, Xiaojin, Kandola, Jaz, Ghahramani, Zoubin and Lafferty, John (2004). Nonparametric Transforms of Graph Kernels for Semi-Supervised Learning. *Neural Information Processing Systems*.

Zhu, Xiaojin (2005). Semi-Supervised Learning with Graphs. PhD Thesis, CMU.

Zhu, Xiaojin and Lafferty, John (2005). Harmonic mixtures: combining mixture models and graph-based methods for inductive and scalable semi-supervised learning. *International Conference on Machine Learning*.

Appendix A Dataset Description and Pre-processing

In this appendix, we describe the datasets used in experiments, their source, property and how they are pre-processed. As we said before, we were unable to exactly reproduce the pre-processing according to significant difference in test accuracy. Now we will pay attention to two questions:

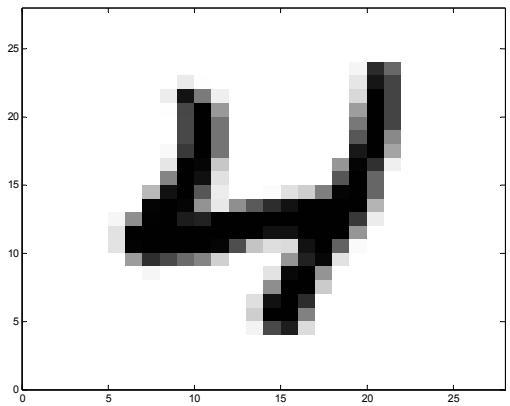
1. How to produce the *probe form* of the datasets by introducing useless features?
2. What should be the correct *original form* corresponding to the probe form, i.e., by just removing the useless features while the other pre-processing steps like helpful feature selection are still applied?

Some descriptions in this appendix are directly cited from the technical memorandum of data preparation for NIPS 2003 Workshop on feature selection, which was in turn partly copied from other sources. *We will put all such copied descriptions in quotation mark*, and describe the pre-processing in a way catering for the experiments in this thesis. The NIPS workshop technical memorandum is downloadable from: http://clopinet.com/isabelle /Prjects/NIPS2003/Slides/NIPS2003-Datasets.pdf.

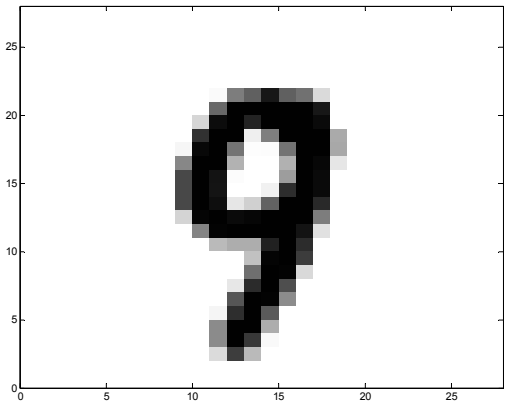
A.1. Handwritten digits discrimination: 4 vs 9

The original source of this dataset is MNIST, which is constructed from NIST database. MNIST is downloadable from http://yann.lecun.com/exdb/mnist/, and it has been very extensively used as a benchmark dataset. "MNIST size-normalized and centred digits in a

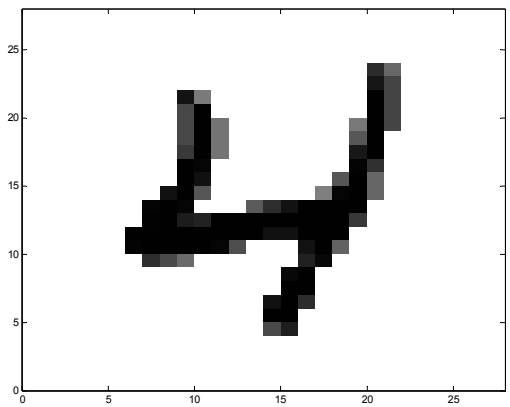
fixed size image of dimension 28×28", with each pixel having 256 grey levels. On average, the grey level (feature value) of about 87% pixels is 0. Though MNIST covers digits from 0 to 9, we are only concerned with distinguishing between 4 and 9. The following Figure A.1 (a) and (b) are two example images from MNIST.



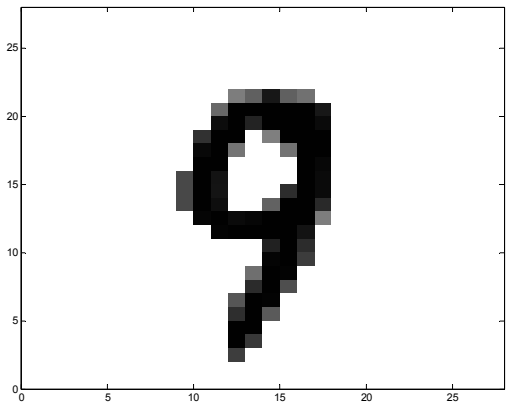
(a) example of 4 in MNIST



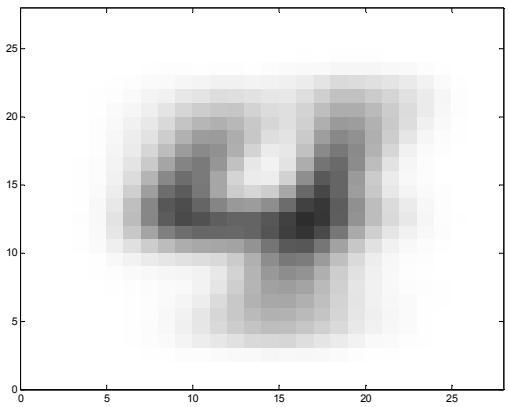
(b) example of 9 in MNIST



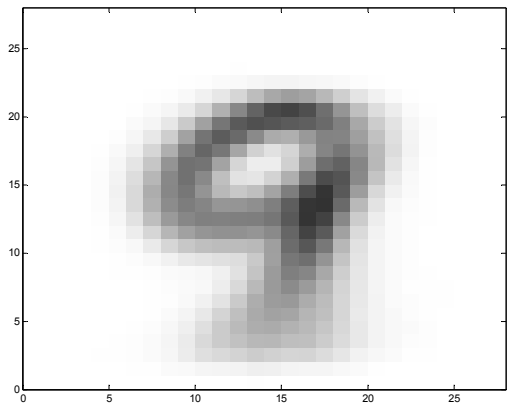
(c) example of 4 after thresholding



(d) example of 9 after thresholding



(e) average of 4 in the whole MNIST



(f) average of 9 in the whole MNIST

Figure A.1 Image examples of handwritten digit recognition

The size of the original dataset is very large. Combining training and testing set, there are 70,000 examples in all for the 10 classes. Each example is a vector with 784 components, varying from 0 to 255 in discrete values. So storing them in a binary file using 32-bit integer will cost over 400 MB. The data file was compressed in a special format to 54 MB. We used the Matlab program written by Bryan Russell and Kevin Murphy to parse the compressed file: http://www.cs.ubc.ca/~murphyk/Software/readMNIST.m. We randomly extracted 1000 examples for '4' and '9' each, which formed *digitOri.dat*[1]. We only applied five simple preprocess steps here as detailed in *genDigitOri.cpp*. Suppose the columns are features and the rows are examples.

1. Among the 784 original features, eliminate all the features which have only zero values or constant values;
2. Standardize by linearly transforming each feature (each column) so that the values in each column would be in the range $[0,1]$;
3. "Threshold values below 0.5 to increase data sparsity" (after this step the examples of Figure A.1 (a) and (b) become (c) and (d) respectively);
4. Discard all the features which appear (nonzero) for less than 3 times;
5. Normalize each example (in each row) so that the length of feature vector is 1 for all examples.

Now we get the so-called "*original version*" of the data in the experiment.

[1] As we redid the pre-processing for 4 vs 9 dataset, we put the processing code and resulting dataset online and we mention the file names in this section.

To produce the probe version (stored in file *digitProbe.dat*), we further applied the following steps, detailed in source code *genDigitProbe.cpp*.

6. "Of the remaining features, complement them by constructing a new feature set as follows to attain the number of 2500 features". Each new feature is built upon two randomly picked original features f_1 and f_2. Then for each example, its new feature value is defined as the product of this example's f_1 and f_2 value. "The pairs were sampled such that each pair member is normally distributed in a region of the image slightly biased upwards. The rationale beyond this choice is that pixels that are discriminative of the "4/9" separation are more likely to fall in that region."

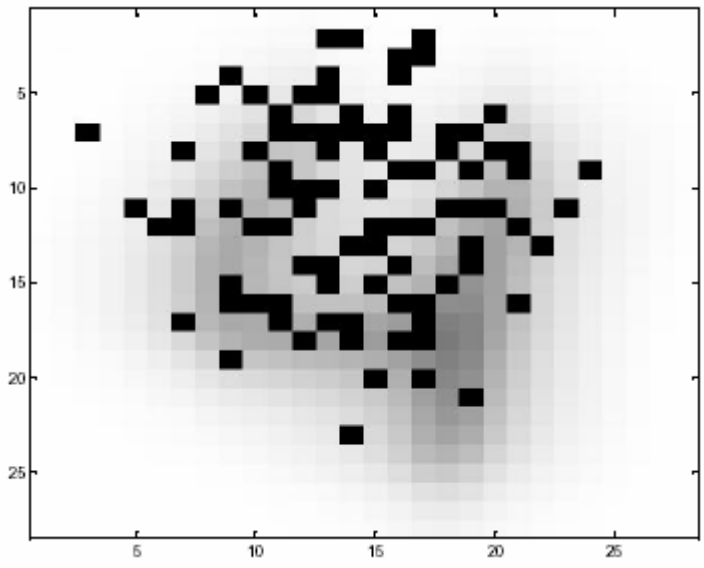


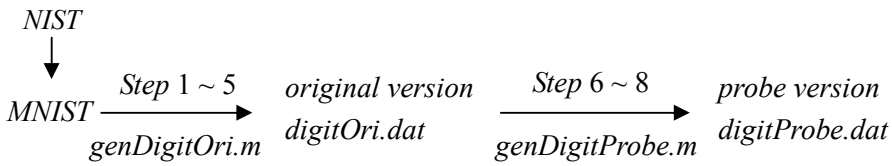
Figure A.2 Probe feature sampling example for handwritten digit recognition

7. "Another 2500 pairs were used to construct "probes"". But in addition to the procedures in 6, we permute the new features' value among all examples. In this way we obtained probes that are similarly distributed to the other features.

8. "Quantize the data to 1000 levels".

Now we get the *probe version* of the dataset.

The whole process can be summarized as:



Originally, there are $28 \times 28 = 784$ features in MNIST. After the steps 1 to 5, there are only 475 non-zero features. Then we added additional features (product of pairs) by step 6, which finally makes 2500 features. Furthermore, step 7 added another 2500 features. We also calculated the sparsity of the datasets, using the following formula:

$$sparsity \triangleq \frac{\sum_i number\ of\ \ zero\ valued\ \ features\ in\ example\ i}{\left(total\ number\ of\ \ features\ in\ whole\ dataset\right) \times \left(number\ of\ \ examples\right)}.$$

Then the sparsity of original and probe version is 79.7% and 85.8% respectively. Probe version is sparser than original version because the product of two numbers is zero as long as one number is zero.

A.2. Cancer vs normal

This task is to distinguish cancer versus normal patterns from mass-spectrometric data. It is a two-class classification problem with *continuous* input variables. The original dataset is from three sources:

	download URL	No. of spectra	No. of cancer spectra	No. of control spectra	No. of features
NCI ovarian data	http://home.ccr.cancer.gov/ncifdaproteomics/ppatterns.asp	253	162	91	15154 continuously valued
NCI prostate cancer data	http://home.ccr.cancer.gov/ncifdaproteomics/ppatterns.asp	322	69	253	15154 continuously valued
EVMS prostate cancer data	http://www.evms.edu/vpc/seldi/	326	167	159	48538 continuously valued

Table A.1 Cancer dataset summary

Merging the three sources, we obtain a dataset of mass-spectra obtained with the SELDI technique. The samples include patients with cancer (ovarian or prostate cancer), and healthy (or control) patients. The total number of examples is 253+322+326 = 901, among which 162+69+167 = 398 examples are cancer samples (positive class) and 91+253+159 = 503 are control samples (negative class). It is interesting to put the two kinds of cancer together. Due to lack of domain knowledge, we did not redo the pre-processing and we describe how the probe version was produced by the NIPS workshop. All examples were merged at first, by adding 48538 and 15154 zero-valued features to NCI and EVMS examples respectively. Then 9 simple pre-processing steps were applied. Suppose the columns

are features and the rows are examples (the 9 steps below are all cited from NIPS workshop).

1. "Limit the mass range. Eliminate small masses under m/z=200 that usually include chemical noise specific to the MALDI/SELDI process (influence of the "matrix"). Also eliminate large masses over m/z = 10000 because few features are usually relevant in that domain and we needed to compress the data.
2. Average the technical repeats. In the EVMS data, two technical repeats were available. They were averaged in order to make examples in the test set independent so that simple statistical tests could be applied.
3. Remove the baseline. Subtract in a window the median of the 20% smallest values.
4. Smoothing. The spectra were slightly smoothed with an exponential kernel in a window of size 9.
5. Re-scaling. The spectra were divided by the median of the 5% top values.
6. Take the square root of all feature values.
7. Align the spectra. Slightly shift the spectra collections of the three datasets so that the peaks of the average spectrum would be better aligned. As a result, the mass-over-charge (m/z) values that identify the features in the aligned data were imprecise. The NCI prostate cancer m/z was taken as reference.
8. Soft thresholding the values. After examining the distribution of values in the data matrix, we subtracted a threshold and equaled to zero all the resulting values that were negative. In this way, only about 50% of non-zero values were kept, which represented significant data compression.
9. Quantization. Quantize the values to 1000 levels."

Now we get the so-called "*original version*" of the data in the experiment.

To produce the probe version, we apply the following processing:

10. "Identify the region of the spectra with least information content using an interval search for the region that gave worst prediction performance of a linear SVM (indices 2250-4750). The features in that region were replaced by "random probes" obtained by randomly permuting the values in the columns of the data matrix.
11. Identify another region of low information content: 6500-7000, and then add 500 random probes that are permutations of those features."

Now we get the *probe version* of the dataset.

A.3. Reuters text categorization: "corporate acquisitions" or not

This task is to classify whether a text is about "corporate acquisitions" or not. This is a two-class classification problem with sparse continuous input variables.

The original source of the dataset is the well-known Reuters text categorization benchmark, hosted at http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html. Another valuable source about this data is http://www.daviddlewis.com/resources/testcollections/reuters21578/. We used a subset of it: the "corporate acquisition" text classification class pre-processed by Thorsten Joachims. As one of the examples of the software SVM^{light}, it is downloadable

from http://download.joachims.org/svm_light/examples/example1.tar.gz.

"The data formatted by Joachims is in the "bag-of-words" representation. There are 9947 features (of which 2562 are always zero for all the examples) that represent frequencies of occurrence of word stems in text. Some normalization has been applied that are not detailed by Joachims in his documentation." We randomly extracted 2000 examples (class balanced) which formed the *original version*. The only pre-processing applied on top of Joachims' data is to discard all the features which appear (nonzero) for less than 3 times.

"The frequency of appearance of words in text is known to follow approximately Zipf's law (for details, see e.g. http://www.nslij-genetics.org/wli/zipf/). According to that law, the frequency of occurrence of words, as a function of the rank k when the rank is determined by the frequency of occurrence, is a power-law function $P_k \sim k^{-a}$ with the exponent a close to 1. The estimation $a = 0.9$ gives us a reasonable approximation of the distribution of the data."

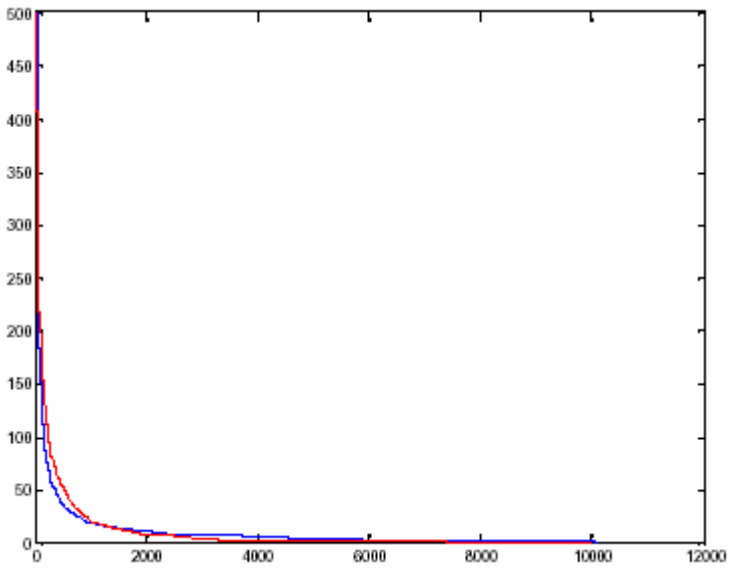


Figure A.3 Comparison of the real data and the random probe data distributions

Figure A.3 plots the number of non-zero values of a given feature as a function of the rank of the feature. The rank is given by the number of non-zero features. Red: real data. Blue: simulated data.

To produce the probe version, the following pre-processing steps were applied:

1. "Add to the original 9947 features, 10053 features drawn at random according to Zipf's law, to obtain a total of 20000 features. Fraction of non-zero values in the real data is 0.46% while fraction of non-zero values in the simulated data: 0.5%.
2. The feature values were quantized to 1000 levels."

Now we get the *probe version* of the dataset.

A.4. Compounds binding to Thrombin

This task is to predict which compounds bind to thrombin. It is a two-class classification problem with sparse binary input variables.

"Drugs are typically small organic molecules that achieve their desired activity by binding to a target site on a receptor. The first step in the discovery of a new drug is usually to identify and isolate the receptor to which it should bind, followed by testing many small molecules for their ability to bind to the target site. This leaves researchers with the task of determining what separates the active (binding) compounds from the inactive (nonbinding) ones. Such a determination can then be used in the design of new compounds that not only bind, but also

have all the other properties required for a drug (solubility, oral absorption, lack of side effects, appropriate duration of action, toxicity, etc.). Biological activity in general, and receptor binding affinity in particular, correlate with various structural and physical properties of small organic molecules. In this task, we are concerned with compounds' ability to bind to a target site on thrombin, a key receptor in blood clotting.

The original combined dataset consisted of 2543 compounds, of which 192 are active (bind well) and the others (2351) are inactive. To simulate the real-world drug design environment, the test set contained 634 compounds that were in fact generated based on the assay results recorded for the training set (1909 compounds). Of the test compounds, 150 bind well and the others are inactive. The compounds in the test set were made after chemists saw the activity results for the training set, so the test set had a higher fraction of actives than did the training set in the original data split. In evaluating the accuracy, a differential cost model was used, so that the sum of the costs of the actives will be equal to the sum of the costs of the inactive.

Each compound is described by a single feature vector comprised of 139,351 binary features, which describe three-dimensional properties of the molecule. The chemical structures of these compounds are not necessary for our analysis. On average, less than 1% feature values are non-zero. The original source of the data is from KDD Cup 2001: http://www.cs.wisc.edu/~dpage/kddcup2001."

All the examples from the original source KDD Cup were used and 5 pre-processing steps were applied.

1. To capitalize on the differences in the distribution between the training and test dataset, a technique in (Weston et al., 2002) was adopted. For all features f_j, compute a score

$$f_j = \sum_{i:y_i=1} X_{ij} - \lambda \sum_{i:y_i=-1} X_{ij}$$

where the matrix X is the *testing* dataset, with columns being features and rows being examples. "The features were sorted according to the f_j criterion with $\lambda = 3$, computed using the original *test* set, which is richer in positive (active) examples."

2. "Only the top ranking 100,000 original features were kept."
3. Discard all the remaining features which appear (nonzero) for less than 3 times.
4. "The all-zero patterns were removed, except one that was given label –1 (inactive)."

Now we get the so-called "*original version*" of the data in the experiment.

To produce the probe version, the following pre-processing step was applied:

5. "For the second half lowest ranked features, the order of the patterns was individually randomly permuted, in order to create "random probes". So now half of the features are real original features while the other half are probe features."

Now we get the *probe version* of the dataset.

The following table presents the class distribution and range of feature values (fea. val) before pre-processing and after steps 1 to 5. l is the number of labelled examples and α = 0.097. Note after pre-processing, there are 1950 example in all, 593 less than original due to step 4.

	before pre-processing		after pre-processing				
	# in training set	# in test set	# in training set	# in test set	max fea. val	min fea. val	median fea. val
positive examples	42	150	$l \times \alpha$	$(1950 - l) \times \alpha$	687	11475	846
negative examples	1867	484	$l \times (1 - \alpha)$	$(1950 - l) \times (1 - \alpha)$	653	3185	783

Table A.2 Compound binding dataset summary

A.5. Ionosphere

This dataset was downloaded from UCI repository (Newman et al., 1998). We did no pre-processing on the dataset, except normalizing the feature vector of each example to have length 1.

Appendix B Details of Experiment Settings

In this appendix, we present miscellaneous details of experiments, including algorithm implementation, dataset choosing, cross validation, and experimental settings.

B.1. Toolboxes used for learning algorithm implementation

All the 12 algorithms were implemented by the author in C++, mixing two toolboxes:

1. Matlab C++ Math library for four matrix operations: matrix multiplication, matrix power, matrix inversion and matrix eigen-decomposition. Matlab offers almost the most efficient implementation of these operations. In particular, it has reduced the cost of full matrix inversion to be far below $O(u^3)$. The detail of its implementation is not released by Mathworks.
2. Toolkit for Advanced Optimization (TAO) by (Benson et al., 2004) for gradient descent optimization in all the algorithms compared except HEM. This package implements three variations of conjugate gradient methods: the Fletcher-Reeves method, the Polak-Ribiére method, and Polak-Ribiére-Plus method. It also implemented a limited memory variable metric method (*lmvm*) for unconstrained nonlinear optimization that requires only function value and gradient information. It keeps a limited history of previous points and previous gradients to approximate the second-order information. We use *lmvm* for all optimizations.

B.2. Dataset size choice

The dataset size was chosen carefully. For convenience we copied Table 5.2 here.

	feature property: continuous or discrete (how many levels), sparsity	number of features	*dataset* size (#positive: #negative: #unlabelled)[1]
handwritten digits (4 vs 9)	Original: 256 levels, 79.7 % Probe: 1000 levels, 85.8 %	Original: 475 Probe: 4525	1000 : 1000 : 0
cancer vs normal	Original: continuous, 50.3 % Probe: 1000 levels, 45.4 %	Original: 18543 Probe: 9961[2]	88 : 112 : 0
Reuters text categorization	Original: continuous, 99.0 % Probe: 1000 levels, 99.3 %	Original: 4608 Probe: 13459	300 : 300 : 1400
compounds bind to thrombin	Original: binary, 99.43 % Probe: binary, 98.55 %	Original: 31976 Probe: 58417	112 : 1038 : 0
Ionosphere	continuous, 38.3 %	33	225 : 126: 0

The objective is to make the dataset size as large as possible, under the constraints of data availability and computational and spatial complexity. Since the matrix inversion takes a lot of time, we upper limit the dataset size to 2000 as in handwritten digits recognition and Reuters text categorization. The UCI repository only has 351 instances in ionosphere and we have already used them all. The dataset of compounds binding to thrombin has a huge

[1] This is common to both original and probe forms.
[2] There are more features in original form than in probe form, because the feature selection process in the NIPS workshop is beyond our domain knowledge and thus only the very original dataset was used.

number of features in the probe form. Since the probe form is directly downloaded from web site, we do not know the detailed meanings of each number. So we keep all features. That makes the spatial cost huge: over 2.2 GB memory used for 1200 examples in our LOOHL implementation. So we stop at 1200. The number of features in cancer (origin) is only around 1/1.6 of thrombin (origin). However, the cancer dataset has a significantly lower sparsity (50.3/45.4 vs 99.43/98.55). So we used a small dataset size.

B.3. Cross validation complexity analysis

The unit of cross validation complexity analysis is the process denoted by the shorthand:

$$Score(X^l, Y^l, X_{te}, Y_{te}, X^u, A, \lambda)$$

For example, in the HEM algorithm, A stands for HEM and λ stands for the coefficients such as the bandwidth of the similarity measure. Now suppose we have 10 labelled articles $(X^l, Y^l) = (X_{1...10}, Y_{1...10})$ and 80 unlabelled articles $X^u = (X_{11...90})$. Besides we are given other 10 testing articles $X_{te} = (X_{91...100})$ and we wish to compare the prediction of the algorithm against the correct labels of these 10 testing articles $Y_{te} = (Y_{91...100})$. As the HEM algorithm is transductive, we can get the model's labelling on X_{te} directly by learning from the 100 articles in whole. However, we can also use A = *TSVM*, λ = {type of kernel, RBF kernel bandwidth, C (trade-off factor between loss and margin), fraction of unlabelled examples to be classified into the positive class}. Different from HEM, TSVM can produce an inductive model, i.e., we can get a model M: $f(x) = \sum_i \alpha_i y_i k(x_i, x)$ by learning from X^l, Y^l, X_{te}, X^u. Then we simply feed X_{te} to M and compare the prediction against Y_{te}. By using the notation $Score(X^l, Y^l, X_{te}, Y_{te}, X^u, A, \lambda)$, we can treat transductive and

inductive semi-supervised learning algorithms in the same way, without worrying about whether there is a intermediate (inductive) model *M*.

The *k*-fold cross validation in Table 5.4 is fair for comparing different algorithms, as long as the discretization of model coefficients λ is done fairly. The problem with this scheme is the expensive computational complexity. Suppose each call of $Score(\cdot)$ costs unit time, which is about $U = 60$ minutes for our leave-one-out hyperparameter learning algorithm on the handwritten digit recognition dataset with $n = 2000$. Suppose the test is run for r times and then average is finally reported. When are c coefficients of the model, each discretized into v levels, the complexity of the above scheme is $O(rkv^c)$. Furthermore, we suppose that we want to test on d datasets, and vary the number of labelled data points by l different values. So the total number of *days* required for computing on P CPUs is: $D = \frac{d \cdot l \cdot r \cdot k \cdot v^c \cdot U}{60 \times 24 \times P}$. Generally speaking, letting $d = 6$, $l = 3$, $r = 10$, $k = 5$, $v = 5$, $c = 2$, $P = 40$, we get $D = 23.4378$ days.

Moreover, we have 4 regularizers for LOOHL so 93.8 days will be needed. Besides, the computing cluster administrator normally does not allow us to use the facility with such intensity. So in practice, we only applied the above *k*-fold cross validation to HEM, and used some fixed rules or fixed coefficients for LOOHL across all datasets to achieve both fairness and efficiency.

B.4. Miscellaneous settings for experiments

In this section, we put together the details of all the experiment settings. Miscellaneous as

it is, we will try to organize them as clearly as possible. The experiments are random in nature. However in order to make the experiments totally reproducible, we have stored all the source programs (both for pre-processing and for learning algorithm implementation), data files (raw data with and without probe features), main control files (scripts), and seed of random number generator. All these files and seeds are available at http://www.comp.nus.edu.sg/~zhangxi2/thesis.html. A short manual is also written to accompany the programs. The raw results and the performance under different coefficients during cross validation model selection are also available on the web site.

The experiments were divided into two parts of comparisons:

1. Compare the performance of HEM, MinEnt, and LOOHL+sqr, each with both threshold form and CMN form, on the same $4 \times 2 + 1 = 9$ datasets by using 5 fold cross validation as model selection, or by fixing the value of coefficients across all datasets;
2. Compare the four regularizers of LOOHL on the same 9 datasets by enumerating the accuracy under a number of coefficient values without model selection.

We will refer to these two parts as part 1 and part 2.

1. Data pre-processing. The details of data pre-processing is in the Appendix A. We just emphasize that all the input feature vectors were normalized to have length 1. Before that, all features which appear (non-zero) in less than three instances in the whole dataset were filtered out. This will help compress data because in many datasets, especially text dataset according to Zipf's law, such infrequently appearing features constitute a large bulk of the total features. This pruning also helps to avoid overfitting,

similar to avoiding such rules as (birth date → score) in decision tree learning.

2. Model selection. In part 1, we used 5 fold cross validation to select the model for HEM. For MinEnt, we tested the accuracy on all the discretized coefficients and then report the highest accuracy without cross validation, which is an unfair advantage for MinEnt. The only exceptions are *cancer* and *ionosphere*, for which we used 5 fold cross validation as well since the computational cost is mild on these two datasets.

   For LOOHL+Sqr, we chose the prior mean of the bandwidth in the regularizer as the bandwidth selected by HEM+Thrd via cross validation. This value also served as the initial value for iterative optimization in LOOHL+Sqr. Once it caused numerical problems in optimization, e.g., line search failure, not-a-number or infinity due to inverting a matrix close to singular, etc., we just increased the bandwidth (both the prior mean and the initial value) by 0.05, until the graph was connected strongly enough and there was no numerical problem in the optimization process. All other coefficients were fixed across all datasets as detailed in point 4 below. In practice, we found that the resulting accuracy was always reasonable when the optimization proceeded smoothly, i.e., no numerical problems occurred.

In part 2, we present the performance by enumerating all the coefficients on a chosen grid, without doing model selection via cross validation. This is because the LOOHL+ regularizers is very time consuming and requires a lot of computing recourses. We have already shown the performance of regularizer Sqr in part 1, by fixing all the coeffi-

cient values and thus on a fair basis of comparison. So in part 2, we aim at showing the promise of the other three regularizers, and it will also provide some insight into the good choices of coefficients.

3. Cross validation was applied only in part 1 for HEM (all datasets) and MinEnt (cancer and ionosphere). We used 5 fold cross validation. As the input vectors were all normalized, we could fix the grid/discretization for candidate models in HEM across all datasets. The candidate σ's are 0.05, 0.1, 0.15, 0.2, 0.25. MinEnt also used these five σ's as initial values for gradient descent, while its candidate ε's are 0.01, 0.05, 0.1, 0.2, 0.3. We ran some initial experiments and found that the models picked for HEM and MinEnt are almost always in the interior of this grid and the step is refined enough.

4. Fixed coefficients. For all LOOHL algorithms in both parts, we fixed ε to 0.01. The LOO loss transformation function was fixed to the polynomial form with exponent 3 (ref. Figure 3.2), and the eigenvalue penalty exponent was fixed to 3 as well (ref. (4.1)). The prior q in CMN (2.7) was always set to the class ratio in training set, except for the thrombin dataset as detailed in the below point 5.

   In part 1, if the objective is written as C_1 * *LOO_loss* + C_2 * *regularizer*, then we fixed C_1:C_2 to 100:1 for LOOHL+Sqr across all datasets. In part 2, we will show the influence of different C_1:C_2 and different bandwidth σ under the other three regularizers. C_1:C_2 are chosen to be 1:10, 1:100, and 1:1000 for all the datasets on regularizers Eigen

and REnt. This is because the value of these two regularizers is bounded by a constant. However, as the first-hit time is not bounded, we carefully chose some candidate C_1:C_2 values for Step regularizer so that the optimization could proceed properly. For 4vs9, cancer and ionosphere, we chose 1:10, 1:100, 1:1000. For text and thrombin, we chose 1:1000, $1:10^4$, $1:10^5$. The absolute values of C_1 and C_2 will be discussed in point 6.

5. Performance measure. We used accuracy as performance measure on all datasets except thrombin binding, where the two classes are very unbalanced at about 1:10. In the NIPS workshop, the balanced error rate (*BER*) was used: $BER \triangleq \frac{1}{2}\left(\frac{b}{a+b}+\frac{c}{c+d}\right)$, where

number of examples in test set		prediction	
		positive	negative
truth	positive r_+	a	b
	negative r_-	c	d

This is equivalent to super-sampling the truly positive examples by (c+d) times, and super-sampling the truly negative examples by (a+b) times. We used $1-BER$ as our balanced accuracy for thrombin binding dataset. The whole learning algorithm of HEM and LOOHL did not change. However, we gave some advantage to HEM+CMN for thrombin dataset, by allowing it to know the *correct* class distribution in the test set. Suppose in labelled data, there are r_+ positive instances and r_- negative instances. Then previously we set $q_1 = r_+/(r_+ + r_-)$. Now given that we know there are

s_+ positive instances and s_- negative instances in the *test* set, we set $q_2 = s_+ / (s_+ + s_-)$, the correct proportion. However, the weighted cost elicited some other considerations. If $s_+ : s_- = 1:10$, then by the definition of *BER*, the loss of misclassifying a truly positive point into negative will be 10 times as much as the opposite misclassification. So we may prefer to be conservative and be more willing to bias the prediction towards the positive class, i.e., avoid making costly mistakes, by setting $q_3 = s_- / (s_+ + s_-)$. In HEM+CMN for thrombin dataset, we will report the average of the highest accuracy for each test point among using q_1, q_2 and q_3.

6. Optimization configurations. We always used the *lmvm* algorithm to perform gradient descent for LOOHL with various regularizers. Although σ appears only in the form of σ^2, we still used σ as optimization variable because otherwise if we use $\alpha \triangleq \sigma^2$ as variable, we will have to add a constraint $\alpha \geq 0$, and constrained programming is usually more complicated than unconstrained programming. Although optimizing over σ will result in multiple optimal solutions, the optimization problem is non-convex by itself, and the gradient descent will just find a local minimum. Empirically, we noticed that the local minimum found by the solver is usually close to the initial value.

Although the absolute value of C_1 and C_2 in above point 3 does not influence the optimal variable value in theory, the solver of the optimization toolkit *is* sensitive to the absolute value in practice. So we picked C_1 and C_2 according to the behaviour of the solver, tuning it among 10^n (where n is an integer), so that it will proceed properly without

looking at the test accuracy. Poorly chosen C_1 and C_2 will quickly cause numerical problems in optimization. We chose C_1 = 10000 for 4vs9 and cancer datasets, C_1 = 100 for text and thrombin binding, and C_1 = 1000 for ionosphere. For each dataset, the same C_1 applied to both part 1 and part 2 of the experiment.

In part 1, the initial value of σ for gradient descent in LOOHL+Sqr was chosen as the 5 fold cross validation result of HEM+Thrd, and then increased by 0.05 until no numerical problem occurs (as detailed in point 2). The initial values of σ for MinEnt were 0.05, 0.1, 0.15, 0.2, and 0.25. In part 2, the initial values of σ chosen were 0.15, 0.2, 0.25, 0.3, and 0.35. Different from Sqr which also used the initial σ as prior mean, the other three regularizers only used it as an initial value for optimization.

Finally we limited the number of function evaluation to 35 on all datasets except ionosphere. This was to avoid the situations of extremely slow convergence. In most of the time, *lmvm* converged in 35 times of function evaluation (not 35 iterations because the number of function evaluation in each iteration is not fixed). We will also report the result under this cut-off optimization because usually this is believed to occur at some flat region and it is still believed to be a reasonable value.

7. Platform consideration. How to make full use of the computing resources is also an important problem in practice. We summarize the software and hardware configurations of the computing facilities we have access to.

	CPU number, type and frequency	bit	memory (GB)	Matlab run-time library available?	node type
access0-13	2 × P4 2.8GHz	32	2.5	✓	access (interactive)
sma0-14	4 × Opteron 2.2GHz	64	2.0	×	access (interactive)
comp0-41	2 × P4 2.8GHz	32	1.0[1]	✓	compute (batch)
compsma0-34	2 × Opteron 2.2GHz	64	2.0	×	compute (batch)

Table B.1 Hardware and software configurations of computing cluster

On none of the above nodes can we compile Matlab code. We compiled Matlab code on twinkle.ddns.comp.nus.edu.sg, which is a 32-bit machine. Then we transferred the executable to the 32-bit compute cluster nodes. Let us also consider the four other algorithms mentioned Table 5.1. Implementation of SVM/TSVM is available at http://svmlight.joachims.org/, written in pure C. We have implemented st-mincut in pure C++ by making use of the max flow implementation by Yuri Boykov and Vladimir Kolmogorov (http://www.cs.cornell.edu/People/vnk/software.html). SGT is available at http://sgt.joachims.org/, which uses Matlab Math library for matrix inversion and eigen-decomposition just like HEM, MinEnt, and LOOHL. In sum, we can dispatch jobs in the following way:

LOOHL	HEM	MinEnt	SGT	SVM/TSVM	st-mincut
access0-13	comp0-41			compsma0-34 or comp0-41	

[1] It is relatively small, especially under the multi-user environment.

Appendix C Detailed Result of Regularizers

In this appendix, we give the detailed results of the regularizers for LOOHL when enumerating on the grid of σ (initial bandwidth for optimization) and C_1:C_2. For Eigen, Step and REnt, the initial values of σ chosen were 0.15, 0.2, 0.25, 0.3, and 0.35, which correspond to the five sub-plots from left to right in the following 24 figures. C_1:C_2 were chosen to be 1:10, 1:100, and 1:1000 for all the datasets on Eigen and REnt regularizers. As for Step regularizer, we chose 1:10, 1:100, 1:1000 for 4vs9, cancer and ionosphere, and chose $1:10^3$, $1:10^4$, $1:10^5$ for text and thrombin (ref. point 4 and 6 in appendix section B.4). We will call these C_1:C_2 as r_{max}, r_{medium}, r_{min} respectively in a uniform way across all datasets, i.e. r_{max} = 1:10 for cancer while $r_{max} = 1:10^3$ for text. The x-axis stands for the number of labelled points and the y-axis stands for accuracy (balanced accuracy for thrombin). Some points are missing because optimization always failed at the corresponding coefficient setting.

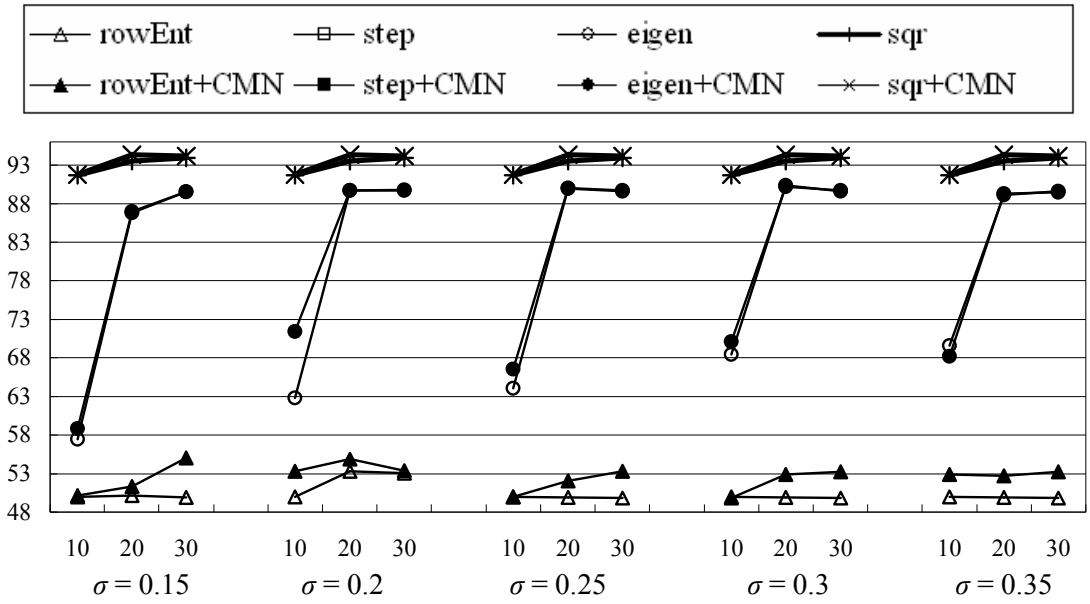


Figure C.1 Accuracy of four regularizers on 4vs9 (original) under r_{max}
The Step regularizer met numerical problems in optimization for all σ under this C_1:C_2.

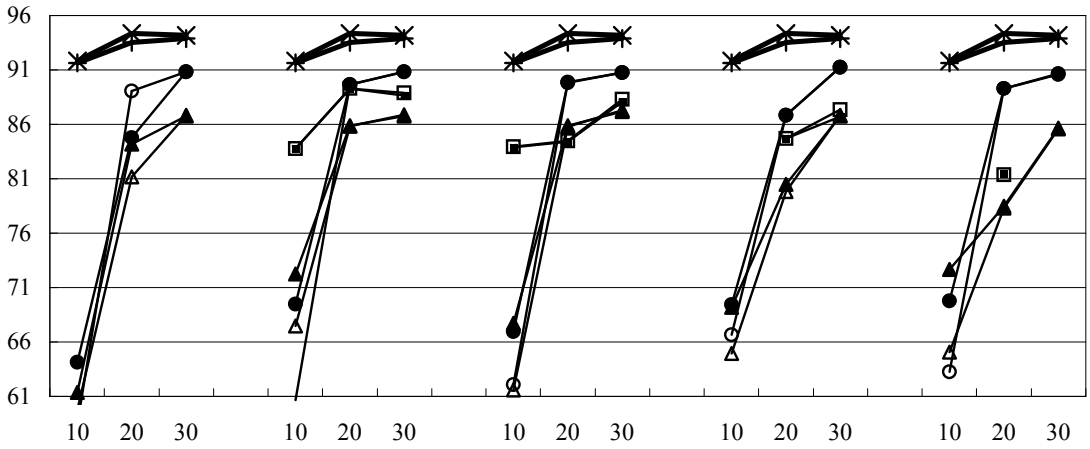


Figure C.2 Accuracy of four regularizers on 4vs9 (original) under r_{medium}

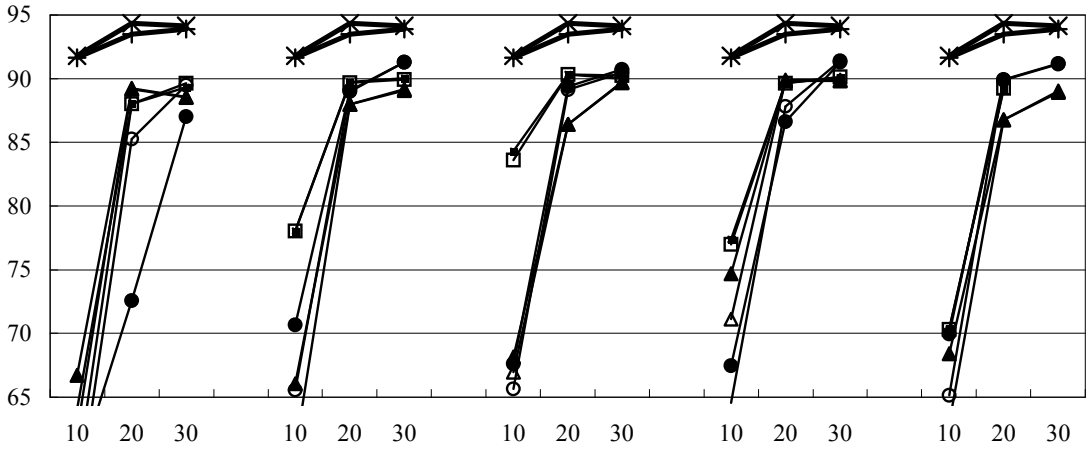


Figure C.3 Accuracy of four regularizers on 4vs9 (original) under r_{min}

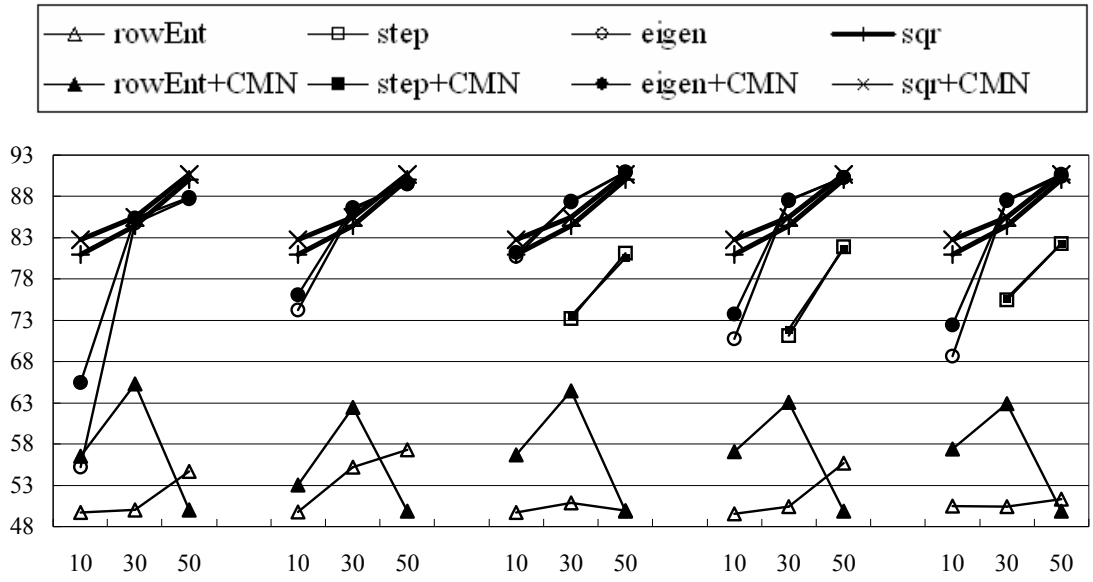


Figure C.4 Accuracy of four regularizers on 4vs9 (probe) under r_{max}

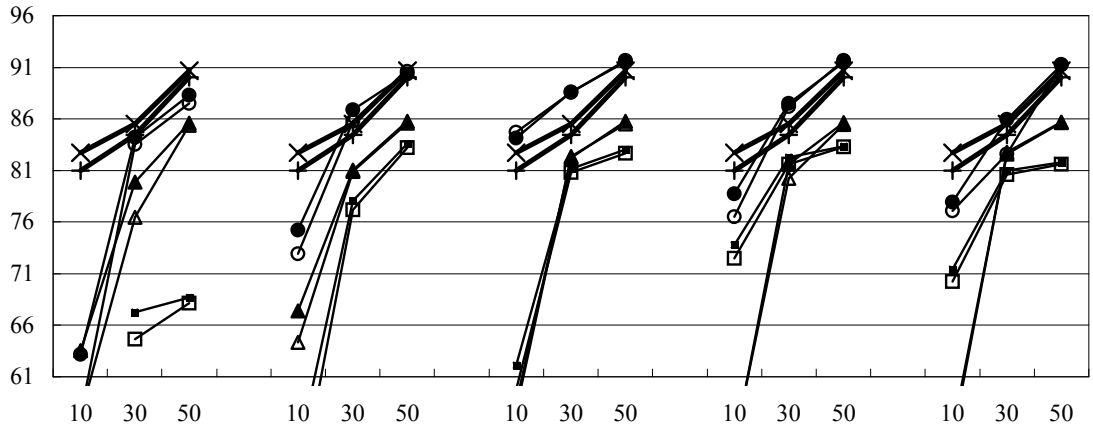


Figure C.5 Accuracy of four regularizers on 4vs9 (probe) under r_{medium}

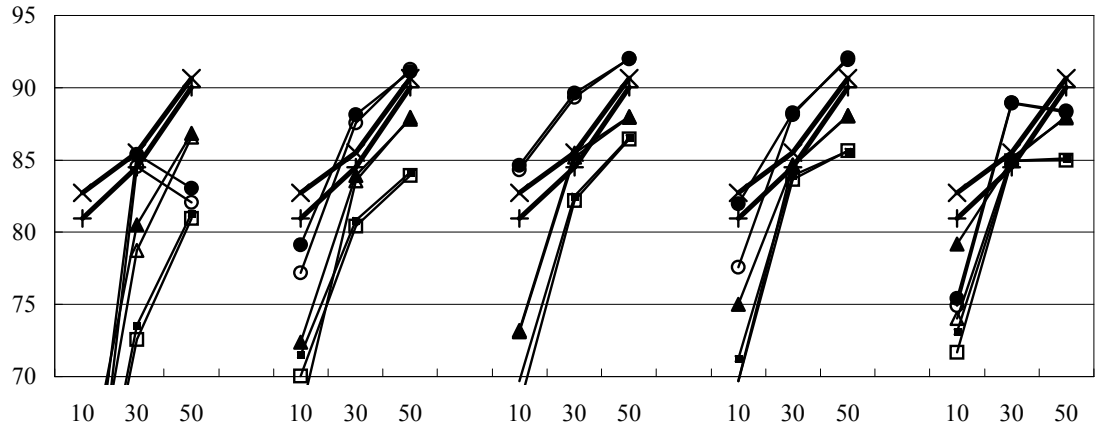


Figure C.6 Accuracy of four regularizers on 4vs9 (probe) under r_{min}

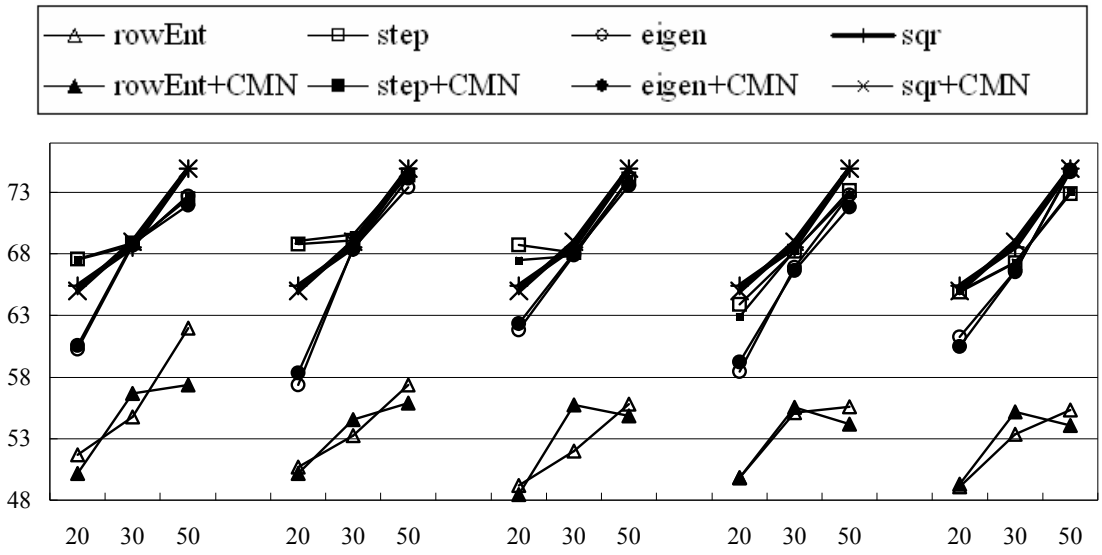


Figure C.7 Accuracy of four regularizers on cancer (original) under r_{max}

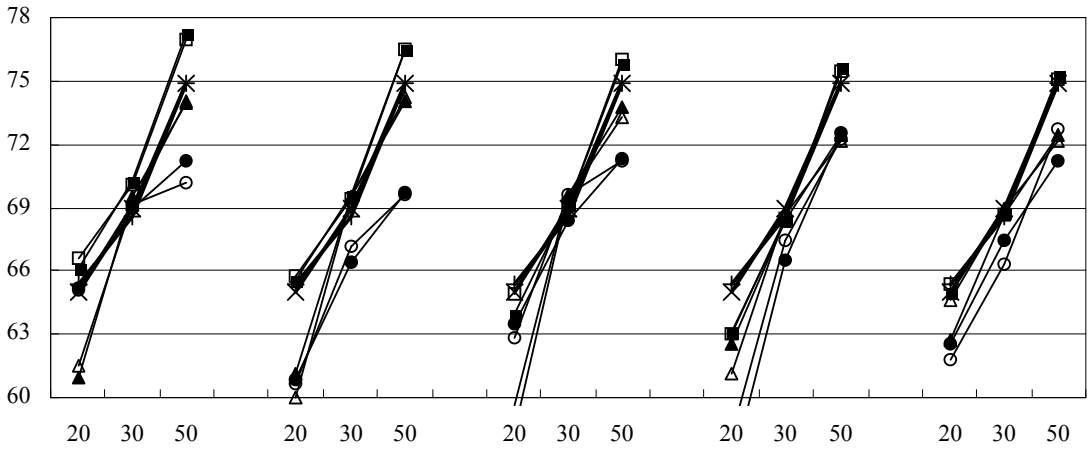


Figure C.8 Accuracy of four regularizers on cancer (original) under r_{medium}

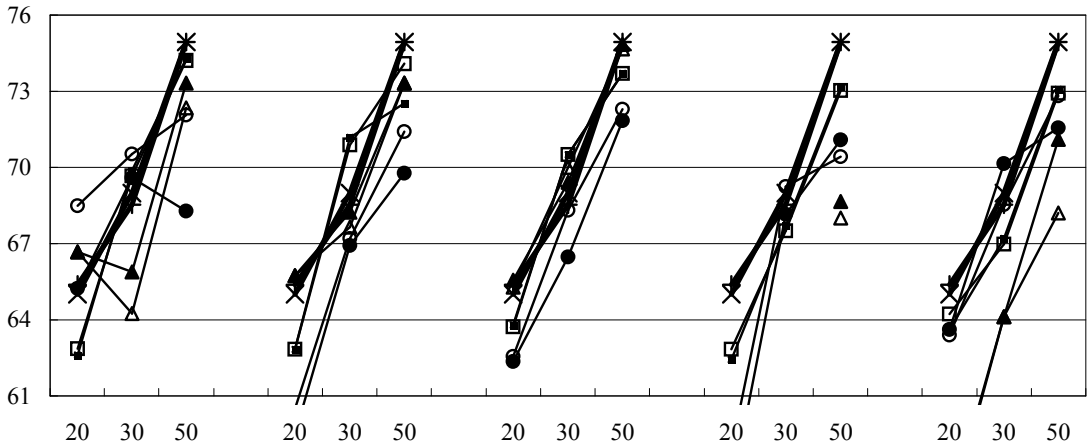


Figure C.9 Accuracy of four regularizers on cancer (original) under r_{min}

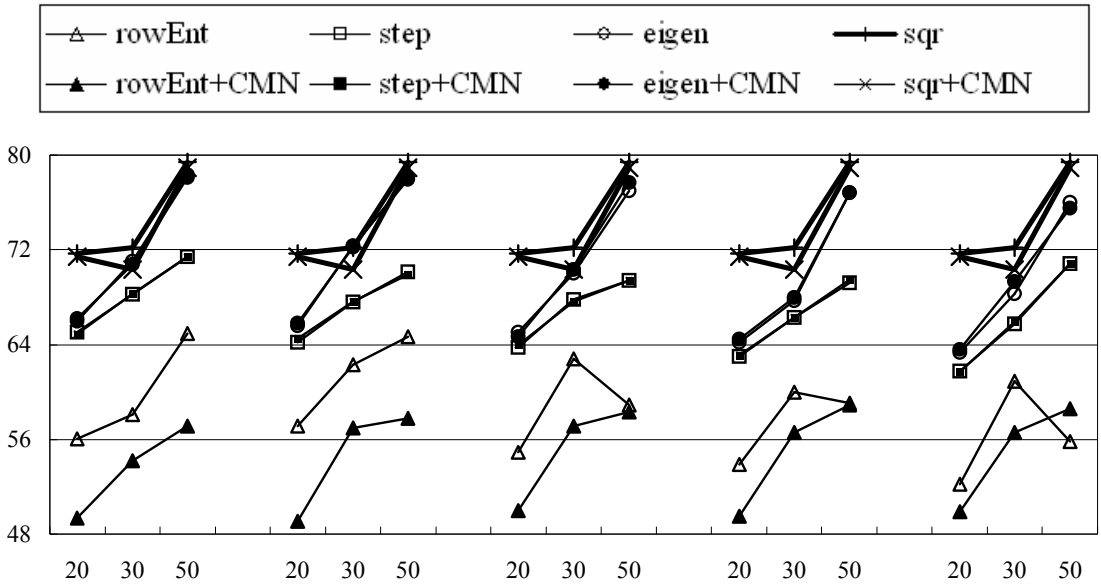


Figure C.10 Accuracy of four regularizers on cancer (probe) under r_{max}

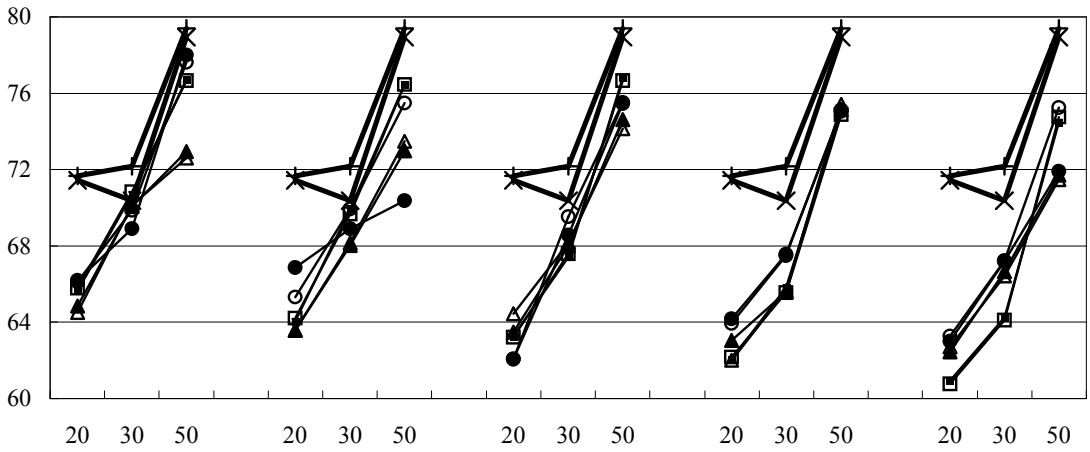


Figure C.11 Accuracy of four regularizers on cancer (probe) under r_{medium}

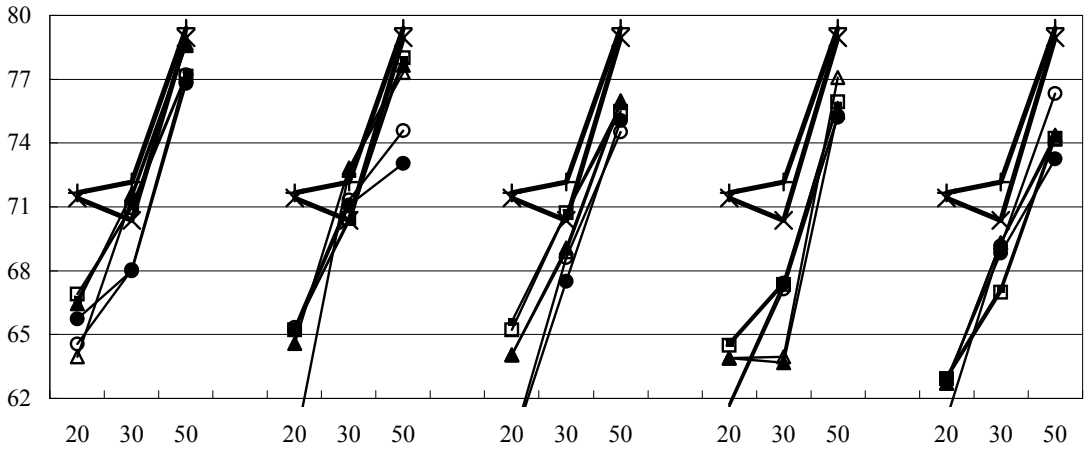


Figure C.12 Accuracy of four regularizers on cancer (probe) under r_{min}

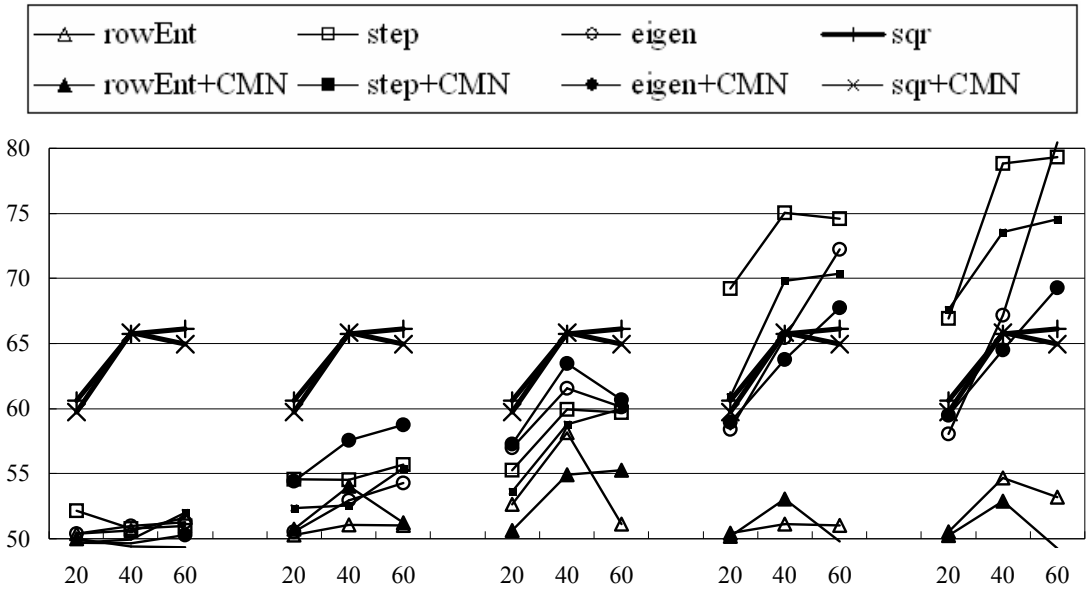


Figure C.13 Accuracy of four regularizers on text (original) under r_{max}

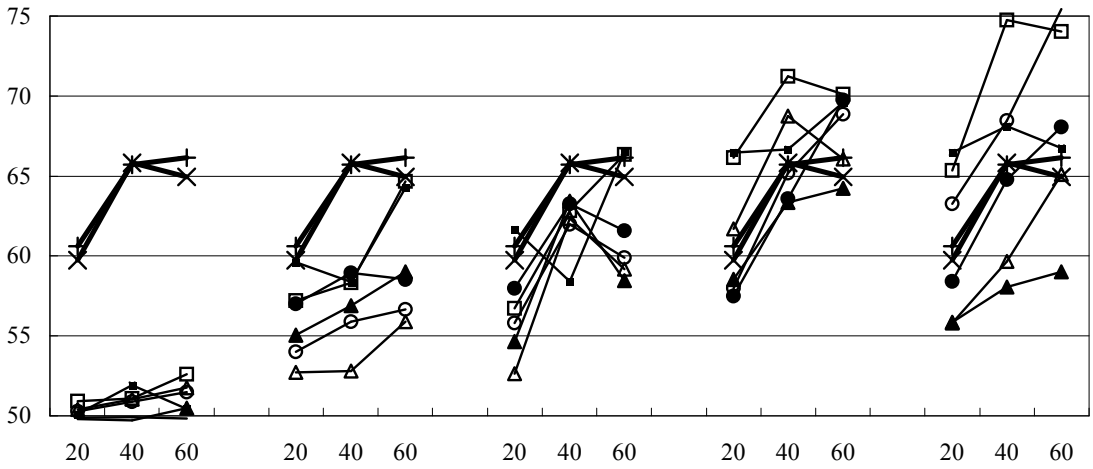


Figure C.14 Accuracy of four regularizers on text (original) under r_{medium}

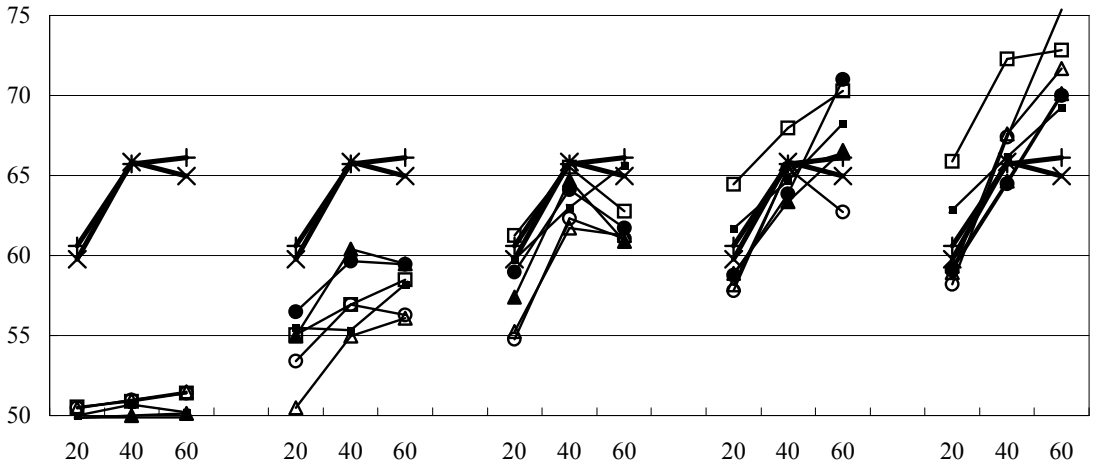


Figure C.15 Accuracy of four regularizers on text (original) under r_{min}

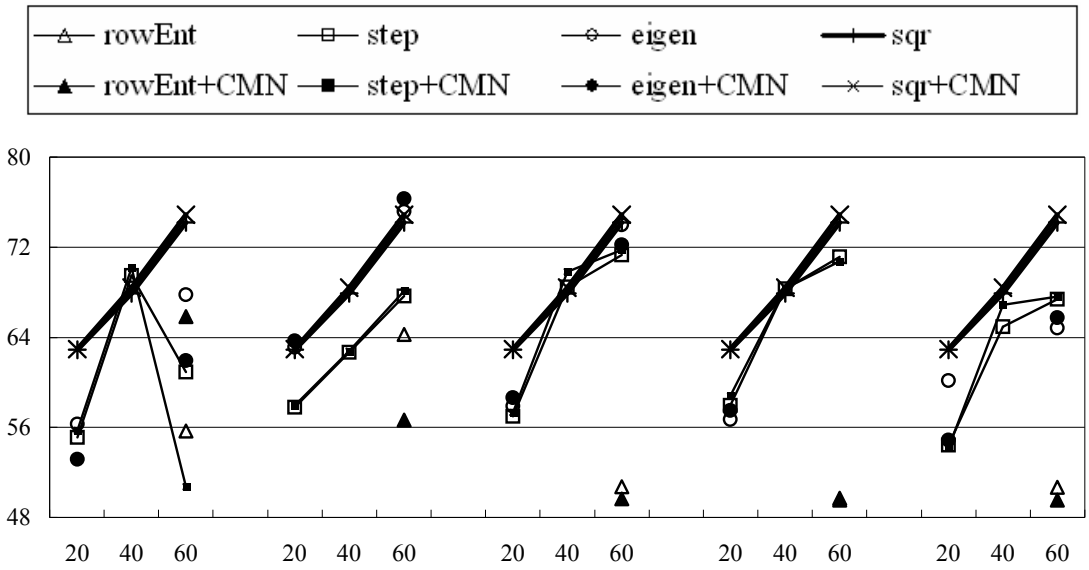


Figure C.16 Accuracy of four regularizers on text (probe) under r_{max}

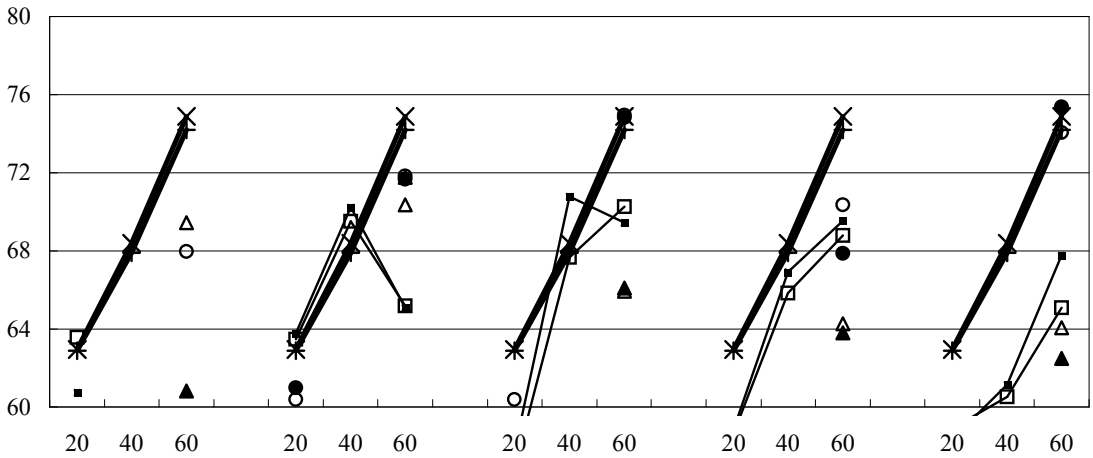


Figure C.17 Accuracy of four regularizers on text (probe) under r_{medium}

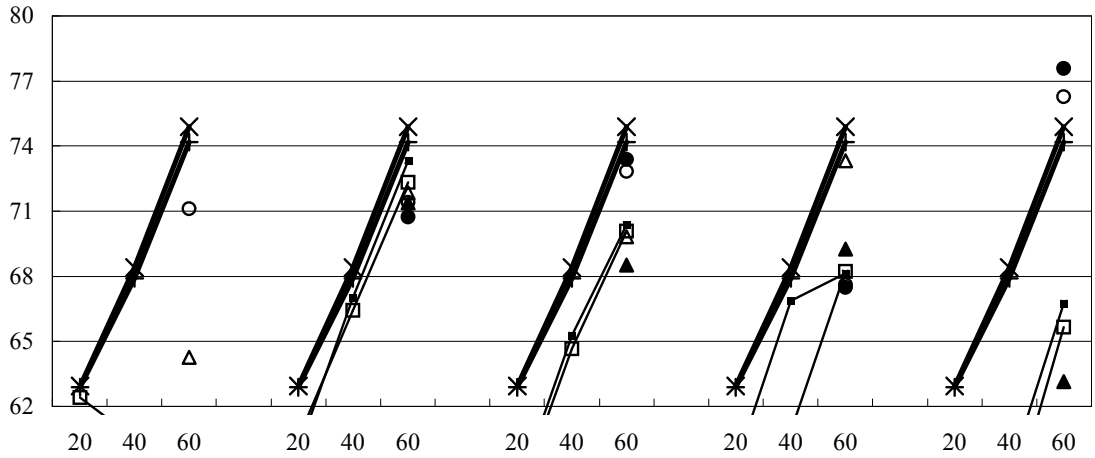


Figure C.18 Accuracy of four regularizers on text (probe) under r_{min}

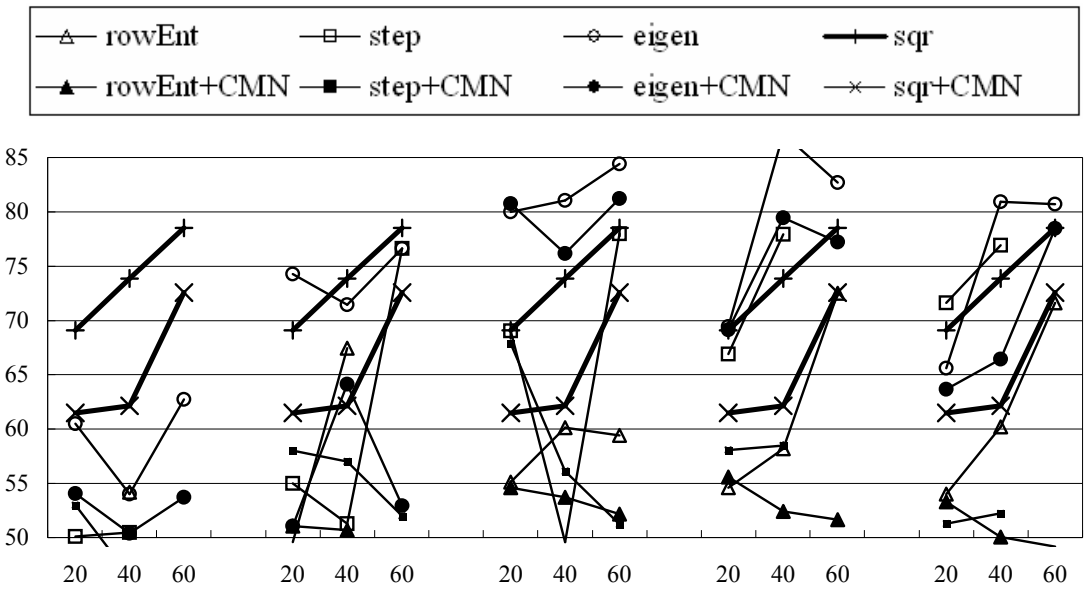


Figure C.19 Accuracy of regularizers on thrombin (original) under r_{max}

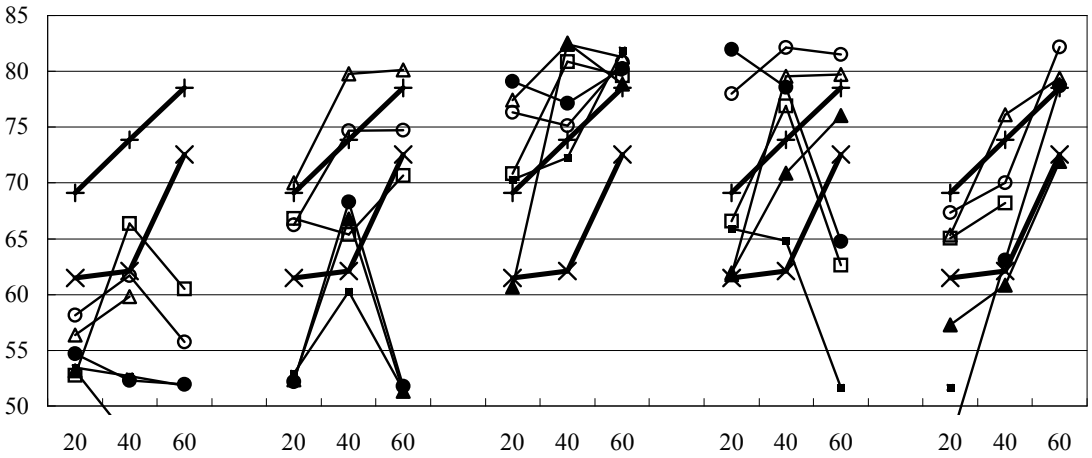


Figure C.20 Accuracy of regularizers on thrombin (original) under r_{medium}

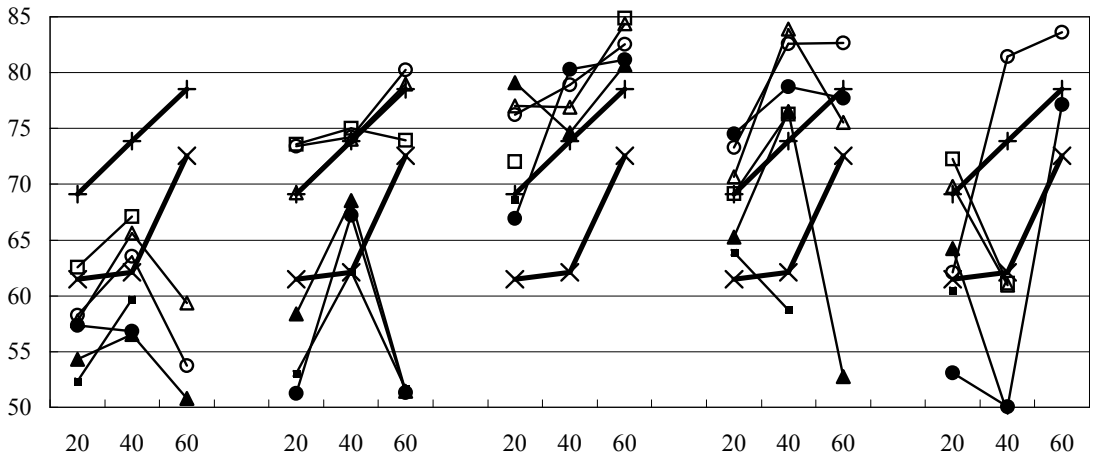


Figure C.21 Accuracy of regularizers on thrombin (original) under r_{min}

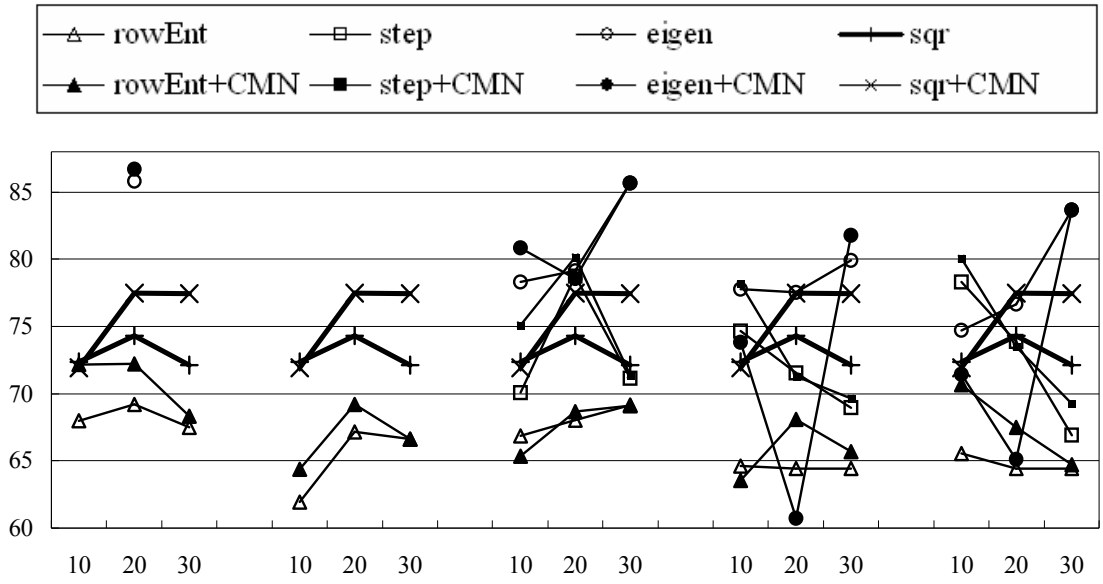


Figure C.22 Accuracy of four regularizers on ionosphere under r_{max}

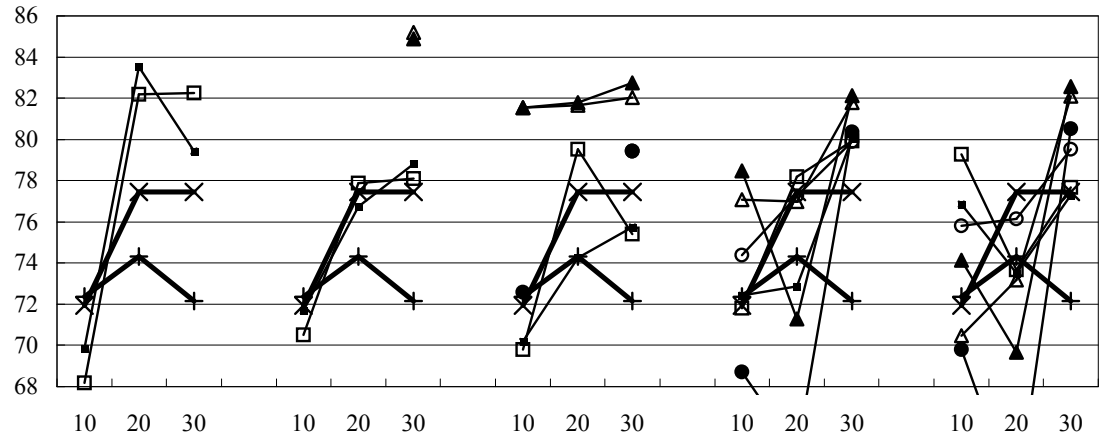


Figure C.23 Accuracy of four regularizers on ionosphere under r_{medium}

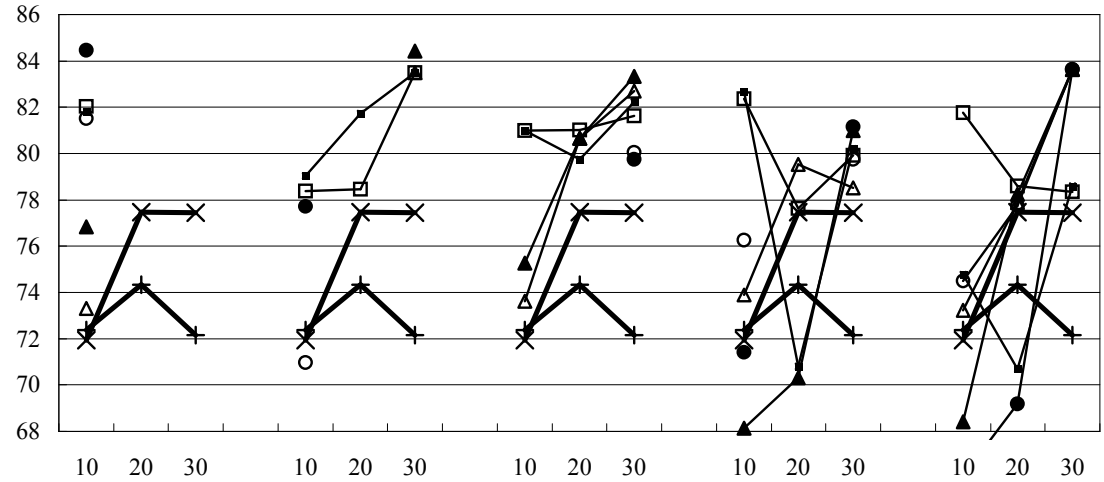


Figure C.24 Accuracy of four regularizers on ionosphere under r_{min}