修士論文

Study on Semi-Supervised Learning with the Framework of Generative Model

生成モデルを用いた半教師付き学習に関する研究

2019年6月26日

学籍番号　17507781

NGUYEN HOANG NGHIA

指導教員　山田　耕一

長岡技術科学大学大学院工学研究科
情報・経営システム工学専攻

Abstract

Machine learning from the view of input data consists of supervised and unsupervised learning. In this traditional facet, the learning models are considered purely all labeled or unlabeled data. However, in many cases, labeling requires a mass effort of human works while it is much easier to collect the data instances only. Furthermore, the unlabeled data in an unsupervised task cannot match with a predefined output. There is a great interest in motivation to combine both types of data in semi-supervised learning whereas we have a massive amount of unlabeled data and only a small proportion of labeled ones.

In this thesis, we present the semi-supervised methods for the binary classification problem using a framework of the generative model. The idea of generative inference is to implicitly predict the label through modeling the joint density of data instances and its corresponding labels. More precisely, we leverage the assumption of mixture model which implied that the distribution of data is laid on different underlying distributions.

What we expect in this thesis are practical learning models that can be straightforwardly handled on hand with affordable complexities and also produce acceptable performances. Therefore, we consider the two different classes of solution. The first traditionally deals with the probabilistic setup of the mixed distributions and solves the problem through the assumptions of prior and conditional distributions. The second solution comes with the more recent inspiration of deriving the graph structure to represent data, the graph-based methods. We construct a graph with vertices from both labeled and unlabeled data. The edges show the relationship between the vertices. In both schemes of solution, we focus on the potential to settle down with the different scales of unlabeled data and identify the underlying distributions. In this meaning, we propose our modification on the origin models and work on several particular setups of each solution to show their abilities to adapt with the condition of semi-supervised learning.

Acknowledgments

First of all I would like to send my sincere thanks to my supervisor, professor YAMADA Koichi, who gave me the opportunity to continue on my learning journey. For his openness allowed me to responsibly decide and have my own space in my studies and to his supportive, valuable academic guidance. Moreover, I have learned from him to be careful and patient in doing research.

I had only two years here at Nagaoka University of Technology. I would like to thank all the faculty, staff, fellow students and friends for their kind support and sharing memories: Assistant professor UNEHARA Muneyuki, SUZUKI Izumi and Yamada-lab's members had greatly followed all of my research seminars and were always warmly available to assist me; the Nagaoka University of Technology Library had provided all of my needed documents and facilities; Division of International Affairs for their useful information, supportive aiding and various interesting programs that made my student life more fascinating; the Japanese Language Department had assisted the necessary basic Japanese courses.

I would be thankful to the Ministry of Education, Culture, Sports, Science and Technology who generously supported my Master course with the Monbukagakusho Scholarship.

Finally, I thank my family. My parents and brothers for their encouragement and endowment in my decisions, my cat for entertaining me all the time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Semi-Supervised Learning

In the field of machine learning, a learning model can be described by the data. With different input data, we have different learning schedules. A problem usually starts with a given set of observed data $\mathcal{X}$ that basically is in the form of feature vectors. Unsupervised models take this input data and commonly specify the clusters, reduce the feature vector dimensionality or generally look for the structure of $\mathcal{X}$. Then the observation comes more with the foresee output $\mathcal{Y}$ called labels for each instance of $\mathcal{X}$. The supervised models join in and want to learn the mapping from $\mathcal{X}$ to $\mathcal{Y}$. When $\mathcal{Y}$ is a discrete set, we have the traditional classification problem.

The semi-supervised term is widely used in the learning models that infer from both labeled and unlabeled data, when there is much of unlabeled and a few labeled data. The model expects to promote better performance than the methods that only use the labeled ones. In this thesis, we consider the semi-supervised inference model with the classification problem. For simplification purposes, we only examine the problem of binary classification, in which the label only receives positive or negative value.

It is natural that we have some learning processes that follow the condition of semi-supervised learning, which falls in between the regimes of unsupervised and supervised models. Let's have some examples on this situation

- The auto collecting news system has a list of categories and needs to classify hundreds of updated news everyday. There is a core set of news that has been categorized before. The traditional classifier has a constant ability in performing this daily task and if we hire someone to update the core, it must be taking much of time to significantly improve the performance of the model.

- The marketing analysis agent based on the community response on some topics on social network sites like Facebook, Twitter, etc., is flooded in the enormous range of user comments, shares or reactions. A few of them have tag names or come from the same groups that we can treat as labels. For the large remaining part, they are unlabeled data.

Furthermore, human beings besides directly gaining knowledge on a new concept from the verified sources also be absorbed from a large unintended information

around them. If we receive a new definition of "Japanese bobtail cat", then it might be reminiscent of other cat breeds if we have met many cats before.

There are two different learning settings in semi-supervised including *inductive* and *transductive* learnings. The inductive learning finds the answer for the whole domain of input data and the transductive learning takes into consideration the input data only. In other words, inductive setup aims to predict the future data which is unknown now through the input data and transductive setup only wants to know the label of the given unlabeled input data.

1.2 Learning from Unlabeled Data

In the statistical classification view, we are looking for the estimation of probability of $\mathcal{Y}$ if we have known $\mathcal{X}$, $P(\mathcal{Y}|\mathcal{X})$. The supervised classification can be divided into generative and discriminative classifiers [1]. The generative models implicitly estimate this probability through the joint probability of $P(\mathcal{X}, \mathcal{Y})$ using Bayesian inference. In contrast, discriminative model directly estimates the $P(\mathcal{Y}|\mathcal{X})$. It has been argued that, in purely classification task, discriminative model outperforms a generative one since it only focuses on the assignment at hand and also has the simpler solution without any assumption on $\mathcal{X}$ [2].

In the same aspect, from [3], there are two groups of approaches in semi-supervised learning including generative and diagnostic models. Semi-supervised learning seems like a greedy strategy that grasps all the data in hand and pushes to the learning process. Unlike supervised models, what makes us believe that the unlabeled data are useful and satisfied our expectations? This should be laid on the basic assumption of semi-supervised that the relationship between what we have known on the distribution of unlabeled data is corresponding with the target labels presented in labeled ones. In other words, the information that unlabeled data bring to is the knowledge of distribution of $P(\mathcal{X})$ and that is the improved factor for our models since what we are planing is the increasing of unlabeled data. In our intuition, does not it mean that the generative approach now plays the explicit role in modeling $P(\mathcal{X})$? Furthermore, the obvious advantage of generative models is the estimation of the structure of our data. From the application perspective, when a large amount of unlabeled data pour in, it also increases the prospect of lost information, this can be quickly fixed by our generative models. Even if we want to reorganize our target definition by merging or splitting the labels, it does not require us to retrain our system.

We do not have the intention to compare between the generative and diagnostic learning schemes of semi-supervised learning. In practical machine learning, the models are a set of toolbox for our problem, in different situations, we should choose the suitable tools. To know which are the good fits, besides understanding our data, it is to have a large box of tools and get to know how they work. With our intuition, we want to examine the idea of a generative approach—the assumption of the situation—and provide through rough investigations on the learning models to match up with the data assumption. By this meaning, given a specific assumption about the data distribution with generative idea, the contribution of this thesis is entirely providing the modified applicable models that adapt to this assumption.

The next section will give more details on the generative approach and the learning models with our modifications.

1.3 The Generative Framework

Let $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ be our random variables for an observed data and its corresponding label. The domains of $\mathcal{X}$ and $\mathcal{Y}$ are restricted by the learning setup. With inductive setting, it means the domain is the entire spectrum of data in the same way of observation in future. On the other hand, the transductive setting constrains only the data we have in the current data set.

Basically, we want to find the best match of $\mathcal{X}$ to $\mathcal{Y}$. It means to classify x with the estimation $\hat{y}$ that maximizing $P(y|x)$. We can denote our solutions such as

$$\hat{y} = \text{argmax}_y \, P(y|x) \tag{1.1}$$

In generative schemes, this can be determined through Bayes theorem

$$P(y|x) = \frac{P(x,y)}{P(x)} = \frac{P(x,y)}{\sum_y P(x|y)P(y)} \tag{1.2}$$

The denominator of (1.2) is the summation over all labels and does not affect our maximization of $P(y|x)$, so that

$$P(y|x) \propto P(x,y) \tag{1.3}$$

Consequently, our problem turns to estimate the joint probability of $P(x,y)$. Moreover, we only take into account the binary classification problem. Thus the value of y belongs to positive or negative value as is in $\{-1,+1\}$. Hence we can directly present our marginal distribution as the sum of two components

$$P(x) = P(x|y=-1)P(y=-1) + P(x|y=+1)P(y=+1) \tag{1.4}$$

With this structure of $P(x)$, we can easily generate, look for the missing features of input data or reorganize the data structure.

The first model that we employ here is the traditional *mixture model* which goes straight with this probabilistic formula to extract the joint probability into the class prior and conditional probabilities. Then comes with an observed data set, the joint log-likelihood is incorporated with EM algorithm [4] to estimate the parameters of the model. We continue further with the *many-to-one assumption*. It states that the class conditional probability is a compound of different components and we continue to infer the model through them. In practice, we consider the application on text data and deploy the Naive Bayes multinomial distribution for the class conditional $P(x|y)$ and the Dirichlet distribution for the prior P(y) [5]. The issue that we tackle here is how to construct the sub-components. It seems like a trivial problem, but in practice, we will show that it has a specific impact on our model.

Following the recent interest in leverage the graph structure in modeling data, we continue with the graph-based approaches. At first, we map our data into a graph and find the way to reasoning the unlabeled label through it. Then we transform our data distribution assumption into a target function that we will try to optimize. The conventional solution utilizes the minimum cut algorithm on this graph [6] that to be shown to have the issue when the boundary between our data distributions are overlapped. To overcome this problem, we employ a graphical model and modify the conventional one by introduce the *influence index* to adapt with our data distribution assumption. Furthermore, we also extend the basic graphical model to work on an arbitrary graph by derivering the approximate *loopy belief propagation algorithm*.

1.4 The Structure of Thesis

Accordingly, the thesis is structured as follows:

Chapter 2 begins with the traditional solution using a mixture model and the many-to-one assumption. Then we will discuss the different methods to initialize the model and later to show it has the influence in the model performance through the experiments.

Chapter 3 presents a graph-based method to our problem with the conventional mincut approach. After that, we derive a graphical model and show how it can be a promising candidate for the replacement of mincut approach using our influence index with the extension of loopy belief propagation inference.

Chapter 4 finalizes the thesis with the discussions and conclusions on the performance and violation of model assumptions and to some prospective works in the future.

Chapter 2

Multinomial Naive Bayes Models

Mixture models are one of the very first ideas applied in semi-supervised learning. In this chapter, we directly derive the mixture model to the text data with the assumption of many components in the label conditional distribution and consider how we initialize these components.

2.1 Problem Statement

We are given a set of independently and identically distributed data $\mathcal{D} = \{\mathcal{D}_L, \mathcal{D}_U\}$ which consists of two subsets of labeled data $\mathcal{D}_L$ and unlabeled data $\mathcal{D}_U$. Denote that the number of labeled data $|\mathcal{D}_L| = l$ and unlabeled data $|\mathcal{D}_U| = u$. We always have the condition $u \gg l$.

The labeled set has the form of pairs of data instances and its corresponding label, $\mathcal{D}_L = \{(x^{(1)}, y^{(1)}), ..., (x^{(l)}, y^{(l)})\}$. The unlabeled set only contains the instances, $\mathcal{D}_U = \{x^{(l+1)}, ..., x^{(l+u)}\}$. In addition, we denote the set of labeled instances $\mathcal{X}_L = \{x^{(1)}, ..., x^{(l)}\}$ and the set of unlabeled instances $\mathcal{X}_U = \{x^{(l+1)}, ..., x^{(l+u)}\}$, then the set of data instances $\mathcal{X} = \mathcal{X}_L \cup \mathcal{X}_U$. Correspondingly we have the set of labels $\mathcal{Y}_L = \{y^{(1)}, ..., y^{(l)}\}$ for $\mathcal{X}_L$, the set of true label $\mathcal{Y}_U = \{y^{(l+1)}, ..., y^{(l+u)}\}$ for $\mathcal{X}_U$ and the set of data label $\mathcal{Y} = \mathcal{Y}_L \cup \mathcal{Y}_U$.

This thesis only focuses on binary classification problem where the labels in $\mathcal{Y}$ only receive positive or negative value. Precisely, we have the condition $\mathcal{Y} \in \{-1, +1\}^{l+u}$. Hence the task is looking for an inference $\mathcal{X} \to \mathcal{Y}$ with inductive learning and estimating $\{\mathcal{X}, \mathcal{Y}_L\} \to \mathcal{Y}_U$ with transductive learning.

2.2 Mixture of Components for Text Data

Suppose we are given the distribution $P(\mathcal{X})$ with the unknown parameter set θ that we want to estimate through the observed data set $\mathcal{D}$. The distribution $P(\mathcal{X})$ has the form of a mixture of different independent components. Denote that the set of these components is M, then we have

$$P(\mathcal{X}) = \sum_{M_j \in M} P(\mathcal{X}|\theta_j) P(M_j) \tag{2.1}$$

where $\theta_j \in \theta$ is the parameters of component M_j.

Now we want to look for the correspondent between M and the label values $\{-1, +1\}$. The first idea here is that each component is equivalent with a label value [4], thus we have two components and directly express the distribution through the label value set as in (1.4). Then the extension to at least one component on a label value was generalized in [7]. We have a predicted label of y for the input data x such that

$$\hat{y} = \text{argmax}_{y \in \{-1,+1\}} \sum_{M_j \in M} P(y|M_j)P(x|\theta_j)P(M_j) \tag{2.2}$$

With the equivalent setting, Nigam *et al.* [5] was applying this model to the text data. We also adapted the term many-to-one assumption from there. The model is restricted that $P(y|M_j) \in \{0, 1\}$ as the *predefined components* on each label value, which means we have decided how many and which components are for each label value. Furthermore, the distribution of the components in the same label value y is given and constrained of sum to equal to 1, so that

$$\sum_{M_j : P(y|M_j)=1} P(M_j|y) = 1 \tag{2.3}$$

and component maps for the given instance and label input

$$P(M_j|x) = P(M_j|y)P(y|x) \tag{2.4}$$

We are now reconstructing the model on text data and estimate the parameter for each component. Given a dictionary as list of ordered words $\mathbf{w} = (w_1, ..., w_d)$ where d is the number of different words in $\mathbf{w}$. We represent the input as word count vector $x = \langle x_1, ..., x_d \rangle$ where each x_k is the count of word w_k in the text x. Consider the joint distribution of input data x and component M_j, we have

$$P(x, M_j|\theta) = P(x|M_j, \theta)P(M_j|\theta) \tag{2.5}$$

with x is assumed with the Naive Bayes assumption on the independence of word features. Hence what we have for component conditional is a multinomial distribution

$$P(x|M_j, \theta) = (\sum_k x_k)! \prod_{k=1}^{d} \frac{P(w_k|M_j, \theta)^{x_k}}{x_k!} \tag{2.6}$$

On this step, with Naive Bayes assumption, there is one more different setup of Bernoulli distribution, where the word count feature be replaced by the word appearance—a word w_k appears in x or not. But it was empirically demonstrated that the Multinomial model has a better performance than the Bernoulli one [8].

We continue with the component prior distribution, which is frequently conjugated with a Dirichlet distribution. In this way, eventually, the estimate also assembles with the smoothing models [9]. So we set concentration parameters of the Dirichlet distribution with the same value and are equal to 2 for a component prior distribution. This gives us an estimate with the uniform distribution and Laplace smoothing. So that

$$P(M_j|\theta) \propto \prod_{k=1}^{d} P(w_k|M_j, \theta) \tag{2.7}$$

So far we have specified the model for our components, we turn back to the estimator of θ. Here we derive the maximum log-likelihood

$$\hat{\theta} = \text{argmax}_\theta \ln(P(\mathcal{D}|\theta)) \tag{2.8}$$

We first consider the basic case when there is only labeled data, the solution is straightforward with partial derivatives of Lagrangian with the constraint of $\sum_{k=1}^{d} P(w_k|M_j) = 1$. More details about the solution can be found in [10]. We have an estimate of the word conditional probability

$$P(w_k|M_j, \hat{\theta}) = \frac{\sum_{i=1}^{l} x_k P(M_j|x) + 1}{\sum_{k=1}^{d} \sum_{i=1}^{l} x_k P(M_j|x) + d} \tag{2.9}$$

and the prior probability

$$P(M_j|\hat{\theta}) = \frac{\sum_{i=1}^{l} P(M_j|x) + 1}{l + 2} \tag{2.10}$$

Next we cooperate the data with the unlabeled set. In this case, we cannot directly solve the optimization of the log-likelihood and to only get to the local convergence using EM algorithm [4]. Basically, the algorithm consists of two basic steps, the expectation (E-step) and the maximization (M-step). The process starts with an initial set of parameter, $\theta^{(0)}$. Then for each iteration, it repeatedly computes the E-step and M-step. At t-th time, the E-step calculates the data conditional probability on the latent components for unlabeled data using the previously observed parameters $\theta^{(t-1)}$—the partial labels. After that at M-step, we would consider that our data have been temporally and partially labeled to components and estimate the parameter $\theta^{(t)}$ using (2.9) and (2.10) correspondingly. The EM algorithm guarantees that the next parameter set $\theta^{(t)}$ is improved from $\theta^{(t-1)}$, which means we have the log-likelihoods $\ln(P(\mathcal{D}|\theta^{(t)})) \geq \ln(P(\mathcal{D}|\theta^{(t-1)}))$ [11]. Algorithm 1 presents the general framework of EM algorithm in our case. The convergence condition could be the threshold of loop number or better is the different of log-likelihood between $\theta^{(t-1)}$ and $\theta^{(t)}$.

Algorithm 1 EM algorithm with labeled and unlabeled data

```
function EmAlgorithm(D)
    t = 0
    Initialize θ^(0)
    repeat
        t = t + 1
        # E-step
        Q(x) ← P(M|x, θ^(t−1))
        # M-step
        θ^(t) ← argmax_θ Q(x) ln(P(x, M|θ))
    until θ^(t) is converged
    return θ^(t)
```

The initialization of $\theta^{(0)}$ setups the predefined component $P(y|M)$ for each label value and the label conditional on components $P(M_j|y)$. Basically, the predefined component is searched from an appropriate range of numbers and the problem left is the assignment of label conditional on components. We will discuss this in the next section.

2.3 Component Initialization Problem

Literally, the initialization step should be done by deciding the component number for each label value and assigning labeled data for the components. The conventional method does it by randomly distributing the components probability throughout the label of the labeled instances. In this way, the only advantage of labeled data is restricted into the constraint on all components of its label and the same initialization is applied for all input components. Which means the inflexible with the change of data, in case of which we may upgrade the labeled set. The positive point of Naive Bayes is simple to implement and flexible with large amounts of data. Consequently, we should think about the methods that are not merely treating all the components in the same meaning, but elaborate the characteristics of input labeled set to enhance the learning model.

We may contemplate that the assignment of labeled data to components is an unsupervised task where we only know the component number. Here we derive two different unsupervised approaches to setup the assignment schemes. The first is constructing a hierarchical tree for each label value and the second is k-means clustering. These two are the basic clustering models of unsupervised learning, we can find a detailed description in [12]. In this section, we only present their application with our model.

We begin with the hierarchical tree assignment. For each label value, we consider the similarity between components. A hierarchical clustering establishes and treats each cluster as a component using agglomerative structure. Figure 2.1 gives the illustration of an agglomerative tree for a label value. The clustering processes through the iterated steps. From the beginning, each data instance is considered as a component. At each iteration, we merge the two most similar components until only one remains. We continuously name and index all components that are merged at each iteration in a layer. A component sampling will be done with the cutting process. A cut between two continuous layers i and j $(i < j)$ returns a set of assigned components. For example, the layer 3 and 4 are $\{\{1,2,3\},\{4\},\{5,6\}\}$ and $\{\{1,2,3,4\},\{5,6\}\}$ respectively. A cut between layer 3 and 4 returns three components corresponded with an initialized component setting $\{1,2,3\},\{4\}$ and $\{5,6\}$. The next cut will return 4 components, it splits one of the current components into two smaller ones. This guarantees that when we sampling a different component setup, we still use the same assignment structure as the previous one.

The problem turns out to find a good similarity estimate between two components which are the groups of word vectors. Denote $S(H,G)$ is the similarity measure function between two groups of word vectors H and G. In other words, the magnitude of S based on the similarity of word distributions on H and G. With each group, we represent it by the means of all vectors in this group. In practice, we use the convenient cosine distance between word vectors. So that

$$s(H,G) = \frac{\sum_{k=1}^{d} H_i G_i}{\sqrt{\sum_{k=1}^{d} H_i^2}\sqrt{\sum_{k=1}^{d} G_i^2}} \tag{2.11}$$

Classifier constructs a tree in each label value, then we decide how components are assigned to them.

We continue with the k-means method. In the same logic as hierarchical tree assignment, we construct a k-means clustering on each label value. The

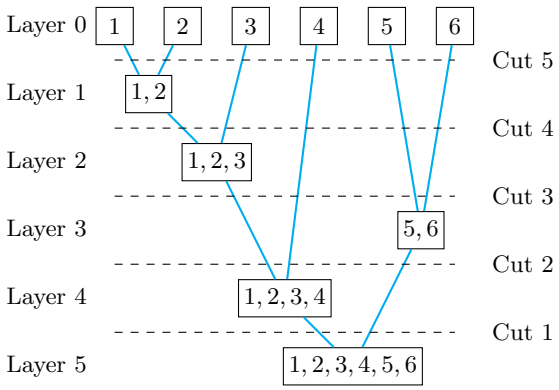


Figure 2.1: Example of an agglomerative tree for one class.

method disjoints the data instance of the same label value into the components by minimizing the variance of all data in the same component—minimize the within-cluster sum of squares. For a given label value y, denote $\mu^{(j)}$ is the vector mean of all instances in M_j such that $P(y|M_j) = 1$, we have the target of k-means clustering given as

$$\sum_{i=1}^{l} \min_{M_j \in M} \sum_{k=1}^{d} (x_k^{(i)} - \mu_k^{(j)})^2 \tag{2.12}$$

The next section will provide experimental evidence for our claim on this unsupervised initialization assertion.

2.4 Experimental Results

We set the experiment on three different tasks of 20Newsgroup data set including comp versus rec groups, 8870 instances; comp versus sci groups, 8843 instances and comp versus talk groups, 8144 instances. These are the major label groups in the data set. The text data was constructed with tf-idf vectorizer, the size of vocabulary reduced to 600 words by sum of tf-idf.

In all data cases, we first separate the data into train-test sets with a fixed ratio at 60:40, 60% of all data for train set and the remaining for test set. After that, the train set continues to split in labeled and unlabeled sets with different split ratio. Both splitting steps also use 5-fold cross validation. We used the splitting ratio of labeled and unlabeled data on train set by the percentage of number of instances in the unlabeled set. For example, the split-70 indicates a split that divides 70% of instances of train data for unlabeled and 30% left for labeled. Figure 2.2 gives the overall view of this scheme. In our experiments, we constructed each data case with 3 splits of split-70, split-90 and split-97. In total we have 5-fold for train-test split and 5-fold for each of 3 labeled-unlabeled splitting, $(5 \times 5 \times 3) = 75$ cases for each classifier.

The experiments were examined with supervised multinomial Naive Bayes (NB) and the corresponding semi-supervised model with three different initialization methods. They are the conventional randomized (EM-random), hierarchical tree (EM-tree) and k-means clustering (EM-kmeans) assignments. The

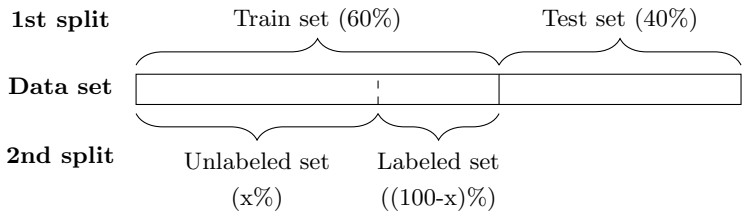


Figure 2.2: The split-x on a specific data case for multinomial models.

search component numbers are in $[1, 15]$. Ultimately, we have 3 data cases with 4 inference models, $(3 \times 4) = 12$ classifiers. Since the results are mostly balanced between precision and recall criteria, we only show the measurement of f1-score here. Accordingly, table 2.1, 2.2 and 2.3 present the results on comp versus rec, comp versus sci and comp versus talk respectively. The complete table of experimental results can be found in appendix A.

In all of the data cases and splits, we can observe that the semi-supervised models outperform the supervised NB model. We now consider only the different component initialization methods of semi-supervised classifiers. Starting with the split-70, the performance between the clustering assignments—EM-tree and EM-kmeans—and randomized assignment—EM-random—are merely the same. It is better in some cases, table 2.1, but lower or equal in the others, and the differences are small. The same situation is also in split-90. Finally, with a narrow source of labeled data in split-97, it gives us more clarity to see the advantage of clustering assignment over the random method. Where in most of the cases, we have the better performances in EM-tree and EM-kmeans. In addition, we may notice that in all of the setup, the models show a good adaption with the changes of unlabeled data. On a different scale, the classifiers still keep it performing with small variances.

Table 2.1: Multinomial model evaluation, 20Newsgroups data, f1-score measurement, comp versus rec.

Label	**NB**	**EM-random**	**EM-tree**	**EM-kmeans**
Split-70, labeled:unlabeled = 1064:2484				
−1	0.83	0.86	0.87	0.87
+1	0.77	0.79	0.82	0.81
avg.	0.80	0.82	0.85	0.84
Split-90, labeled:unlabeled = 354:3194				
−1	0.78	0.86	0.86	0.86
+1	0.77	0.80	0.80	0.80
avg.	0.78	0.83	0.83	0.83
Split-97, labeled:unlabeled = 106:3442				
−1	0.67	0.77	0.83	0.78
+1	0.66	0.70	0.72	0.76
avg.	0.66	0.74	0.77	0.77

Table 2.2: Multinomial model evaluation, 20Newsgroups data, f1-score measurement, comp versus sci.

Label	**NB**	**EM-random**	**EM-tree**	**EM-kmeans**
Split-70, labeled:unlabeled = 1061:2476				
−1	0.90	0.92	0.92	0.92
+1	0.85	0.89	0.89	0.89
avg.	0.88	0.91	0.91	0.90
Split-90, labeled:unlabeled = 353:3184				
−1	0.91	0.92	0.91	0.91
+1	0.88	0.89	0.89	0.89
avg.	0.89	0.90	0.90	0.90
Split-97, labeled:unlabeled = 106:3431				
−1	0.88	0.89	0.91	0.91
+1	0.83	0.82	0.88	0.88
avg.	0.86	0.86	0.89	0.89

Table 2.3: Multinomial model evaluation, 20Newsgroups data, f1-score measurement, comp versus talk.

Label	**NB**	**EM-random**	**EM-tree**	**EM-kmeans**
Split-70, labeled:unlabeled = 977:2280				
−1	0.94	0.95	0.94	0.95
+1	0.91	0.91	0.91	0.91
avg.	0.93	0.93	0.93	0.93
Split-90, labeled:unlabeled = 325:2932				
−1	0.94	0.94	0.94	0.94
+1	0.89	0.91	0.90	0.91
avg.	0.91	0.93	0.92	0.93
Split-97, labeled:unlabeled = 97:3160				
−1	0.91	0.93	0.94	0.94
+1	0.84	0.87	0.91	0.91
avg.	0.88	0.90	0.92	0.92

Chapter 3

Components Separation with Graph

We construct a graph $\mathcal{G}$ from the set of instances $\mathcal{X}$, where each data instance is a vertex and the edges represent the relationship between vertices. A model builds on this graph, combines with the information of labeled vertices to estimate the label for unlabeled vertices. In semi-supervised learning, the inferences that utilize the graph structure like this are called graph-based methods.

As an application of graph-based methods, in this chapter, we consider our data assumption in the form of an objective function on $\mathcal{G}$. Then the inference process aims to recognize the boundary between groups of different label vertices.

3.1 Graph Construction

Basically most graph-based methods are transductive learning. A common graph-based setup consists of two basic steps. The first is graph construction where the data is mapped to a graph, $\mathcal{X} \to \mathcal{G}$. Then in graph inference, we estimate labels for unlabeled data, $\{\mathcal{G}, \mathcal{Y}_L\} \to \mathcal{Y}_U$.

In this chapter, we discuss the first step of graph construction. Denote that n is the total number of instances, $n \triangleq l + u$. We initialize a graph $\mathcal{G} = (V, E, W)$, where $V = \{1, ..., n\}$ is the set of vertices; $E \subseteq V \times V$ is the set of edges and W is a $n \times n$ weight matrix on E. First, we need to establish and clarify the basic conditions on $\mathcal{G}$

$\mathcal{G}$ is a simple weighted graph: $\mathcal{G}$ does not contain self-loop and multiple edges; each edge is undirected and assigned with a weight value.

V is constructed from data instances: V is constructed by an one-to-one correspondence with data instances and its label, $\{(\mathcal{X}, \mathcal{Y})\} \to V$. This means each vertex contains information of an instance and its corresponding label. In practice, $\mathcal{Y}_U$ is initialized as non-clarified labels $\mathcal{Y}_U = \{0\}^u$.

W is positive and symmetric: $W \in \mathbb{R}^{n \times n}, W > 0$ and $W_{i,j} = W_{j,i}$. If $W_{i,j} = 0$ that means there is no edge between i and j. In this thesis, we consider the weight on an edge between two vertices as the similarity between them. In other words, the larger the weight is, the more similar the two vertices.

Because the set of vertices V is fixed, the remaining task involves estimating E and W. Actually we only take into account the estimation of W, because it entirely defines the setup of E.

Typically the graph construction process contains two smaller steps: graph sparsification and graph re-weighting. Denote a similarity measure $S : V \times V \to \mathbb{R}^{n \times n}$ is a function that returns the similarity between two vertices. From the beginning, we initialize $\mathcal{G}$ as a fully connected graph $W_{i,j} = S(x^{(i)}, x^{(j)})$. In graph sparsification, we intend to sparsify W through a binary mass matrix $P \in \{0, 1\}^{n \times n}$, where $P_{i,j} = 1$ means there is an edge between i and j and 0 otherwise, start with P of all zeros. There are many sparsification algorithms, though we only discuss the primary two that will be set in our practice

K-nearest neighbors (kNN): The typical kNN sparsification searches for the k most similar vertices of each vertex. We set $P_{i,j} = 1$ if j is one of the most similar neighbors of i. Also, if i is a selected neighbor of j, then the reverse state does not hold. Thus to meet the symmetry condition of W, after searching, we have to re-symmetrize $P = max(P, P^T)$. This may lead to the misconception of the result kNN graph when after all we have $\sum_j P_{i,j} \geq k$, which means the graph would have more edges than expected.

Maximum spanning tree (MST): In this construction, we look for a spanning tree—a tree containing all the vertices in $\mathcal{G}$—that has the highest sum of weights. We may further expand this structure by executing the MST on each component of the kNN graph. The benefit of this construction comes from its spanning tree structure, where we always receive the graph with a fixed number of $(n - 1)$ edges. But this might be a trade-off if the graph is sparse, it can cause the lost information of vertices connection.

Continuing with the graph re-weighting, from the solution matrix P, we continue to re-estimate the weight of the selected edges to produce the final matrix W. The motivation of this task is to focus on some target characteristics that we aim at inference step and try to omit the misbehavior of P from sparsification step. There are also some possible approaches to this task [13], [14]. For simplification purposes, we utilize the simplest schedule using the element-wise product to set $W = W \cdot P$. The graph construction process is summarized in algorithm 2. It takes the instances set $\mathcal{X}$, label of labeled instances $\mathcal{Y}_L$ and similarity measure S as input and outputs a graph as a pair of vertices set and weight matrix (V, W).

Algorithm 2 Graph construction process

function GRAPH_CONSTRUCTION($\mathcal{X}, \mathcal{Y}_L, S$)
  $\mathcal{Y}_U \leftarrow \{0\}^u$ // Initialize label for unlabeled instances
  $\mathcal{Y} \leftarrow \mathcal{Y}_L \cup \mathcal{Y}_U$ // Merge the complete label sets
  $V \leftarrow \{(\mathcal{X}, \mathcal{Y})\}$ // Set V as set of (instance, label)
  $W \leftarrow S(V \times V)$ // Initialize W by similarity measure S
  $P = Sparsify(W)$ // Sparsify W
  $W \leftarrow W \cdot P$ // Re-weigh W
  return (V, W)

So far we have been constructing the graph. In the next section we will discuss how to inferring the labels for unlabeled vertices.

3.2 Minimum Cut Inference

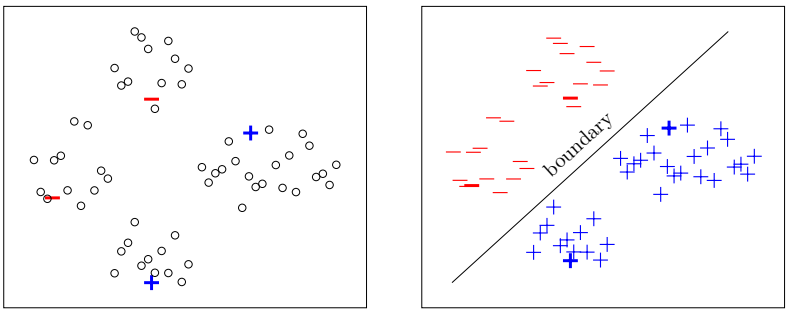
Given our graph after construction $\mathcal{G}$. We presume that the vertices set V is the union of V_L and V_U respectively are the subsets of labeled and unlabeled vertices, $V = V_L \cup V_U$. Moreover, we assume $U_L = U_{L^+} \cup U_{L^-}$, where U_{L^+} is the positive vertices set and U_{L^-} is the negative vertices set of labeled vertices. We construct a simple target function

$$\text{argmin}_f \quad \frac{1}{2} \sum_{(i,j)\in E} W_{i,j} |f(i) - f(j)| \tag{3.1}$$

where $f(i) \in \{-1, +1\}$ represents labels for $i \in V$. With $i \in V_L$, $f(i) = -1$ if $i \in U_{L^-}$ and otherwise, $f(i) = +1$ if $i \in U_{L^+}$.

Now let's consider the intuition viewing of (3.1). In [6], we have an interesting expression of our implied generative framework in there. We assume that the underlying distribution is a fixed number of combined components—groups of vertices—and each has a unique label. Components are separated by a minimum dissimilarity value. An instance is generated by randomly picking a vertex in one of these components then giving it the label of component which it belongs to.

The hypothesis here is that with proper graph construction, there is a sparse connection between two components and more densely connected edges inside oneself. Then with a large enough unlabeled data, the component will be more dense and the connection between them will be the loosest one. So they become the boundary to separate the components. Thus, with only a few labeled vertices on each component, we can label all of them. Figure 3.1 gives an illustration of this view with four regions of positive and negative labels.



(a) (b)

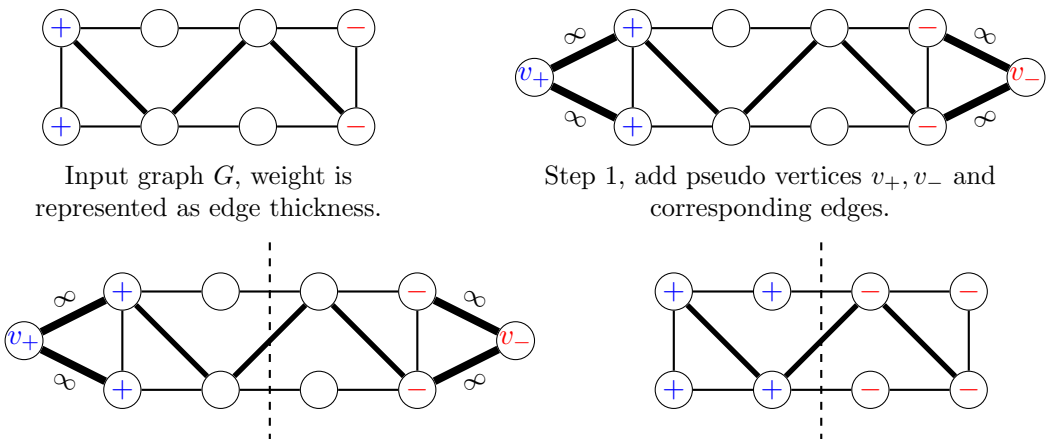
Figure 3.1: Illustration of the combination of graph components assumption. There are two positive and two negative regions. When there is enough unlabeled data, each component will form as a dense graph—figure (a). Then with only one labeled vertex on each component, we can decide the boundary and label all unlabeled vertices—figure (b).

We are now looking for the inference which can be adapted to the optimization condition (3.1) above. Literally, the solution is a configuration of $f(i), i \in V_U$ that guarantees the minimization. With a small set of unlabeled data, a brute-force approach that finds all the 2^u configurations of unlabeled vertices seems like a possible solution. But the basic condition of semi-supervised learning is a large scale of u. A common solution for this inference is the mincut approach using a minimum cut on $\mathcal{G}$ [6]. It finds a minimum cut on which it separates all positive

and negative vertices. Recall a cut $C = \{C_1, C_2\}$ on $\mathcal{G}$ is a segmentation of V such that $C_1 \cap C_2 = \emptyset, C_1 \cup C_2 = V$. The size of cut C is the total weight of edges between C_1 and C_2. More precisely, denote W_C is the size of the cut, we have $W_C = \sum_{i \in C_1, j \in C_2} W_{i,j}$. Given a pair of source and sink vertices (s, t), the minimum cut algorithm finds a cut with smallest weight such that $s \in C_1$ and $t \in C_2$. The solution for finding minimum cut is a well-known algorithm in graph theory and can be archived in polynomial running time. The detailed descriptions and tutorials of this algorithm can be found in [15], [16].

Define that $MinCut(\mathcal{G}, s, t)$ is a function which searches for a minimum cut on $\mathcal{G}$ with source vertex t and sink vertex s. The output returns a cut C with $s \in C_1$ and $t \in C_2$. The mincut approach starts by adding two new pseudo vertices v_+ and v_- to V, then connects v_+ with all positive and v_- with all negative vertices. These pseudo vertices are treated as pins that keep all positive and negative vertices in separate parts. When we have a minimum cut on $\mathcal{G}$ with v_+ and v_- are source and sink, MinCut$(\mathcal{G}, v_+, v_-)$. We can label all unlabeled vertices using its correspondence with v_+ or v_-. All vertices in the part containing v_+ will have the positive label and all vertices in the part of v_- will have the negative label. Figure 3.2 illustrates the process of mincut approach and we summarize the process in algorithm 3. It receives the graph $\mathcal{G}$ as input and returns the vertices set V after labeling all unlabeled vertices.

It is obvious that the complexity of this approach mainly comes from the function MinCut$(\mathcal{G}, s, t)$. Constantly, the problem of finding a minimum s-t cut in graph is similar to looking for a maximum flow on $\mathcal{G}$. For a quick implementation, we may take the well known Edmonds–Karp algorithm [17] which costs us $\mathcal{O}(n|E|^2)$ to find the solution. One of the more advanced methods, the push-relabel algorithm [18], is commonly supported in difference graph frameworks having the complexity of $\mathcal{O}(n|E| \log \frac{n^2}{|E|})$. There are also other methods that would save more time, but they are far more complicated to be considered in our meaning time.



Input graph G, weight is represented as edge thickness.

Step 1, add pseudo vertices v_+, v_- and corresponding edges.

Step 2, find a minimum cut in G with source v_+ and sink v_-.

Step 3, use this cut to label unlabeled vertices.

Figure 3.2: Example of mincut approach.

Algorithm 3 Semi-supervised mincut approach

function MINCUT_SSL($\mathcal{G}, s, t$)
    `# Step 1`
    $V \leftarrow V \cup \{v_+, v_-\}$ // Add in $\mathcal{G}$ two new vertices v_+ and v_-
    for $i \in V_{L^+}$ **do** // For each positive vertices:
        $W_{i,v_+} \leftarrow +\infty$ // Connect to v_+ with $+\infty$ weight
    for $i \in V_{L^-}$ **do** // For each negative vertices:
        $W_{i,v_-} \leftarrow +\infty$ // Connect to v_- with $+\infty$ weight
    `# Step 2`
    $C = MinCut(\mathcal{G}, v_+, v_-)$ // Find a minimum cut on $\mathcal{G}$
    `# Step 3`
    for $i \in C_1$ **do** // For each vertex in the part contains v_+:
        $V_{L^+} \leftarrow V_{L^+} \cup i$ // Add to the positive label set
    for $i \in C_2$ **do** // For each vertex in the part contains v_-:
        $V_{L^-} \leftarrow V_{L^-} \cup i$ // Add to the negative label set
    $V \leftarrow V_{L^+} \cup V_{L^-}$ // V is now all labeled
    return V

In the case of having more than one minimum cut, which means more than one configuration of f that matches the condition (3.1), the mincut approach is prone to a cut that is close to the positive or negative region. Thus this label will have a smaller proportion of vertices. For instance, in figure 3.3, the selected cut is more likely to be C_1 or C_2 than others. From an overall perspective, this missing behavior may not tell much about the performance of this approach, since we do not know the exact global proportion of data set. However, many experiments had been conducted on this problem and showed that it is better to have the solution with more balance between labels [6], [19]–[21]. Furthermore, the balanced cut solution has been proved to be an NP-hard problem [21].

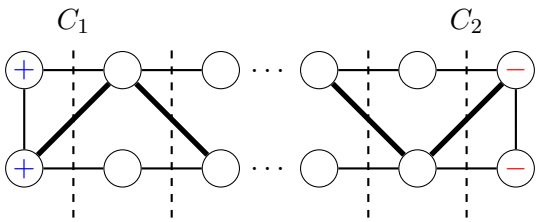


Figure 3.3: Example of a graph has more than one minimum cut.

In [19], we have an idea of the *randomized mincut approach*. The method repeatedly disturbs the edges set E by adding a small amount of noise into the weight matrix. At each iteration, it finds a new minimum cut. The cut is then verified and will be removed if it gives an unbalanced prediction on the train set. Finally, with a set of cuts, the label will be decided on the majority agreements of all cuts. However, through our experimental investigation, this method has no advantage over the original mincut approach. Following this unbalanced issue, in the next chapter, we are going to examine an alternative solution for the mincut approach.

3.3 Graphical Model

A possible replacement for the mincut approach employs a graphical model. This idea was raised by Blum and Chawla [6] and had the first fundamental experimental verification with a tree graph construction in [19]. Graphical model is a wide range of research field which is approaching the graph structure on exploring the connected relationship of a set of random variables. This thesis does not try to cover the general technical terms and algorithms of this model. Accordingly, it only focuses on the application of this model in case of solving the target function on binary classification. Gentle introductions, tutorials and examples for graphical model can be found in [22]–[24].

Currently, the conventional deployment of graphical model on (3.1) only works with a graph forming as a tree using *max-sum* algorithm. The algorithm itself does not have much ability to adapt to the balancing problem. This thesis introduces the *influence index* that supports the inference decision on each vertex when we have more than one applicable decision. In addition, there is no affordable learning algorithm that can work on an arbitrary graph. To complement the conventional approach, we extend this approach in the case of graph having cycles using the approximate inference algorithm *loopy belief propagation*. Combine them together and we have a complete alternative solution for the mincut approach that can be assembled with general graph constructions. Algorithm 4 gives a brief sketch for this combination.

Algorithm 4 Graphical model for arbitrary graph using influence index

```
function GraphicalModel_SSL(G, y_L)
    influence ← <Extract influence index for i ∈ V_U>
    if G is Tree then
        inference ← Max_Sum(G, y_L, influence)
    else
        inference ← Loopy_Belief(G, y_L)
    y_U ← <Trace back label from inference>
    return y_U
```

3.3.1 Inference on Tree

To keep this chapter consistent and self-contained, we first go through with the model setup and the fundamental idea of the max-sum algorithm. Suppose we have a set of random variables $X = \{X_i : i \in V, X_i = f(i)\}$. Define a set of factors

$$\Phi = \{\phi_{i,j} : (i,j) \in E, \phi_{i,j} = \exp(-W_{i,j}(X_i - X_j)^2)\} \tag{3.2}$$

We assume that the implied distribution of X_i only depends on Φ; $\phi(u,v)$ is independent at each other. We can define our model as a joint distribution

$$P_\Phi(X) = \frac{1}{Z} \prod_{(i,j)\in E} \phi_{i,j} \tag{3.3}$$

where the normailizing constant (partition function)

$$Z = \sum_{X} \prod_{(i,j)\in E} \phi_{i,j}$$

Consider the inference to maximize the joint distribution

$$\begin{aligned} \text{argmax}_X\, P_\Phi(X) &= \text{argmax}_X\, \frac{1}{Z} \prod_{(i,j)\in E} \phi_{i,j} \\ &= \text{argmax}_X \prod_{(i,j)\in E} \phi_{i,j} \end{aligned} \tag{3.4}$$

Take the logarithm, then we have

$$\begin{aligned} (3.4) &= \text{argmax}_X \ln(\prod_{(i,j)\in E} \phi_{i,j}) \\ &= \text{argmax}_X \ln(\prod_{(i,j)\in E} \exp(-W_{i,j}(X_i - X_j)^2)) \\ &= \text{argmax}_X \sum_{(u,v)\in E} -W_{i,j}(X_i - X_j)^2 \end{aligned} \tag{3.5}$$

Combine the condition $W_{u,v}(X_u - X_v)^2 \geq 0$, then from (3.5) we can obtain that the result of this inference is similar to our target function (3.1).

Looking back at our inference (3.4), define X_{max} is an observed set of X that maximizes the joint distribution, we have

$$X_{max} = \text{argmax}_X\, P_\Phi(X) \tag{3.6}$$

Then from (3.3), the maximize probability can be represented as

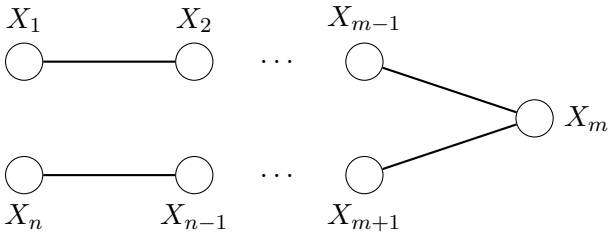
$$\begin{aligned} P_\Phi(X_{max}) &= \max_X P_\Phi \\ &= \max_{X_1,\ldots,X_n} \prod_{(i,j)\in E} \phi_{i,j} \end{aligned} \tag{3.7}$$

Let's consider a simple case when we have a tree as a chain of vertices in figure 3.4a, we have

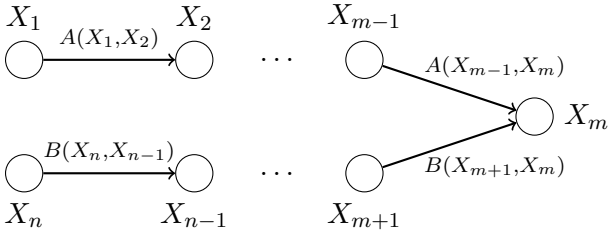
$$(3.7) \Leftrightarrow \max_{X_1,\ldots,X_n} (\phi_{1,2}\ \phi_{2,3} \ldots \phi_{m-1,m}\ \phi_{m,m+1} \ldots \phi_{n-1,n}) \tag{3.8}$$

$$= \max_{X_1} \left[\max_{X_2} \left[\ldots \left[\max_{X_n} (\phi_{1,2}\ \phi_{2,3} \ldots \phi_{m-1,m}\ \phi_{m,m+1} \ldots \phi_{n-1,n}) \right] \ldots \right] \right] \tag{3.9}$$

We first assume that all the random variables are unobserved. Follow the maximization of all random variables in (3.9) gives a brute force approach that takes all positive combination of X having 2^n cases. Now we are looking for a better way to compute it. We may have noticed that each maximization on a random variable only affects on the factor functions which have this variable as one of its parameters and there are at most two factors that is related to a maximization. For example, the maximization on X_m only affects on $\phi_{m-1,m}$ and $\phi_{m,m+1}$. Furthermore, in figure 3.4a, we have two special random variables that



(a) A simple graph forms a chain of vertices.



(b) Messages send from X_1 to X_m and from X_n to X_m

Figure 3.4: Inference in a chain of vertices.

belong to only one factor, X_1 and X_n. Also, they are corresponding with two leaf vertices on this tree. Then we can rearrange (3.9) to group only related factors with its maximization, so that

$$\begin{aligned}(3.9) = \max_{X_m}\Bigg[&\max_{X_{m-1}} \phi_{m-1,m}\left[\dots\left[\max_{X_2}\phi_{2,3}\left(\max_{X_1}\phi_{1,2}\right)\right]\dots\right] \\ &\max_{X_{m+1}} \phi_{m+1,m}\left[\dots\left[\max_{X_{n-1}}\phi_{n-1,n-2}\left(\max_{X_n}\phi_{n,n-1}\right)\right]\dots\right]\Bigg]\end{aligned} \tag{3.10}$$

We may observe that (3.10) contains two parts, the first part is group of maximization of $[X_1, ..., X_{m-1}]$ and the second part is of $[X_{m+1}, ..., X_n]$. Consider the first part, we define a function A recursively

$$\begin{aligned}&A(X_1, X_2) = \max_{X_1} \phi_{1,2}, \\ &A(X_2, X_3) = \max_{X_2} (\phi_{2,3} A(X_1, X_2)), \\ &..., \\ &A(X_{m-1}, X_m) = \max_{X_{m-1}} (\phi_{m-1,m} A(X_{m-2}, X_{m-1}))\end{aligned}$$

Similarly, we define a function B for the second group

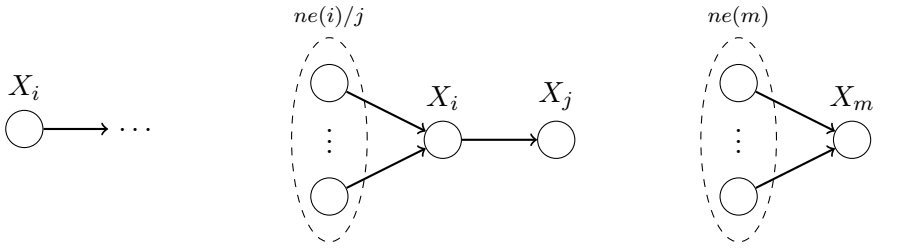
$$\begin{aligned}&B(X_n, X_{n-1}) = \max_{X_n} \phi_{n,n-1}, \\ &B(X_{n-1}, X_{n-2}) = \max_{X_{n-1}} (\phi_{n-1,n-2} B(X_n, X_{n-1})), \\ &..., \\ &B(X_{m+1}, X_m) = \max_{X_{m+1}} (\phi_{m+1,m} B(X_{m+2}, X_{m+1}))\end{aligned}$$

Combine them together and we have

$$(3.10) \Leftrightarrow \max_{X_m} (A(X_{m-1}, X_m) B(X_{m+1}, X_m)) \tag{3.11}$$

Consequently, the inference through recursive functions A and B reduces our calculation significantly. The size of each maximization now is $2^2 = 4$ cases—a factor that inputs 2 variables and each has value in $\{-1, 1\}$—and we only need to execute it $|E|$ times. In addition, we may consider A and B as messages sent on the graph, A is messages sent from vertex of X_1 to X_m and B is messages sent from vertex of X_n to X_m. The figure 3.4b shows this view of message sending process.

Let's formalize this idea to a general tree. At first, on each edge, we need to decide a sending schedule where message will be sent starting from the leaf vertices to a root. Moreover, the sending schedule follows the condition that before a message is sent from a vertex, it must receive all the messages sent to it. We can handle this schedule using a breadth-first search starting from any vertex as the root and invert the search route to get the sending schedule. Figure 3.5 summarizes the different types of vertices we need to handle in this sending process.



(a) A leaf vertex (b) A typical vertex (c) The root vertex

Figure 3.5: Types of vertices on a tree.

Assume that a message is sent from i to j. Then on edge $(i, j) \in E$ which is not a leave or root vertex, we define the message

$$\psi(X_i, X_j) = \max_{X_i} \left[\phi_{i,j} \prod_{k \in ne(i)/j} \psi(X_k, X_i) \right] \tag{3.12}$$

Where $ne(i)$ is a function returns set of all neighbor vertices of i. In practice, we utilize the logarithm function to leverage our calculation by converting between product and sum. This approach names the max-sum algorithm. So that

$$(3.12) \Leftrightarrow \psi(X_i, X_j) = \max_{X_i} \left[\phi_{i,j} \sum_{k \in ne(i)/j} \ln\left(\psi(X_k, X_i)\right) \right] \tag{3.13}$$

In case of i is a leaf, we have

$$\psi(X_i, X_j) = \max_{X_i} \left(\ln\left(\phi_{i,j}\right)\right) \tag{3.14}$$

Finally, after all the messages are sent, suppose our root is m, then we take all the neighbor messages and get the maximum result

$$P_\Phi(X_{max}) = \max_{X_m} \sum_{k \in ne(m)} \psi(X_k, X_m) \tag{3.15}$$

After we have done with sending message to the root, we can trace back this root to get the corresponding values of X_{max}, we will discuss this process further in next section. Furthermore, in the case X_i is observed—labeled vertex. We directly assign this value to get the message without taking the maximization.

3.3.2 Influence Index and Loopy Belief Propagation

When we release the assumption of the minimum distance between components to accepting the overlaps but not collapse of the boundary between the components. We have the problem of having more than one possible solution with our target function. With the mincut approach, we cannot intervene in the finding minimum cut process and can only determine the final cut value until we have finished the flow detecting process. In the case of max-sum algorithm, this can be detected when we trace back the result in the maximization of (3.13) and (3.15).

Usually, when we have equal inputs and look for the maximization index, the function will return a default index or randomly choose the result. This does not affect the final result because these all produce maximum probability. But while the number of equal occasions notably increases, this default behavior may cause a degenerate solution when all equal probabilities are set to the same label. Therefore, it is better to have a second criteria to decide which one should have higher priority in this situation. To bypass this problem, we heuristically define an influence index for each vertex using the total weight by the similarity measure from graph construction S. The similarity takes between unlabeled vertices V_U and sets of positive and negative label vertices. If there is a tide decision to trace back the process, the decision will be made on which label has a larger total weight. So that, we define

$$\text{influence}_{i\in V_U} = \text{argmax}_{X_i} \sum_{X_i = X_j, j\in V_L} S(X_i, X_j) \tag{3.16}$$

Because the max-sum algorithm sends a message on each edge only one time, hence it is convenient for us to catch the trace right after a message is sent. We denote that the trace on an unlabeled vertex i is the decision label to our message $\psi(X_i, X_j)$. Then we define the trace on each edge

$$\text{trace}_{i\in V_U} = \begin{cases} \text{influence}_i & \text{if } \psi(+1, X_j) = \psi(-1, X_j) \\ \text{argmax}_{X_i}\, \psi(X_i, X_j) & \text{otherwise.} \end{cases} \tag{3.17}$$

The same implementation is applied for root m to decide the label at root vertex. For $k \in ne(m)$, we have

$$\text{trace}_m = \begin{cases} \text{influence}_m & \text{if } \sum_k \psi(X_k, +1) = \sum_k \psi(X_k, -1) \\ \text{argmax}_{X_m} \sum_k \psi(X_k, X_m) & \text{otherwise.} \end{cases} \tag{3.18}$$

Algorithm 5 summarizes the message sending process of max-sum algorithm using influence index. Start by constructing the sending schedule by reversing the breadth-first search with default root V_0. Then follow the edges list in sending schedule, the algorithm defines the message on these edges and sets trace for unlabeled vertices. Finally, it returns the sent messages list.

Algorithm 5 Max-sum algorithm using influence index

function MAX_SUM($\mathcal{G}, y_L$, influence)
  schedule $\leftarrow$ reverse_BFS(V_0)
  for $(i, j) \in$ schedule **do**
    if i **is** Leaf **then**
      $\psi[X_i, X_j] = \max_{X_i} (\ln (\phi_{i,j}))$
    else
      $\psi[X_i, X_j] = \max_{X_i} \left(\phi_{i,j} \sum_{k \in ne(i)/j} \ln (\psi[X_k, X_i])\right)$
      if $i \in V_U$ **then**
        Set trace[i]
  return ψ

In general, the additional influence index does not affect the complexity of our model, we can handle with its calculation in graph construction before going to the inference process. Now, consider the implemented algorithm 5, the cost for building the sending schedule is $\mathcal{O}(|E|)$ when we use the adjacency list to store the edges. Then the complexity of this algorithm depends on the loop of sending messages. This loop only iterate $|E|$ times with the messages only sends one time on each edge. Each time, we only send one message, if it is sent from a leaf, the complexity is constant when finding the maximize of $2^2 = 4$ inputs. Moreover, because the factors are fixed, then we can re-estimate all these messages before and the complexity reduce to constant $\mathcal{O}(1)$. In the case of other vertices, we sum over the neighbors and maximize the same 4 inputs. The summation is taken on all neighbors vertices, each is checked one time, then in total we have to check them $2|E|$ times. Finally, we have our sending process complexity is $\mathcal{O}(|E|^2)$.

Now we extend our work to an arbitrary graph. In this case, we employ the loopy belief propagation algorithm and integrate with our influence index. The loopy belief propagation is an approximate algorithm which does not guarantee the convergence of joint probability. The algorithm is merely an extension of the max-sum algorithm when the messages are sent in both directions. It is possible that we have a positive solution with convergence on most of the vertices when the max-sum algorithm only handles the message locally where each message only depends on its neighbors. There are many works that have been done on this algorithm and tried to extend the range of graph types and also arbitrary graph with positive results [25]–[28].

Let's define the message sending process. Now we have an arbitrary graph with cycles and there is no leaf or root. It is certain that some path of the bridge edges which are outside of the cycles can have the exact inference as in a tree, but now we are supposed to only take into consideration messages on the cycles. For each edge (i, j), we define two messages using the same formula in (3.13) on both directions, $\psi(X_i, X_j)$ and $\psi(X_j, X_i)$. On a cycle, a message, say $\psi(X_i, X_j)$, would be re-estimated many times and this process should be triggered every time one of a neighbor of the two ends vertices $\psi(X_{ne(i)/j}, X_i)$ is updated.

At first, we initialize all messages on each vertex and directions using the formula of (3.14). The process of sending messages is repeatedly executed. In this measure, there are two common schedules for sending messages on a cycle [24]. The first is serials schedule when we only send one message at each time. The second, which will be implemented here, is a flooding schedule. At each iteration,

we send messages on all edges and in both directions. This schedule makes it easier to follow the track of our messages and utilize our influence index. We can determine whenever a message from a labeled vertex is sent to an unlabeled one. Since a message is being sent many times, we can not catch the trace immediately through its process. At later when the sending process is finished, the trace of label on each vertex is determined by sum over all messages sending from its neighbors, same as the calculation on root vertex of max-sum algorithm in (3.18) where we also deploy our influence index.

Finally, there must be a state to stop the loop or a convergence condition. In our case, there is no rational evidence to check whether all or major parts of the vertices are converged or they will not. Hence the explicit condition is the loop count threshold. It is reasonable that we should set the limit at least equal to the number of edges $|E|$. In this case, each message has enough time to reach out to all vertices in the graph.

In summary, we have algorithm 6 to shortly describe loopy belief propagation. We first initialize all messages using factors on each edge, these first messages on an edge are similar in both directions. Then we repeatedly send out messages all over the graph at the same time before we reach the threshold condition. With the same logic in algorithm 5, the complexity of loopy belief propagation merely depends on the sending message process. In this case, the complexity exponentially increases with threshold value. Suppose we set the threshold to be the number of edges, then we have the cost of the algorithm is $\mathcal{O}(|V|^3)$.

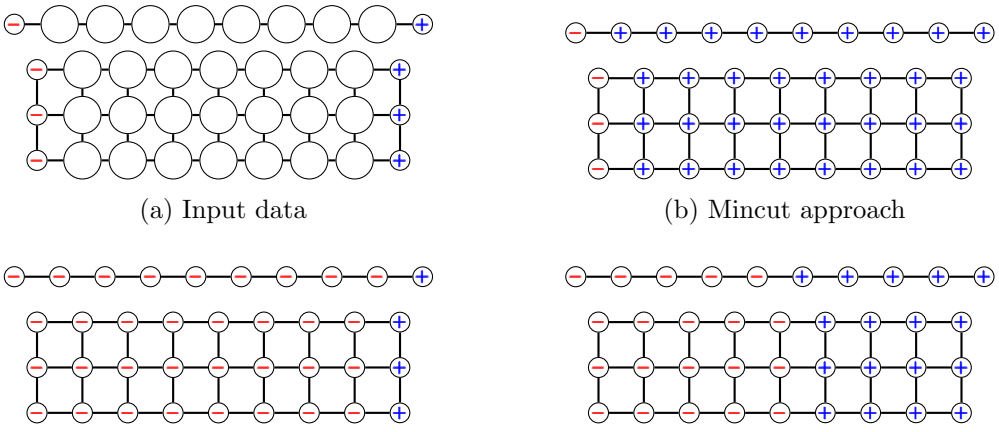
Algorithm 6 Loopy belief propagation algorithm

function LOOPY_BELIEF($\mathcal{G}, y_L$)
  for $(i,j) \in E$ **do**
    $\psi[X_i, X_j] = \max_{X_i} (\ln(\phi_{i,j}))$
    $\psi[X_j, X_i] = \psi[X_i, X_j]$
  loop_count $= 0$
  while loop_count $<$ threshold **do**
    for $(i,j) \in E$ **do**
      $\psi[X_i, X_j] = \max_{X_i} \left(\phi_{i,j} \sum_{k \in ne(i)/j} \ln(\psi[X_k, X_i])\right)$
      $\psi[X_j, X_i] = \max_{X_j} \left(\phi_{j,i} \sum_{k \in ne(j)/i} \ln(\psi[X_k, X_j])\right)$
    loop_count $=$ loop_count $+1$
  return ψ

3.4 Synthetic Data Analysis

We give some illustration examples on synthetic data to see how our discussed models work and to show the advantages when using influence index. The data is constructed in a 2-dimension euclidean space.

Let's begin with chain and grid forms of graph in figure 3.6. We have 2 types of graph here. A chain of 10 continuously connected vertices and a grid of (3×9) vertices. The data are equally separated and the labeled data are distributed to the left and right sides on each graph giving a bunch of unlabeled data in between. So any set of vertical edges would be our separation boundary. These options are treated equally in mincut approach and conventional graphical model.



(a) Input data

(b) Mincut approach

(c) Conventional graphical model

(d) Graphical model using Influence index

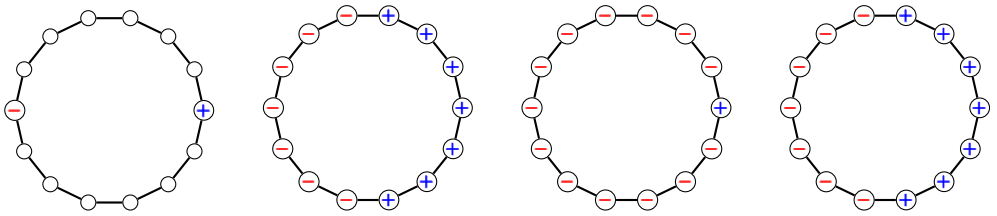
Figure 3.6: Inference in synthetic data forms a chain and grid of vertices.

They prefer the nearest boundary of one of the labeled groups, figure 3.6b and 3.6c. In contrast, the graphical model using the influence index does better when choosing the separation near the centre of the graph, where the summations of total weight to both groups of positive and negative labels are nearly balance, figure 3.6d. Note that in the graphical model, the chain graph is inferred by max-sum algorithm and the grid graph is by loopy belief propagation.

Continue to examine more on the case of loopy belief propagation. From section 3.3.2, we have known that it is possible to detect when a message is balanced in the sending process immediately with max-sum algorithm. But with loopy belief propagation we need to wait until this process is finished and only be able to examine the equal condition when summing up all the messages on each node. In figure 3.7 we have a circle graph with 14 vertices equally separated, which represents a simplified loop in arbitrary graph. With this form of graph, any combination of 2 edges on upper and lower semicircles is a valid boundary to separate the label groups. The influence index still keeps getting a good separation, figure 3.7d.

What we pay attention to is the conventional loop belief propagation in different iterations. The third iteration, figure 3.7b, messages are on the way to send out on both sides. If we get the results right there, it seems be good. But actually, the messages have not fully spread through the graph yet and this is only a lucky gamble decision that we rarely have. So if we continue the loop, from iteration fourth to seventh, figure 3.7c, we have a mix of equal messages from both sides then the decision is set to the default value, negative label in this case. Besides, the interesting fact is if we continue the process many times after then, where all the messages have been received all over the graph and resent again, we still have the same result as at seventh iteration. The reason is when the new round of messages starts to resend again, the sum of all the messages sent to one vertex will sum up with the old one. Then more or less, both directions self-balanced themselves, we have the equal results.

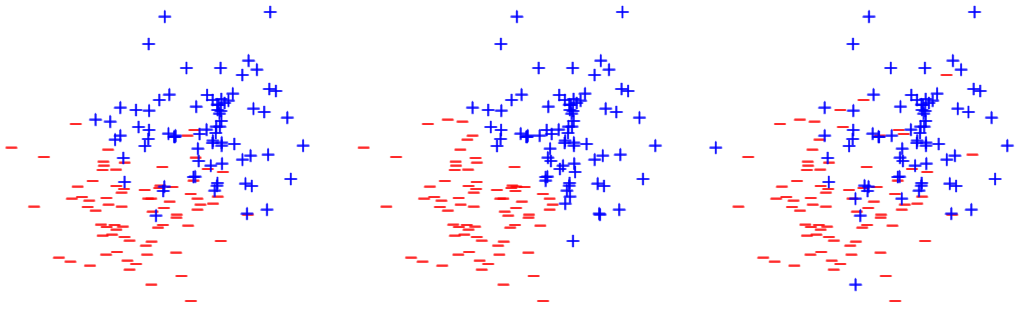
Lastly, we want to see whether all the things we have made are working on our data distribution assumption. Then we only compare the mincut approach and the graphical model using influence index. In figure 3.8, we have a set of



(a) Input data (b) After 3 iterations (c) After 7 iterations (d) Influence index

Figure 3.7: Graphical inference in a graph forms a circle of vertices.

150 instances made from two normal distributions, $\mathcal{N}(1, 0.6^2)$ for positive and $\mathcal{N}(0, 0.6^2)$ for negative label. The labels were equivalently distributed, with 75 instances including 7 labeled and 68 unlabeled for each. The mincut approach, figure 3.8b, gives us a neat boundary of the separation which hardly divides the center density of each group. The graphical model, figure 3.8c, returns in a tangled boundary. Some measurements of this result are in table 3.1. There is not much to be concluded here, where both models are on track to recognize the boundary and the mixing region is still small.



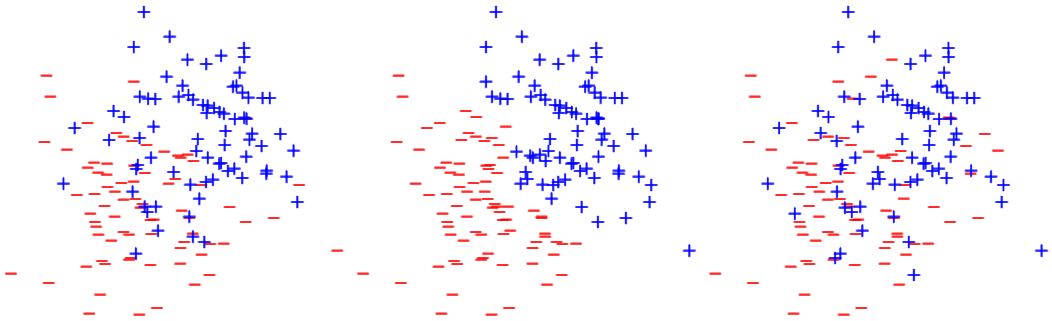
(a) True data (b) Mincut approach (c) Graphical model

Figure 3.8: Inference outputs on $\mathcal{N}(0, 0.6^2)$ and $\mathcal{N}(1, 0.6^2)$. 150 instances, 7 labeled and 68 unlabeled for each label.

Table 3.1: Measurement of inferences on $\mathcal{N}(0, 0.6^2)$ and $\mathcal{N}(1, 0.6^2)$.

Label	Precision	Recall	F1	Precision	Recall	F1
	Mincut approach			**Graphical model**		
-1	0.88	0.87	0.87	0.87	0.91	0.89
$+1$	0.87	0.88	0.88	0.90	0.87	0.89
avg.	0.88	0.88	0.87	0.89	0.89	0.89

Now we twist the two distributions and let them bend more with $\mathcal{N}(0, 0.8^2)$ and $\mathcal{N}(1, 0.8^2)$ on the same set up. The results are in figure 3.9. The true data regions are more chaotic—figure 3.9a—and the mincut approach appears to be shrink to positive region in figure 3.9b. The graphical model using influence index in figure 3.9c seems to have better match in shape of out true data. The estimation indices in table 3.2 also shows us the same view of this illustration. If we try to squeeze these distributions more, the objective condition will be violated and the models have nothing to deal with the case.



(a) True data (b) Mincut approach (c) Graphical model

Figure 3.9: Inference outputs on $\mathcal{N}(0, 0.8^2)$ and $\mathcal{N}(1, 0.8^2)$. 150 instances, 7 labeled and 68 unlabeled for each label.

Table 3.2: Measurement of inferences on $\mathcal{N}(0, 0.8^2)$ and $\mathcal{N}(1, 0.8^2)$.

Label	Precision	Recall	F1	Precision	Recall	F1
	Mincut approach			**Graphical model**		
-1	0.76	0.80	0.78	0.82	0.87	0.85
$+1$	0.79	0.75	0.77	0.86	0.82	0.84
avg.	0.78	0.78	0.78	0.84	0.84	0.84

To give more demonstrations for graphical models is a reasonable replacement for the mincut approach, and to examine our graph-based models in practical data, the next section gives us the experimental results taken on some common data sets.

3.5 Experimental Results

In this section, we evaluate our graph-based models by both views of application: transductive semi-supervised learning when predicting unlabeled data in section 3.5.1 and in compare with some supervised learning in section 3.5.2.

In all data cases, we continue to use the splitting ratio of labeled and unlabeled data on the train set from section 2.4. Because the learning methods are transductive, we entirely treat the data set as train data. Then, using a split we separate the data set into labeled and unlabeled sets. Figure 3.10 illustrates this overall splitting process.

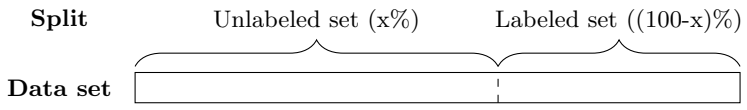


Figure 3.10: The split-x on a specific data cases for graph-based models.

Continuing with similarity measurement, there were 3 types of function in used including Euclidean, cosine and Gaussian kernel using default bandwidth equals to number of features. As graph sparsity methods are described in previous section 3.1, the experiments constructed three different kinds of graph with modifications

kNN graph: Search for the most k similar vertices. The search range for k was $[1, 10]$. In the case of sparse graph, it is possible to have a component with all data that are unlabeled. We bypass this problem by constraining that there must be at least one labeled data in neighbor search.

MST tree: Construct a maximum spanning tree on $\mathcal{G}$. The greedy Prim algorithm [29] was implemented.

kNN-MST graph: Construct a kNN graph first, then find a MST tree for each component of this kNN graph.

We only present the inference results of the mincut approach and graphical model using the influence index—we now short its name to the graphical model. The reason is that the conventional graphical model for our problem is not a completed one, it only works on a tree. Moreover, as has been shown in previous section, it does not have the adaption to our data assumption. Hence more or less, even with only the tree construction, it is no better than the mincut approach. Besides that, we have conducted the evaluation on the randomized mincut approach in section 3.2, and it did not turn well either. The complete output of experiments and more details about the data resources and experiment reproduction are described in appendix A.

3.5.1 Semi-Supervised Evaluation

The semi-supervised evaluation considers the ability of models to predict the unlabeled data in various scaling size of unlabeled set. The model learns all data instances and predicts the label for the unlabeled set. For each experimental data set, the experiment ran on 3 splits, split-70, split-90 and split-97, and the result was an average of 5-folds cross validation. In total we have 3 split sizes and 5-folds cross validation, $(3 \times 5) = 15$ evaluations for each model configuration.

Abalone and Digit data

Abalone data was set on a binary task that determines the age range of abalone is $[5, 9]$ or $[10, 15]$ with 3842 instances, 7 features. The Digit data task was to clarify whether an 8×8 digit image—64 features—is odd or even digit with 1797 instances. Both data were set on Euclidean similarity and Gaussian kernel with all three graph sparsity functions. Here we have 2 data cases, 2 similarity measurements, 3 graph types, $(2 \times 2 \times 3) = 12$ configurations.

The result of Abalone data using Euclidean similarity is in table 3.3, the trend is similar with Gaussian kernel. At first glance, we can observe that the mincut approach has the equivalent or better performance in case of split-70, especially with kNN graph. Here the kNN graph also illustrates its lead over MST and kNN-MST graphs. The situation changes when we significantly shrink the labeled size with split-97. Inform that the split-70 specifies that the unlabeled size is about 2 times bigger than the labeled one, while the split-97 means the difference is now about 32 times. In this case, the graphical model using kNN graph outperforms the mincut approach. Also the MST and kNN-MST graphs take off the gap with kNN graph and the results are nearly the same on both inference models.

We will continue with Digit data. In the same way, table 3.4 shows the results of Digit data with Euclidean similarity. Take a look at graphical model, we may

allegedly observe that, in kNN graph, the loopy belief propagation could not be converged on a group of vertices. This pushes the recall on positive label high up, but downgrades precision significantly compared with the mincut approach, and the reverse situation is applied to negative label. The equivalent outputs are held on MST and kNN-MST graph.

20Newsgroups data

The 20Newsgroups data was constructed with tf-idf vectorization, the size of vocabulary reduced to 600 words by sum of tf-idf. The original data set consists of many groups of different topic labels. Here we set up an experiment between comp versus rec groups, in 8870 instances. We only use cosine similarity measurement with 3 types of graph construction. Thus we have 1 classifier, 1 similarity measurement, 3 graph types, $(1 \times 1 \times 3) = 3$ configurations.

The results are in table 3.5. This time, the performance of kNN graph prevails the other two graphs in both mincut approach and graphical model. With split-70 and split-90, we do not have much to say. But there in split-97, the mincut approach has fallen to a degenerate solution when vastly going for the negative label. If we solely examine the MST and kNN-MST graphs, the slightly better outcome comes from the mincut approach.

3.5.2 Supervised Evaluation

In the same data setup of semi-supervised evaluation, for each data set, we added-in two supervised models including Logistic Regression and Support Vector Machines (SVM, using RBF kernel, $\gamma = \frac{1}{\#features}$) to predict the unlabeled data. In total we have 2 added supervised models, 3 split sizes and 5-folds cross validation, $(2 \times 3 \times 5) = 30$ additional evaluations for each data case.

From the previous evaluation, we would claim the presume that with the carefully selected graph construction for a specific data, the graphical model can have the advantage over the mincut approach. Therefore, in this section, we only compare the supervised model with the graphical model. We also assume that we already evaluated the graph construction methods and utilize the best one here.

In table 3.6 we have the comparison on Abalone data, the graphical model works on Euclidean kNN graph. The table starts with the split-70 where we have merely equivalent outcomes. Then go down with the split-90 and split-97, the logistic regression has the most recognizable downgrade. The graphical model has its solid performance and states first on split-97 in comparison with the other two. A similar result for Digit data is in table 3.7 with Euclidean kNN-MST graph for graphical model. But at this time, the logistic regression has been downed from the beginning. Finally, the result of 20newsgroup data is in table 3.8, graphical model utilizes cosine kNN graph. The supervised keeps its benefits in split-70 and split-90. In split-97, SVM has fallen into a degenerate trap when predicting most of the label for negative. It is the same situation with kNN graph of the mincut approach for this data case.

Table 3.3: Semi-supervised evaluation, Abalone data using Euclidean similarity.

Graph type	Label	Precision	Recall	F1	Precision	Recall	F1
		Mincut approach			**Graphical model**		
Split-70, labeled:unlabeled = 1152:2690							
kNN	−1	0.73	0.79	0.76	0.69	0.79	0.73
	+1	0.80	0.74	0.76	0.78	0.67	0.72
	avg.	0.76	0.76	0.76	0.74	0.73	0.73
MST	−1	0.72	0.71	0.71	0.72	0.71	0.71
	+1	0.74	0.75	0.74	0.74	0.75	0.74
	avg.	0.73	0.73	0.73	0.73	0.73	0.73
kNN-MST	−1	0.71	0.70	0.70	0.71	0.70	0.70
	+1	0.73	0.75	0.74	0.73	0.75	0.74
	avg.	0.72	0.72	0.72	0.72	0.72	0.72
Split-90, labeled:unlabeled = 384:3458							
kNN	−1	0.71	0.74	0.72	0.71	0.70	0.70
	+1	0.76	0.73	0.74	0.73	0.74	0.74
	avg.	0.73	0.73	0.73	0.72	0.72	0.72
MST	−1	0.70	0.67	0.68	0.70	0.67	0.68
	+1	0.71	0.74	0.73	0.71	0.74	0.73
	avg.	0.71	0.70	0.70	0.71	0.70	0.70
kNN-MST	−1	0.69	0.66	0.67	0.69	0.66	0.67
	+1	0.71	0.73	0.72	0.71	0.73	0.72
	avg.	0.70	0.69	0.69	0.70	0.69	0.69
Split-97, labeled:unlabeled = 115:3727							
kNN	−1	0.69	0.66	0.67	0.69	0.73	0.71
	+1	0.70	0.73	0.72	0.74	0.71	0.72
	avg.	0.69	0.69	0.69	0.72	0.72	0.72
MST	−1	0.70	0.64	0.67	0.70	0.64	0.67
	+1	0.70	0.75	0.72	0.70	0.75	0.72
	avg.	0.70	0.70	0.70	0.70	0.70	0.70
kNN-MST	−1	0.69	0.64	0.66	0.69	0.64	0.66
	+1	0.70	0.73	0.71	0.70	0.73	0.71
	avg.	0.69	0.69	0.69	0.69	0.69	0.69

Table 3.4: Semi-supervised evaluation, Digit data using Euclidean similarity.

Graph type	Label	Precision	Recall	F1	Precision	Recall	F1
		Mincut approach			**Graphical model**		
Split-70, data number labeled-unlabeled : 539-1258							
kNN	−1	1.00	0.99	0.99	1.00	0.99	0.99
	+1	0.99	1.00	0.99	0.99	1.00	0.99
	avg.	0.99	0.99	0.99	0.99	0.99	0.99
MST	−1	1.00	0.99	0.99	1.00	0.99	0.99
	+1	0.99	1.00	0.99	0.99	1.00	0.99
	avg.	0.99	0.99	0.99	0.99	0.99	0.99
kNN-MST	−1	1.00	0.99	0.99	1.00	0.99	0.99
	+1	0.99	1.00	0.99	0.99	1.00	0.99
	avg.	0.99	0.99	0.99	0.99	0.99	0.99
Split-90, data number labeled-unlabeled : 179-1618							
kNN	−1	0.98	0.98	0.98	0.93	0.84	0.88
	+1	0.98	0.98	0.98	0.86	0.94	0.90
	avg.	0.98	0.98	0.98	0.89	0.89	0.89
MST	−1	0.99	0.98	0.99	0.99	0.98	0.99
	+1	0.99	0.99	0.99	0.99	0.99	0.99
	avg.	0.99	0.99	0.99	0.99	0.99	0.99
kNN-MST	−1	0.99	0.98	0.99	0.99	0.98	0.99
	+1	0.98	0.99	0.99	0.98	0.99	0.99
	avg.	0.99	0.99	0.99	0.99	0.99	0.99
Split-97, data number labeled-unlabeled : 53-1744							
kNN	−1	0.93	0.97	0.95	0.91	0.77	0.83
	+1	0.97	0.93	0.95	0.80	0.92	0.86
	avg.	0.95	0.95	0.95	0.86	0.84	0.84
MST	−1	0.90	0.98	0.93	0.90	0.98	0.93
	+1	0.98	0.88	0.93	0.98	0.88	0.93
	avg.	0.94	0.93	0.93	0.94	0.93	0.93
kNN-MST	−1	0.94	0.96	0.95	0.94	0.96	0.95
	+1	0.96	0.94	0.95	0.96	0.94	0.95
	avg.	0.95	0.95	0.95	0.95	0.95	0.95

Table 3.5: Semi-supervised evaluation, 20Newsgroups data, comp versus rec using cosine similarity.

Graph type	Label	Precision	Recall	F1	Precision	Recall	F1
		Mincut approach			**Graphical model**		
Split-70, labeled:unlabeled = 2661:6209							
kNN	−1	0.89	0.84	0.86	0.82	0.96	0.88
	+1	0.81	0.88	0.84	0.94	0.73	0.82
	avg.	0.85	0.86	0.85	0.88	0.85	0.85
MST	−1	0.86	0.80	0.83	0.82	0.83	0.83
	+1	0.77	0.84	0.81	0.79	0.78	0.79
	avg.	0.82	0.82	0.82	0.81	0.81	0.81
kNN-MST	−1	0.86	0.80	0.83	0.82	0.83	0.83
	+1	0.78	0.84	0.81	0.79	0.78	0.79
	avg.	0.82	0.82	0.82	0.81	0.81	0.81
Split-90, labeled:unlabeled = 860:8010							
kNN	−1	0.91	0.82	0.86	0.80	0.96	0.88
	+1	0.80	0.89	0.85	0.94	0.71	0.81
	avg.	0.85	0.86	0.85	0.87	0.84	0.84
MST	−1	0.82	0.78	0.80	0.79	0.82	0.80
	+1	0.75	0.79	0.77	0.77	0.73	0.75
	avg.	0.79	0.79	0.79	0.78	0.77	0.78
kNN-MST	−1	0.84	0.80	0.82	0.80	0.83	0.82
	+1	0.77	0.81	0.79	0.78	0.75	0.77
	avg.	0.80	0.80	0.80	0.79	0.79	0.79
Split-97, labeled:unlabeled = 266:8604							
kNN	−1	0.56	0.97	0.71	0.82	0.94	0.87
	+1	0.59	0.06	0.11	0.91	0.74	0.81
	avg.	0.57	0.51	0.41	0.86	0.84	0.84
MST	−1	0.80	0.77	0.79	0.77	0.81	0.79
	+1	0.73	0.77	0.75	0.75	0.70	0.73
	avg.	0.77	0.77	0.77	0.76	0.76	0.76
kNN-MST	−1	0.81	0.79	0.80	0.78	0.82	0.80
	+1	0.75	0.78	0.76	0.77	0.72	0.74
	avg.	0.78	0.78	0.78	0.77	0.77	0.77

Table 3.6: Supervised evaluation, Abalone data.

Label	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
	Logistic Regression			**SVM**			**Graphical model**		
				Split-70					
-1	0.72	0.73	0.72	0.69	0.78	0.73	0.69	0.79	0.73
$+1$	0.75	0.75	0.75	0.78	0.68	0.72	0.78	0.67	0.72
avg.	0.74	0.74	0.74	0.73	0.73	0.73	0.74	0.73	0.73
				Split-90					
-1	0.71	0.62	0.66	0.69	0.74	0.71	0.71	0.70	0.70
$+1$	0.69	0.77	0.73	0.75	0.70	0.72	0.73	0.74	0.74
avg.	0.70	0.69	0.69	0.72	0.72	0.72	0.72	0.72	0.72
				Split-97					
-1	0.70	0.65	0.67	0.66	0.79	0.72	0.69	0.73	0.71
$+1$	0.71	0.74	0.72	0.78	0.63	0.69	0.74	0.71	0.72
avg.	0.70	0.70	0.69	0.72	0.71	0.70	0.72	0.72	0.72

Table 3.7: Supervised evaluation, Digit data.

Label	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
	Logistic Regression			**SVM**			**Graphical model**		
				Split-70					
-1	0.90	0.89	0.90	1.00	0.98	0.99	1.00	0.99	0.99
$+1$	0.90	0.91	0.90	0.98	1.00	0.99	0.99	1.00	0.99
avg.	0.90	0.90	0.90	0.99	0.99	0.99	0.99	0.99	0.99
				Split-90					
-1	0.88	0.86	0.87	0.98	0.95	0.97	0.99	0.98	0.99
$+1$	0.87	0.88	0.88	0.96	0.98	0.97	0.98	0.99	0.99
avg.	0.87	0.87	0.87	0.97	0.97	0.97	0.99	0.99	0.99
				Split-97					
-1	0.83	0.82	0.82	0.94	0.88	0.90	0.94	0.96	0.95
$+1$	0.83	0.83	0.83	0.89	0.94	0.91	0.96	0.94	0.95
avg.	0.83	0.83	0.83	0.91	0.91	0.91	0.95	0.95	0.95

Table 3.8: Supervised evaluation, 20Newsgroups data, comp versus rec.

Label	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
	Logistic Regression			**SVM**			**Graphical model**		
				Split-70					
-1	0.92	0.89	0.90	0.90	0.90	0.90	0.82	0.96	0.88
$+1$	0.87	0.90	0.89	0.87	0.88	0.88	0.94	0.73	0.82
avg.	0.89	0.90	0.90	0.89	0.89	0.89	0.88	0.85	0.85
				Split-90					
-1	0.90	0.90	0.90	0.79	0.97	0.87	0.80	0.96	0.88
$+1$	0.88	0.88	0.88	0.95	0.68	0.79	0.94	0.71	0.81
avg.	0.89	0.89	0.89	0.87	0.82	0.83	0.87	0.84	0.84
				Split-97					
-1	0.82	0.93	0.87	0.55	1.00	0.71	0.82	0.94	0.87
$+1$	0.90	0.74	0.81	1.00	0.01	0.01	0.91	0.74	0.81
avg.	0.86	0.84	0.84	0.78	0.50	0.36	0.86	0.84	0.84

Chapter 4

Discussions and Concluding Remarks

4.1 What We Have Done

We have examined two different generative models of semi-supervised learning including multinomial Naive Bayes and graph-based methods for optimizing the components separation measurement. First and foremost, we have claimed the assumption that the label values are distributed as the dense components and these components are separated in its data representation form. Then what we have wanted to do is improve the methods that have better way to distinguish these components. So then, we have

- The multinomial model, constructed with Naive Bayes and many-to-one assumptions, performs its head over the traditional model with one component per label value. To leverage the use of labeled data, through the experiments, we showed that it is better to consider the initialization with unsupervised methods. It gives us not only the better use of labeled data, but also the more stable with different components sampling than the conventional randomized method

- The graph-based methods setup on the minimization of similarity between groups of label components. The problem comes from the overlap region of components when it may produce many optimized solutions. In this situation, the original mincut approach has an inflexible solution when we cannot have a change to intervene in the inference process. Then we changed to utilize the graphical model with the support of influence index. We also extended the model itself to work on a more general case when we employ the loopy belief propagation algorithm. Through our investigations, we hold the observation that when giving the advantage to examine and choose the proper graph construction for a specific data, the graphical model shows its outperform to the mincut approach. Moreover, the model has proved itself to be a stable solution in the condition of semi-supervised assumption when we are drowning in the unlabeled data in comparison with our tested supervised models.

Ultimately, besides giving more useful modifications in some particular cases, our works have given more evidence on the reasonable of the semi-supervised learning in practice.

4.2 Shortcomings and Long Term Views

The limitations go directly from the restrictions of our assumptions. It is also from simplifications and omissions when we modify our models. Nevertheless, it will be the room of our future work. Here we outline them in the objective for both models and subjective for each individual.

- First, we go back again to the question of the benefit of unlabeled data. It is argued that the unlabeled data does not always give positive value, and given more unlabeled data may hurt our classifier. In our experiments, we did not cover this problem since it is outside of what we intended to do. Now, in a better situation, we should find a reasonable amount of unlabeled data or the weighting factor between labeled-unlabeled data that still keeps the classifier working well.

- When the components are bent too close to each other, the assumption fails in matching the separation boundary. In this case, the multinomial seems to be more stable when it utilizes the statistical model rather than the similarity of data instances as in graph-based methods. With this case, we should look for a way to determine when it should be interchange between these types of models to uphold its advantages.

- From the beginning of chapter 2, we have introduced the general form of many-to-one assumption as in (2.1). But when we deployed the multinomial model, we constrained the component to a fixed label value, the predetermined components. However, Miller and Uyar [7] suggested that it is better to release more to a component that is distributed to all the label values. It may be interesting to derive this claim and to try with our assignments.

- The last obvious one is multi-classification. It is natural to extend the multi-classes to multinomial model, but it has no hope with our graph-based methods, we need to change the target function in this case. There are some common schemes for combining binary classifiers like one-versus-one or one-versus-rest. We may refer to them if we want to work on a multi-classification problem.

4.3 Conclusions

Finally, we agree with the view point of Olivier, Schölkopf, and Zien [3], semi-supervised learning seems to be more practical than theoretical problem. By this time, semi-supervised learning has not been kept in the main stream of machine learning interest. But when we construct a learning system, it frequently has the situations when we need to consult the solution from the semi-supervised models. So that is the reason why we prefer the practical solution in specific cases of semi-supervised than the overall one which competes with the wide range of different methods.

Appendix A

Experimental Resources

Data Description

All the data were fetched from UCI repository [30]

Abalone data consists of 7 features of abalone physical measurements like sex, length, shell weight, etc,. The task is to classify abalone by their ages. We only picked the age data in $[5, 15]$ which has more than 100 records per each class and constructed a binary classifier on the ages ranges from $[5, 9]$ and $[10, 15]$. The final data contains 3842 instances—1820 negative and 2022 positive.

Digit data contains 1797 instances of 8×8 images hand-written digits, 64 features. We constructed a binary classifier between odd and even digits—891 negative, 906 positive.

20Newsgroup data, 20news-bydate version is a collection of nearly 20000 newsgroup documents, divided into 20 different groups. In total, the origin set consists of 18774 instances with a vocabulary size of 61188 words. In our experiment, we only set binary classifiers for the major groups. So we constructed

- comp versus rec groups, 8870 instances—4891 negative, 3979 positive
- comp versus sci groups, 8843 instances—4891 negative, 3952 positive
- comp versus talk groups, 8144 instances—4891 negative, 3253 positive

Experiments Reproduction and Complete Results

The experiment was conducted on Python 3.6 programming language [31] with libraries Numpy [32], Scikit-Learn [33] and Igraph [34]. All of the resources are uploaded online at https://github.com/nghiapickup/master_thesis with detailed descriptions. We can reconstruct the experiments or get the complete results there.

Bibliography

[1] A. Y. Ng and M. I. Jordan, "On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes," in *Advances in Neural Information Processing Systems 14*, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds., MIT Press, 2002, pp. 841–848.

[2] V. Vapnik, *Statistical learning theory. 1998*. 1998, ISBN: 1368-4973.

[3] C. Olivier, B. Schölkopf, and A. Zien, *Semi-Supervised Learning*, 2. 2006, vol. 1, p. 524, ISBN: 9780262251150.

[4] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977, ISSN: 00359246.

[5] K. Nigam, A. K. McCallum, S. Thrun, and T. Mitchell, "Text Classification from Labeled and Unlabeled Documents Using EM," *Machine Learning - Special issue on information retrieval*, vol. 39, no. 2-3, pp. 103–134, May 2000, ISSN: 0885-6125.

[6] A. Blum and S. Chawla, "Learning from Labeled and Unlabeled Data Using Graph Mincuts," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 19–26, ISBN: 1-55860-778-1.

[7] D. J. Miller and H. S. Uyar, "A Mixture of Experts Classifier with Learning Based on Both Labelled and Unlabelled Data," in *Advances in Neural Information Processing Systems 9*, M. C. Mozer, M. I. Jordan, and T. Petsche, Eds., MIT Press, 1997, pp. 571–577.

[8] A. McCallum and K. Nigam, "A Comparison of Event Models for Naive Bayes Text Classification," in *Learning for Text Categorization: Papers from the 1998 {AAAI} Workshop*, 1998, pp. 41–48.

[9] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," *Computer Speech & Language*, vol. 13, no. 4, pp. 359–394, Oct. 1999, ISSN: 0885-2308.

[10] M. Collins, "The Naive Bayes Model, Maximum-Likelihood Estimation, and the EM Algorithm," Columbia University, Tech. Rep., 2013, pp. 1–21.

[11] S. Borman, "The Expectation Maximization Algorithm A short tutorial," School of computing, the university of Utah, Tech. Rep., 2006.

[12] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008, ISBN: 0521865719, 9780521865715.

[13] T. Jebara, J. Wang, and S.-F. Chang, "Graph Construction and B-matching for Semi-supervised Learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09, New York, NY, USA: ACM, 2009, pp. 441–448, ISBN: 978-1-60558-516-1.

[14] A. Subramanya and P. P. Talukdar, *Graph-Based Semi-Supervised Learning*. Morgan & Claypool Publishers, 2014, ISBN: 1627052011, 9781627052016.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[16] D. B. West, *Introduction to Graph Theory*, 2nd. Upper Saddle River (New Jersey) : Prentice-Hall., 2001.

[17] J. Edmonds and R. M. Karp, "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems," *J. ACM*, vol. 19, no. 2, pp. 248–264, Apr. 1972, ISSN: 0004-5411.

[18] V. King, S. Rao, and R. Tarjan, "A Faster Deterministic Maximum Flow Algorithm," *Journal of Algorithms*, vol. 17, no. 3, pp. 447–474, 1994, ISSN: 0196-6774.

[19] A. Blum, J. Lafferty, M. R. Rwebangira, and R. Reddy, "Semi-supervised Learning Using Randomized Mincuts," in *Proceedings of the Twenty-first International Conference on Machine Learning*, ser. ICML '04, New York, NY, USA: ACM, 2004, pp. 13–, ISBN: 1-58113-838-5.

[20] T. Joachims, "Transductive Learning via Spectral Graph Partitioning," in *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ser. ICML'03, AAAI Press, 2003, pp. 290–297, ISBN: 1-57735-189-4.

[21] J. Malik and J. Shi, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000, ISSN: 01628828.

[22] M. I. Jordan, "Graphical Models," *Statist. Sci.*, vol. 19, no. 1, pp. 140–155, 2004.

[23] D. Koller and N. Friedman, *Probabilistic graphical models : principles and techniques*. MIT Press, 2009, p. 1233, ISBN: 9780262013192.

[24] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[25] M. Bayati, D. Shah, and M. Sharma, "Maximum Weight Matching via Max-Product Belief Propagation," *CoRR*, vol. abs/cs/050, 2005.

[26] B. Huang and T. Jebara, "Loopy Belief Propagation for Bipartite Maximum Weight b-Matching," in *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, M. Meila and X. Shen, Eds., ser. Proceedings of Machine Learning Research, vol. 2, San Juan, Puerto Rico: PMLR, 2007, pp. 195–202.

[27] S. M. Aji, G. B. Horn, and R. J. Mceliece, "On the Convergence of Iterative Decoding on Graphs with a Single Cycle," in *In Proc. 1998 ISIT*, 1998, p. 276.

[28] Y. Weiss and W. T. Freeman, "Correctness of Belief Propagation in Gaussian Graphical Models of Arbitrary Topology," *Neural Computation*, vol. 13, no. 10, pp. 2173–2200, Oct. 2001, ISSN: 0899-7667.

[29] R. C. Prim, "Shortest connection networks and some generalizations," *The Bell System Technical Journal*, vol. 36, no. 6, pp. 1389–1401, Nov. 1957, ISSN: 0005-8580.

[30] D. Dheeru and E. Karra Taniskidou. (2017). {UCI} Machine Learning Repository, [Online]. Available: http://archive.ics.uci.edu/ml.

[31] Python Software Foundation. (2017). Python Programming Language, version 3.6, [Online]. Available: https://www.python.org/.

[32] T. Oliphant, *NumPy: A guide to NumPy*, USA: Trelgol Publishing.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[34] G. Csardi and T. Nepusz, "The igraph software package for complex network research," *InterJournal*, vol. Complex Sy, p. 1695, 2006.