5 Partially Supervised Learning

With *Wee Sun Lee*

In supervised learning, the learning algorithm uses labeled training examples from every class to generate a classification function. One of the drawbacks of this classic paradigm is that a large number of labeled examples are needed in order to learn accurately. Since labeling is often done manually, it can be very labor intensive and time consuming. In this chapter, we study two **partially supervised learning** tasks. As their names suggest, these two learning tasks do not need full supervision, and thus are able to reduce the labeling effort. The first is the task of **learning from labeled and unlabeled examples**, which is commonly known as **semi-supervised learning**. In this chapter, we also call it **LU learning** (L and U stand for "labeled" and "unlabeled" respectively). In this learning setting, there is a small set of labeled examples of every class, and a large set of unlabeled examples. The objective is to make use of the unlabeled examples to improve learning.

The second is the task of **learning from positive and unlabeled examples**. This problem assumes two-class classification. However, the training data only has a set of labeled positive examples and a set of unlabeled examples, but no labeled negative examples. In this chapter, we also call this problem **PU learning** (P and U stand for "positive" and "unlabeled" respectively). The objective is to build an accurate classifier without labeling any negative examples. We study these two problems in the context of text classification and Web page classification in this chapter. However, the general ideas and the algorithms are also applicable to other kinds of classification tasks.

5.1 Learning from Labeled and Unlabeled Examples

As we described in Chap. 3, the common approach to learning a classification function is to label a set of examples with some pre-defined categories or classes, and then use a learning algorithm to produce a classifier. This classifier is applied to assign classes to future instances (or test data). In the context of text classification and Web page classification, the examples are text documents and Web pages. This approach to building a classifier is called supervised learning because the training documents/pages have been labeled with pre-defined classes.

The main bottleneck of building such a classifier is that a large, often prohibitive, number of labeled training documents are needed to build accurate classifiers. In text classification, the labeling is typically done manually by reading the documents, which is a time consuming task. However, we cannot eliminate labeling completely because without it a machine learning algorithm will not know what the user is interested in. Although unsupervised learning or clustering may help to some extent, clustering does not guarantee to produce the categorization results required by the user. This raises an important question: Can the manual labeling effort be reduced, and can other sources of information be used so that the number of labeled examples required for learning would not be too large?

This section addresses the problem of learning from a small set of **labeled examples** and a large set of **unlabeled examples**, i.e., LU learning. Thus, in this setting only a small set of examples needs to be labeled for each class. However, since a small set of labeled examples is not sufficient for building an accurate classifier, a large number of unlabeled examples are utilized to help. One key point to note is that although the number may be small, every class must have some labeled examples.

In many applications, unlabeled examples are easy to come by. This is especially true for online documents. For example, if we want to build a classifier to classify news articles into different categories or classes, it is fairly easy to collect a huge number of unlabeled news articles from the Web. In fact, in many cases, the new data that need to be classified (which have no class labels) can be used as the unlabeled examples.

The question is: why do the unlabeled data help? In the context of text classification, one reason is that the unlabeled data provide information on the joint probability distribution over words. For example, using only the labeled data we find that documents containing the word "homework" tend to belong to a particular class. If we use this fact to classify the unlabeled documents, we may find that "lecture" co-occurs with "homework" frequently in the unlabeled set. Then, "lecture" may also be an indicative word for the class. Such correlations provide a helpful source of information to increase classification accuracy, especially when the labeled data are scarce.

Several researchers have shown that unlabeled data help learning. That is, under certain conditions using both labeled and unlabeled data in learning is better than using a small set of labeled data alone. Their techniques can thus alleviate the labor-intensive labeling effort. We now study some of these learning techniques, and also discuss their limitations.

5.1.1 EM Algorithm with Naïve Bayesian Classification

One of the LU learning techniques uses the Expectation–Maximization (EM) algorithm [127]. EM is a popular iterative algorithm for maximum likelihood estimation in problems with missing data. The EM algorithm consists of two steps, the **Expectation step** (or **E-step**), and the **Maximization step** (or **M-step**). The E-step basically fills in the missing data based on the current estimation of the parameters. The M-step, which maximizes the likelihood, re-estimates the parameters. This leads to the next iteration of the algorithm, and so on. EM converges to a local minimum when the model parameters stabilize.

The ability of EM to work with missing data is exactly what is needed for learning from labeled and unlabeled examples. The documents in the labeled set (denoted by L) all have class labels (or values). The documents in the unlabeled set (denoted by U) can be regarded as having missing class labels. We can use EM to estimate them based on the current model, i.e., to assign probabilistic class labels to each document d_i in U, i.e., $\Pr(c_j|d_i)$. After a number of iterations, all probabilities will converge.

Note that the EM algorithm is not really a specific "algorithm", but is a framework or strategy. It simply runs a base algorithm iteratively. We will use the naïve Bayesian (NB) algorithm discussed in Sect. 3.7 as the base algorithm, and run it iteratively. The parameters that EM estimates in this case are the probability of each word given a class and the class prior probabilities (see Equation (27) and (28) in Sect. 3.7 of Chap. 3).

Although it is quite involved to derive the EM algorithm with the NB classifier, it is fairly straightforward to implement and to apply the algorithm. That is, we use a NB classifier in each iteration of EM, Equation (29) in Chap. 3 for the E-step, and Equations (27) and (28) in Chap. 3 for the M-step. Specifically, we first build a NB classifier f using the labeled examples in L. We then use f to classify the unlabeled examples in U, more accurately to assign a probability to each class for every unlabeled example, i.e., $\Pr(c_j|d_i)$, which takes the value in [0, 1] instead of {0, 1}. Some explanations are in order here.

Let the set of classes be $C = \{c_1, c_2, \ldots, c_{|C|}\}$. Each iteration of EM will assign every example d_i in U a probability distribution on the classes that it may belong to. That is, it assigns d_i the class probabilities of $\Pr(c_1|d_i)$, $\Pr(c_2|d_i)$, …, $\Pr(c_{|C|}|d_i)$. This is different from the example in the labeled set L, where each document belongs to only a single class c_k, i.e., $\Pr(c_k|d_i) = 1$ and $\Pr(c_j|d_i) = 0$ for $j \neq k$.

Based on the assignments of $\Pr(c_j|d_i)$ to each document in U, a new NB classifier can be constructed. This new classifier can use both the labeled set L and the unlabeled set U as the examples in U now have probabilistic

```
Algorithm EM($L$, $U$)
1  Learn an initial naïve Bayesian classifier $f$ from only the labeled set $L$ (us-
      ing Equations (27) and (28) in Chap. 3);
2  repeat
      // E-Step
3     for each example $d_i$ in $U$ do
4         Using the current classifier $f$ to compute $\Pr(c_j|d_i)$ (using Equation
            (29) in Chap. 3).
5     end
      // M-Step
6     learn a new naïve Bayesian classifier $f$ from $L \cup U$ by computing $\Pr(c_j)$
         and $\Pr(w_t|c_j)$ (using Equations (27) and (28) in Chap. 3).
7  until the classifier parameters stabilize
Return the classifier $f$ from the last iteration.
```

Fig. 5.1. The EM algorithm with naïve Bayesian classification

labels, $\Pr(c_j|d_i)$. This leads to the next iteration. The process continues until the classifier parameters ($\Pr(w_t|c_j)$ and $\Pr(c_j)$) no longer change (or have minimum changes).

The EM algorithm with NB classification was proposed for LU learning by Nigam et al. [413]. The algorithm is shown in Fig. 5.1. EM here can also be seen as a clustering method with some initial seeds (labeled data) in each cluster. The class labels of the seeds indicate the class labels of the resulting clusters.

The derivation of the EM algorithm in Fig. 5.1 is quite involved and is given as an appendix at the end of this chapter. Two assumptions are made in the derivation. They are in fact the two mixture model assumptions in Sect. 3.7 of Chap. 3 for deriving the naïve Bayesian classifier for text classification, i.e.,

1. the data is generated by a mixture model, and
2. there is a one-to-one correspondence between mixture components and classes.

It has been shown that the EM algorithm in Fig. 5.1 works well if the two mixture model assumptions for a particular data set are true. Note that although naïve Bayesian classification makes additional assumptions as we discussed in Sect. 3.7 of Chap. 3, it performs surprisingly well despite the obvious violation of the assumptions. The two mixture model assumptions, however, can cause major problems when they do not hold. In many real-life situations, they may be violated. It is often the case that a class (or topic) contains a number of sub-classes (or sub-topics). For example, the class Sports may contain documents about different sub-classes of sports, e.g., Baseball, Basketball, Tennis, and Softball. Worse still, a class c_j may even contain documents from completely different topics, e.g., Science, Politics, and Sports. The first assumption above is usually not a problem. The second assumption is critical. If the condition holds, EM works very well and is particularly useful when the labeled set is very small, e.g., fewer than five labeled documents per class. In such cases, every iteration of EM is able to improve the classifier dramatically. However, if the second condition does not hold, the classifier from each iteration can become worse and worse. That is, the unlabeled set hurts learning instead of helping it.

Two methods are proposed to remedy the situation.

Weighting the Unlabeled Data: In LU learning, the labeled set is small, but the unlabeled set is very large. So the EM's parameter estimation is almost completely determined by the unlabeled set after the first iteration. This means that EM essentially performs unsupervised clustering. When the two mixture model assumptions are true, the natural clusters of the data are in correspondence with the class labels. The resulting clusters can be used as the classifier. However, when the assumptions are not true, the clustering can go very wrong, i.e., the clustering may not converge to mixture components corresponding to the given classes, and are therefore detrimental to classification accuracy. In order to reduce the effect of the problem, we can weight down the unlabeled data during parameter estimation (EM iterations). Specifically, we change the computation of $\Pr(w_t|c_j)$ (Equation (27) in Chap. 3) to the following, where the counts of the unlabeled documents are decreased by a factor of μ, $0 \le \mu \le 1$:

$$\Pr(w_t \mid c_j) = \frac{\lambda + \sum_{i=1}^{|D|} \Lambda(i) N_{ti} \Pr(c_j \mid d_i)}{\lambda |V| + \sum_{s=1}^{|V|} \sum_{i=1}^{|D|} \Lambda(i) N_{ti} \Pr(c_j \mid d_i)}, \tag{1}$$

where

$$\Lambda(i) = \begin{cases} \mu & \text{if } d_i \in U \\ 1 & \text{if } d_i \in L. \end{cases} \tag{2}$$

When $\mu = 1$, each unlabeled document is weighted the same as a labeled document. When $\mu = 0$, the unlabeled data are not considered. The value of μ can be chosen based on leave-one-out cross-validation accuracy on the labeled training data. The μ value that gives the best result is used.

Finding Mixture Components: Instead of weighting unlabeled data low, we can attack the problem head on, i.e., by finding the mixture components (sub-classes) of the class. For example, the original class Sports may consist of documents from Baseball, Tennis, and Basketball, which are three mixture components (sub-classes or sub-topics) of Sports. Instead of using the original class, we try to find these components and treat each of them as a class. That is, if we can find the three mixture components, we can use them to replace the class Sports. There are several automatic approaches for identifying mixture components. For example, a hierarchical clustering technique was proposed in [111] to find the mixture components, which showed good performances. A simple approach based on leave-one-out cross-validation on the labeled training set was also given in [413].

Manually identifying different components may not be a bad option for text documents because one only needs to read the documents in the labeled set (or some sampled unlabeled documents), which is very small.

5.1.2 Co-Training

Co-training is another approach to learning from labeled and unlabeled examples. This approach assumes that the set of attributes (or features) in the data can be partitioned into two subsets. Each of them is sufficient for learning the target classification function. For example, in Web page classification, one can build a classifier using either the text appearing on the page itself, or the anchor text attached to hyperlinks pointing to the page from other pages on the Web. This means that we can use the same training data to build two classifiers using two subsets of features.

Traditional learning algorithms do not consider this division of features (attributes), or feature redundancy. All the features are pooled together in learning. In some cases, feature selection algorithms are applied to remove redundant features. Co-training exploits this feature division to learn separate classifiers over each of the feature sets, and utilizes the fact that the two classifiers must agree on their labeling of the unlabeled data to do LU learning.

Blum and Mitchell [55] formalize the co-training setting and provide a theoretical guarantee for accurate learning subject to certain assumptions. In the formalization, we have an example (data) space $X = X_1 \times X_2$, where X_1 and X_2 provide two different "views" of the example. That is, each example $\mathbf{x}$ (represented as a vector) is given as a pair ($\mathbf{x}_1$, $\mathbf{x}_2$). This simply means that the set of features (or attributes) is partitioned into two subsets. Each "view" or feature subset is sufficient for correct classification. Under some further assumptions, it was proven that co-training algorithms can learn from unlabeled data starting from only a weak classifier built using the small set of labeled training documents.

The first assumption is that the example distribution is **compatible** with the target functions; that is, for most examples, the target classification functions over the feature sets predict the same label. In other words, if f denotes the combined classifier, f_1 denotes the classifier learned from X_1, f_2 denotes the classifier learned from X_2 and c is the actual class label of example $\mathbf{x}$, then $f(\mathbf{x}) = f_1(\mathbf{x}_1) = f_2(\mathbf{x}_2) = c$ for most examples.

The second assumption is that the features in one set of an example are **conditionally independent** of the features in the other set, given the class of the example. In the case of Web page classification, this assumes that the words on a Web page are not related to the words on its incoming hyperlinks, except through the class of the Web page. This is a somewhat unrealistic assumption in practice.

The co-training algorithm explicitly uses the feature split to learn from labeled and unlabeled data. The algorithm is iterative. The main idea is that in each iteration, it learns a classifier from the labeled set L with each subset of the features, and then applies the classifier to classify (or label) the unlabeled examples in U. A number (n_i) of most confidently classified examples in U from each class c_i are added to L. This process ends when U becomes empty (or a fixed number of iterations is reached). In practice, we can set a different n_i for a different class c_i depending on class distributions. For example, if a data set has one third of class 1 examples and two thirds of class 2 examples, we can set $n_1 = 1$ and $n_2 = 2$.

The whole co-training algorithm is shown in Fig. 5.2. Lines 2 and 3 build two classifiers f_1 and f_2 from the two "views" of the data respectively. f_1 and f_2 are then applied to classify the unlabeled examples in U (lines 4 and 5). Some most confidently classified examples are removed from U and added to L. The algorithm then goes to the next iteration.

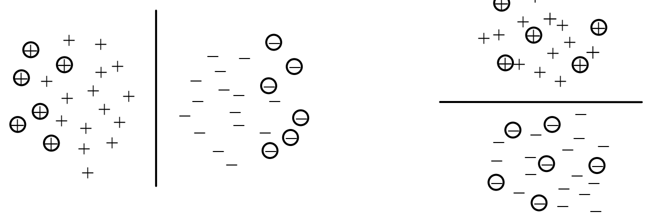
```
Algorithm co-training($L$, $U$)
1  repeat
2     Learn a classifier $f_1$ using $L$ based on only $\mathbf{x}_1$ portion of the examples $\mathbf{x}$.
3     Learn a classifier $f_2$ using $L$ based on only $\mathbf{x}_2$ portion of the examples $\mathbf{x}$.
4     Apply $f_1$ to classify the examples in $U$, for each class $c_i$, pick $n_i$ examples
          that $f_1$ most confidently classifies as class $c_i$, and add them to $L$.
5     Apply $f_2$ to classify the examples in $U$, for each class $c_i$, pick $n_i$ examples
          that $f_2$ most confidently classifies as class $c_i$, and add them to $L$.
6  until $U$ becomes empty or a fixed number of iterations are reached
```

Fig. 5.2. A co-training algorithm

When the co-training algorithm ends, it returns two classifiers. At classification time, for each test example the two classifiers are applied separately and their scores are combined to decide the class. For naïve Bayesian classifiers, we multiply the two probability scores, i.e.,

$$\Pr(c_j|\mathbf{x}) = \Pr(c_j|\mathbf{x}_1)\Pr(c_j|\mathbf{x}_2) \qquad (3)$$

The key idea of co-training is that classifier f_1 adds examples to the labeled set that are used for learning f_2 based on the X_2 view, and vice versa. Due to the conditional independence assumption, the examples added by f_1 can be considered as new and random examples for learning f_2 based on the X_2 view. Then the learning will progress. The situation is illustrated in Fig. 5.3. This example has classes, positive and negative, and assumes linear separation of the two classes. In the X_1 view (Fig. 5.3(A)), the circled examples are most confident positive and negative examples classified (or labeled) by f_1 in the unlabeled set U. In the X_2 view (Fig. 5.3(B)), these circled examples appear randomly. With these random examples from U added to L, a better f_2 will be learned in the next iteration.



(A) X_1 view: data in U labeled by h_1 (B) X_2 view: the same data

Fig. 5.3. Two views of co-training.

However, if the added examples to L are not random examples in the X_2 space but very similar to the situation in Fig. 5.3(A), then these examples are not informative to learning. That is, if the two subsets of features are correlated given the class or the conditional independence assumption is violated, the added examples will not be random but isolated in a specific region similar to those in Fig. 5.3(A). Then they will not be as useful or informative to learning. Consequently, co-training will not be effective.

In [411], it is shown that co-training produces more accurate classifiers than the EM algorithm presented in the previous section, even for data sets whose feature division does not completely satisfy the strict requirements of compatibility and conditional independence.

5.1.3 Self-Training

Self-training, which is similar to both EM and co-training, is another method for LU learning. It is an incremental algorithm that does not use the split of features. Initially, a classifier (e.g., naïve Bayesian classifier) is trained with the small labeled set considering all features. The classifier is then applied to classify the unlabeled set. Those most confidently classified (or unlabeled) documents of each class, together with their predicted class labels, are added to the labeled set. The classifier is then re-trained and the procedure is repeated. This process iterates until all the unlabeled documents are given class labels. The basic idea of this method is that the classifier uses its own predictions to teach itself.

5.1.4 Transductive Support Vector Machines

Support vector machines (SVM) is one of the most effective methods for text classification. One way to use unlabeled data in training SVM is by selecting the labels of the unlabeled data in such a way that the resulting margin of the classifier is maximized. Training for the purpose of labeling known (unlabeled) test instances is referred to as **transduction**, giving rise to the name **transductive SVM** [526]. An example of how transduction can change the decision boundary is shown in Fig. 5.4. In this example, the old decision boundary, constructed using only labeled data, would have a very small margin on the unlabeled data. By utilizing the unlabeled data in the training process, a classifier that has the largest margin on both the labeled and unlabeled data can be obtained.

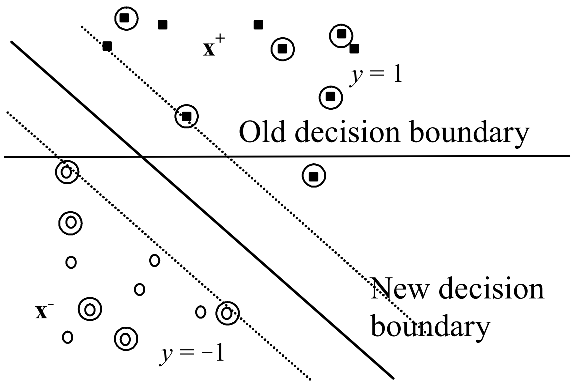


Fig. 5.4. The old decision boundary (before the addition of unlabeled data) and the new decision boundary created by transductive SVM. The unlabeled data are indicated by circles around them

The main difficulty with applying transductive SVM is the computational complexity. When all the labels are observed, training SVM is a convex optimization problem that can be solved efficiently. The problem of assigning labels to unlabeled examples in such a way that the resulting margin of the classifier is maximized can no longer be solved efficiently.

To solve the problem, Joachims [259] used a sub-optimal iterative method that starts by learning a classifier using only the labeled data. The method then treats a subset of unlabeled instances that are most confidently labeled positive by the learned classifier as initial positive examples while the rest of the unlabeled examples are treated as initial negative examples. The number of instances to label as positive can be specified by the user to change the precision–recall trade-off and is maintained throughout the iterations. The method then tries to improve the soft margin cost function by iteratively changing the labels of some of the instances and retraining the SVM. The ratio of positive to negative instances is maintained by selecting one positively labeled instance *p* and one negatively labeled instance *q* to change in each iteration. It was shown in [259] that if the two instances are selected such that the slack variables $\xi_p > 0$, $\xi_q > 0$ and $\xi_p + \xi_q > 2$, the soft margin cost function will decreases at each iteration. Further improvements described in [259] include allowing the soft margin error of unlabeled examples to be penalized differently from the soft margin error of the labeled examples and penalizing the soft margin error on the positive unlabeled examples differently from the soft margin error on the negative unlabeled examples. The penalty on the unlabeled examples is also iteratively increased from a small value to the desired value. This may improve the chances of finding a good local optimum as it may be easier to improve the cost function when the penalty is small. The method was applied successfully to text classification problems.

Like other methods of learning from labeled and unlabeled examples, transductive SVM can be sensitive to its assumptions. When the large margin assumption is correct on the dataset, it may improve performance but when the assumption is incorrect, it can decrease performance compared to supervised learning. As an example, the transductive SVM performed poorly using small labeled data sets when separating Project Web pages from other types of university Web pages in [259]. It was conjectured that, with a small number of labeled data, separating the Web pages according to some of the underlying topics of the Web pages may give a larger margin than separating them according to whether the Web pages are Project pages or not.

5.1.5 Graph-Based Methods

Graph-based LU learning methods can be viewed as extensions of **nearest neighbor supervised learning** algorithms that work with both labeled and unlabeled instances. The basic idea in these methods is to treat labeled and unlabeled instances as vertices in a graph where a similarity function is used to define the edge weights between instances. The graph, with similar instances connected by larger weights, is then used to help label the unlabeled instances in such a way that labels of vertices connected by edges with large weights tend to agree with each other. Methods used for constructing the graphs include connecting each instance to its *k*-nearest neighbors, connecting each instance to other instances within a certain distance δ and using a fully connected graph with an exponentially decreasing similarity function such as the Gaussian function to assign the weights. The assumptions used in these methods are similar to those of the nearest neighbor classifier, that is, near neighbors should have the same labels and we have a good measure of similarity between instances. We discuss three types of graph-based LU learning methods below: **mincut**, **Gaussian fields** and **spectral graph transducer**. All three methods work on binary classification problems but, like the support vector machines, can be used with strategies such as one-against-rest for solving multiple class classification problems.

Mincut: This method was proposed by Blum and Chalwa [54]. A weighted graph $G = (V, E, W)$ is constructed first, where V consists of the labeled and unlabeled instances, E consists of edges between the instances and W is a function on the edges with $W(i, j) = w_{ij}$ denoting the similarity of instances i and j. The vertices associated with labeled instances are then given values from {0, 1} consistent with their binary labels. The idea in the mincut algorithm is to find an assignment of values v_i from the set {0, 1} to the unlabeled instances in V such that the cost function $\sum_{(i,j)\in E} w_{ij} | v_i - v_j |$ is minimized. The advantage of this formulation is that the problem can be solved in polynomial time even though it is a combinatorial optimization problem. One way to do this is to transform the problem into a max-flow problem (see [116] for a description of the max-flow problem). To do that, we convert the graph into a flow network by introducing a source vertex v_+ and a sink vertex v_-, where the source vertex is connected by edges with infinite capacities to the positive labeled instances while the sink vertex is connected by edges with infinite capacities to the negative labeled instances. The other edge weights in the graph are also treated as edge capacities in the flow network. A cut of the network is a partition of the vertices into two subsets V_+ and V_- such that $v_+ \in V_+$ and $v_- \in V_-$. A minimum cut is a partition that has the smallest sum of capacities in the edges connecting V_+ and V_-. Finding a minimum cut is equivalent to minimizing the function $\sum_{(i,j)\in E} w_{ij} | v_i - v_j |$ since all the vertices are assigned values from {0, 1}. Max-flow algorithms can be used to efficiently find a mincut of the network in time $O(|V|^3)$.

Gaussian Fields: Instead of minimizing $\sum_{(i,j)\in E} w_{ij}|v_i - v_j|$, Zhu et al. [619] proposed minimizing $\sum_{(i,j)\in E} w_{ij}(v_i - v_j)^2$ with the value of the vertices being selected from [0, 1] instead of {0, 1}. The advantage of using this formulation is that it allows the solution to be obtained using linear algebra. Let $\boldsymbol{W}$ be the weight matrix corresponding to the graph,

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{W}_{LL} & \boldsymbol{W}_{LU} \\ \boldsymbol{W}_{UL} & \boldsymbol{W}_{UU} \end{bmatrix}, \tag{4}$$

where $\boldsymbol{W}_{LL}$, $\boldsymbol{W}_{LU}$, $\boldsymbol{W}_{UL}$ and $\boldsymbol{W}_{UU}$ are sub-matrices with the subscript L denoting labeled instances and the subscript U denoting the unlabeled instances. Let $\boldsymbol{D}$ be a diagonal matrix where $D_{ii} = \sum_j w_{ij}$ is the sum of the entries in row (or column) i. We also form a vector $\mathbf{v}$, consisting of values assigned to the labeled and unlabeled instances. The labeled instances are assigned fixed values in {0, 1} consistent with their labels while the values v_i assigned to the unlabeled instances are chosen to minimize $\sum_{(i,j)\in E} w_{ij}(v_i - v_j)^2$. The solution can be written as

$$\mathbf{v}_U = (\boldsymbol{D}_{UU} - \boldsymbol{W}_{UU})^{-1}\boldsymbol{W}_{UL}\mathbf{v}_L, \tag{5}$$

where $\mathbf{v}_U$ is the part of the vector $\mathbf{v}$ that contains values assigned to the unlabeled instances, $\mathbf{v}_L$ is the part of the vector that contains values assigned to labeled instances and $\boldsymbol{D}_{UU}$ is the sub-matrix of $\boldsymbol{D}$ consisting of sum of entries of rows in $\boldsymbol{W}$ associated with unlabeled instances.

The optimization problem $\sum_{(i,j)\in E} w_{ij}(v_i - v_j)^2$ can also be written in matrix form as $\mathbf{v}^T\Delta\mathbf{v}$ where $\Delta = \boldsymbol{D} - \boldsymbol{W}$ is known as the **combinatorial Laplacian** of the graph. The matrix Δ is known to be positive semidefinite, so it can be viewed as an inverse covariance matrix of a multivariate Gaussian random variable, giving rise to the name Gaussian field.

Spectral Graph Transducer: One potential problem with the mincut formulation is that the mincut cost function tends to prefer unbalanced cuts where the number of instances in either the positive or negative class vastly outnumbers the number of instances in the other class. Unbalanced cuts tend to have a lower cost in the mincut formulation because the number of edges between V_+ and V_- is maximized when the sizes of V_+ and V_- are equal and is small when either one of them is small. For example, if we have n vertices and V_+ contains a single element, then there are potentially $n-1$ edges between V_+ and V_-. In contrast, if V_+ and V_- are the same size, then there are potentially $n^2/4$ edges between the two sets of vertices.

Let cut(V_+, V_-) be the sum of the edge weights connecting V_+ and V_-. To mitigate the effect of preferring unbalanced cut, Joachims [261] proposed to minimize a cost function of normalized cut $\frac{cut(V_+,V_-)}{|V_+||V_-|}$, where the cut value is normalized by the number of edges between the two sets. Minimizing this cost function is computationally difficult, so Joachims [261] proposed minimizing a relaxed version of the problem.

Let Δ be the combinatorial Laplacian of the graph. It can be shown that minimizing the normalized cut (with no labeled data) using α and β number of instances (α and β are specified by the user) in the two partitions is equivalent to minimizing $\mathbf{v}^T\Delta\mathbf{v}$ for $v_i \in \{\gamma_+, \gamma_-\}$, where

$$\gamma_+ = \sqrt{\frac{\beta}{\alpha}} \quad \text{and} \quad \gamma_- = -\sqrt{\frac{\alpha}{\beta}}. \tag{6}$$

Instead of using $v_i \in \{\gamma_+, \gamma_-\}$, Joachims [261] proposed to allow v_i to take real values under the constraint $\mathbf{v}^T\mathbf{1}=0$ and $\mathbf{v}^T\mathbf{v}=n$, where $\mathbf{1}$ is the all one vector. To make sure that the labeled instances are properly classified, a term $(\mathbf{v}-\gamma)^T\boldsymbol{C}(\mathbf{v}-\gamma)$ is added to the cost function, where $\boldsymbol{C}$ is a diagonal matrix with non-zero entries only for labeled instances and γ is the target vector for approximation by $\mathbf{v}$. The components of γ that correspond to positive and negative instances are set to γ_+ and γ_- respectively, while the components of γ that correspond to unlabeled instances do not affect the cost function because their corresponding diagonal entries of $\boldsymbol{C}$ are set to zero. The values of the non-zero entries of $\boldsymbol{C}$ can be set by the user to give different misclassification costs to each instance. This gives the combined optimization problem of

$$\begin{aligned} &\min_v \mathbf{v}^T\Delta\mathbf{v} + c(\mathbf{v}-\gamma)^T\boldsymbol{C}(\mathbf{v}-\gamma) \\ &s.t. \qquad \mathbf{v}^T\mathbf{1} = 0 \; and \; \mathbf{v}^T\mathbf{v} = n \end{aligned} \tag{7}$$

where c gives a trade-off between the cost for the labeled and unlabeled parts. The solution of Equation (7) is obtained using spectral methods.

The Gaussian field method and spectral graph transduction have been applied to the natural language processing problem of word sense disambiguation in [414, 442]. Word sense disambiguation is the problem of assigning appropriate meanings to words (which may have multiple meanings) according to the context that they appear in. Although some improvements are observed, the success of these methods is still limited.

5.1.6 Discussion

We discuss two issues: (1) whether the unlabeled set *U* is always helpful and (2) the evaluation of LU learning.

Does the Unlabeled Set Always Help? The answer is no. As we have seen, all approaches make strong assumptions. For example, EM makes two mixture model assumptions, and co-training makes the feature split assumption. When the assumptions are true for an application data set, unlabeled data can help learning (even dramatically). When the assumptions are not true, the unlabeled data may harm learning. Automatically detecting bad match of the problem structure with the model assumptions in advance is, however, very hard and remains an open problem.

A related issue is that researchers have not shown that when the labeled data set is sufficiently large, the unlabeled data still help. Manual labeling more text documents may not be as difficult as it seems in some applications, especially when the number of classes is small. In most cases, to label a document one does not even need to read the entire document (if it is long). Typically, the first few sentences can already tell its class. Compounded with the problem of inability to decide whether the unlabeled data indeed help classification, practical applications of LU learning are still limited.

Evaluation: The evaluation of LU learning is commonly done in the same way as traditional classification. However, there is a problem with the availability of sufficient test data. In practice, users always want to have a reasonable guarantee on the predictive accuracy of a classification system before they are willing to use the system. This means that test data sets have to be used to evaluate the system. Existing algorithms for LU learning assume that there is a large set of labeled test data for this purpose. However, this contradicts the LU learning problem statement, which says that the labeled set is very small. If we can ask the user to label more data, then we do not need LU learning because some examples of the test set can be used in training. Evaluation without more labeled data is also an open problem.

One may look at this problem in another way. We first use the classifier generated by LU learning to classify the unlabeled set or a new test set and then sample some classified documents to be checked manually in order to estimate the classification accuracy. If classification is sufficiently accurate, the results of the classifier will be used. Otherwise, improvements need to be made. In this case, additional labeled data obtained during manual inspection can be added to the original labeled set. You see we end up doing more labeling! Hopefully, we do not have to do too much labeling.

5.2 Learning from Positive and Unlabeled Examples

In some applications, the problem is to identify a particular class *P* of documents from a set of mixed documents, which contains documents of class *P* and also other kinds of documents. We call the class of documents that one is interested in the positive class documents, or simply **positive documents**. We call the rest of the documents the negative class documents or simply **negative documents**.

This problem can be seen as a classification problem with two classes, positive and negative. However, there are no labeled negative documents for training. The problem is stated more formally as follows,

Problem Statement: Given a set *P* of *positive documents* that we are interested in, and a set *U* of **unlabeled documents** (*the mixed set*), which contains both *positive documents* and *negative documents*, we want to build a classifier using *P* and *U* that can identify positive documents in *U* or in a separate test set – in other words, we want to accurately classify positive and negative documents in *U* or in the test (or future) data set.

This problem is called **PU learning**. Note that the set *U* can be used in both training and testing because *U* is unlabeled.

The key feature of this problem is that there is no labeled negative document for learning. Traditional supervised learning algorithms are thus not directly applicable because they all require both labeled positive and labeled negative documents to build a classifier. This is also the case for LU learning, although the labeled set for each class may be very small.

5.2.1 Applications of PU Learning

The PU learning problem occurs frequently in Web and text retrieval applications because most of the time the user is only interested in Web pages or text documents of a particular topic. For example, one may be interested in only travel related pages (positive pages). Then all the other types of pages are negative pages. Let us use a concrete example to show the actual setting of a PU learning application.

Example 1: We want to build a repository of data mining research papers. We can start with an initial set of papers from a data mining conference or journal, which are positive examples. We then want to find data mining papers from online journals and conference series in the fields of databases and artificial intelligence. Journals and conferences in these fields all contain some data mining papers. They also contain many other types of papers. The problem is how to extract data mining papers from such conferences and journals, or in other words, how to classify the papers from these sources into data mining papers and non-data mining papers without labeling any negative papers in any source. ■

In practical applications, positive documents are usually available because if one has worked on a particular task for some time one should have accumulated many related documents. Even if no positive document is available initially, collecting some from the Web or any other source is relatively easy. One can then use this set to find the same class of documents from other sources without manually labeling any negative documents. PU learning is particularly useful in the following situations:

1. **Learning with multiple unlabeled sets:** In some applications, one needs to find positive documents from a large number of document collections. For example, we want to identify Web pages that sell printers. We can easily obtain a set of positive pages from an online merchant, e.g., amazon.com. Then we want to find printer pages from other merchants. We can crawl each site one by one, and extract printer pages from each site using PU learning. We do not need to manually label negative pages (non-printer pages) from any site.

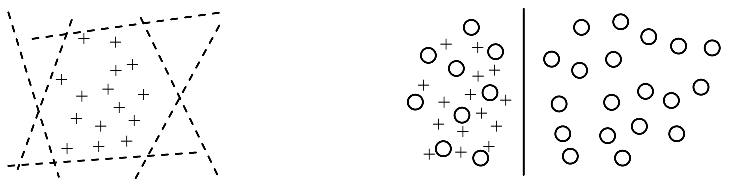
   Although it may not be hard to label some negative pages from a single site, it is difficult to label for every site. Note that in general the classifier built based on the negative pages from one site s_1 may not be used to classify pages from another site s_2 because the negative pages in s_2 can be very different from the negative pages in s_1. The reason is that although both sites sell printers, the other products that they sell can be quite different. Thus using the classifier built for s_1 to classify pages in s_2 may violate the fundamental assumption of machine learning: the distribution of training examples is identical to the distribution of test examples. As a consequence, we may obtain poor accuracy results.
2. **Learning with unreliable negative examples:** This situation often occurs in experimental sciences. For example, in biology, biologists perform experiments to determine some biological functions. They are often quite confident about positive cases that they have discovered. However, they may not be confident about negative cases because laboratory results can be affected by all kinds of conditions. The negative cases are thus unreliable. It is perhaps more appropriate to treat such negative cases as unlabeled examples than negative examples.

PU learning is also useful for the following seemingly unrelated problems:

Detecting unexpected documents in the test set: In many applications, the test set (or future instance set in practice) contains some classes of documents that are not seen in the training set. For instance, our training set has only two classes, Sports and Science. Then, a learning algorithm will produce a classifier to separate Sports and Science documents. However, in the test set, some other types of documents, e.g., Politics and Religion, may appear, which are called **unexpected documents**. In traditional classification, those Politics and Religion documents in the test set will be classified as either Sports or Science documents, which is clearly inappropriate. In PU learning, we can remove Politics and Religion documents by treating the whole training set as the positive data, and the whole test set as the unlabeled data. A study of this problem is reported in [323].

Detecting outliers: Traditional outlier detection algorithms are mainly based on clustering. During clustering, those data points that are too far away from cluster centroids are considered outliers. PU learning can be applied to outlier detection as follows: A random sample is drawn from the original data. The sample is treated as the set of positive examples, and the remaining data is treated as the set of unlabeled examples. The method given in [323] can then be applied. This strategy may work because since the number of outliers is small, the chance of selecting them during sampling will be extremely small. To make the method more robust, one can run the technique multiple times using multiple random samples.

Before discussing theoretical foundations of PU learning, let us first develop some intuition on why PU learning is possible and why unlabeled data are helpful. Figure 5.5 shows the idea.



(A) With only positive data (B) With both positive and unlabeled data

Fig. 5.5. Unlabeled data are helpful

In Fig. 5.5(A), we see only positive documents (data points) represented with +'s. We assume that a linear classifier is sufficient for the classification task. In this case, it is hard to know where to draw the line to separate positive and negative examples because we do not know where the negative examples might be. There are infinite possibilities. However, if the unlabeled data (represented by small circles) are added to the space (Fig. 5.5(B)), it is very clear where the separation line should be. Let us now discuss the theoretical result of PU learning.

5.2.2 Theoretical Foundation

Let $(\mathbf{x}_i, y_i)$ be random variables drawn independently from probability distribution $D_{(\mathbf{x},y)}$ where $y \in \{-1, 1\}$ is the conditional random variable that we wish to estimate given $\mathbf{x}$. $\mathbf{x}_i$ represents a document, and y_i is its class, which can be 1 (positive) or –1 (negative). Let $D_{\mathbf{x}|y=1}$ be the conditional distribution from which the positive examples are independently drawn and let $D_{\mathbf{x}}$ be the marginal distribution from which unlabeled examples are independently drawn. Our objective is to learn a classification function f that can separate positive and negative documents. Since learning is to produce a classifier that has the minimum probability of error, $\Pr(f(\mathbf{x})\neq y)$, let us rewrite it into a more useful form,

$$\Pr(f(\mathbf{x})\neq y) = \Pr(f(\mathbf{x})=1 \text{ and } y=-1) + \Pr(f(\mathbf{x})=-1 \text{ and } y=1). \tag{8}$$

The first term can be rewritten as

$$\begin{aligned}&\Pr(f(\mathbf{x})=1 \text{ and } y=-1)\\ &= \Pr(f(\mathbf{x})=1) - \Pr(f(\mathbf{x})=1 \text{ and } y=1)\\ &= \Pr(f(\mathbf{x})=1) - (\Pr(y=1) - \Pr(f(\mathbf{x})=-1 \text{ and } y=1)).\end{aligned} \tag{9}$$

Substituting (9) into Equation (8), we obtain

$$\begin{aligned}&\Pr(f(\mathbf{x})\neq y)\\ &= \Pr(f(\mathbf{x})=1) - \Pr(y=1) + 2\Pr(f(\mathbf{x})=-1|y=1)\Pr(y=1).\end{aligned} \tag{10}$$

Since $\Pr(y = 1)$ is constant (although it is unknown), we can minimize the probability of error by minimizing

$$\Pr(f(\mathbf{x})=1) + 2\Pr(f(\mathbf{x})=-1|y=1)\Pr(y=1). \tag{11}$$

If we can hold $\Pr(f(\mathbf{x})=-1|y=1)$ small, then learning is approximately the same as minimizing $\Pr(f(\mathbf{x})=1)$. Holding $\Pr(f(\mathbf{x})=-1|y=1)$ small while minimizing $\Pr(f(\mathbf{x})=1)$ is approximately the same as minimizing $\Pr_u(f(\mathbf{x})=1)$ (on the unlabeled set U) while holding $\Pr_P(f(\mathbf{x})=1) \geq r$ (on the positive set P), where r is the **recall**, i.e., $\Pr(f(\mathbf{x})=1|y=1)$. Note that $(\Pr_P(f(\mathbf{x})=1) \geq r)$ is the same as $(\Pr_P(f(\mathbf{x})=-1) \leq 1-r)$.

Two theorems given by Liu et al. [348] state these formally and show that in both the noiseless case (P has no error) and the noisy case (P contains errors, i.e., some negative documents) reasonably good learning results can be achieved if

- the problem is posed as a **constrained optimization problem** where the algorithm tries to minimize the number of unlabeled examples labeled positive subject to the constraint that the fraction of errors on the positive examples is no more than $1- r$.

Example 2: Figure 5.6 illustrates the constrained optimization problem. Assume that positive and negative documents can be linearly separated. Positive documents are represented with +'s, and unlabeled documents with small circles. Assume also that the positive set has no error and we want the recall r on the positive set to be 100%. Each line in the figure is a possible linear classifier. Every document on the left of each line will be labeled (classified) by the line as positive, and every document on the right will be labeled as negative. Lines 1 and 2 are clearly not solutions because the constraint "the fraction of errors on the positive examples must be no more than $1 - r$ (= 0)" is violated, although the number of unlabeled examples labeled (classified) as positive is minimized by line 1. Lines 4, 5, and 6 are poor solutions too because the number of unlabeled examples labeled as positive is not minimized by any of them. Line 3 is the optimal solution. Under the constraint that no positive example is labeled negative, line 3 minimizes the number of unlabeled examples labeled as positive. ■

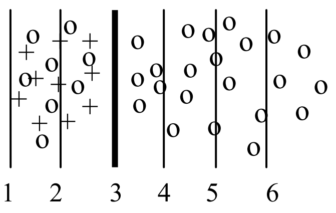


Fig. 5.6. An illustration of the constrained optimization problem

Based on the constrained optimization idea, two kinds of approaches have been proposed to build PU classifiers: the **two-step approach** and the **direct approach**. In the actual learning algorithms, the user may not need to specify a desired recall level r on the positive set because some of these algorithms have their evaluation methods that can automatically determine whether a good solution has been found.

5.2.3 Building Classifiers: Two-Step Approach

As its name suggests the two-step approach works in two steps:

1. Identifying a set of **reliable negative documents** (denoted by RN) from the unlabeled set U.
2. Building a classifier using P, RN and $U - RN$. This step may apply an existing learning algorithm once or iteratively depending on the quality and the size of the RN set.

This two-step approach is illustrated in Fig. 5.7. Here, we assume that step 2 uses an iterative algorithm. In step 1, a set of reliable negative documents (*RN*) is found from the unlabeled set *U*, which divides *U* into two subsets, *RN* and *Q* (= *U* – *RN*). *Q* is called the **likely positive set**. In step 2, the algorithm iteratively improves the results by adding more documents to *RN* until a convergence criterion is met. We can see that the process is trying to minimize the number of unlabeled examples labeled positive since *Q* becomes smaller and smaller while *RN* becomes larger and larger. In other words, it tries to iteratively increase the number of unlabeled examples that are labeled negative while maintaining the positive examples in *P* correctly classified. We present several techniques for each step below.

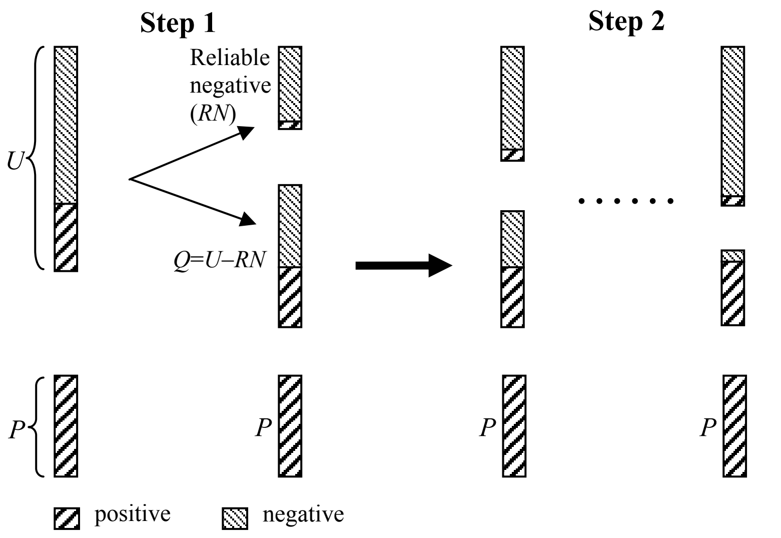


Fig. 5.7. An illustration of the two-step approach

Techniques for Step 1

We introduce four methods to extract reliable negative documents from the unlabeled set *U*.

Spy Technique: This technique works by sending some "spy" documents from the positive set *P* to the unlabeled set *U*. Figure 5.8 gives the algorithm of the technique, which is used in the S-EM system [348]. The algorithm has three sub-steps:

1. It randomly samples a set *S* of positive documents from *P* and put them in *U* (lines 2 and 3). The default sampling ratio of *s*% is 15% in S-EM.

```
    Algorithm Spy($P$, $U$)
1.  $RN \leftarrow \varnothing$;
2.  $S \leftarrow Sample(P, s\%)$;
3.  $Us \leftarrow U \cup S$;
4.  $Ps \leftarrow P - S$;
5.  Assign each document in $Ps$ the class label 1;
6.  Assign each document in $Us$ the class label −1;
7.  NB($Us$, $Ps$);                         // This produces a NB classifier.
8.  Classify each document in $Us$ using the NB classifier;
9.  Determine a probability threshold $t$ using $S$;
10. for each document $d \in Us$ do
11.     if its probability $\Pr(1|d) < t$ then
12.         $RN \leftarrow RN \cup \{d\}$;
13.     endif
14. endfor
```

Fig. 5.8. The spy technique.

The documents in S act as "spy" documents from the positive set to the unlabeled set U. Since the spies behave similarly to the unknown positive documents in U, they allow the algorithm to infer the behavior of the unknown positive documents in U.

2. It randomly samples a set S of positive documents from P and put them in U (lines 2 and 3). The default sampling ratio of $s\%$ is 15% in S-EM. The documents in S act as "spy" documents from the positive set to the unlabeled set U. Since the spies behave similarly to the unknown positive documents in U, they allow the algorithm to infer the behavior of the unknown positive documents in U.
3. It runs the naïve Bayesian (NB) algorithm using the set $P - S$ as positive and the set $U \cup S$ as negative (lines 3–7). The NB classifier is then applied to classify each document d in $U \cup S$ (or Us), i.e., to assign it a probabilistic class label $\Pr(1|d)$, where 1 represents the positive class.
4. It uses the probabilistic labels of the spies to decide which documents are most likely to be negative. A threshold t is employed to make the decision. Those documents in U with lower probabilities ($\Pr(1|d)$) than t are the most likely negative documents, denoted by RN (lines 10–14).

We now discuss how to determine t using spies (line 9). Let the set of spies be $S = \{s_1, s_2, \ldots, s_k\}$, and the probabilistic labels assigned to each s_i be $\Pr(1|s_i)$. Intuitively, we can use the minimum probability in S as the threshold value t, i.e., $t = min\{\Pr(1|s_1), \Pr(1|s_2), \ldots, \Pr(1|s_k)\}$, which means that we want to retrieve all spy documents. In a noiseless case, using the minimum probability is acceptable. However, most real-life document collections have outliers and noise. Using the minimum probability is unreliable. The reason is that the posterior probability Pr(1|s_i) of an outlier document s_i in S could be 0 or smaller than most (or even all) actual negative documents. However, we do not know the noise level of the data. To be safe, the S-EM system uses a large noise level l = 15% as the default. The final classification result is not very sensitive to l as long it is not too small. To determine t, we first sort the documents in S according to their Pr(1|s_i) values. We then use the selected noise level l to decide t: we select t such that l percent of documents in S have probability less than t. Hence, t is not a fixed value. The actual parameter is in fact l.

Note that the reliable negative set *RN* can also be found through multiple iterations. That is, we run the spy algorithm multiple times. Each time a new random set of spies S is selected from P and a different set of reliable negative documents is obtained, denoted by RN_i. The final set of reliable negative documents is the intersection of all RN_i. This may be a better technique because we do not need to worry that one set of random spies S may not be chosen well, especially when the set P is not large.

1DNF Technique: The 1DNF method (Fig. 5.9) is used in [583]. It first builds a positive feature set *PF* containing words that occur in the positive set P more frequently than in the unlabeled set U (lines 1–7). Line 1 collects all the words in $U \cup P$ to obtain a vocabulary V. Lines 8–13 try to identify reliable negative documents from U. A document in U that does not contain any feature in *PF* is regarded as a reliable negative document.

NB Technique: This method is employed in [340]. It simply uses a naïve Bayesian classifier to identify a set of reliable negative documents *RN* from the unlabeled set U. The algorithm is given in Fig. 5.10.

This method may also be run multiple times. Each time we randomly remove a few documents from P to obtain a different set of reliable negative documents, denoted by RN_i. The final set of reliable negative documents *RN* is the intersection of all RN_i.

Rocchio technique: This method is employed in [321]. The algorithm is the same as that in Fig. 5.10 except that NB is replaced with Rocchio. The Rocchio classification method is described in Sect. 6.3.

Techniques for Step 2

There are two approaches for this step.

1. Run a learning algorithm (e.g., NB or SVM) using P and *RN*. The set of documents in *U–RN* is discarded. This method works well if the reliable negative set *RN* is sufficiently large and contains mostly negative docu-

```
Algorithm 1DNF(P, U)
1.  Assume the word feature set be $V = \{w_1,\ldots, w_n\}$, $w_i \in U \cup P$;
2.  Let positive feature set $PF \leftarrow \emptyset$;
3.  for each $w_i \in V$ do                               // $freq(w_i, P)$: number of times
4.    if $(freq(w_i, P) / |P| > freq(w_i, U) / |U|)$ then   // that $w_i$ appears in $P$
5.      $PF \leftarrow PF \cup \{w_i\}$;
6.    endif
7.  endfor;
8.  $RN \leftarrow U$;
9.  for each document $d \in U$ do
10.   if $\exists w_j\ freq(w_j, d) > 0$ and $w_j \in PF$ then
11.     $RN \leftarrow RN - \{d\}$
12.   endif
13. endfor
```

Fig. 5.9. The 1DNF technique for step 1

```
1.  Assign each document in $P$ the class label 1;
2.  Assign each document in $U$ the class label $-1$;
3.  Build a NB classifier using $P$ and $U$;
4.  Use the classifier to classify $U$. Those documents in $U$ that are classified as
    negative form the reliable negative set $RN$.
```

Fig. 5.10. The NB method for Step 1

ments. The spy technique, NB and Rocchio in step 1 are often able to produce a sufficiently large set of negative documents. The 1DNF technique may only identify a very small set of negative documents. Then running a learning algorithm will not be able to build a good classifier.

2. Run a learning algorithm iteratively till it converges or some stopping criterion is met. This method is used when the set *RN* is small.

We will not discuss the first approach as it is straightforward. SVM usually does very well. Below, we introduce two techniques for the second approach, which are based on EM and SVM respectively.

EM Algorithm with Naïve Bayesian Classification: The EM algorithm can be used naturally here [348]. As in LU learning, the Expectation step basically fills in the missing data. In our case, it produces and revises the probabilistic labels of the documents in *U–RN* (see below). The parameters are estimated in the Maximization step after the missing data are filled. This leads to the next iteration of the algorithm. EM converges when its parameters stabilize. Using NB in each iteration, EM employs the same equations as those used in building a NB classifier (Equation (29) for the Expectation step, and Equations (27) and (28) for the Maximization step).

```
   Algorithm EM($P$, $U$, $RN$)
1.  Each document in $P$ is assigned the class label 1;
2.  Each document in $RN$ is assigned the class label –1;
3.  Learn an initial NB classifier $f$ from $P$ and $RN$ (using Equations (27) and
        (28) in Chap. 3);
4   repeat
       // E-Step
5      for each example $d_i$ in $U–RN$ do
6          Using the current classifier $f$ to compute $\Pr(c_j|d_i)$ using Equation (29)
               in Chap. 3.
7      end
       // M-Step
8      learn a new NB classifier $f$ from $P$, $RN$ and $U–RN$ by computing $\Pr(c_j)$
           and $\Pr(w_t|c_j)$ (using Equations (27) and (28) in Chap. 3).
9   until the classifier parameters stabilize
10. Return the classifier $f$ from the last iteration.
```

Fig. 5.11. EM algorithm with the NB classifier

```
   Algorithm I-SVM($P$, $RN$, $Q$)
1.  Every document in $P$ is assigned the class label 1;
2.  Every document in $RN$ is assigned the class label –1;
3.  loop
4.      Use $P$ and $RN$ to train a SVM classifier $f$;
5.      Classify $Q$ using $f$;
6.      Let $W$ be the set of documents in $Q$ that is classified as negative;
7.      if $W = \emptyset$ then exit-loop   // convergence
8.      else $Q \leftarrow Q - W$;
9.           $RN \leftarrow RN \cup W$;
10.     endif;
```

Fig. 5.12. Running SVM iteratively

The class probability given to each document in *U–RN* takes the value in [0, 1] instead of {0, 1}. The algorithm is given in Fig. 5.11.

The EM algorithm here makes the same mixture model assumptions as in LU learning. Thus, it has the same problem of model mismatch. See the discussions in Sect. 5.1.1.

Iterative SVM: In this method, SVM is run iteratively using *P*, *RN* and *Q* (= *U–RN*). The algorithm, called I-SVM, is given in Fig. 5.12. The basic idea is as follows: In each iteration, a new SVM classifier *f* is constructed from *P* and *RN* (line 4). Here *RN* is regarded as the set of negative examples (line 2). The classifier *f* is then applied to classify the documents in *Q* (line 5). The set *W* of documents in *Q* that are classified as negative (line

6) is removed from Q (line 8) and added to RN (line 9). The iteration stops when no document in Q is classified as negative, i.e., $W = \varnothing$ (line 7). The final classifier is the result. This method is used in [321, 582, 583].

Finally, we note again that if the first step is able to identify a large number of reliable negative documents from U, running SVM once in step 2 is sufficient. Iterative approaches may not be necessary, which are also less efficient. The Spy, NB and Rocchio methods for step 1 are often able to identify a large number of reliable negative documents. See [340] for an evaluation of various methods based on two benchmark text collections.

Classifier Selection

The iterative methods discussed above produce a new classifier at each iteration. However, the classifier at the convergence may not be the best classifier. In general, each iteration of the algorithm gives a classifier that may potentially be a better classifier than the classifier produced at convergence. This is true for both EM and SVM.

The main problem with EM is that classes and topics may not have one-to-one correspondence. This is the same problem as in LU learning. SVM may also produce poor classifiers at the convergence because SVM is sensitive to noise. If the RN set is not chosen well or in an iteration some positive documents are classified as negative, then the subsequent iterations may produce very poor results. In such cases, it is often better to stop at an earlier iteration. One simple method is to apply the theory directly. That is, each classifier is applied to classify a positive validation set, P_v. If many documents from P_v (e.g., > 5%) are classified as negative, the algorithm should stop (that is a recall of 95%). If the set P is small, the method can also be applied to P directly. A principled method is given in the next subsection, i.e., Equation (14).

5.2.4 Building Classifiers: Direct Approach

We now present a direct approach, called **biased-SVM**. This approach modifies the SVM formulation slightly so that it is suitable for PU learning. Let the set of training examples be $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i$ is an input vector and y_i is its class label, $y_i \in \{1, -1\}$. Assume that the first k–1 examples are positive examples P (labeled 1), while the rest are unlabeled examples U, which are treated as negative and labeled –1. Thus, the negative set has errors, i.e., containing positive documents. We consider two cases.

1. **Noiseless case:** There is no error in the positive examples but only in unlabeled examples. The theoretical result in Sect. 5.2.2 states that if the sample size is large enough, minimizing the number of unlabeled examples classified as positive while constraining the positive examples to be correctly classified will give a good classifier. Following the theory, in this noiseless case, we have this following SVM formulation

$$\begin{aligned}&\text{Minimize: } \frac{\langle \mathbf{w}\cdot\mathbf{w}\rangle}{2} + C\sum_{i=k}^{n}\xi_i \\ &\text{Subject to: } \langle \mathbf{w}\cdot\mathbf{x}_i\rangle + b \geq 1, \;\; i = 1,2,\ldots,k-1 \\ &\qquad -1(\langle \mathbf{w}\cdot\mathbf{x}_i\rangle + b) \geq 1-\xi_i, \;\; i = k, k+1, \ldots, n \\ &\qquad \xi_i \geq 0, \;\; i = k, k+1, \ldots, n\end{aligned} \tag{12}$$

   In this formulation, we do not allow any error in the positive set *P*, which is the first constraint, but allow errors for the negative set (the original unlabeled set), which is the second constraint. Clearly, the formulation follows the theory exactly due to the second term in the objective function. The subscript in the second term starts from *k*, which is the index of the first unlabeled example. To distinguish this formulation from the classic SVM, we call it the **biased-SVM** [340].
2. **Noisy case:** In practice, the positive set may also contain some errors. Thus, if we allow noise (or error) in positive examples, we have the following soft margin version of the biased-SVM which uses two parameters C_+ and C_- to weigh positive errors and negative errors differently.

$$\begin{aligned}&\text{Minimize: } \frac{\langle \mathbf{w}\cdot\mathbf{w}\rangle}{2} + C_+\sum_{i=1}^{k-1}\xi_i + C_-\sum_{i=k}^{n}\xi_i \\ &\text{Subject to: } y_i(\langle \mathbf{w}\cdot\mathbf{x}_i\rangle + b) \geq 1-\xi_i, \;\; i = 1,2,\ldots,n \\ &\qquad \xi_i \geq 0, \;\; i = 1, 2, \ldots, n\end{aligned} \tag{13}$$

   We can vary C_+ and C_- to achieve our objective. Intuitively, we give a bigger value for C_+ and a smaller value for C_- because the unlabeled set, which is assumed to be negative, contains positive data.

We now focus on Equation (13) as it is more realistic in practice. We need to choose values for C_+ and C_-. The common practice is to try a range of values for both C_+ and C_- and use a separate validation set to verify the performance of the resulting classifier. The C_+ and C_- values that give the best classification results on the validation set are selected as the final parameter values for them. Cross-validation is another possible technique for the purpose. Since the need to learn from positive and unlabeled examples often arises in retrieval situations (retrieving positive documents from the unlabeled set), we employ the commonly used F-score as the performance measure, $F = 2pr/(p+r)$, where p is the precision and r is the recall.

Unfortunately it is not clear how to estimate the F-score without labeled negative examples. In [309], Lee and Liu proposed an alternative performance measure to compare different classifiers. It behaves similarly to the F-score but can be estimated directly from the validation set without the need of labeled negative examples. The measure is

$$\frac{r^2}{\Pr(f(\mathbf{x})=1)}, \tag{14}$$

where f is the classifier and $\Pr(f(\mathbf{x})=1)$ is the probability that a document is classified as positive. It is not easy to see why Equation (14) behaves similarly to the F-score, but we can show that $r^2/\Pr(f(\mathbf{x})=1) = pr/\Pr(y=1)$, where $\Pr(y=1)$ is the probability of positive documents. $pr/\Pr(y=1)$ behaves similarly to the F-score in the sense that it is large when both p and r are large and is small when either p or r is small.

We first write recall (r) and precision (p) in terms of probability:

$$r = \Pr(f(\mathbf{x})=1 \mid y=1) \tag{15}$$

$$p = \Pr(y=1 \mid f(\mathbf{x})=1). \tag{16}$$

According to probability theory, we have

$$\Pr(f(\mathbf{x})=1 \mid y=1)\Pr(y=1) = \Pr(y=1 \mid f(\mathbf{x})=1)\Pr(f(\mathbf{x})=1), \tag{17}$$

which can be written as

$$\frac{r}{\Pr(f(\mathbf{x})=1)} = \frac{p}{\Pr(y=1)}. \tag{18}$$

Multiplying both sides by r, we obtain the result:

$$\frac{r^2}{\Pr(f(\mathbf{x})=1)} = \frac{pr}{\Pr(y=1)}. \tag{19}$$

The quantity $r^2/\Pr(f(\mathbf{x})=1)$ can be estimated based on the validation set, which contains both positive and unlabeled documents. r can be estimated using the positive examples in the validation set and $\Pr(f(\mathbf{x}) = 1)$ can be estimated from the whole validation set.

This criterion in fact reflects the theory in Sect. 5.2.2 very well. The quantity is large when r is large and $\Pr(f(\mathbf{x}) = 1)$ is small, which means that the number of unlabeled examples labeled as positive should be small. In [340], it is shown that biased-SVM works better than two-step techniques.

5.2.5 Discussion

Does PU Learning Always Work? Theoretical results show that it should if the positive set and the unlabeled set are sufficiently large [348]. This has been confirmed by many experimental studies. Interested readers can find the detailed results in [340, 348], which we summarize below:

1. PU learning can achieve about the same classification results as fully supervised learning (i.e., both labeled positive and negative examples are available for training) when the positive set and the unlabeled set are sufficiently large. This implies that labeled negative examples do not provide much information for learning. When the positive set is very small, PU learning is poorer than fully supervised learning.
2. For the two-step approaches, using SVM for the second step performs better than EM. SVM needs to be run only once if step 1 can extract a large number of reliable negative documents. Both Spy and Rocchio (and to some extent NB as well) are able to do that. Thus, the iterative method in step 2 is not necessary.

   The generative model of naïve Bayes with EM in the second step can perform very well if the mixture model assumptions hold [348]. However, if the mixture model assumptions do not hold, the classification results can be very poor [340]. Note that SVM is usually called a **discriminative model** (or **classifier**) because it does not make any model assumptions. It simply finds a hyperplane to separate positive and negative examples in the training data.
3. Biased-SVM performs slightly better than the 2-step approaches. However, it is very slow in training because SVM needs to be run a large number of times in order to select the best values for C_+ and C_-.

Evaluation: Unlike LU learning, here we do not even have labeled negative examples, which makes the evaluation difficult. Although Equation (14) and other heuristics allow a system to choose a "better" classifier among a set of classifiers, it is unable to give the actual accuracy, precision or recall of each classifier. Evaluation is an open problem. The results reported in [340, 348] assume that a set of labeled positive and negative test examples is available, which, of source, is unrealistic in practice because the PU learning model states that no labeled negative example is available.

In some cases, the evaluation can be done with some confidence. For example, if the user needs to extract positive documents from many unlabeled sets as in the example of identifying printer pages from multiple Web sites, a PU learning algorithm can be applied to one site and then the user manually checks the classification result to see whether it is satisfactory. If the result is satisfactory, the algorithm can be applied to the rest of the sites without further manual evaluation.

Appendix: *Derivation of EM for Naïve Bayesian Classification*

EM is a method for performing a classical statistical estimation technique called **maximum likelihood estimation**. In maximum likelihood estimation, the aim is to find the model parameter $\hat{\Theta}$ that maximizes the likelihood function $\Pr(D_o; \Theta)$ for observed data D_o. In other words, maximum likelihood estimation aims to select the model that is most likely to have generated the observed data. In many cases, such as in the naïve Bayesian classification model, the maximum likelihood estimator is easy to find and has a closed form solution when all components of the data D are observed. However, the problem becomes difficult when the data D actually consists of an observed component D_o and an unobserved component D_u. In such cases, iterative methods that converge only to a local maximum, such as the EM method, are usually used.

Maximizing the log likelihood function $\log\Pr(D_o; \Theta)$ produces the same solution as maximizing the likelihood function and is easier to handle mathematically. In the presence of unobserved data D_u, the log likelihood function becomes $\log \Pr(D_o;\Theta) = \log \sum_{D_u} \Pr(D_o, D_u;\Theta)$. Instead of maximizing the log likelihood $\log \sum_{D_u} \Pr(D_o, D_u;\Theta)$ directly, at each iteration T, the EM algorithm finds the value Θ that maximizes the **expected complete log likelihood**

$$\sum_{D_u} E(D_u \mid D_o;\Theta^{T-1}) \log \Pr(D_o, D_u;\Theta), \tag{20}$$

where Θ^{T-1} is the parameter that was produced in iteration T–1. In many cases, such as in the naïve Bayesian model, the expected log likelihood is easy to maximize and has a closed form solution. It can be shown (see [127]) that the log likelihood increases monotonically with each iteration of the EM algorithm.

We now derive the EM update for the naïve Bayesian model. We first consider the complete log likelihood, that is, the log likelihood when all variables are observed. The conditional probability of a document given its class is (see Sect. 3.7.2 in Chap. 3)

$$\Pr(d_i \mid c_j;\Theta) = \Pr(|d_i|)\,|d_i|!\prod_{t=1}^{|V|} \frac{\Pr(w_t \mid c_j;\Theta)^{N_{ti}}}{N_{ti}!} \tag{21}$$

Each document and its class label are assumed to have been sampled independently. Let $c_{(i)}$ be the class label of document i. The likelihood function can hence be written as

$$\prod_{i=1}^{|D|} \Pr(d_i \mid c_{(i)};\Theta)\Pr(c_{(i)};\Theta) = \prod_{i=1}^{|D|} \Pr(|d_i|)\,|d_i|!\prod_{t=1}^{|V|} \frac{\Pr(w_t \mid c_{(i)};\Theta)^{N_{ti}}}{N_{ti}!}\Pr(c_{(i)};\Theta). \qquad (22)$$

Taking logs, we have the complete log likelihood function

$$\sum_{i=1}^{|D|}\sum_{t=1}^{|V|} N_{ti}\log\Pr(w_t \mid c_{(i)};\Theta) + \sum_{i=1}^{|D|}\log\Pr(c_{(i)};\Theta) + \phi, \qquad (23)$$

where ϕ is a constant containing the terms unaffected by Θ. To facilitate the process of taking expectation when some of the class labels are not observed, we introduce indicator variables, h_{ik}, that take the value 1 when document i takes the label k and the value 0 otherwise. The complete log likelihood can be written in the following equivalent form

$$\sum_{i=1}^{|D|}\sum_{t=1}^{|V|}\sum_{k=1}^{|C|} h_{ik}N_{ti}\log\Pr(w_t \mid c_k;\Theta) + \sum_{i=1}^{|D|}\sum_{k=1}^{|C|} h_{ik}\log\Pr(c_k;\Theta) + \phi. \qquad (24)$$

When some of the labels are not observed, we take the conditional expectation for the unobserved variables h_{ik} with respect to Θ^{T-1} to get the expected complete log likelihood

$$\begin{aligned}&\sum_{i=1}^{|D|}\sum_{t=1}^{|V|}\sum_{k=1}^{|C|}\Pr(c_k \mid d_i;\Theta^{T-1})N_{ti}\log\Pr(w_t \mid c_k;\Theta)\\ &\quad+\sum_{i=1}^{|D|}\sum_{k=1}^{|C|}\Pr(c_k \mid d_i;\Theta^{T-1})\log\Pr(c_k;\Theta) + \phi,\end{aligned} \qquad (25)$$

where, for the observed labels $c_{(i)}$, we use the convention that $\Pr(c_k|d_i;\Theta^{T-1})$ takes the value one for $c_k = c_{(i)}$ and zero otherwise. We maximize the expected complete log likelihood subject to the coefficients summing to one using the Lagrange multiplier method. The Lagrangian is

$$\begin{aligned}&\sum_{i=1}^{|D|}\sum_{t=1}^{|V|}\sum_{k=1}^{|C|}\Pr(c_k \mid d_i;\Theta^{T-1})N_{ti}\log\Pr(w_t \mid c_k;\Theta)\\ &+\sum_{i=1}^{|D|}\sum_{k=1}^{|C|}\Pr(c_k \mid d_i;\Theta^{T-1})\log\Pr(c_k;\Theta)\\ &+\lambda\left(1-\sum_{k=1}^{|C|}\Pr(c_k;\Theta)\right)+\sum_{t=1}^{|V|}\sum_{k=1}^{|C|}\lambda_{tk}\left(1-\sum_{t=1}^{|V|}\Pr(w_t \mid c_k;\Theta)\right)+\phi.\end{aligned} \qquad (26)$$

Differentiating the Lagrangian with respect to λ, we get $\sum_{k=1}^{|C|}\Pr(c_k;\Theta) = 1$. Differentiating with respect to $\Pr(c_k;\Theta)$, we get

$$\sum_{i=1}^{|D|} \Pr(c_k \mid d_i; \Theta^{T-1}) = \lambda \Pr(c_k; \Theta) \qquad \textit{for } k = 1, \ldots, |C|. \tag{27}$$

Summing the left and right-hand side over k and using $\sum_{k=1}^{|C|} \Pr(c_k; \Theta) = 1$, we get $\lambda = \sum_{i=1}^{|D|} \sum_{k=1}^{|C|} \Pr(c_k \mid d_i; \Theta^{T-1}) = |D|$. Substituting back, we obtain the update equation

$$\Pr(c_j; \Theta^T) = \frac{\sum_{i=1}^{|D|} \Pr(c_j \mid d_i; \Theta^{T-1})}{|D|}. \tag{28}$$

Working similarly, we can get the update equation for $\Pr(w_t|c_j; \Theta^T)$,

$$\Pr(w_t \mid c_j; \Theta^T) = \frac{\sum_{i=1}^{|D|} N_{ti} \Pr(c_j \mid d_i; \Theta^{T-1})}{\sum_{s=1}^{|V|} \sum_{i=1}^{|D|} N_{si} \Pr(c_j \mid d_i; \Theta^{T-1})}. \tag{29}$$

To handle the 0 count problem (see Sect. 3.7.2 in Chap. 3), we can use Lidstone smoothing (Equation (27) in Chap. 3).

Bibliographic Notes

Learning with labeled and unlabeled examples (LU learning) using naïve Bayes and EM was proposed by Nigam et al. [413]. They also noted the problem of having mixtures of subclasses in the classes and proposed to identify and use such subclasses as a possible solution. A hierarchical clustering technique was also proposed by Cong et al. [111] for handling the mixture of subclasses problem. Castelli and Cover [83] presented a theoretical study of LU learning using mixture models.

Co-training was introduced by Blum and Mitchell [55]. Follow-on works include those by Collins and Singer [109], Goldman and Zhou [201], etc. Generalization error bounds within the Probably Approximately Correct (PAC) framework was given in [121] by Dasgupta et al. Nigam and Ghani [411] examined the importance of feature division in co-training and compared it to the EM algorithm and self-training.

Transduction was proposed by Vapnik [526] as learning when the test instances are known. Joachims described a heuristic algorithm and built a system for performing transduction using SVM [259]. The transductive SVM given in [259] can also be used for induction, i.e. classifying future unseen instances. In contrast, most graph-based methods are more suited for transduction, i.e. classifying test instances that are known during training. The graph-based mincut algorithm was introduced by Blum and Chalwa [54]. The graph-based Gaussian field method was proposed by Zhu et al. [619] while the spectral graph transducer was proposed by Joachims [261]. The edited book by Chapelle et al. [93] gives a comprehensive coverage of various LU learning algorithms.

On learning from positive and unlabeled examples (or PU learning), Denis [129] reported a theoretical study of PAC learning in this setting under the statistical query model [272], which basically assumes that the proportion of positive instances in the unlabeled set is known. Letouzey et al. [315] presented a learning algorithm based on a modified decision tree method in this model. PU learning is also studied theoretically by Muggleton [398] from the Bayesian framework where the distribution of functions and examples are assumed known. Liu et al. [348] gives another theoretical study, in which both the noiseless case and the noisy case are considered. It was concluded that a reasonable generalization (learning) can be achieved if the problem is posed as a constrained optimization problem (see Sect. 5.2.2). Most existing algorithms for solving the problem are based on this constrained optimization model.

Over the years, several practical algorithms were proposed. The first class of algorithms deals with the problem in two steps. These algorithms include S-EM [348], PEBL [582, 583], and Roc-SVM [321], which have been studied in this chapter. The second class of algorithm follows the theoretical result directly. Lee and Liu [309] described a weighted logistic regression technique. Liu et al. [340] described a biased-SVM technique. They both require a performance criterion to determine the quality of the classifier. The criterion is given in [309], which has been presented in Sect. 5.2.4. Liu et al. reported a comprehensive comparison of various techniques in [340]. It was shown that biased-SVM performed better than other techniques. Some other works on PU learning include those of Agichtein [6], Barbara et al. [35], Deng et al. [128], Denise et al. [130], Fung, et al. [188], Li and Liu [322], Zhang and Lee [603], etc.

A closely related work to PU learning is one-class SVM, which uses only positive examples to build a classifier. This method was proposed by Scholkopf et al. [478]. Manevitz and Yousef [360] studied text classification using one-class SVM. Li and Liu [321] showed that its accuracy results were poorer than PU learning for text classification. Unlabeled data does help classification significantly.