Chapter 11

Text Classification

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown Heights, NY
charu@us.ibm.com

ChengXiang Zhai
University of Illinois at Urbana-Champaign
Urbana, IL
czhai@cs.uiuc.edu

11.1 Introduction

The problem of classification has been widely studied in the database, data mining, and information retrieval communities. The problem of classification is defined as follows. Given a set of records $\mathcal{D} = \{X_1, \ldots, X_N\}$ and a set of k different discrete values indexed by $\{1 \ldots k\}$, each representing a category, the task is to assign one category (equivalently the corresponding index value) to each record X_i. The problem is usually solved by using a supervised learning approach where a set of training data records (i.e., records with known category labels) are used to construct a *classification model*, which relates the features in the underlying record to one of the class labels. For a given *test instance* for which the class is unknown, the training model is used to predict a class label for this instance. The problem may also be solved by using unsupervised approaches that do not require labeled training data, in which case keyword queries characterizing each class are often manually created, and bootstrapping may be used to heuristically obtain pseudo training data. Our review focuses on supervised learning approaches.

There are some variations of the basic problem formulation given above for text classifcation. In the *hard version* of the classification problem, a particular label is explicitly assigned to the instance, whereas in the *soft version* of the classification problem, a probability value is assigned to the test instance. Other variations of the classification problem allow ranking of different class choices for a test instance, or allow the assignment of multiple labels [60] to a test instance. The classification problem assumes categorical values for the labels, though it is also possible to use continuous values as labels. The latter is referred to as the regression modeling problem. The problem of text classification is closely related to that of classification of records with set-valued features [34]; however, this model assumes that only information about the presence or absence of words is used in a document. In reality, the frequency of words also plays a helpful role in the classification process, and the typical domain-size of text data (the entire lexicon size) is much greater than a typical set-valued classification problem.

A broad survey of a wide variety of classification methods may be found in [50,72], and a survey that is specific to the text domain may be found in [127]. A relative evaluation of different kinds of text classification methods may be found in [153]. A number of the techniques discussed in this chapter have also been converted into software and are publicly available through multiple toolkits such as the *BOW* toolkit [107], Mallot [110], WEKA,[1] and LingPipe.[2]

The problem of text classification finds applications in a wide variety of domains in text mining. Some examples of domains in which text classification is commonly used are as follows:

- **News Filtering and Organization:** Most of the news services today are electronic in nature in which a large volume of news articles are created every single day by the organizations. In such cases, it is difficult to organize the news articles manually. Therefore, automated methods can be very useful for news categorization in a variety of Web portals [90]. In the special case of binary categorization where the goal is to distinguish news articles interesting to a user from those that are not, the application is also referred to as *text filtering*.

- **Document Organization and Retrieval:** The above application is generally useful for many applications beyond news filtering and organization. A variety of supervised methods may be used for document organization in many domains. These include large digital libraries of doc-

[1] http://www.cs.waikato.ac.nz/ml/weka/
[2] http://alias-i.com/lingpipe/

uments, Web collections, scientific literature, or even social feeds. Hierarchically organized document collections can be particularly useful for browsing and retrieval [24].

- **Opinion Mining:** Customer reviews or opinions are often short text documents that can be mined to determine useful information from the review. Details on how classification can be used in order to perform opinion mining are discussed in [101]. A common classification task is to classify an opinionated text object (e.g., a product review) into positive or negative sentiment categories.
- **Email Classification and Spam Filtering:** It is often desirable to classify email [29, 33, 97] in order to determine either the subject or to determine junk email [129] in an automated way. This is also referred to as *spam filtering* or *email filtering*.

A wide variety of techniques have been designed for text classification. In this chapter, we will discuss the broad classes of techniques, and their uses for classification tasks. We note that these classes of techniques also generally exist for other data domains such as quantitative or categorical data. Since text may be modeled as quantitative data with frequencies on the word attributes, it is possible to use most of the methods for quantitative data directly on text. However, text is a particular kind of data in which the word attributes are sparse, and high dimensional, with low frequencies on most of the words. Therefore, it is critical to design classification methods that effectively account for these characteristics of text. In this chapter, we will focus on the specific changes that are applicable to the text domain. Some key methods, that are commonly used for text classification are as follows:

- **Decision Trees:** Decision trees are designed with the use of a hierarchical division of the underlying data space with the use of different text features. The hierarchical division of the data space is designed in order to create class partitions that are more skewed in terms of their class distribution. For a given text instance, we determine the partition that it is most likely to belong to, and use it for the purposes of classification.
- **Pattern (Rule)-Based Classifiers:** In rule-based classifiers we determine the word patterns that are most likely to be related to the different classes. We construct a set of rules, in which the left-hand side corresponds to a word pattern, and the right-hand side corresponds to a class label. These rules are used for the purposes of classification.
- **SVM Classifiers:** SVM Classifiers attempt to partition the data space with the use of linear or non-linear delineations between the different classes. The key in such classifiers is to determine the optimal boundaries between the different classes and use them for the purposes of classification.
- **Neural Network Classifiers:** Neural networks are used in a wide variety of domains for the purposes of classification. In the context of text data, the main difference for neural network classifiers is to adapt these classifiers with the use of word features. We note that neural network classifiers are related to SVM classifiers; indeed, they both are in the category of discriminative classifiers, which are in contrast with the *generative classifiers* [116].
- **Bayesian (Generative) Classifiers:** In Bayesian classifiers (also called generative classifiers), we attempt to build a probabilistic classifier based on modeling the underlying word features in different classes. The idea is then to classify text based on the posterior probability of the documents belonging to the different classes on the basis of the word presence in the documents.
- **Other Classifiers:** Almost all classifiers can be adapted to the case of text data. Some of the other classifiers include nearest neighbor classifiers, and genetic algorithm-based classifiers. We will discuss some of these different classifiers in some detail and their use for the case of text data.

The area of text categorization is so vast that it is impossible to cover all the different algorithms in detail in a single chapter. Therefore, our goal is to provide the reader with an overview of the most important techniques, and also the pointers to the different variations of these techniques.

Feature selection is an important problem for text classification. In feature selection, we attempt to determine the features that are most relevant to the classification process. This is because some of the words are much more likely to be correlated to the class distribution than others. Therefore, a wide variety of methods have been proposed in the literature in order to determine the most important features for the purpose of classification. These include measures such as the gini-index or the entropy, which determine the level at which the presence of a particular feature skews the class distribution in the underlying data. We will discuss the different feature selection methods that are commonly used for text classification.

The rest of this chapter[3] is organized as follows. In the next section, we will discuss methods for feature selection in text classification. In Section 11.3, we will describe decision tree methods for text classification. Rule-based classifiers are described in detail in Section 11.4. We discuss naive Bayes classifiers in Section 11.5. The nearest neighbor classifier is discussed in Section 11.7. In Section 11.6, we will discuss a number of linear classifiers, such as the SVM classifier, direct regression modeling, and the neural network classifier. A discussion of how the classification methods can be adapted to text and Web data containing hyperlinks is discussed in Section 11.8. In Section 11.9, we discuss a number of different meta-algorithms for classification such as boosting, bagging, and ensemble learning. Methods for enhancing classification methods with additional training data are discussed in Section 11.10. Section 11.11 contains the conclusions and summary.

11.2 Feature Selection for Text Classification

Before any classification task, one of the most fundamental tasks that needs to be accomplished is that of document representation and feature selection. While feature selection is also desirable in other classification tasks, it is especially important in text classification due to the high dimensionality of text features and the existence of irrelevant (noisy) features. In general, text can be represented in two separate ways. The first is as a bag of words, in which a document is represented as a set of words, together with their associated frequency in the document. Such a representation is essentially independent of the sequence of words in the collection. The second method is to represent text directly as *strings*, in which each document is a sequence of words. Most text classification methods use the bag-of-words representation because of its simplicity for classification purposes. In this section, we will discuss some of the methods that are used for feature selection in text classification.

The most common feature selection that is used in both supervised and unsupervised applications is that of stop-word removal and stemming. In stop-word removal, we determine the common words in the documents that are not specific or discriminatory to the different classes. In stemming, different forms of the same word are consolidated into a single word. For example, singular, plural, and different tenses are consolidated into a single word. We note that these methods are not specific

[3] Another survey on text classification may be found in [1]. This chapter is an up-to-date survey, and contains material on topics such as SVM, Neural Networks, active learning, semisupervised learning, and transfer learning.

to the case of the classification problem, and are often used in a variety of unsupervised applications such as clustering and indexing. In the case of the classification problem, it makes sense to supervise the feature selection process with the use of the class labels. This kind of selection process ensures that those features that are highly skewed towards the presence of a particular class label are picked for the learning process. A wide variety of feature selection methods are discussed in [154, 156]. Many of these feature selection methods have been compared with one another, and the experimental results are presented in [154]. We will discuss each of these feature selection methods in this section.

11.2.1 Gini Index

One of the most common methods for quantifying the discrimination level of a feature is the use of a measure known as the *gini-index*. Let $p_1(w) \ldots p_k(w)$ be the fraction of class-label presence of the k different classes for the word w. In other words, $p_i(w)$ is the conditional probability that a document belongs to class i, given the fact that it contains the word w. Therefore, we have:

$$\sum_{i=1}^{k} p_i(w) = 1. \tag{11.1}$$

Then, the gini-index for the word w, denoted by $G(w)$, is defined[4] as follows:

$$G(w) = \sum_{i=1}^{k} p_i(w)^2 \tag{11.2}$$

The value of the gini-index $G(w)$ always lies in the range $(1/k, 1)$. Higher values of the gini-index $G(w)$ represent a greater discriminative power of the word w. For example, when all documents that contain word w belong to a particular class, the value of $G(w)$ is 1. On the other hand, when documents containing word w are evenly distributed among the k different classes, the value of $G(w)$ is $1/k$.

One criticism with this approach is that the global class distribution may be skewed to begin with, and therefore the above measure may sometimes not accurately reflect the discriminative power of the underlying attributes. Therefore, it is possible to construct a normalized gini-index in order to reflect the discriminative power of the attributes more accurately. Let $P_1 \ldots P_k$ represent the global distributions of the documents in the different classes. Then, we determine the normalized probability value $p_i'(w)$ as follows:

$$p_i'(w) = \frac{p_i(w)/P_i}{\sum_{j=1}^{k} p_j(w)/P_j}. \tag{11.3}$$

Then, the gini-index is computed in terms of these normalized probability values.

$$G(w) = \sum_{i=1}^{k} p_i'(w)^2. \tag{11.4}$$

The use of the global probabilities P_i ensures that the gini-index more accurately reflects class-discrimination in the case of biased class distributions in the whole document collection. For a document corpus containing n documents, d words, and k classes, the complexity of the information gain computation is $O(n \cdot d \cdot k)$. This is because the computation of the term $p_i(w)$ for all the different words and the classes requires $O(n \cdot d \cdot k)$ time.

[4]The gini-index is also sometimes defined as $1 - \sum_{i=1}^{k} p_i(w)^2$, with lower values indicating greater discriminative power of the feature w.

11.2.2 Information Gain

Another related measure that is commonly used for text feature selection is that of information gain or entropy. Let P_i be the global probability of class i, and $p_i(w)$ be the probability of class i, given that the document contains the word w. Let $F(w)$ be the fraction of the documents containing the word w. The information gain measure $I(w)$ for a given word w is defined as follows:

$$I(w) = -\sum_{i=1}^{k} P_i \cdot \log(P_i) + F(w) \cdot \sum_{i=1}^{k} p_i(w) \cdot \log(p_i(w)) + \\ +(1 - F(w)) \cdot \sum_{i=1}^{k} (1 - p_i(w)) \cdot \log(1 - p_i(w)).$$

The greater the value of the information gain $I(w)$, the greater the discriminatory power of the word w. For a document corpus containing n documents and d words, the complexity of the information gain computation is $O(n \cdot d \cdot k)$.

11.2.3 Mutual Information

This *mutual information measure* is derived from information theory [37], and provides a formal way to model the mutual information between the features and the classes. The pointwise mutual information $M_i(w)$ between the word w and the class i is defined on the basis of the level of co-occurrence between the class i and word w. We note that the expected co-occurrence of class i and word w on the basis of mutual independence is given by $P_i \cdot F(w)$. The true co-occurrence is of course given by $F(w) \cdot p_i(w)$. In practice, the value of $F(w) \cdot p_i(w)$ may be much larger or smaller than $P_i \cdot F(w)$, depending upon the level of correlation between the class i and word w. The mutual information is defined in terms of the ratio between these two values. Specifically, we have:

$$M_i(w) = \log\left(\frac{F(w) \cdot p_i(w)}{F(w) \cdot P_i}\right) = \log\left(\frac{p_i(w)}{P_i}\right). \tag{11.5}$$

Clearly, the word w is positively correlated to the class i, when $M_i(w) > 0$, and the word w is negatively correlated to class i, when $M_i(w) < 0$. We note that $M_i(w)$ is specific to a particular class i. We need to compute the overall mutual information as a function of the mutual information of the word w with the different classes. These are defined with the use of the average and maximum values of $M_i(w)$ over the different classes.

$$M_{avg}(w) = \sum_{i=1}^{k} P_i \cdot M_i(w)$$
$$M_{max}(w) = \max_i\{M_i(w)\}.$$

Either of these measures may be used in order to determine the relevance of the word w. The second measure is particularly useful, when it is more important to determine high levels of positive correlation of the word w with any of the classes.

11.2.4 χ^2-Statistic

The χ^2 statistic is a different way to compute the lack of independence between the word w and a particular class i. Let n be the total number of documents in the collection, $p_i(w)$ be the conditional probability of class i for documents that contain w, P_i be the global fraction of documents containing the class i, and $F(w)$ be the global fraction of documents that contain the word w. The χ^2-statistic of the word between word w and class i is defined as follows:

$$\chi_i^2(w) = \frac{n \cdot F(w)^2 \cdot (p_i(w) - P_i)^2}{F(w) \cdot (1 - F(w)) \cdot P_i \cdot (1 - P_i))} \tag{11.6}$$

As in the case of the mutual information, we can compute a global χ^2 statistic from the class-specific values. We can use either the average or maximum values in order to create the composite value:

$$\chi^2_{avg}(w) = \sum_{i=1}^{k} P_i \cdot \chi^2_i(w)$$

$$\chi^2_{max}(w) = \max_i \chi^2_i(w)$$

We note that the χ^2-statistic and mutual information are different ways of measuring the correlation between terms and categories. One major advantage of the χ^2-statistic over the mutual information measure is that it is a normalized value, and therefore these values are more comparable across terms in the same category.

11.2.5 Feature Transformation Methods: Unsupervised and Supervised LSI

While feature selection attempts to reduce the dimensionality of the data by picking from the original set of attributes, feature transformation methods create a new (and smaller) set of features as a function of the original set of features. The motivation of these feature transformation methods is to capture semantic associations of words so that words related to the same latent concept would be grouped together into one latent concept feature. A typical example of such a feature transformation method is Latent Semantic Indexing (LSI) [46], and its probabilistic variant PLSA [67]. The LSI method transforms the text space of a few hundred thousand word features to a new axis system (of size about a few hundred), which is a linear combination of the original word features. In order to achieve this goal, Principal Component Analysis techniques [81] are used to determine the axis-system that retains the greatest level of information about the variations in the underlying attribute values. The main disadvantage of using techniques such as LSI is that these are unsupervised techniques that are blind to the underlying class-distribution. Thus, the features found by LSI are not necessarily the directions along which the *class-distribution* of the underlying documents can best be separated. A modest level of success has been obtained in improving classification accuracy by using boosting techniques in conjunction with the conceptual features obtained from unsupervised pLSA method [22]. A more recent study has systematically compared pLSA and LDA (which is a Bayesian version of pLSA) in terms of their effectiveness in transforming features for text categorization, and has drawn a similar conclusion and found that pLSA and LDA tend to perform similarly [104]. In the case of classification tasks that require fine-grained discrimination, such as search engine applications where the task is to distinguish relevant from non-relevant documents to a query, the latent low-dimensional features obtained by LSI or PLSA alone are often insufficient, and a more discriminative representation based on keywords may have to be used as well to obtain a multiscale representation [161].

A number of techniques have also been proposed to perform the feature transformation methods by using the class labels for effective supervision. The most natural method is to adapt LSI in order to make it work more effectively for the supervised case. A number of different methods have been proposed in this direction. One common approach is to perform local LSI on the subsets of data representing the individual classes, and identify the discriminative eigenvectors from the different reductions with the use of an iterative approach [142]. This method is known as SLSI (Supervised Latent Semantic Indexing), and the advantages of the method seem to be relatively limited, because the experiments in [142] show that the improvements over a standard SVM classifier, which did not use a dimensionality reduction process, are relatively limited. The work in [149] uses a combination of class-specific LSI and global analysis. As in the case of [142], class-specific LSI representations are created. Test documents are compared against each LSI representation in order to create the most discriminative reduced space. One problem with this approach is that the different local LSI representations use a different subspace, and therefore it is difficult to compare the similarities of the different documents across the different subspaces. Furthermore, both methods in [142, 149] tend to

be computationally expensive. Similar ideas were also implemented in a probabilistic topic model, called supervised LDA [16], where class labels are used to "supervise" the discovery of topics in text data [16].

A method called *sprinkling* is proposed in [26], in which artificial terms are added to (or "sprinkled" into) the documents, which correspond to the class labels. In other words, we create a term corresponding to the class label, and add it to the document. LSI is then performed on the document collection with these added terms. The sprinkled terms can then be removed from the representation, once the eigenvectors have been determined. The sprinkled terms help in making the LSI more sensitive to the class distribution during the reduction process. It has also been proposed in [26] that it can be generally useful to make the sprinkling process *adaptive*, in which all classes are not necessarily treated equally, but the relationships between the classes are used in order to regulate the sprinkling process. Furthermore, methods have also been proposed in [26] to make the sprinkling process adaptive to the use of a particular kind of classifier.

11.2.6 Supervised Clustering for Dimensionality Reduction

One technique that is commonly used for feature transformation is that of text clustering [8, 83, 95, 140]. In these techniques, the clusters are constructed from the underlying text collection, with the use of supervision from the class distribution. The exception is [95] in which supervision is not used. In the simplest case, each class can be treated as a separate cluster, though better results may be obtained by using the classes for supervision of the clustering process. The frequently occurring words in these supervised clusters can be used in order to create the new set of dimensions. The classification can be performed with respect to this new feature representation. One advantage of such an approach is that it retains interpretability with respect to the original words of the document collection. The disadvantage is that the optimum directions of separation may not necessarily be represented in the form of clusters of words. Furthermore, the underlying axes are not necessarily orthonormal to one another. The use of supervised methods [2, 8, 83, 140] has generally led to good results either in terms of improved classification accuracy, or significant performance gains at the expense of a small reduction in accuracy. The results with the use of unsupervised clustering [95, 99] are mixed. For example, the work in [95] suggests that the use of unsupervised term-clusters and phrases is generally not helpful [95] for the classification process. The key observation in [95] is that the loss of granularity associated with the use of phrases and term clusters is not necessarily advantageous for the classification process. The loss of discrimination due to dimension reduction is also recently discussed in the context of using probabilistic topic models to obtain a low-dimensional feature representation for text data [161]. The work in [9] has shown that the use of the information bottleneck method for feature distributional clustering can create clustered pseudo-word representations that are quite effective for text classification.

11.2.7 Linear Discriminant Analysis

Another method for feature transformation is the use of linear discriminants, which explicitly try to construct directions in the feature space, along which there is best separation of the different classes. A common method is the *Fisher's linear discriminant* [54]. The main idea in the Fisher's discriminant method is to determine the directions in the data along which the points are as well separated as possible. The subspace of lower dimensionality is constructed by iteratively finding such unit vectors α_i in the data, where α_i is determined in the ith iteration. We would also like to ensure that the different values of α_i are orthonormal to one another. In each step, we determine this vector α_i by discriminant analysis, and project the data onto the remaining orthonormal subspace. The next vector α_{i+1} is found in this orthonormal subspace. The quality of vector α_i is measured by an objective function that measures the separation of the different classes. This objective function reduces in each iteration, since the value of α_i in a given iteration is the optimum discriminant in that

subspace, and the vector found in the next iteration is the optimal one from a smaller search space. The process of finding linear discriminants is continued until the class separation, as measured by an objective function, reduces below a given threshold for the vector determined in the current iteration. The power of such a dimensionality reduction approach has been illustrated in [23], in which it has been shown that a simple decision tree classifier can perform much more effectively on this transformed data, as compared to more sophisticated classifiers.

Next, we discuss how the Fisher's discriminant is actually constructed. First, we will set up the objective function $J(\overline{\alpha})$, which determines the level of separation of the different classes along a given direction (unit-vector) $\overline{\alpha}$. This sets up the crisp optimization problem of determining the value of $\overline{\alpha}$, which maximizes $J(\overline{\alpha})$. For simplicity, let us assume the case of binary classes. Let D_1 and D_2 be the two sets of documents belonging to the two classes. Then, the projection of a document $\overline{X} \in D_1 \cup D_2$ along $\overline{\alpha}$ is given by $\overline{X} \cdot \overline{\alpha}$. Therefore, the squared class separation $S(D_1, D_2, \overline{\alpha})$ along the direction $\overline{\alpha}$ is given by:

$$S(D_1, D_2, \overline{\alpha}) = \left(\frac{\sum_{\overline{X} \in D_1} \overline{\alpha} \cdot \overline{X}}{|D_1|} - \frac{\sum_{\overline{X} \in D_2} \overline{\alpha} \cdot \overline{X}}{|D_2|} \right)^2. \tag{11.7}$$

In addition, we need to normalize this absolute class separation with the use of the underlying class variances. Let $Var(D_1, \overline{\alpha})$ and $Var(D_2, \overline{\alpha})$ be the individual class variances along the direction α. In other words, we have:

$$Var(D_1, \overline{\alpha}) = \frac{\sum_{\overline{X} \in D_1} (\overline{X} \cdot \overline{\alpha})^2}{|D_1|} - \left(\frac{\sum_{\overline{X} \in D_1} \overline{X} \cdot \overline{\alpha}}{|D_1|} \right)^2. \tag{11.8}$$

The value of $Var(D_2, \overline{\alpha})$ can be defined in a similar way. Then, the normalized class-separation $J(\overline{\alpha})$ is defined as follows:

$$J(\overline{\alpha}) = \frac{S(D_1, D_2, \overline{\alpha})}{Var(D_1, \overline{\alpha}) + Var(D_2, \overline{\alpha})} \tag{11.9}$$

The optimal value of α needs to be determined subject to the constraint that $\overline{\alpha}$ is a unit vector. Let μ_1 and μ_2 be the means of the two data sets D_1 and D_2, and C_1 and C_2 be the corresponding covariance matrices. It can be shown that the optimal (unscaled) direction $\overline{\alpha} = \overline{\alpha^*}$ can be expressed in closed form, and is given by the following:

$$\overline{\alpha^*} = \left(\frac{C_1 + C_2}{2} \right)^{-1} (\mu_1 - \mu_2). \tag{11.10}$$

The main difficulty in computing the above equation is that this computation requires the inversion of the covariance matrix, which is sparse and computationally difficult in the high-dimensional text domain. Therefore, a gradient descent approach can be used in order to determine the value of $\overline{\alpha}$ in a more computationally effective way. Details of the approach are presented in [23].

Another related method that attempts to determine projection directions that maximize the topical differences between different classes is the *Topical Difference Factor Analysis* method proposed in [84]. The problem has been shown to be solvable as a generalized eigenvalue problem. The method was used in conjunction with a k-nearest neighbor classifier, and it was shown that the use of this approach significantly improves the accuracy over a classifier that uses the original set of features.

11.2.8 Generalized Singular Value Decomposition

While the method discussed above finds one vector $\overline{\alpha_i}$ at a time in order to determine the relevant dimension transformation, it is possible to be much more direct in finding the optimal subspaces simultaneously by using a generalized version of dimensionality reduction [68, 69]. It is important to

note that this method has really been proposed in [68, 69] as an *unsupervised* method that preserves the underlying *clustering* structure, assuming the data has already been clustered in a pre-processing phase. Thus, the generalized dimensionality reduction method has been proposed as a much more aggressive dimensionality reduction technique, which preserves the underlying clustering structure rather than the individual points. This method can however also be used as a *supervised* technique in which the different classes are used as input to the dimensionality reduction algorithm, instead of the clusters constructed in the pre-processing phase [152]. This method is known as the *Optimal Orthogonal Centroid Feature Selection Algorithm (OCFS)*, and it directly targets at the maximization of inter-class scatter. The algorithm is shown to have promising results for supervised feature selection in [152].

11.2.9 Interaction of Feature Selection with Classification

Since the classification and feature selection processes are dependent upon one another, it is interesting to test how the feature selection process interacts with the underlying classification algorithms. In this context, two questions are relevant:

- Can the feature-specific insights obtained from the intermediate results of some of the classification algorithms be used for creating feature selection methods that can be used more generally by other classification algorithms?
- Do the different feature selection methods work better or worse with different kinds of classifiers?

Both these issues were explored in some detail in [113]. In regard to the first question, it was shown in [113] that feature selection, which was derived from *linear* classifiers, provided very effective results. In regard to the second question, it was shown in [113] that the sophistication of the feature selection process itself was more important than the specific pairing between the feature selection process and the classifier.

Linear Classifiers are those for which the output of the linear predictor is defined to be $p = \overline{A} \cdot \overline{X} + b$, where $\overline{X} = (x_1 \ldots x_n)$ is the normalized document word frequency vector, $\overline{A} = (a_1 \ldots a_n)$ is a vector of linear coefficients with the same dimensionality as the feature space, and b is a scalar. Both the basic neural network and basic SVM classifiers [75] (which will be discussed later in this chapter) belong to this category. The idea here is that if the coefficient a_i is close to zero, then the corresponding feature does not have a significant effect on the classification process. On the other hand, since large absolute values of a_j may significantly influence the classification process, such features should be selected for classification. In the context of the SVM method, which attempts to determine linear planes of separation between the different classes, the vector $\overline{A}$ is essentially the normal vector to the corresponding plane of separation between the different classes. This intuitively explains the choice of selecting features with large values of $|a_j|$. It was shown in [113] that this class of feature selection methods was quite robust, and performed well even for classifiers such as the Naive Bayes method, which were unrelated to the linear classifiers from which these features were derived. Further discussions on how SVM and maximum margin techniques can be used for feature selection may be found in [59, 65].

11.3 Decision Tree Classifiers

A decision tree [122] is essentially a hierarchical decomposition of the (training) data space, in which a *predicate* or a condition on the attribute value is used in order to divide the data space

hierarchically. In the context of text data, such predicates are typically conditions on the presence or absence of one or more words in the document. The division of the data space is performed recursively in the decision tree, until the leaf nodes contain a certain minimum number of records, or some conditions on class purity. The majority class label (or cost-weighted majority label) in the leaf node is used for the purposes of classification. For a given test instance, we apply the sequence of predicates at the nodes, in order to traverse a path of the tree in top-down fashion and determine the relevant leaf node. In order to further reduce the overfitting, some of the nodes may be pruned by holding out a part of the data, which are not used to construct the tree. The portion of the data which is held out is used in order to determine whether or not the constructed leaf node should be pruned or not. In particular, if the class distribution in the training data (for decision tree construction) is very different from the class distribution in the training data that is used for pruning, then it is assumed that the node overfits the training data. Such a node can be pruned. A detailed discussion of decision tree methods may be found in [20, 50, 72, 122].

In the particular case of text data, the predicates for the decision tree nodes are typically defined in terms of the terms in the underlying text collection. For example, a node may be partitioned into its children nodes depending upon the presence or absence of a particular term in the document. We note that different nodes at the same level of the tree may use different terms for the partitioning process.

Many other kinds of predicates are possible. It may not be necessary to use individual terms for partitioning, but one may measure the similarity of documents to correlated sets of terms. These correlated sets of terms may be used to further partition the document collection, based on the similarity of the document to them. The different kinds of splits are as follows:

- **Single Attribute Splits:** In this case, we use the presence or absence of particular words (or even phrases) at a particular node in the tree in order to perform the split. At any given level, we pick the word that provides the maximum discrimination between the different classes. Measures such as the gini-index or information gain can be used in order to determine the level of entropy. For example, the DT-min10 algorithm [93] is based on this approach.

- **Similarity-Based Multi-Attribute Split:** In this case, we use documents (or meta-documents such as frequent word clusters), and use the similarity of the documents to these word clusters in order to perform the split. For the selected word cluster, the documents are further partitioned into groups by rank ordering the documents by similarity value, and splitting at a particular threshold. We select the word-cluster for which rank-ordering by similarity provides the best separation between the different classes.

- **Discriminant-Based Multi-Attribute Split:** For the multi-attribute case, a natural choice for performing the split is to use discriminants such as the Fisher discriminant for performing the split. Such discriminants provide the directions in the data along which the classes are best separated. The documents are projected on this discriminant vector for rank ordering, and then split at a particular coordinate. The choice of split point is picked in order to maximize the discrimination between the different classes. The work in [23] uses a discriminant-based split, though this is done indirectly because of the use of a feature transformation to the discriminant representation, before building the classifier.

Some of the earliest implementation of classifiers may be found in [92, 93, 99, 147]. The last of these is really a rule-based classifier, which can be interpreted either as a decision tree or a rule-based classifier. Most of the decision tree implementations in the text literature tend to be small variations on standard packages such as ID3 and C4.5, in order to adapt the model to text classification. Many of these classifiers are typically designed as baselines for comparison with other learning models [75].

A well known implementation of the decision tree classifier based on the C4.5 taxonomy of algorithms [122] is presented in [99]. More specifically, the work in [99] uses the successor to the C4.5 algorithm, which is also known as the C5 algorithm. This algorithm uses single-attribute splits at each node, where the feature with the highest information gain [37] is used for the purpose of the split. Decision trees have also been used in conjunction with boosting techniques. An adaptive boosting technique [56] is used in order to improve the accuracy of classification. In this technique, we use n different classifiers. The ith classifier is constructed by examining the errors of the $(i-1)$th classifier. A voting scheme is applied among these classifiers in order to report the final label. Other boosting techniques for improving decision tree classification accuracy are proposed in [134].

The work in [51] presents a decision tree algorithm based on the Bayesian approach developed in [28]. In this classifier, the decision tree is grown by recursive greedy splits, where the splits are chosen using Bayesian posterior probability of model structure. The structural prior penalizes additional model parameters at each node. The output of the process is a class probability rather than a deterministic class label for the test instance.

11.4 Rule-Based Classifiers

Decision trees are also generally related to *rule-based classifiers*. In rule-based classifiers, the data space is modeled with a set of rules, in which the left-hand side is a condition on the underlying feature set, and the right-hand side is the class label. The rule set is essentially the model that is generated from the training data. For a given test instance, we determine the set of rules for which the test instance satisfies the condition on the left-hand side of the rule. We determine the predicted class label as a function of the class labels of the rules that are satisfied by the test instance. We will discuss more on this issue slightly later.

In its most general form, the left-hand side of the rule is a boolean condition, which is expressed in Disjunctive Normal Form (DNF). However, in most cases, the condition on the left-hand side is much simpler and represents a set of terms, all of which must be present in the document for the condition to be satisfied. The *absence* of terms is rarely used, because such rules are not likely to be very informative for sparse text data, in which most words in the lexicon will typically not be present in it by default (sparseness property). Also, while the *set intersection* of conditions on term presence is used often, the union of such conditions is rarely used in a single rule. This is because such rules can be split into two separate rules, each of which is more informative on its own. For example, the rule $Honda \cup Toyota \Rightarrow Cars$ can be replaced by two separate rules $Honda \Rightarrow Cars$ and $Toyota \Rightarrow Cars$ without any loss of information. In fact, since the confidence of each of the two rules can now be measured separately, this can be more useful. On the other hand, the rule $Honda \cap Toyota \Rightarrow Cars$ is certainly much more informative than the individual rules. Thus, in practice, for sparse data sets such as text, rules are much more likely to be expressed as a simple conjunction of conditions on term presence.

We note that decision trees and decision rules both tend to encode rules on the feature space, except that the decision tree tends to achieve this goal with a hierarchical approach. In fact, the original work on decision tree construction in C4.5 [122] studied the decision tree problem and decision rule problem within a single framework. This is because a particular path in the decision tree can be considered a rule for classification of the text instance. The main difference is that the decision tree framework is a strict hierarchical partitioning of the data space, whereas rule-based classifiers allow for overlaps in the decision space. The general principle is to create a rule set, such that all points in the decision space are covered by *at least* one rule. In most cases, this is achieved

by generating a set of targeted rules that are related to the different classes, and one default *catch-all* rule, which can cover all the remaining instances.

A number of criteria can be used in order to generate the rules from the training data. Two of the most common conditions that are used for rule generation are those of *support* and *confidence*. These conditions are common to all rule-based pattern classifiers [100] and may be defined as follows:

- **Support:** This quantifies the **absolute number** of instances in the training data set that are relevant to the rule. For example, in a corpus containing 100,000 documents, a rule in which **both** the left-hand set and right-hand side are satisfied by 50,000 documents is more important than a rule that is satisfied by 20 documents. Essentially, this quantifies the statistical *volume* that is associated with the rule. However, it does not encode the strength of the rule.

- **Confidence:** This quantifies the **conditional probability** that the right-hand side of the rule is satisfied, if the left-hand side is satisfied. This is a more direct measure of the strength of the underlying rule.

We note that the aforementioned measures are not the only measures that are possible, but are widely used in the data mining and machine learning literature [100] for both textual and non-textual data, because of their intuitive nature and simplicity of interpretation. One criticism of the above measures is that they do not normalize for the a-priori presence of different terms and features, and are therefore prone to misinterpretation, when the feature distribution or class-distribution in the underlying data set is skewed.

The training phase constructs all the rules, which are based on measures such as the above. For a given test instance, we determine all the rules that are relevant to the test instance. Since we allow overlaps, it is possible that more than one rule may be relevant to the test instance. If the class labels on the right-hand sides of all these rules are the same, then it is easy to pick this class as the relevant label for the test instance. On the other hand, the problem becomes more challenging when there are conflicts between these different rules. A variety of different methods are used to rank-order the different rules [100], and report the most relevant rule as a function of these different rules. For example, a common approach is to rank-order the rules by their confidence, and pick the top-k rules as the most relevant. The class label on the right-hand side of the greatest number of these rules is reported as the relevant one.

An interesting rule-based classifier for the case of text data has been proposed in [6]. This technique uses an iterative methodology, which was first proposed in [148] for generating rules. Specifically, the method determines the single best rule related to any particular class in the training data. The best rule is defined in terms of the confidence of the rule, as defined above. This rule along with its corresponding instances are removed from the training data set. This approach is continuously repeated, until it is no longer possible to find strong rules in the training data, and complete predictive value is achieved.

The transformation of decision trees to rule-based classifiers is discussed generally in [122], and for the particular case of text data in [80]. For each path in the decision tree a rule can be generated, which represents the conjunction of the predicates along that path. One advantage of the rule-based classifier over a decision tree is that it is not restricted to a strict hierarchical partitioning of the feature space, and it allows for overlaps and inconsistencies among the different rules. Therefore, if a new set of training examples are encountered, which are related to a new class or new part of the feature space, then it is relatively easy to modify the rule set for these new examples. Furthermore, rule-based classifiers also allow for a tremendous interpretability of the underlying decision space. In cases in which domain-specific expert knowledge is known, it is possible to encode this into the classification process by manual addition of rules. In many practical scenarios, rule-based techniques are more commonly used because of their ease of maintenance and interpretability.

One of the most common rule-based techniques is the *RIPPER* technique discussed in [32–34]. The *RIPPER* technique uses the sequential covering paradigm to determine the combinations of

words that are related to a particular class. The *RIPPER* method has been shown to be especially effective in scenarios where the number of training examples is relatively small [31]. Another method called *sleeping experts* [32,57] generates rules that take the placement of the words in the documents into account. Most of the classifiers such as *RIPPER* [32–34] treat documents as set-valued objects, and generate rules based on the co-presence of the words in the documents. The rules in *sleeping experts* are different from most of the other classifiers in this respect. In this case [32,57], the left-hand side of the rule consists of a *sparse phrase*, which is a group of words close to one another in the document (though not necessarily completely sequential). Each such rule has a weight, which depends upon its classification specificity in the training data. For a given test example, we determine the sparse phrases that are present in it, and perform the classification by combining the weights of the different rules that are fired. The *sleeping experts* and *RIPPER* systems have been compared in [32], and have been shown to have excellent performance on a variety of text collections.

11.5 Probabilistic and Naive Bayes Classifiers

Probabilistic classifiers are designed to use an implicit mixture model for generation of the underlying documents. This mixture model typically assumes that each class is a component of the mixture. Each mixture component is essentially a generative model, which provides the probability of sampling a particular term for that component or class. This is why this kind of classifiers are often also called generative classifiers. The naive Bayes classifier is perhaps the simplest and also the most commonly used generative classifier. It models the distribution of the documents in each class using a probabilistic model with independence assumptions about the distributions of different terms. Two classes of models are commonly used for naive Bayes classification. Both models essentially compute the posterior probability of a class, based on the distribution of the words in the document. These models ignore the actual position of the words in the document, and work with the "bag of words" assumption. The major difference between these two models is the assumption in terms of taking (or not taking) word frequencies into account, and the corresponding approach for sampling the probability space:

- **Multivariate Bernoulli Model:** In this model, we use the presence or absence of words in a text document as features to represent a document. Thus, the frequencies of the words are not used for the modeling a document, and the word features in the text are assumed to be binary, with the two values indicating presence or absence of a word in text. Since the features to be modeled are binary, the model for documents in each class is a multivariate Bernoulli model.

- **Multinomial Model:** In this model, we capture the frequencies of terms in a document by representing a document with a bag of words. The documents in each class can then be modeled as samples drawn from a multinomial word distribution. As a result, the conditional probability of a document given a class is simply a product of the probability of each observed word in the corresponding class.

No matter how we model the documents in each class (be it a multivariate Bernoulli model or a multinomial model), the component class models (i.e., generative models for documents in each class) can be used in conjunction with the Bayes rule to compute the posterior probability of the class for a given document, and the class with the highest posterior probability can then be assigned to the document.

There has been considerable confusion in the literature on the differences between the multivariate Bernoulli model and the multinomial model. A good exposition of the differences between these two models may be found in [108]. In the following, we describe these two models in more detail.

11.5.1 Bernoulli Multivariate Model

This class of techniques treats a document as a set of distinct words with no frequency information, in which an element (term) may be either present or absent. The seminal work on this approach may be found in [94].

Let us assume that the lexicon from which the terms are drawn are denoted by $V = \{t_1 \ldots t_n\}$. Let us assume that the bag-of-words (or text document) in question contains the terms $Q = \{t_{i_1} \ldots t_{i_m}\}$, and the class is drawn from $\{1 \ldots k\}$. Then, our goal is to model the posterior probability that the document (which is assumed to be generated from the term distributions of one of the classes) belongs to class i, given that it contains the terms $Q = \{t_{i_1} \ldots t_{i_m}\}$. The best way to understand the Bayes method is by understanding it as a sampling/generative process from the underlying mixture model of classes. The Bayes probability of class i can be modeled by sampling a set of terms T from the term distribution of the classes:

If we sampled a term set T of any size **from the term distribution of one of the randomly chosen classes***, and the final outcome is the set Q, then what is the posterior probability that we had originally picked class i for sampling? The a-priori probability of picking class i is equal to its fractional presence in the collection.*

We denote the class of the sampled set T by C^T and the corresponding posterior probability by $P(C^T = i|T = Q)$. This is essentially what we are trying to find. It is important to note that since we do not allow replacement, we are essentially picking a subset of terms from V with no frequencies attached to the picked terms. Therefore, the set Q may not contain duplicate elements. Under the naive Bayes assumption of independence between terms, this is essentially equivalent to either selecting or not selecting each term with a probability that depends upon the underlying term distribution. Furthermore, it is also important to note that this model has no restriction on the number of terms picked. As we will see later, these assumptions are the key differences with the multinomial Bayes model. The Bayes approach classifies a given set Q based on the posterior probability that Q is a sample from the data distribution of class i, i.e., $P(C^T = i|T = Q)$, and it requires us to compute the following two probabilities in order to achieve this:

1. What is the prior probability that a set T is a sample from the term distribution of class i? This probability is denoted by $P(C^T = i)$.

2. If we sampled a set T of any size *from the term distribution of class i*, then what is the probability that our sample is the set Q? This probability is denoted by $P(T = Q|C^T = i)$.

We will now provide a more mathematical description of Bayes modeling. In other words, we wish to model $P(C^T = i|Q$ is sampled). We can use the Bayes rule in order to write this conditional probability in a way that can be *estimated* more easily from the underlying corpus. In other words, we can simplify as follows:

$$P(C^T = i|T = Q) = \frac{P(C^T = i) \cdot P(T = Q|C^T = i)}{P(T = Q)}$$

$$= \frac{P(C^T = i) \cdot \prod_{t_j \in Q} P(t_j \in T|C^T = i) \cdot \prod_{t_j \notin Q} (1 - P(t_j \in T|C^T = i))}{P(T = Q)}.$$

We note that the last condition of the above sequence uses the *naive independence assumption*, because we are assuming that the probabilities of occurrence of the different terms are independent of one another. This is practically necessary, in order to transform the probability equations to a form that can be estimated from the underlying data.

The class assigned to Q is the one with the highest posterior probability given Q. It is easy to see that this decision is not affected by the denominator, which is the marginal probability of observing Q. That is, we will assign the following class to Q:

$$\begin{aligned}\hat{i} &= \arg\max_i P(C^T = i | T = Q) \\ &= \arg\max_i P(C^T = i) \cdot \\ &\quad \prod_{t_j \in Q} P(t_j \in T | C^T = i) \cdot \prod_{t_j \notin Q} (1 - P(t_j \in T | C^T = i)).\end{aligned}$$

It is important to note that all terms in the right hand-side of the last equation can be estimated from the training corpus. The value of $P(C^T = i)$ is estimated as the global fraction of documents belonging to class i, the value of $P(t_j \in T | C^T = i)$ is the fraction of documents in the ith class that contain term t_j. We note that all of the above are maximum likelihood estimates of the corresponding probabilities. In practice, Laplacian smoothing [144] is used, in which small values are added to the frequencies of terms in order to avoid zero probabilities of sparsely present terms.

In most applications of the Bayes classifier, we only care about the *identity* of the class with the highest probability value, rather than the actual probability value associated with it, which is why we do not need to compute the normalizer $P(T = Q)$. In fact, in the case of **binary** classes, a number of simplifications are possible in computing these Bayes "probability" values by using the logarithm of the Bayes expression, and removing a number of terms that do not affect the ordering of class probabilities. We refer the reader to [124] for details.

Although for classification, we do not need to compute $P(T = Q)$, some applications necessitate the exact computation of the posterior probability $P(C^T = i | T = Q)$. For example, in the case of supervised anomaly detection (or rare class detection), the exact posterior probability value $P(C^T = i | T = Q)$ is needed in order to fairly compare the probability value over different test instances, and rank them for their anomalous nature. In such cases, we would need to compute $P(T = Q)$. One way to achieve this is simply to take a sum over all the classes:

$$P(T = Q) = \sum_i P(T = Q | C^T = i) P(C^T = i).$$

This is based on the conditional independence of features for each class. Since the parameter values are estimated for each class separately, we may face the problem of data sparseness. An alternative way of computing it, which may alleviate the data sparseness problem, is to further make the assumption of (global) independence of terms, and compute it as:

$$P(T = Q) = \prod_{j \in Q} P(t_j \in T) \cdot \prod_{t_j \notin Q} (1 - P(t_j \in T))$$

where the term probabilities are based on global term distributions in *all* the classes.

A natural question arises, as to whether it is possible to design a Bayes classifier that does not use the naive assumption, and models the dependencies between the terms during the classification process. Methods that generalize the naive Bayes classifier by not using the independence assumption do not work well because of the higher computational costs and the inability to estimate the parameters accurately and robustly in the presence of limited data. The most interesting line of work in relaxing the independence assumption is provided in [128]. In this work, the tradeoffs in spectrum of allowing different levels of dependence among the terms have been explored. On the one extreme, an assumption of complete dependence results in a Bayesian network model that turns out to be computationally very expensive. On the other hand, it has been shown that allowing limited levels of dependence can provide good tradeoffs between accuracy and computational costs. We note that while the independence assumption is a practical approximation, it has been shown

in [35,47] that the approach does have some theoretical merit. Indeed, extensive experimental tests have tended to show that the naive classifier works quite well in practice.

A number of papers [24,74,86,91,124,129] have used the naive Bayes approach for classification in a number of different application domains. The classifier has also been extended to modeling temporally aware training data, in which the importance of a document may decay with time [130]. As in the case of other statistical classifiers, the naive Bayes classifier [129] can easily incorporate *domain-specific knowledge* into the classification process. The particular domain that the work in [129] addresses is that of filtering junk email. Thus, for such a problem, we often have a lot of additional domain knowledge that helps us determine whether a particular email message is junk or not. For example, some common characteristics of the email that would make an email to be more or less likely to be junk are as follows:

- The domain of the sender such as *.edu* or *.com* can make an email to be more or less likely to be junk.
- Phrases such as *"Free Money"* or over emphasized punctuation such as "!!!" can make an email more likely to be junk.
- Whether the recipient of the message was a particular user, or a mailing list.

The Bayes method provides a natural way to incorporate such additional information into the classification process, by creating new features for each of these characteristics. The standard Bayes technique is then used in conjunction with this augmented representation for classification. The Bayes technique has also been used in conjunction with the incorporation of other kinds of domain knowledge, such as the incorporation of hyperlink information into the classification process [25, 118].

The Bayes method is also suited to hierarchical classification, when the training data is arranged in a taxonomy of topics. For example, the Open Directory Project (ODP), *Yahoo!* Taxonomy, and a variety of news sites have vast collections of documents that are arranged into hierarchical groups. The hierarchical structure of the topics can be exploited to perform more effective classification [24, 86], because it has been observed that context-sensitive feature selection can provide more useful classification results. In hierarchical classification, a Bayes classifier is built at *each node*, which then provides us with the next branch to follow for classification purposes. Two such methods are proposed in [24, 86], in which node specific features are used for the classification process. Clearly, much fewer features are required at a particular node in the hierarchy, because the features that are picked are relevant to that branch. An example in [86] suggests that a branch of the taxonomy that is related to *Computer* may have no relationship with the word "cow." These node-specific features are referred to as *signatures* in [24]. Furthermore, it has been observed in [24] that in a given node, the most discriminative features for a given class may be different from their parent nodes. For example, the word "health" may be discriminative for the *Yahoo*! category *@Health*, but the word "baby" may be much more discriminative for the category *@Health@Nursing*. Thus, it is critical to have an appropriate feature selection process at each node of the classification tree. The methods in [24, 86] use different methods for this purpose.

- The work in [86] uses an information-theoretic approach [37] for feature selection, which takes into account the dependencies between the attributes [128]. The algorithm greedily eliminates the features one-by-one so as to least disrupt the conditional class distribution at that node.
- The node-specific features are referred to as *signatures* in [24]. These node-specific signatures are computed by calculating the ratio of intra-class variance to inter-class variance for the different words at each node of the tree. We note that this measure is the same as that optimized by the Fisher's discriminant, except that it is applied to the original set of words, rather than solved as a general optimization problem in which arbitrary directions in the data are picked.

A Bayesian classifier is constructed at each node in order to determine the appropriate branch. A small number of context-sensitive features provide one advantage of these methods, i.e., Bayesian classifiers work much more effectively with a much smaller number of features. Another major difference between the two methods is that the work in [86] uses the Bernoulli model, whereas that in [24] uses the multinomial model, which will be discussed in the next subsection. This approach in [86] is referred to as the *Pachinko Machine classifier* and that in [24] is known as *TAPER (Taxonomy and Path Enhanced Retrieval System)*.

Other noteworthy methods for hierarchical classification are proposed in [13, 109, 151]. The work [13] addresses two common problems associated with hierarchical text classification: (1) error propagation; (2) non-linear decision surfaces. The problem of error propagation occurs when the classification mistakes made at a parent node are propagated to its children nodes. This problem was solved in [13] by using cross validation to obtain a training data set for a child node that is more similar to the actual test data passed to the child node from its parent node than the training data set normally used for training a classifier at the child node. The problem of non-linear decision surfaces refers to the fact that the decision boundary of a category at a higher level is often non-linear (since its members are the union of the members of its children nodes). This problem is addressed by using the tentative class labels obtained at the children nodes as features for use at a parent node. These are general strategies that can be applied to any base classifier, and the experimental results in [13] show that both strategies are effective.

11.5.2 Multinomial Distribution

This class of techniques treats a document as a set of words with frequencies attached to each word. Thus, the set of words is allowed to have duplicate elements.

As in the previous case, we assume that the set of words in document is denoted by Q, drawn from the vocabulary set V. The set Q contains the distinct terms $\{t_{i_1} \ldots t_{i_m}\}$ with associated frequencies $F = \{F_{i_1} \ldots F_{i_m}\}$. We denote the terms and their frequencies by $[Q,F]$. The total number of terms in the document (or document length) is denoted by $L = \sum_{j=1}^{m} F(i_j)$. Then, our goal is to model the posterior probability that the document T belongs to class i, given that it contains the terms in Q *with the associated frequencies* F. The Bayes probability of class i can be modeled by using the following sampling process:

If we sampled L terms sequentially **from the term distribution of one of the randomly chosen classes** *(allowing repetitions) to create the term set T, and the final outcome for sampled set T is the set Q with the corresponding frequencies F, then what is the posterior probability that we had originally picked class i for sampling? The a-priori probability of picking class i is equal to its fractional presence in the collection.*

The aforementioned probability is denoted by $P(C^T = i|T = [Q,F])$. An assumption that is commonly used in these models is that the length of the document is independent of the class label. While it is easily possible to generalize the method, so that the document length is used as a prior, independence is usually assumed for simplicity. As in the previous case, we need to estimate two values in order to compute the Bayes posterior.

1. What is the prior probability that a set T is a sample from the term distribution of class i? This probability is denoted by $P(C^T = i)$.

2. If we sampled L terms *from the term distribution of class i* (with repetitions), then what is the probability that our sampled set T is the set Q with associated frequencies F? This probability is denoted by $P(T = [Q,F]|C^T = i)$.

Then, the Bayes rule can be applied to this case as follows:

$$P(C^T = i|T = [Q,F]) = \frac{P(C^T = i) \cdot P(T = [Q,F]|C^T = i)}{P(T = [Q,F])}$$

$$\propto P(C^T = i) \cdot P(T = [Q,F]|C^T = i). \tag{11.11}$$

As in the previous case, it is not necessary to compute the denominator, $P(T = [Q,F])$, for the purpose of deciding the class label for Q. The value of the probability $P(C^T = i)$ can be estimated as the fraction of documents belonging to class i. The computation of $P([Q,F]|C^T = i)$ is more complicated. When we consider the sequential order of the L different samples, the number of possible ways to sample the different terms so as to result in the outcome $[Q,F]$ is given by $\frac{L!}{\prod_{i=1}^{m} F_i!}$. The probability of *each* of these sequences is given by $\prod_{t_j \in Q} P(t_j \in T)^{F_j}$, by using the naive independence assumption. Therefore, we have:

$$P(T = [Q,F]|C^T = i) = \frac{L!}{\prod_{i=1}^{m} F_i!} \cdot \prod_{t_j \in Q} P(t_j \in T|C^T = i)^{F_j}. \tag{11.12}$$

We can substitute Equation 11.12 in Equation 11.11 to obtain the class with the highest Bayes posterior probability, where the class priors are computed as in the previous case, and the probabilities $P(t_j \in T|C^T = i)$ can also be easily estimated as previously with Laplacian smoothing [144]. Note that for the purpose of choosing the class with the highest posterior probability, we do not really have to compute $\frac{L!}{\prod_{i=1}^{m} F_i!}$, as it is a constant not depending on the class label (i.e., the same for all the classes). We also note that the probabilities of class absence are not present in the above equations because of the way in which the sampling is performed.

A number of different variations of the multinomial model have been proposed in [61, 82, 96, 109, 111, 117]. In the work [109], it is shown that a category hierarchy can be leveraged to improve the estimate of multinomial parameters in the naive Bayes classifier to significantly improve classification accuracy. The key idea is to apply shrinkage techniques to smooth the parameters for data-sparse child categories with their common parent nodes. As a result, the training data of related categories are essentially "shared" with each other in a weighted manner, which helps improve the robustness and accuracy of parameter estimation when there are insufficient training data for each individual child category. The work in [108] has performed an extensive comparison between the Bernoulli and the multinomial models on different corpora, and the following conclusions were presented:

- The multi-variate Bernoulli model can sometimes perform better than the multinomial model at small vocabulary sizes.
- The multinomial model outperforms the multi-variate Bernoulli model for large vocabulary sizes, and almost always beats the multi-variate Bernoulli when vocabulary size is chosen optimally for both. On the average a 27% reduction in error was reported in [108].

The afore-mentioned results seem to suggest that the two models may have different strengths, and may therefore be useful in different scenarios.

11.5.3 Mixture Modeling for Text Classification

We note that the afore-mentioned Bayes methods simply assume that each component of the mixture corresponds to the documents belonging to a class. A more general interpretation is one in which the components of the mixture are created by a clustering process, and the class membership probabilities are modeled in terms of this mixture. Mixture modeling is typically used for unsupervised (probabilistic) clustering or topic modeling, though the use of clustering can also help

in enhancing the effectiveness of probabilistic classifiers [98, 117]. These methods are particularly useful in cases where the amount of training data is limited. In particular, clustering can help in the following ways:

- The Bayes method implicitly estimates the word probabilities $P(t_i \in T|C^T = i)$ of a large number of terms in terms of their fractional presence in the corresponding component. This is clearly noisy. By treating the clusters as separate entities from the classes, we now only need to relate (a much smaller number of) cluster membership probabilities to class probabilities. This reduces the number of parameters and greatly improves classification accuracy [98].

- The use of clustering can help in incorporating unlabeled documents into the training data for classification. The premise is that unlabeled data is much more copiously available than labeled data, and when labeled data is sparse, it should be used in order to assist the classification process. While such unlabeled documents do not contain class-specific information, they do contain a lot of information about the clustering behavior of the underlying data. This can be very useful for more robust modeling [117], when the amount of training data is low. This general approach is also referred to as *co-training* [10, 17, 45].

The common characteristic of both methods [98, 117] is that they both use a form of supervised clustering for the classification process. While the goal is quite similar (limited training data), the approach used for this purpose is quite different. We will discuss both of these methods in this section.

In the method discussed in [98], the document corpus is modeled with the use of supervised word clusters. In this case, the k mixture components are clusters that are correlated to, but are distinct from the k groups of documents belonging to the different classes. The main difference from the Bayes method is that the term probabilities are computed indirectly by using clustering as an intermediate step. For a sampled document T, we denote its class label by $C^T \in \{1 \ldots k\}$, and its mixture component by $M^T \in \{1 \ldots k\}$. The k different mixture components are essentially word-clusters whose frequencies are generated by using the frequencies of the terms in the k different classes. This ensures that the word clusters for the mixture components are correlated to the classes, but they are not assumed to be drawn from the same distribution. As in the previous case, let us assume that the A document contains the set of words Q. Then, we would like to estimate the probability $P(T = Q|C^T = i)$ for each class i. An interesting variation of the work in [98] from the Bayes approach is that it does not attempt to determine the posterior probability $P(C^T = i|T = Q)$. Rather, it simply reports the class with the highest likelihood $P(T = Q|C^T = i)$. This is essentially equivalent to assuming, in the Bayes approach, that the prior distribution of each class is the same.

The other difference of the approach is in terms of how the value of $P(T = Q|C^T = i)$ is computed. As before, we need to estimate the value of $P(t_j \in T|C^T = i)$, according to the naive Bayes rule. However, unlike the standard Bayes classifier, this is done very indirectly with the use of mixture modeling. Since the mixture components do not directly correspond to the class, this term can only be estimated by summing up the expected value over all the mixture components:

$$P(t_j \in T|C^T = i) = \sum_{s=1}^{k} P(t_j \in T|M^T = s) \cdot P(M^T = s|C^T = i). \tag{11.13}$$

The value of $P(t_j \in T|M^T = s)$ is easy to estimate by using the fractional presence of term t_j in the sth mixture component. The main unknown here is the set of model parameters $P(M^T = s|C^T = i)$. Since a total of k classes and k mixture-components are used, this requires the estimation of only k^2 model parameters, which is typically quite modest for a small number of classes. An EM-approach has been used in [98] in order to estimate this small number of model parameters in a robust way. It is important to understand that the work in [98] is an interesting combination of

supervised topic modeling (dimensionality reduction) and Bayes classification after reducing the effective dimensionality of the feature space to a much smaller value by clustering. The scheme works well because of the use of supervision in the topic modeling process, which ensures that the use of an intermediate clustering approach does not lose information for classification. We also note that in this model, the number of mixtures can be made to vary from the number of classes. While the work in [98] does not explore this direction, there is really no reason to assume that the number of mixture components is the same as the number of classes. Such an assumption can be particularly useful for data sets in which the classes may not be contiguous in the feature space, and a natural clustering may contain far more components than the number of classes.

Next, we will discuss the second method [117], which uses unlabeled data. The approach in [117] uses the unlabeled data in order to improve the training model. Why should unlabeled data help in classification at all? In order to understand this point, recall that the Bayes classification process effectively uses k mixture components, which are assumed to be the k different classes. If we had an infinite amount of training data, it would be possible to create the mixture components, but it would not be possible to assign labels to these components. However, the most data-intensive part of modeling the mixture is that of determining the shape of the mixture components. The actual assignment of mixture components to class labels can be achieved with a relatively small number of class labels. It has been shown in [30] that the accuracy of assigning components to classes increases exponentially with the number of labeled samples available. Therefore, the work in [117] designs an EM-approach [44] to simultaneously determine the relevant mixture model and its class assignment.

It turns out that the EM-approach, as applied to this problem, is quite simple to implement. It has been shown in [117] that the EM-approach is equivalent to the following iterative methodology. First, a naive Bayes classifier is constructed by estimating the model parameters from the labeled documents only. This is used in order to assign probabilistically weighted class labels to the unlabeled documents. Then, the Bayes classifier is reconstructed, except that we also use the newly labeled documents in the estimation of the underlying model parameters. We again use this classifier to reclassify the (originally unlabeled) documents. The process is continually repeated till convergence is achieved. This process is in many ways similar to pseudo-relevance feedback in information retrieval where a portion of top-ranked documents returned from an initial round of retrieval would be assumed to be relevant document examples (may be weighted), which can then be used to learn an improved query representation for improving ranking of documents in the next round of retrieval [150].

The ability to significantly improve the quality of text classification with a small amount of labeled data, and the use of clustering on a large amount of unlabeled data, has been a recurring theme in the text mining literature. For example, the method in [141] performs purely unsupervised clustering (with no knowledge of class labels), and then as a final step assigns all documents in the cluster to the *dominant* class label of that cluster (as an evaluation step for the unsupervised clustering process in terms of its ability in matching clusters to known topics).[5] It has been shown that this approach is able to achieve a comparable accuracy of matching clusters to topics as a supervised naive Bayes classifier trained over a small data set of about 1000 documents. Similar results were obtained in [55] where the quality of the unsupervised clustering process was shown to be comparable to an SVM classifier that was trained over a small data set.

[5] In a supervised application, the last step would require only a small number of class labels in the cluster to be known to determine the dominant label very accurately.

11.6 Linear Classifiers

Linear Classifiers are those for which the output of the linear predictor is defined to be $p = \overline{A} \cdot \overline{X} + b$, where $\overline{X} = (x_1 \ldots x_n)$ is the normalized document word frequency vector, $\overline{A} = (a_1 \ldots a_n)$ is a vector of linear coefficients with the same dimensionality as the feature space, and b is a scalar. A natural interpretation of the predictor $p = \overline{A} \cdot \overline{X} + b$ in the *discrete scenario* (categorical class labels) would be as a *separating hyperplane* between the different classes. *Support Vector Machines* [36, 145] are a form of classifiers that attempt to determine "good" linear separators between the different classes. One characteristic of linear classifiers is that they are closely related to many *feature transformation methods* (such as the Fisher discriminant), which attempt to use these directions in order to transform the feature space, and then use other classifiers on this transformed feature space [59, 65, 113]. Thus, linear classifiers are intimately related to linear feature transformation methods as well.

Regression modeling (such as the least squares method) is a more direct and traditional statistical method for text classification. However, it is generally used in cases where the target variable to be learned is numerical rather than categorical, though logistic regression [116] is commonly used for text classification (indeed, classification in general). A number of methods have been proposed in the literature for adapting such methods to the case of text data classification [155]. A comparison of different linear regression techniques for classification, including SVM, may be found in [159].

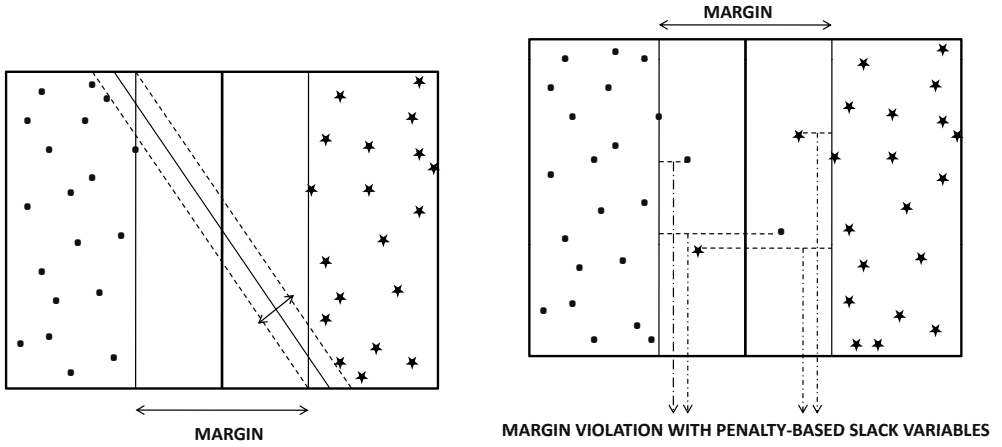
Finally, simple neural networks are also a form of linear classifiers, since the function computed by a set of neurons is essentially linear. The simplest form of neural network, known as the *perceptron* (or single layer network) is essentially designed for linear separation, and works well for text. However, by using multiple layers of neurons, it is also possible to generalize the approach for non-linear separation. In this section, we will discuss the different linear methods for text classification.

11.6.1 SVM Classifiers

Support vector machines are generally defined for binary classification problems. Therefore, the class variable y_i for the ith training instance $\overline{X_i}$ is assumed to be drawn from $\{-1, +1\}$. One advantage of the SVM method is that since it attempts to determine the optimum direction of discrimination in the feature space by examining the appropriate combination of features, it is quite robust to high dimensionality. It has been noted in [74] that text data is ideally suited for SVM classification because of the sparse high-dimensional nature of text, in which few features are irrelevant, but they tend to be correlated with one another and generally organized into linearly separable categories. We note that it is not necessary to use a linear function for the SVM classifier. Rather, with the kernel trick [7], SVM can construct a non-linear *decision surface* in the original feature space by mapping the data instances non-linearly to an inner product space where the classes can be separated linearly with a hyperplane.

The most important criterion, which is commonly used for SVM classification, is that of the *maximum margin hyperplane*. In order to understand this point, consider the case of linearly separable data illustrated in Figure 11.1(a). Two possible separating hyperplanes, with their corresponding *support vectors* and *margins* have been illustrated in the figure. It is evident that one of the separating hyperplanes has a much larger margin than the other, and is therefore more desirable because of its greater generality for unseen test examples. Therefore, one of the important criteria for support vector machines is to achieve maximum margin separation of the hyperplanes.

In general, it is assumed for d dimensional data that the separating hyperplane is of the form $\overline{W} \cdot \overline{X} + b = 0$. Here $\overline{W}$ is a d-dimensional vector representing the coefficients of the hyperplane of separation, and b is a constant. Without loss of generality, it may be assumed (because of appropriate coefficient scaling) that the two symmetric support vectors have the form $\overline{W} \cdot \overline{X} + b = 1$ and $\overline{W} \cdot$



(a) Hard separation (b) Soft separation

FIGURE 11.1: Hard and soft support vector machines.

$\overline{X} + b = -1$. The coefficients $\overline{W}$ and b need to be learned from the training data $\mathcal{D}$ in order to maximize the margin of separation between these two parallel hyperplanes. It can be shown from elementary linear algebra that the distance between these two hyperplanes is $2/||\overline{W}||$. Maximizing this objective function is equivalent to minimizing $||\overline{W}||^2/2$. The constraints are defined by the fact that the training data points for each class are on one side of the support vector. Therefore, these constraints are as follows:

$$\overline{W} \cdot \overline{X_i} + b \geq +1 \quad \forall i : y_i = +1 \tag{11.14}$$

$$\overline{W} \cdot \overline{X_i} + b \leq -1 \quad \forall i : y_i = -1. \tag{11.15}$$

This is a constrained convex quadratic optimization problem, which can be solved using Lagrangian methods. In practice, an off-the-shelf optimization solver may be used to achieve the same goal.

In practice, the data may not be linearly separable. In such cases, soft-margin methods may be used. A slack $\xi_i \geq 0$ is introduced for training instance, and a training instance is allowed to violate the support vector constraint, for a penalty, which is dependent on the slack. This situation is illustrated in Figure 1.2(b). Therefore, the new set of constraints are now as follows:

$$\overline{W} \cdot \overline{X} + b \geq +1 - \xi_i \quad \forall i : y_i = +1 \tag{11.16}$$

$$\overline{W} \cdot \overline{X} + b \leq -1 + \xi_i \quad \forall i : y_i = -1 \tag{11.17}$$

$$\xi_i \geq 0. \tag{11.18}$$

Note that additional non-negativity constraints also need to be imposed in the slack variables. The objective function is now $||\overline{W}||^2/2 + C \cdot \sum_{i=1}^{n} \xi_i$. The constant C regulates the importance of the margin and the slack requirements. In other words, small values of C make the approach closer to soft-margin SVM, whereas large values of C make the approach more of the hard-margin SVM. It is also possible to solve this problem using off-the-shelf optimization solvers.

It is also possible to use transformations on the feature variables in order to design non-linear SVM methods. In practice, non-linear SVM methods are learned using kernel methods. The key idea here is that SVM formulations can be solved using only pairwise dot products (similarity values) between objects. In other words, the optimal decision about the class label of a test instance, from the solution to the quadratic optimization problem in this section, can be expressed in terms of the following:

1. Pairwise dot products of different training instances.

2. Pairwise dot product of the test instance and different training instances.

The reader is advised to refer to [143] for the specific details of the solution to the optimization formulation. The dot product between a pair of instances can be viewed as notion of similarity among them. Therefore, the aforementioned observations imply that it is possible to perform SVM classification, with pairwise similarity information between training data pairs and training-test data pairs. The actual feature values are not required.

This opens the door for using transformations, which are represented by their similarity values. These similarities can be viewed as kernel functions $K(\overline{X}, \overline{Y})$, which measure similarities between the points $\overline{X}$ and $\overline{Y}$. Conceptually, the kernel function may be viewed as dot product between the pair of points in a newly transformed space (denoted by mapping function $\Phi(\cdot)$). However, this transformation does not need to be explicitly computed, as long as the kernel function (dot product) $K(\overline{X}, \overline{Y})$ is already available:

$$K(\overline{X}, \overline{Y}) = \Phi(\overline{X}) \cdot \Phi(\overline{Y}). \tag{11.19}$$

Therefore, all computations can be performed in the original space using the dot products implied by the kernel function. Some interesting examples of kernel functions include the Gaussian radial basis function, polynomial kernel, and hyperbolic tangent, which are listed below in the same order.

$$K(\overline{X_i}, \overline{X_j}) = e^{-||\overline{X_i} - \overline{X_j}||^2/2\sigma^2}. \tag{11.20}$$

$$K(\overline{X_i}, \overline{X_j}) = (\overline{X_i} \cdot \overline{X_j} + 1)^h. \tag{11.21}$$

$$K(\overline{X_i}, \overline{X_j}) = \tanh(\kappa \overline{X_i} \cdot \overline{X_j} - \delta). \tag{11.22}$$

These different functions result in different kinds of nonlinear decision boundaries in the original space, but they correspond to a linear separator in the transformed space. The performance of a classifier can be sensitive to the choice of the kernel used for the transformation. One advantage of kernel methods is that they can also be extended to arbitrary data types, as long as appropriate pairwise similarities can be defined.

The first set of SVM classifiers, as adapted to the text domain, were proposed in [74–76]. A deeper theoretical study of the SVM method has been provided in [77]. In particular, it has been shown why the SVM classifier is expected to work well under a wide variety of circumstances. This has also been demonstrated experimentally in a few different scenarios. For example, the work in [49] applied the method to email data for classifying it as spam or non-spam data. It was shown that the SVM method provides much more robust performance as compared to many other techniques such as boosting decision trees, the rule based RIPPER method, and the Rocchio method. The SVM method is flexible and can easily be combined with interactive user-feedback methods [123].

The major downside of SVM methods is that they are slow. Our discussion in this section shows that the problem of finding the best separator is a Quadratic Programming problem. The number of constraints is proportional to the number of data points. This translates directly into the number of Lagrangian relaxation variables in the optimization problem. This can sometimes be slow, especially for high dimensional domains such as text. It has been shown [51] that by breaking a large Quadratic Programming problem (QP problem) into a set of smaller problems, an efficient solution can be derived for the task.

A number of other methods have been proposed to scale up the SVM method for the special structure of text. A key characteristic of text is that the data is high dimensional, but an individual document contains very few features from the full lexicon. In other words, the number of *non-zero* features is small. A number of different approaches have been proposed to address these issues. The first approach [78], referred to as *SVMLight*, shares a number of similarities with [51]. The first approach also breaks down the quadratic programming problem into smaller subproblems. This achieved by using a working set of Lagrangian variables, which are optimized, while keeping the

other variables fixed. The choice of variables to select is based on the gradient of the objective function with respect to these variables. This approach does not, however, fully leverage the sparsity of text. A second approach, known as *SVMPerf* [79] also leverages the sparsity of text data. This approach reduces the number of slack variables in the quadratic programming formulation, while increasing the constraints. A cutting plane algorithm is used to solve the optimization problem efficiently. The approach is shown to require $O(n \cdot s)$ time, where n is the number of training examples, and s is the average number of non-zero features per training document. The reader is referred to Chapter 10 on big-data classification, for a detailed discussion of this approach. The SVM approach has also been used successfully [52] in the context of a hierarchical organization of the classes, as often occurs in Web data. In this approach, a different classifier is built at different positions of the hierarchy.

SVM methods are very popular and tend to have high accuracy in the text domain. Even the linear SVM works rather well for text in general, though the specific accuracy is obviously data-set dependent. An introduction to SVM methods may be found in [36, 40, 62, 132, 133, 145]. Kernel methods for support vector machines are discussed in [132].

11.6.2 Regression-Based Classifiers

Regression modeling is a method that is commonly used in order to learn the relationships between real-valued attributes. Typically, these methods are designed for real valued attributes, as opposed to binary attributes. This, however, is not an impediment to its use in classification, because the binary value of a class may be treated as a rudimentary special case of a real value, and some regression methods such as logistic regression can also naturally model discrete response variables.

An early application of regression to text classification is the Linear Least Squares Fit (LLSF) method [155], which works as follows. Suppose the predicted class label is $p_i = \overline{A} \cdot \overline{X_i} + b$, and y_i is known to be the true class label, then our aim is to learn the values of A and b, such that the *Linear Least Squares Fit (LLSF)* $\sum_{i=1}^{n} (p_i - y_i)^2$ is minimized. In practice, the value of b is set to 0 for the learning process. Let P be $1 \times n$ vector of binary values indicating the binary class to which the corresponding class belongs. Thus, if X is the $n \times d$ term-matrix, then we wish to determine the $1 \times d$ vector of regression coefficients A for which $||A \cdot X^T - P||$ is minimized, where $||\cdot||$ represents the Froebinus norm. The problem can be easily generalized from the binary class scenario to the multi-class scenario with k classes, by using P as a $k \times n$ matrix of binary values. In this matrix, exactly one value in each column is 1, and the corresponding row identifier represents the class to which that instance belongs. Similarly, the set A is a $k \times d$ vector in the multi-class scenario. The LLSF method has been compared to a variety of other methods [153, 155, 159], and has been shown to be very robust in practice.

A more natural way of modeling the classification problem with regression is the logistic regression classifier [116], which differs from the LLSF method in that the objective function to be optimized is the likelihood function. Specifically, instead of using $p_i = \overline{A} \cdot \overline{X_i} + b$ directly to fit the true label y_i, we assume that the probability of observing label y_i is:

$$p(C = y_i | X_i) = \frac{exp(\overline{A} \cdot \overline{X_i} + b)}{1 + exp(\overline{A} \cdot \overline{X_i} + b).}$$

This gives us a conditional generative model for y_i given X_i. Putting it in another way, we assume that the logit transformation of $p(C = y_i | X_i)$ can be modeled by the linear combination of features of the instance X_i, i.e.,

$$\log \frac{p(C = y_i | X_i)}{1 - p(C = y_i | X_i)} = \overline{A} \cdot \overline{X_i} + b.$$

Thus logistic regression is also a linear classifier as the decision boundary is determined by a linear function of the features. In the case of binary classification, $p(C = y_i | X_i)$ can be used to determine

the class label (e.g., using a threshold of 0.5). In the case of multi-class classification, we have $p(C = y_i|X_i) \propto exp(\overline{A} \cdot \overline{X_i} + b)$, and the class label with the highest value according to $p(C = y_i|X_i)$ would be assigned to X_i. Given a set of training data points $\{(X_1, y_i), ...(X_n, y_n)\}$, the logistic regression classifier can be trained by choosing parameters $\overline{A}$ to maximize the conditional likelihood $\prod_{i=1}^{n} p(y_i|X_i)$.

In some cases, the domain knowledge may be of the form where some sets of words are more important than others for a classification problem. For example, in a classification application, we may know that certain domain-words (*Knowledge Words (KW)*) may be more important to classification of a particular target category than other words. In such cases, it has been shown [43] that it may be possible to encode such domain knowledge into the logistic regression model in the form of prior on the model parameters and use Bayesian estimation of model parameters.

It is clear that the regression classifiers are extremely similar to the SVM model for classification. Indeed, since LLSF, Logistic Regression, and SVM are all linear classifiers, they are thus identical at a conceptual level; the main difference among them lies in the details of the optimization formulation and implementation. As in the case of SVM classifiers, training a regression classifier also requires an expensive optimization process. For example, fitting LLSF requires expensive matrix computations in the form of a singular value decomposition process.

11.6.3 Neural Network Classifiers

Neural networks attempt to simulate biological systems, corresponding to the human brain. In the human brain, neurons are connected to one another via points, which are referred to as *synapses*. In biological systems, learning is performed by changing the strength of the synaptic connections, in response to impulses.

This biological analogy is retained in an artificial neural network. The basic computation unit in an artificial neural network is a *neuron* or *unit*. These units can be arranged in different kinds of architectures by connections between them. The most basic architecture of the neural network is a perceptron, which contains a set of input nodes and an output node. The output unit receives a set of inputs from the input units. There are d different input units, which is exactly equal to the dimensionality of the underlying data. The data is assumed to be numerical. Categorical data may need to be transformed to binary representations, and therefore the number of inputs may be larger. The output node is associated with a set of weights $\overline{W}$, which are used in order to compute a function $f(\cdot)$ of its inputs. Each component of the weight vector is associated with a connection from the input unit to the output unit. The weights can be viewed as the analogue of the synaptic strengths in biological systems. In the case of a perceptron architecture, the input nodes do not perform any computations. They simply transmit the input attribute forward. Computations are performed only at the output nodes in the basic perceptron architecture. The output node uses its weight vector along with the input attribute values in order to compute a function of the inputs. A typical function, which is computed at the output nodes, is the signed linear function:

$$z_i = \text{sign}\{\overline{W} \cdot \overline{X_i} + b\}. \tag{11.23}$$

The output is a predicted value of the binary class variable, which is assumed to be drawn from $\{-1, +1\}$. The notation b denotes the bias. Thus, for a vector $\overline{X_i}$ drawn from a dimensionality of d, the weight vector $\overline{W}$ should also contain d elements. Now consider a binary classification problem, in which all labels are drawn from $\{+1, -1\}$. We assume that the class label of $\overline{X_i}$ is denoted by y_i. In that case, the sign of the predicted function z_i yields the class label. Thus, the goal of the approach is to *learn* the set of weights $\overline{W}$ with the use of the training data, so as to minimize the least squares error $(y_i - z_i)^2$. The idea is that we start off with random weights and gradually update them, when a mistake is made by applying the current function on the training example. The magnitude of the update is regulated by a learning rate λ. This update is similar to the updates in gradient descent,

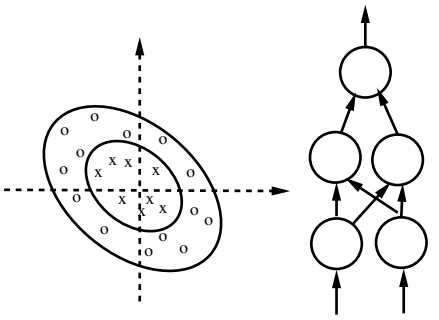


FIGURE 11.2: Multi-layered neural networks for nonlinear separation.

which are made for least-squares optimization. In the case of neural networks, the update function is as follows.

$$\overline{W}^{t+1} = \overline{W}^t + \lambda(y_i - z_i)\overline{X_i} \tag{11.24}$$

Here, $\overline{W}^t$ is the value of the weight vector in the tth iteration. It is not difficult to show that the incremental update vector is related to the negative gradient of $(y_i - z_i)^2$ with respect to $\overline{W}$. It is also easy to see that updates are made to the weights only when mistakes are made in classification. When the outputs are correct, the incremental change to the weights is zero. The overall perceptron algorithm is illustrated below.

Perceptron Algorithm
Inputs: Learning Rate: λ
Training Data $(\overline{X_i}, y_i)$ $\forall i \in \{1 \ldots n\}$
Initialize weight vectors in $\overline{W}$ and b to small random numbers
repeat
Apply each training data to the neural network to check if the
sign of $\overline{W} \cdot \overline{X_i} + b$ matches y_i;
if sign of $\overline{W} \cdot \overline{X_i} + b$ does **not** match y_i, then
update weights $\overline{W}$ based on learning rate λ
until weights in $\overline{W}$ converge

The similarity to support vector machines is quite striking, in the sense that a linear function is also learned in this case, and the sign of the linear function predicts the class label. In fact, the perceptron model and support vector machines are closely related, in that both are linear function approximators. In the case of support vector machines, this is achieved with the use of maximum margin optimization. In the case of neural networks, this is achieved with the use of an incremental learning algorithm, which is approximately equivalent to least squares error optimization of the prediction.

The constant λ regulates the learning rate. The choice of learning rate is sometimes important, because learning rates that are too small will result in very slow training. On the other hand, if the learning rates are too fast, this will result in oscillation between suboptimal solutions. In practice, the learning rates are fast initially, and then allowed to gradually slow down over time. The idea here is that initially large steps are likely to be helpful, but are then reduced in size to prevent oscillation between suboptimal solutions. For example, after t iterations, the learning rate may be chosen to be proportional to $1/t$.

The aforementioned discussion was based on the simple perceptron architecture, which can model only linear relationships. A natural question arises as to how a neural network may be used, if all the classes may not be neatly separated from one another with a linear separator. For example,

in Figure 11.2, we have illustrated an example in which the classes may not be separated with the use of a single linear separator. The use of *multiple layers of neurons* can be used in order to induce such non-linear classification boundaries. The effect of such multiple layers is to induce multiple piece-wise linear boundaries, which can be used to approximate enclosed regions belonging to a particular class. In such a network, the outputs of the neurons in the earlier layers feed into the neurons in the later layers. The training process of such networks is more complex, as the errors need to be back-propagated over different layers. Some examples of such classifiers include those discussed in [87, 126, 146, 153]. However, the general observation [135, 149] for text has been that linear classifiers generally provide comparable results to non-linear data, and the improvements of non-linear classification methods are relatively small. This suggests that the additional complexity of building more involved non-linear models does not pay for itself in terms of significantly better classification.

In practice, the neural network is arranged in three layers, referred to as the *input layer*, *hidden layer*, and the *output layer*. The input layer only transmits the inputs forward, and therefore, there are really only two layers to the neural network, which can perform computations. Within the hidden layer, there can be any number of layers of neurons. In such cases, there can be an arbitrary number of layers in the neural network. In practice, there is only one hidden layer, which leads to a two-layer network. The perceptron can be viewed as a very special kind of neural network, which contains only a single layer of neurons (corresponding to the output node). Multilayer neural networks allow the approximation of nonlinear functions, and complex decision boundaries, by an appropriate choice of the network topology, and non-linear functions at the nodes. In these cases, a logistic or sigmoid function, known as a *squashing function*, is also applied to the inputs of neurons in order to model non-linear characteristics. It is possible to use different non-linear functions at different nodes. Such general architectures are very powerful in approximating arbitrary functions in a neural network, given enough training data and training time. This is the reason that neural networks are sometimes referred to as *universal function approximators*.

In the case of single layer perceptron algorithms, the training process is easy to perform by using a gradient descent approach. The major challenge in training multilayer networks is that it is no longer known for intermediate (hidden layer) nodes what their "expected" output should be. This is only known for the final output node. Therefore, some kind of "error feedback" is required, in order to determine the changes in the weights at the intermediate nodes. The training process proceeds in two phases, one of which is in the forward direction, and the other is in the backward direction.

1. *Forward Phase:* In the forward phase, the activation function is repeatedly applied to propagate the inputs from the neural network in the forward direction. Since the final output is supposed to match the class label, the final output at the output layer provides an error value, depending on the training label value. This error is then used to update the weights of the output layer, and propagate the weight updates backwards in the next phase.

2. *Backpropagation Phase:* In the backward phase, the errors are propagated backwards through the neural network layers. This leads to the updating of the weights in the neurons of the different layers. The gradients at the previous layers are learned as a function of the errors and weights in the layer ahead of it. The learning rate λ plays an important role in regulating the rate of learning.

In practice, any arbitrary function can be approximated well by a neural network. The price of this generality is that neural networks are often quite slow in practice. They are also sensitive to noise, and can sometimes overfit the training data.

The previous discussion assumed only binary labels. It is possible to create a k-label neural network, by either using a multiclass "one-versus-all" meta-algorithm, or by creating a neural network architecture in which the number of output nodes is equal to the number of class labels. Each output represents prediction to a particular label value. A number of implementations of neural network methods have been studied in [41, 102, 115, 135, 149], and many of these implementations

are designed in the context of text data. It should be pointed out that both neural networks and SVM classifiers use a linear model that is quite similar. The main difference between the two is in how the optimal linear hyperplane is determined. Rather than using a direct optimization methodology, neural networks use a *mistake-driven* approach to data classification [41]. Neural networks are described in detail in [15, 66].

11.6.4 Some Observations about Linear Classifiers

While the different linear classifiers have been developed independently from one another in the research literature, they are surprisingly similar at a basic conceptual level. Interestingly, these different lines of work have also resulted in a number of similar conclusions in terms of the effectiveness of the different classifiers. We note that the main difference between the different classifiers is in terms of the details of the objective function that is optimized, and the iterative approach used in order to determine the optimum direction of separation. For example, the SVM method uses a Quadratic Programming (QP) formulation, whereas the LLSF method uses a closed-form least-squares formulation. On the other hand, the perceptron method does not try to formulate a closed-form objective function, but works with a softer iterative hill climbing approach. This technique is essentially inherited from the iterative learning approach used by neural network algorithms. However, its goal remains quite similar to the other two methods. Thus, the differences between these methods are really at a detailed level, rather than a conceptual level, in spite of their very different research origins.

Another general observation about these methods is that all of them can be implemented with non-linear versions of their classifiers. For example, it is possible to create non-linear decision surfaces with the SVM classifier, just as it is possible to create non-linear separation boundaries by using layered neurons in a neural network [153]. However, the general consensus has been that the linear versions of these methods work very well, and the additional complexity of non-linear classification does not tend to pay for itself, except for some special data sets. The reason for this is perhaps because text is a high dimensional domain with highly correlated features and small non-negative values on sparse features. For example, it is hard to easily create class structures such as that indicated in Figure 11.2 for a sparse domain such as text containing only small non-negative values on the features. On the other hand, the high dimensional nature of correlated text dimensions is especially suited to classifiers that can exploit the redundancies and relationships between the different features in separating out the different classes. Common text applications have generally resulted in class structures that are linearly separable over this high dimensional domain of data. This is one of the reasons that linear classifiers have shown an unprecedented success in text classification.

11.7 Proximity-Based Classifiers

Proximity-based classifiers essentially use distance-based measures in order to perform the classification. The main thesis is that documents which belong to the same class are likely to be close to one another based on similarity measures such as the dot product or the cosine metric [131]. In order to perform the classification for a given test instance, two possible methods can be used:

- We determine the k-nearest neighbors in the training data to the test instance. The majority (or most abundant) class from these k neighbors are reported as the class label. Some examples of such methods are discussed in [31, 63, 155]. The choice of k typically ranges between 20 and

40 in most of the afore-mentioned work, depending upon the size of the underlying corpus. In practice, it is often set empirically using cross validation.

- We perform training data aggregation during pre-processing, in which clusters or groups of documents belonging to the same class are created. A representative meta-document is created from each group. The same k-nearest neighbor approach is applied as discussed above, except that it is applied to this new set of meta-documents (or *generalized instances* [88]) rather than to the original documents in the collection. A pre-processing phase of summarization is useful in improving the efficiency of the classifier, because it significantly reduces the number of distance computations. In some cases, it may also boost the accuracy of the technique, especially when the data set contains a large number of outliers. Some examples of such methods are discussed in [64, 88, 125].

A method for performing nearest neighbor classification in text data is the *WHIRL* method discussed in [31]. The *WHIRL* method is essentially a method for performing soft similarity joins on the basis of text attributes. By *soft* similarity joins, we refer to the fact that the two records may not be exactly the same on the joined attribute, but may be approximately similar based on a pre-defined notion of similarity. It has been observed in [31] that any method for performing a similarity-join can be adapted as a nearest neighbor classifier, by using the relevant text documents as the joined attributes.

One observation in [155] about nearest neighbor classifiers was that feature selection and document representation play an important part in the effectiveness of the classification process. This is because most terms in large corpora may not be related to the category of interest. Therefore, a number of techniques were proposed in [155] in order to learn the associations between the words and the categories. These are then used to create a feature representation of the document, so that the nearest neighbor classifier is more sensitive to the classes in the document collection. A similar observation has been made in [63], in which it has been shown that the addition of weights to the terms (based on their class-sensitivity) significantly improves the underlying classifier performance. The nearest neighbor classifier has also been extended to the temporally-aware scenario [130], in which the timeliness of a training document plays a role in the model construction process. In order to incorporate such factors, a temporal weighting function has been introduced in [130], which allows the importance of a document to gracefully decay with time.

For the case of classifiers that use grouping techniques, the most basic among such methods is that proposed by Rocchio in [125]. In this method, a *single* representative meta-document is constructed from each of the representative classes. For a given class, the weight of the term t_k is the normalized frequency of the term t_k in documents belonging to that class, minus the normalized frequency of the term in documents which do not belong to that class. Specifically, let f_p^k be the expected weight of term t_k in a randomly picked document belonging to the positive class, and f_n^k be the expected weight of term t_k in a randomly picked document belonging to the negative class. Then, for weighting parameters α_p and α_n, the weight $f_{rocchio}^k$ is defined as follows:

$$f_{rocchio}^k = \alpha_p \cdot f_p^k - \alpha_n \cdot f_n^k \tag{11.25}$$

The weighting parameters α_p and α_n are picked so that the positive class has much greater weight as compared to the negative class. For the relevant class, we now have a vector representation of the terms $(f_{rocchio}^1, f_{rocchio}^2 \ldots f_{rocchio}^n)$. This approach is applied separately to each of the classes, in order to create a separate meta-document for each class. For a given test document, the closest meta-document to the test document can be determined by using a vector-based dot product or other similarity metric. The corresponding class is then reported as the relevant label. The main distinguishing characteristic of the Rocchio method is that it creates a single profile of the entire class. This class of methods is also referred to as the *Rocchio framework*. The main disadvantage of this method is that if a single class occurs in multiple disjoint clusters that are not very well

connected in the data, then the centroid of these examples may not represent the class behavior very well. This is likely to be a source of inaccuracy for the classifier. The main advantage of this method is its extreme simplicity and efficiency; the training phase is linear in the corpus size, and the number of computations in the testing phase are linear to the number of classes, since all the documents have already been aggregated into a small number of classes. An analysis of the Rocchio algorithm, along with a number of different variations may be found in [74].

In order to handle the shortcomings of the Rocchio method, a number of classifiers have also been proposed [2, 19, 64, 88], which explicitly perform the clustering of each of the classes in the document collection. These clusters are used in order to generate class-specific profiles. These profiles are also referred to as *generalized instances* in [88]. For a given test instance, the label of the closest generalized instance is reported by the algorithm. The method in [19] is also a centroid-based classifier, but is specifically designed for the case of text documents. The work in [64] shows that there are some advantages in designing schemes in which the similarity computations take account of the dependencies between the terms of the different classes.

We note that the nearest neighbor classifier can be used in order to generate a ranked list of categories for each document. In cases where a document is related to multiple categories, these can be reported for the document, as long as a thresholding method is available. The work in [157] studies a number of thresholding strategies for the k-nearest neighbor classifier. It has also been suggested in [157] that these thresholding strategies can be used to understand the thresholding strategies of other classifiers that use ranking classifiers.

11.8 Classification of Linked and Web Data

In recent years, the proliferation of the Web and social network technologies has led to a tremendous amount of document data, which are expressed in the form of linked networks. The simplest example of this is the Web, in which the documents are linked to one another with the use of hyperlinks. Social networks can also be considered a noisy example of such data, because the comments and text profiles of different users are connected to one another through a variety of links. Linkage information is quite relevant to the classification process, because documents of similar subjects are often linked together. This observation has been used widely in the *collective classification literature* [14], in which a subset of network nodes are labeled, and the remaining nodes are classified on the basis of the linkages among the nodes.

In general, a content-based network may be denoted by $G = (N, A, C)$, where N is the set of nodes, A is the set of edges between the nodes, and C is a set of text documents. Each node in N corresponds to a text document in C, and it is possible for a document to be empty, when the corresponding node does not contain any content. A subset of the nodes in N are labeled. This corresponds to the training data. The classification problem in this scenario is to determine the labels of the remaining nodes with the use of the training data. It is clear that both the content and structure can play a useful and complementary role in the classification process.

An interesting method for combining linkage and content information for classification was discussed in [25]. In this paper, a hypertext categorization method was proposed, which uses the content and labels of neighboring Web pages for the classification process. When the labels of all the nearest neighbors are available, a Bayesian method can be adapted easily for classification purposes. Just as the presence of a word in a document can be considered a Bayesian feature for a text classifier, the presence of a link between the target page and a page for which the label is known can be considered a feature for the classifier. The real challenge arises when the labels of all

the nearest neighbors are not available. In such cases, a relaxation labeling method was proposed in order to perform the classification. Two methods have been proposed in this work:

- **Fully Supervised Case of Radius One Enhanced Linkage Analysis:** In this case, it is assumed that all the neighboring class labels are known. In such a case, a Bayesian approach is utilized in order to treat the labels on the nearest neighbors as features for classification purposes. In this case, the linkage information is the sole information that is used for classification purposes.

- **When the class labels of the nearest neighbors are not known:** In this case, an iterative approach is used for combining text and linkage based classification. Rather than using the pre-defined labels (which are not available), we perform a first labeling of the neighboring documents with the use of document content. These labels are then used to classify the label of the target document, with the use of *both* the local text and the class labels of the neighbors. This approach is used iteratively for re-defining the labels of both the target document and its neighbors until convergence is achieved.

The conclusion from the work in [25] is that a combination of text and linkage based classification always improves the accuracy of a text classifier. Even when none of the neighbors of the document have known classes, it always seemed to be beneficial to add link information to the classification process. When the class labels of all the neighbors are known, the advantages of using the scheme seem to be quite significant.

An additional idea in the paper is that of the use of *bridges* in order to further improve the classification accuracy. The core idea in the use of a bridge is the use of *2-hop* propagation for link-based classification. The results with the use of such an approach are somewhat mixed, as the accuracy seems to reduce with an increasing number of hops. The work in [25] shows results on a number of different kinds of data sets such as the *Reuters database*, *US patent database*, and *Yahoo!* Since the *Reuters database* contains the least amount of noise, pure text classifiers were able to do a good job. On the other hand, the *US patent database* and the *Yahoo! database* contain an increasing amount of noise, which reduces the accuracy of text classifiers. An interesting observation in [25] was that a scheme that simply absorbed the neighbor text into the current document performed *significantly worse* than a scheme that was based on pure text-based classification. This is because there are often significant cross-boundary linkages between topics, and such linkages are able to confuse the classifier. A publicly available implementation of this algorithm may be found in the *NetKit* tool kit available in [106].

Another relaxation labeling method for graph-based document classification is proposed in [5]. In this technique, the probability that the end points of a link take on a particular pair of class labels is quantified. We refer to this as the *link-class pair probability*. The posterior probability of classification of a node T into class i is expressed as the sum of the probabilities of pairing all possible class labels of the neighbors of T with class label i. We note a significant percentage of these (exponential number of) possibilities are pruned, since only the currently most probable[6] labelings are used in this approach. For this purpose, it is assumed that the class labels of the different neighbors of T (while dependent on T) are independent of each other. This is similar to the naive assumption, which is often used in Bayes classifiers. Therefore, the probability for a particular combination of labels on the neighbors can be expressed as the product of the corresponding link-class pair probabilities. The approach starts off with the use of a standard content-based Bayes or SVM classifier in order to assign the initial labels to the nodes. Then, an iterative approach is used to refine the labels, by using the most probable label estimations from the previous iteration in order to refine the labels in the current iteration. We note that the link-class pair probabilities can

[6] In the case of *hard labeling*, the single most likely labeling is used, whereas in the case of *soft labeling*, a small set of possibilities is used.

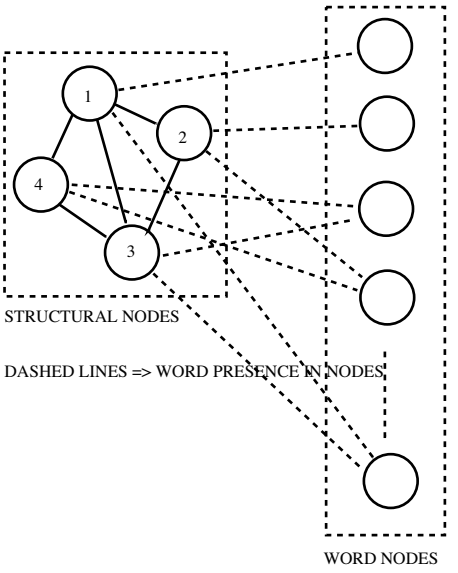


FIGURE 11.3: The Semi-bipartite transformation.

be estimated as the smoothed fraction of edges in the last iteration that contains a particular pair of classes as the end points (hard labeling), or it can also be estimated as the average product of node probabilities over all edges which take on that particular class pair (soft labeling). This approach is repeated to convergence.

Another method that uses a naive Bayes classifier to enhance link-based classification is proposed in [118]. This method incrementally assigns class labels, starting off with a temporary assignment and then gradually making them permanent. The initial class assignment is based on a simple Bayes expression based on both the terms and links in the document. In the final categorization, the method changes the term weights for Bayesian classification of the target document with the terms in the neighbor of the current document. This method uses a broad framework which is similar to that in [25], except that it differentiates between the classes in the neighborhood of a document in terms of their influence on the class label of the current document. For example, documents for which the class label was either already available in the training data, or for which the algorithm has performed a final assignment, have a different confidence weighting factor than those documents for which the class label is currently temporarily assigned. Similarly, documents that belong to a completely different subject (based on content) are also removed from consideration from the assignment. Then, the Bayesian classification is performed with the re-computed weights, so that the document can be assigned a final class label. By using this approach the technique is able to compensate for the noise and inconsistencies in the link structures among different documents.

One major difference between the work in [25] and [118] is that the former is focussed on using link information in order to propagate the labels, whereas the latter attempts to use the content of the neighboring pages. Another work along this direction, which uses the content of the neighboring pages more explicitly, is proposed in [121]. In this case, the content of the neighboring pages is broken up into different fields such as titles, anchor text, and general text. The different fields are given different levels of importance, which is learned during the classification process. It was shown in [121] that the use of title fields and anchor fields is much more relevant than the general text. This accounts for much of the accuracy improvements demonstrated in [121].

The work in [3] proposes a method for dynamic classification in text networks with the use of a random-walk method. The key idea in the work is to *transform* the combination of structure and

content in the network into a pure network containing only content. Thus, we transform the original network $G = (N, A, C)$ into an *augmented* network $G^A = (N \cup N_c, A \cup A_c)$, where N_c and A_c are an additional set of nodes and edges added to the original network. Each node in N_c corresponds to a distinct word in the lexicon. Thus, the augmented network contains the original structural nodes N, and a new set of word nodes N_c. The added edges in A_c are undirected edges added between the structural nodes N and the word nodes N_c. Specifically, an edge (i, j) is added to A_c, if the word $i \in N_c$ occurs in the text content corresponding to the node $j \in N$. Thus, this network is *semi-bipartite*, in that there are no edges between the different word nodes. An illustration of the semi-bipartite content-structure transformation is provided in Figure 11.3.

It is important to note that once such a transformation has been performed, any of the collective classification methods [14] can be applied to the structural nodes. In the work in [3], a random-walk method has been used in order to perform the collective classification of the underlying nodes. In this method, repeated random walks are performed starting at the unlabeled nodes that need to be classified. The random walks are defined only on the structural nodes, and each hop may either be a *structural* hop or a *content* hop. We perform l different random walks, each of which contains h nodes. Thus, a total of $l \cdot h$ nodes are encountered in the different walks. The class label of this node is predicted to be the label with the highest frequency of presence in the different $l \cdot h$ nodes encountered in the different walks. The error of this random walk-based sampling process has been bounded in [14]. In addition, the method in [14] can be adapted to dynamic content-based networks, in which the nodes, edges, and their underlying content continuously evolve over time. The method in [3] has been compared to that proposed in [23] (based on the implementation in [106]), and it has been shown that the classification methods of [14] are significantly superior.

Another method for classification of linked text data is discussed in [160]. This method designs two separate regularization conditions; one is for the text-only classifier (also referred to as the *local* classifier), and the other is for the link information in the network structure. These regularizers are expressed in the terms of the underlying kernels; the link regularizer is related to the standard graph regularizer used in the machine learning literature, and the text regularizer is expressed in terms of the kernel gram matrix. These two regularization conditions are combined in two possible ways. One can either use linear combinations of the regularizers, or linear combinations of the associated kernels. It was shown in [160] that both combination methods perform better than either pure structure-based or pure text-based methods. The method using a linear combination of regularizers was slightly more accurate and robust than the method that used a linear combination of the kernels.

A method in [38] designs a classifier that combines a naive Bayes classifier (on the text domain), and a rule-based classifier (on the structural domain). The idea is to invent a set of predicates, which are defined in the space of links, pages, and words. A variety of predicates (or relations) are defined depending upon the presence of the word in a page, linkages of pages to each other, the nature of the anchor text of the hyperlink, and the neighborhood words of the hyperlink. These essentially encode the graph structure of the documents in the form of boolean predicates, and can also be used to construct relational learners. The main contribution in [38] is to combine the relational learners on the structural domain with the naive Bayes approach in the text domain. We refer the reader to [38, 39] for the details of the algorithm, and the general philosophy of such relational learners.

One of the interesting methods for collective classification in the context of email networks was proposed in [29]. The technique in [29] is designed to classify *speech acts* in email. Speech acts essentially characterize whether an email refers to a particular kind of action (such as scheduling a meeting). It has been shown in [29] that the use of sequential thread-based information from the email is very useful for the classification process. An email system can be modeled as a network in several ways, one of which is to treat an email as a node, and the edges as the thread relationships between the different emails. In this sense, the work in [29] devises a network-based mining procedure that uses both the content and the structure of the email network. However, this work is rather specific to the case of email networks, and it is not clear whether the technique can be adapted (effectively) to more general networks.

A different line of solutions to such problems, which are defined on a heterogeneous feature space, is to use latent space methods in order to simultaneously homogenize the feature space, and also determine the latent factors in the underlying data. The resulting representation can be used in conjunction with any of the text classifiers that are designed for latent space representations. A method in [162] uses a matrix factorization approach in order to construct a latent space from the underlying data. Both supervised and unsupervised methods were proposed for constructing the latent space from the underlying data. It was then shown in [162] that this feature representation provides more accurate results, when used in conjunction with an SVM-classifier.

Finally, a method for Web page classification is proposed in [138]. This method is designed for using intelligent agents in Web page categorization. The overall approach relies on the design of two functions that correspond to scoring Web pages and links respectively. An advice language is created, and a method is proposed for mapping advice to neural networks. It is has been shown in [138] how this general purpose system may be used in order to find home pages on the Web.

11.9 Meta-Algorithms for Text Classification

Meta-algorithms play an important role in classification strategies because of their ability to enhance the accuracy of existing classification algorithms by combining them, or making a general change in the different algorithms to achieve a specific goal. Typical examples of classifier meta-algorithms include *bagging*, *stacking*, and *boosting* [50]. Some of these methods change the underlying distribution of the training data, others combine classifiers, and yet others change the algorithms in order to satisfy specific classification criteria. We will discuss these different classes of methods in this section.

11.9.1 Classifier Ensemble Learning

In this method, we use *combinations* of classifiers in conjunction with a voting mechanism in order to perform the classification. The idea is that since different classifiers are susceptible to different kinds of overtraining and errors, a combination classifier is likely to yield much more robust results. This technique is also sometimes referred to as *stacking* or *classifier committee construction*.

Ensemble learning has been used quite frequently in text categorization. Most methods simply use weighted combinations of classifier outputs (either in terms of scores or ranks) in order to provide the final classification result. For example, the work by Larkey and Croft [91] used weighted linear combinations of the classifier scores or ranks. The work by Hull [70] used linear combinations of probabilities for the same goal. A linear combination of the normalized scores was used for classification [158]. The work in [99] used classifier selection techniques and voting in order to provide the final classification result. Some examples of such voting and selection techniques are as follows:

- In a binary-class application, the class label that obtains the majority vote is reported as the final result.
- For a given test instance, a specific classifier is selected, depending upon the performance of the classifiers that are closest to that test instance.
- A weighted combination of the results from the different classifiers are used, where the weight is regulated by the performance of the classifier on validation instances that are most similar to the current test instance.

The last two methods above try to select the final classification in a smarter way by discriminating between the performances of the classifiers in different scenarios. The work by [89] used category-averaged features in order to construct a different classifier for each category.

The major challenge in ensemble learning is to provide the appropriate combination of classifiers for a particular scenario. Clearly, this combination can significantly vary with the scenario and the data set. In order to achieve this goal, the method in [12] proposes a method for probabilistic combination of text classifiers. The work introduces a number of variables known as *reliability variables* in order to regulate the importance of the different classifiers. These reliability variables are learned dynamically for each situation, so as to provide the best classification.

11.9.2 Data Centered Methods: Boosting and Bagging

While ensemble techniques focus on combining different classifiers, data-centered methods such as boosting and bagging typically focus on training the same classifier on different parts of the training data in order to create different models. For a given test instance, a combination of the results obtained from the use of these different models is reported. Another major difference between ensemble-methods and boosting methods is that the training models in a boosting method are not constructed independently, but are constructed sequentially. Specifically, after i classifiers are constructed, the $(i+1)$th classifier is constructed on those parts of the training data that the first i classifiers are unable to accurately classify. The results of these different classifiers are combined together carefully, where the weight of each classifier is typically a function of its error rate. The most well known meta-algorithm for boosting is the *AdaBoost* algorithm [56]. Such boosting algorithms have been applied to a variety of scenarios such as decision tree learners, rule-based systems, and Bayesian classifiers [57, 71, 85, 114, 134, 137].

We note that boosting is also a kind of ensemble learning methodology, except that we train the same model on different subsets of the data in order to create the ensemble. One major criticism of boosting is that in many data sets, some of the training records are noisy, and a classification model should be resistant to overtraining on the data. Since the boosting model tends to weight the error-prone examples more heavily in successive rounds, this can cause the classification process to be more prone to overfitting. This is particularly noticeable in the case of noisy data sets. Some recent results have suggested that all convex boosting algorithms may perform poorly in the presence of noise [103]. These results tend to suggest that the choice of boosting algorithm may be critical for a successful outcome, depending upon the underlying data set.

Bagging methods [21] are generally designed to reduce the model overfitting error that arises during the learning process. The idea in bagging is to pick *bootstrap samples* (samples with replacement) from the underlying collection, and train the classifiers in these samples. The classification results from these different samples are then combined together in order to yield the final result. Bagging methods are generally used in conjunction with decision trees, though these methods can be used in principle with any kind of classifier. The main criticism of the bagging method is that it can sometimes lead to a reduction in accuracy because of the smaller size of each individual training sample. Bagging is useful only if the model is unstable to small details of the training algorithm, because it reduces the overfitting error. An example of such an algorithm would be the decision tree model, which is highly sensitive to how the higher levels of the tree are constructed in a high dimensional feature space such as text. The main goal in bagging methods is to reduce the variance component of the underlying classifier.

11.9.3 Optimizing Specific Measures of Accuracy

We note that the use of the absolute classification accuracy is not the only measure that is relevant to classification algorithms. For example, in skewed-class scenarios, as often arise in the context of applications such as fraud detection, and spam filtering, it is more costly to misclassify examples

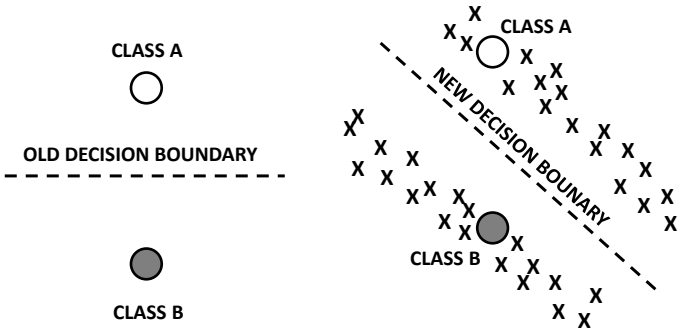
of one class than another. For example, while it may be tolerable to misclassify a few spam emails (thereby allowing them into the inbox), it is much more undesirable to incorrectly mark a legitimate email as spam. Cost-sensitive classification problems also naturally arise in cases in which one class is more rare than the other, and it is therefore more desirable to identify the rare examples. In such cases, it is desirable to optimize the *cost-weighted accuracy* of the classification process. We note that many of the broad techniques that have been designed for non-textual data [48, 50, 53] are also applicable to text data, because the specific feature representation is not material to how standard algorithms are modified to the cost-sensitive case. A good understanding of cost-sensitive classification both for the textual and non-textual case may be found in [4, 48, 53]. Some examples of how classification algorithms may be modified in straightforward ways to incorporate cost-sensitivity are as follows:

- In a decision-tree, the split condition at a given node tries to maximize the accuracy of its children nodes. In the cost-sensitive case, the split is engineered to maximize the cost-sensitive accuracy.
- In rule-based classifiers, the rules are typically quantified and ordered by measures corresponding to their predictive accuracy. In the cost-sensitive case, the rules are quantified and ordered by their *cost-weighted accuracy*.
- In Bayesian classifiers, the posterior probabilities are weighted by the cost of the class for which the prediction is made.
- In linear classifiers, the optimum hyperplane separating the classes is determined in a cost-weighted sense. Such costs can typically be incorporated in the underlying objective function. For example, the least-square error in the objective function of the LLSF method can be weighted by the underlying costs of the different classes.
- In a k-nearest neighbor classifier, we report the cost-weighted majority class among the k nearest neighbors of the test instance.

We note that the use of a cost-sensitive approach is essentially a change of the objective function of classification, which can also be formulated as an optimization problem. While the standard classification problem generally tries to optimize accuracy, the cost-sensitive version tries to optimize a cost-weighted objective function. A more general approach was proposed in [58] in which a meta-algorithm was proposed for optimizing a specific figure of merit such as the accuracy, precision, recall, or F_1-measure. Thus, this approach generalizes this class of methods to *any arbitrary objective function*, making it essentially an *objective-centered classification method*. A generalized probabilistic descent algorithm (with the desired objective function) is used in conjunction with the classifier of interest in order to derive the class labels of the test instance. The work in [58] shows the advantages of using the technique over a standard SVM-based classifier.

11.10 Leveraging Additional Training Data

In this class of methods, *additional labeled or unlabeled data* are used to enhance classification. While the use of additional unlabeled training data was discussed briefly in the section on Bayes classification, this section will discuss it in more detail. Both these methods are used when there is a direct paucity of the underlying training data. In the case of transfer learning (to be discussed later), additional training (labeled) data from a different domain or problem are used to supervise the classification process. On the other hand, in the case of semi-supervised learning, unlabeled data



(a) only labeled examples (b) labeled and unlabeled examples

FIGURE 11.4: Impact of unsupervised examples on classification process.

are used to enhance the classification process. These methods are briefly described in this section. A survey on transfer learning methods may be found in [68].

11.10.1 Semi-Supervised Learning

Co-training and semi-supervised learning methods improve the effectiveness of learning methods with the use of *unlabeled* data [112]. In domains such as text, unlabeled data is copious and freely available from many sources such as the Web. On the other hand, it is much harder to obtain labeled data. The main difference of semi-supervised learning from transfer learning methods is that *unlabeled* data with the same features is used in the former, whereas external labeled data (possibly from a different source) is used in the latter. A key question arises as to why unlabeled data should improve the effectiveness of classification in any way, when it does not provide any additional labeling knowledge. The reason for this is that unlabeled data provides a good idea of the manifolds in which the data is embedded, as well as the density structure of the data in terms of the clusters and sparse regions. The key assumption is that the classification labels exhibit a smooth variation over different parts of the manifold structure of the underlying data. This manifold structure can be used to determine feature correlations, and joint feature distributions, which are very helpful for classification. The semi-supervised setting is also sometimes referred to as the *transductive* setting, when all the unlabeled examples need to be specified together with the labeled training examples.

The motivation of semisupervised learning is that knowledge of the dense regions in the space and correlated regions of the space are helpful for classification. Consider the two-class example illustrated in Figure 11.4(a), in which only a single training example is available for each class. In such a case, the decision boundary between the two classes is the straight line perpendicular to the one joining the two classes. However, suppose that some additional unsupervised examples are available, as illustrated in Figure 11.4(b). These unsupervised examples are denoted by "x". In such a case, the decision boundary changes from Figure 11.4(a). The major assumption here is that the classes vary *less* in dense regions of the training data, because of the *smoothness* assumption. As a result, even though the added examples do not have labels, they contribute significantly to improvements in classification accuracy.

In this example, the *correlations* between feature values were estimated with unlabeled training data. This has an intuitive interpretation in the context of text data, where *joint* feature distributions can be estimated with unlabeled data. For example, consider a scenario where training data is available about predicting whether a document is in the "*politics*" category. It may be possible that the word "*Obama*" (or some of the less common words) may not occur in any of the (small number of) training documents. However, the word "*Obama*" may often co-occur with many features of the

"*politics*" category in the unlabeled instances. Thus, the unlabeled instances can be used to learn the relevance of these less common features to the classification process, especially when the amount of available training data is small.

Similarly, when the data are clustered, each cluster in the data is likely to predominantly contain data records of one class or the other. The identification of these clusters only requires unsupervised data rather than labeled data. Once the clusters have been identified from unlabeled data, only a small number of labeled examples are required in order to determine confidently which label corresponds to which cluster. Therefore, when a test example is classified, its clustering structure provides critical information for its classification process, even when a smaller number of labeled examples are available. It has been argued in [117] that the accuracy of the approach may increase exponentially with the number of labeled examples, as long as the assumption of smoothness in label structure variation holds true. Of course, in real life, this may not be true. Nevertheless, it has been shown repeatedly in many domains that the addition of unlabeled data provides significant advantages for the classification process. An argument for the effectiveness of semi-supervised learning, which uses the spectral clustering structure of the data, may be found in [18]. In some domains such as graph data, semisupervised learning is the only way in which classification may be performed. This is because a given node may have very few neighbors of a specific class.

Text classification from labeled and unlabeled documents uses EM. Semi-supervised methods are implemented in a wide variety of ways. Some of these methods directly try to label the unlabeled data in order to increase the size of the training set. The idea is to incrementally add the most confidently predicted label to the training data. This is referred to as *self training*. Such methods have the downside that they run the risk of overfitting. For example, when an unlabeled example is added to the training data with a specific label, the label might be incorrect because of the specific characteristics of the feature space, or the classifier. This might result in further propagation of the errors. The results can be quite severe in many scenarios.

Therefore, semisupervised methods need to be carefully designed in order to avoid overfitting. An example of such a method is *co-training* [17], which partitions the attribute set into two subsets, on which classifier models are independently constructed. The top label predictions of one classifier are used to augment the training data of the other, and vice-versa. Specifically, the steps of co-training are as follows:

1. Divide the feature space into two disjoint subsets f_1 and f_2.
2. Train two independent classifier models $\mathcal{M}_1$ and $\mathcal{M}_2$, which use the disjoint feature sets f_1 and f_2, respectively.
3. Add the unlabeled instance with the most confidently predicted label from $\mathcal{M}_1$ to the training data for $\mathcal{M}_2$ and vice-versa.
4. Repeat all the above steps.

Since the two classifiers are independently constructed on different feature sets, such an approach avoids overfitting. The partitioning of the feature set into f_1 and f_2 can be performed in a variety of ways. While it is possible to perform random partitioning of features, it is generally advisable to leverage redundancy in the feature set to construct f_1 and f_2. Specifically, each feature set f_i should be picked so that the features in f_j (for $j \neq i$) are redundant with respect to it. Therefore, each feature set represents a different view of the data, which is sufficient for classification. This ensures that the "confident" labels assigned to the other classifier are of high quality. At the same time, overfitting is avoided to at least some degree, because of the disjoint nature of the feature set used by the two classifiers. Typically, an erroneously assigned class label will be more easily detected by the disjoint feature set of the other classifier, which was not used to assign the erroneous label. For a test instance, each of the classifiers is used to make a prediction, and the combination

score from the two classifiers may be used. For example, if the naive Bayes method is used as the base classifier, then the product of the two classifier scores may be used.

The aforementioned methods are generic meta-algorithms for semi-supervised leaning. It is also possible to design variations of existing classification algorithms such as the EM-method, or transductive SVM classifiers. EM-based methods [117] are very popular for text data. These methods attempt to model the joint probability distributions of the features and the labels with the use of partially supervised clustering methods. This allows the estimation of the conditional probabilities in the Bayes classifier to be treated as missing data, for which the EM-algorithm is very effective. This approach shows a connection between the partially supervised clustering and partially supervised classification problems. The results show that partially supervised classification is most effective when the clusters in the data correspond to the different classes. In transductive SVMs, the labels of the unlabeled examples are also treated as integer decision variables. The SVM formulation is modified in order to determine the maximum margin SVM, with the best possible label assignment of unlabeled examples. The SVM classifier has also been shown to be useful in large scale scenarios in which a large amount of unlabeled data and a small amount of labeled data is available [139]. This is essentially a semi-supervised approach because of its use of unlabeled data in the classification process. This technique is also quite scalable because of its use of a number of modified quasi-Newton techniques, which tend to be efficient in practice. Surveys on semi-supervised methods may be found in [27, 163].

11.10.2 Transfer Learning

As in the case of semi-supervised learning, transfer learning methods are used when there is a direct paucity of the underlying training data. However, the difference from semi-supervised learning is that, instead of using unlabeled data, labeled data from a different domain is used to enhance the learning process. For example, consider the case of learning the class label of Chinese documents, where enough training data is not available about the documents. However, similar English documents may be available, that contain training labels. In such cases, the knowledge in training data for the English documents can be *transferred* to the Chinese document scenario for more effective classification. Typically, this process requires some kind of "bridge" in order to relate the Chinese documents to the English documents. An example of such a "bridge" could be pairs of similar Chinese and English documents, though many other models are possible. In many cases, a small amount of auxiliary training data, in the form of labeled Chinese training documents, may also be available in order to further enhance the effectiveness of the transfer process. This general principle can also be applied to cross-category or cross-domain scenarios where knowledge from one classification category is used to enhance the learning of another category [120], or the knowledge from one data domain (e.g., text) is used to enhance the learning of another data domain (e.g., images) [42, 120, 121]. In the context of text data, transfer learning is generally applied in two different ways:

1. *Crosslingual Learning:* In this case, the documents from one language are transferred to the other. An example of such an approach is discussed in [11].

2. *Crossdomain learning:* In this case, knowledge from the text domain is typically transferred to multimedia, and vice-versa. An example of such an approach, which works between the text and image domain, is discussed in [119, 121].

Broadly speaking, transfer learning methods fall into one of the following four categories:

1. *Instance-based Transfer:* In this case, the feature space of the two domains are highly overlapping; even the class labels may be the same. Therefore, it is possible to transfer knowledge from one domain to the other by simply re-weighting the features.

2. *Feature-based Transfer:* In this case, there may be some overlaps among the features, but a significant portion of the feature space may be different. Often, the goal is to perform a transformation of each feature set into a new low dimensional space, which can be shared across related tasks.

3. *Parameter-Based Transfer:* In this case, the motivation is that a good training model has typically learned a lot of structure. Therefore, if two tasks are related, then the structure can be transferred to learn the target task.

4. *Relational-Transfer Learning:* The idea here is that if two domains are related, they may share some similarity relations among objects. These similarity relations can be used for transfer learning across domains.

The major challenge in such transfer learning methods is that *negative* transfer can be caused in some cases when the side information used is very noisy or irrelevant to the learning process. Therefore, it is critical to use the transfer learning process in a careful and judicious way in order to truly improve the quality of the results. A survey on transfer learning methods may be found in [105], and a detailed discussion on this topic may be found in Chapter 21.

11.10.3 Active Learning

A different way of enhancing the classification process is to focus on *label acquisition* actively during the learning process, so as to enhance the training data. Most classification algorithms assume that the learner is a passive recipient of the data set, which is then used to create the training model. Thus, the data collection phase is cleanly separated out from modeling, and is generally not addressed in the context of model construction. However, data collection is costly, and is often the (cost) bottleneck for many classification algorithms. In active learning, the goal is to collect more labels *during the learning process* in order to improve the effectiveness of the classification process at a low cost. Therefore, the learning process and data collection process are tightly integrated with one another and enhance each other. Typically, the classification is performed in an interactive way with the learner providing well chosen examples to the user, for which the user may then provide labels.

In general, the examples are typically chosen for which the learner has the greatest level of uncertainty based on the current training knowledge and labels. This choice evidently provides the greatest additional information to the learner in cases where the greatest uncertainty exists about the current label. As in the case of semi-supervised learning, the assumption is that unlabeled data are copious, but acquiring labels for them is expensive. Therefore, by using the help of the learner in choosing the appropriate examples to label, it is possible to greatly reduce the effort involved in the classification process. Active learning algorithms often use support vector machines, because the latter is particularly good at determining the boundaries between the different classes. Examples that lie on these boundaries are good candidates to query the user, because the greatest level of uncertainty exists for these examples. Numerous criteria exist for training example choice in active learning algorithms, most of which try to either reduce the uncertainty in classification or reduce the error associated with the classification process. A survey on active learning methods may be found in [136]. Active learning is discussed in detail in Chapter 22.

11.11 Conclusions and Summary

The classification problem is one of the most fundamental problems in the machine learning and data mining literature. In the context of text data, the problem can also be considered similar to that of classification of *discrete set-valued* attributes, when the frequencies of the words are ignored.

The domains of these sets are rather large, as it comprises the entire lexicon. Therefore, text mining techniques need to be designed to effectively manage large numbers of elements with varying frequencies. Almost all the known techniques for classification such as decision trees, rules, Bayes methods, nearest neighbor classifiers, SVM classifiers, and neural networks have been extended to the case of text data. Recently, a considerable amount of emphasis has been placed on linear classifiers such as neural networks and SVM classifiers, with the latter being particularly suited to the characteristics of text data. In recent years, the advancement of Web and social network technologies have led to a tremendous interest in the classification of text documents containing links or other meta-information. Recent research has shown that the incorporation of linkage information into the classification process can significantly improve the quality of the underlying results.

Bibliography

[1] C. C. Aggarwal, C. Zhai. A survey of text classification algorithms, In *Mining Text Data*, pages 163–222, Springer, 2012.

[2] C. C. Aggarwal, S. C. Gates, and P. S. Yu. On using partial supervision for text categorization, *IEEE Transactions on Knowledge and Data Engineering*, 16(2):245–255, 2004.

[3] C. C. Aggarwal and N. Li. On node classification in dynamic content-based networks, *SDM Conference*, 2011.

[4] I. Androutsopoulos, J. Koutsias, K. Chandrinos, G. Paliouras, and C. Spyropoulos. An evaluation of naive Bayesian anti-spam filtering. *Proceedings of the* Workshop on *Machine Learning in the New Information Age*, in conjunction with *ECML Conference*, 2000. `http://arxiv.org/PS_cache/cs/pdf/0006/0006013v1.pdf`

[5] R. Angelova and G. Weikum. Graph-based text classification: Learn from your neighbors, *ACM SIGIR Conference*, pages 485–492, 2006.

[6] C. Apte, F. Damerau, and S. Weiss. Automated learning of decision rules for text categorization, *ACM Transactions on Information Systems*, 12(3):233–251, 1994.

[7] M. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning, *Automation and Remote Control*, 25: 821–837, 1964.

[8] L. Baker and A. McCallum. Distributional clustering of words for text classification, *ACM SIGIR Conference*, pages 96–103, 1998.

[9] R. Bekkerman, R. El-Yaniv, Y. Winter, and N. Tishby. On feature distributional clustering for text categorization, *ACM SIGIR Conference*, pages 146–153, 2001.

[10] S. Basu, A. Banerjee, and R. J. Mooney. Semi-supervised clustering by seeding, *ICML Conference*, pages 27–34, 2002.

[11] N. Bel, C. Koster, and M. Villegas. Cross-lingual text categorization. In *Research and advanced technology for digital libraries*, Springer, Berlin Heidelberg, pages 126–139, 2003.

[12] P. Bennett, S. Dumais, and E. Horvitz. Probabilistic combination of text classifiers using reliability indicators: Models and results, *ACM SIGIR Conference*, pages 207, 214, 2002.

[13] P. Bennett and N. Nguyen. Refined experts: Improving classification in large taxonomies. *Proceedings of the 32nd ACM SIGIR Conference*, pages 11–18, 2009.

[14] S. Bhagat, G. Cormode, and S. Muthukrishnan. Node classification in social networks, In *Social Network Data Analytics*, Ed. Charu Aggarwal, Springer, 2011.

[15] C. Bishop. *Neural Networks for Pattern Recognition*, Oxford University Press, 1996.

[16] D. M. Blei and J. D. McAuliffe. Supervised topic models, *NIPS 2007*.

[17] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training, *COLT*, pages 92–100, 1998.

[18] M. Belkin and P. Niyogi. Semi-supervised learning on Riemannian manifolds, *Machine Learning*, 56:209–239, 2004.

[19] D. Boley, M. Gini, R. Gross, E.-H. Han, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, and J. Moore. Partitioning-based clustering for web document categorization, *Decision Support Systems*, 27(3):329–341, 1999.

[20] L. Brieman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*, CRC Press, Boca Raton, FL, 1984.

[21] L. Breiman. Bagging predictors, *Machine Learning*, 24(2):123–140, 1996.

[22] L. Cai, T. Hofmann. Text categorization by boosting automatically extracted concepts, *ACM SIGIR Conference*, pages 182–189, 2003.

[23] S. Chakrabarti, S. Roy, and M. Soundalgekar. Fast and accurate text classification via multiple linear discriminant projections, *VLDB Journal*, 12(2):172–185, 2003.

[24] S. Chakrabarti, B. Dom. R. Agrawal, and P. Raghavan. Using taxonomy, discriminants and signatures for navigating in text databases, *VLDB Conference*, pages 446–455, 1997.

[25] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks, *ACM SIGMOD Conference*, pages 307–318, 1998.

[26] S. Chakraborti, R. Mukras, R. Lothian, N. Wiratunga, S. Watt, and D. Harper. Supervised latent semantic indexing using adaptive sprinkling, *IJCAI*, pages 1582–1587, 2007.

[27] O. Chapelle, B. Scholkopf, and A. Zien. *Semi-Supervised Learning*, Vol. 2, MIT Press, Cambridge, MA, 2006.

[28] D. Chickering, D. Heckerman, and C. Meek. A Bayesian approach for learning Bayesian networks with local structure, *Thirteenth Conference on Uncertainty in Artificial Intelligence*, pages 80–89, 1997.

[29] V. R. de Carvalho and W. Cohen. On the collective classification of email "speech acts", *ACM SIGIR Conference*, pages 345–352, 2005.

[30] V. Castelli and T. M. Cover. On the exponential value of labeled samples, *Pattern Recognition Letters*, 16(1):105–111, 1995.

[31] W. Cohen and H. Hirsh. Joins that generalize: text classification using WHIRL, *ACM KDD Conference*, pages 169–173, 1998.

[32] W. Cohen and Y. Singer. Context-sensitive learning methods for text categorization, *ACM Transactions on Information Systems*, 17(2):141–173, 1999.

[33] W. Cohen. Learning rules that classify e-mail, *AAAI Conference*, pages 18–25, 1996.

[34] W. Cohen. Learning trees and rules with set-valued features, *AAAI Conference*, pages 709–716, 1996.

[35] W. Cooper. Some inconsistencies and misnomers in probabilistic information retrieval, *ACM Transactions on Information Systems*, 13(1):100–111, 1995.

[36] C. Cortes and V. Vapnik. Support-vector networks, *Machine Learning*, 20(3):273–297, 1995.

[37] T. M. Cover and J. A. Thomas. *Elements of Information Theory*, New York: John Wiley and Sons, 1991.

[38] M. Craven and S. Slattery. Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning*, 43(1-2):97–119, 2001.

[39] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to extract symbolic knowledge from the worldwide web, *AAAI Conference*, pages 509–516, 1998.

[40] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge University Press, 2000.

[41] I. Dagan, Y. Karov, and D. Roth. Mistake-driven learning in text categorization, *Proceedings of EMNLP*, pages 55–63, 1997.

[42] W. Dai, Y. Chen, G.-R. Xue, Q. Yang, and Y. Yu. Translated learning: Transfer learning across different feature spaces, *Proceedings of Advances in Neural Information Processing Systems*, 2008.

[43] A. Dayanik, D. Lewis, D. Madigan, V. Menkov, and A. Genkin. Constructing informative prior distributions from domain knowledge in text classification, *ACM SIGIR Conference*, pages 493–500, 2006.

[44] 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*, 39(1): pp. 1–38, 1977.

[45] F. Denis and A. Laurent. Text Classification and Co-Training from Positive and Unlabeled Examples, *ICML 2003 Workshop: The Continuum from Labeled to Unlabeled Data.* `http://www.grappa.univ-lille3.fr/ftp/reports/icmlws03.pdf`.

[46] S. Deerwester, S. Dumais, T. Landauer, G. Furnas, and R. Harshman. Indexing by latent semantic analysis, *JASIS*, 41(6):391–407, 1990.

[47] P. Domingos and M. J. Pazzani. On the the optimality of the simple Bayesian classifier under zero-one loss, *Machine Learning*, 29(2–3), 103–130, 1997.

[48] P. Domingos. MetaCost: A general method for making classifiers cost-sensitive, *ACM KDD Conference*, pages 155–164, 1999.

[49] H. Drucker, D. Wu, and V. Vapnik. Support vector machines for spam categorization, *IEEE Transactions on Neural Networks*, 10(5):1048–1054, 1999.

[50] R. Duda, P. Hart, and W. Stork. *Pattern Classification*, Wiley Interscience, 2000.

[51] S. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization, *CIKM Conference*, pages 148–155, 1998.

[52] S. Dumais and H. Chen. Hierarchical classification of web content, *ACM SIGIR Conference*, pages 256–263, 2000.

[53] C. Elkan. The foundations of cost-sensitive learning, *IJCAI Conference*, pages 973–978, 2001.

[54] R. Fisher. The use of multiple measurements in taxonomic problems, *Annals of Eugenics*, 7(2):179–188, 1936.

[55] R. El-Yaniv and O. Souroujon. Iterative double clustering for unsupervised and semi-supervised learning, *NIPS Conference*, pages 121–132, 2002.

[56] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of Second European Conference on Computational Learning Theory*, pages 23–37, 1995.

[57] Y. Freund, R. Schapire, Y. Singer, and M. Warmuth. Using and combining predictors that specialize, *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 334–343, 1997.

[58] S. Gao, W. Wu, C.-H. Lee, and T.-S. Chua. A maximal figure-of-merit learning approach to text categorization, *SIGIR Conference*, pages 190–218, 2003.

[59] R. Gilad-Bachrach, A. Navot, and N. Tishby. Margin based feature selection – theory and algorithms, *ICML Conference*, pages 43–50, 2004.

[60] S. Gopal and Y. Yang. Multilabel classification with meta-level features, *ACM SIGIR Conference*, pages 315–322, 2010.

[61] L. Guthrie and E. Walker. Document classification by machine: Theory and practice, *COLING*, pages 1059–1063, 1994.

[62] L. Hamel. *Knowledge Discovery with Support Vector Machines*, Wiley, 2009.

[63] E.-H. Han, G. Karypis, and V. Kumar. Text categorization using weighted-adjusted k-nearest neighbor classification, *PAKDD Conference*, pages 53–65, 2001.

[64] E.-H. Han and G. Karypis. Centroid-based document classification: Analysis and experimental results, *PKDD Conference*, pages 424–431, 2000.

[65] D. Hardin, I. Tsamardinos, and C. Aliferis. A theoretical characterization of linear SVM-based feature selection, *ICML Conference*, 2004.

[66] S. Haykin. *Neural Networks and Learning Machines*, Prentice Hall, 2008.

[67] T. Hofmann. Probabilistic latent semantic indexing, *ACM SIGIR Conference*, pages 50–57, 1999.

[68] P. Howland, M. Jeon, and H. Park. Structure preserving dimension reduction for clustered text data based on the generalized singular value decomposition, *SIAM Journal of Matrix Analysis and Applications*, 25(1):165–179, 2003.

[69] P. Howland and H. Park. Generalizing discriminant analysis using the generalized singular value decomposition, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(8):995–1006, 2004.

[70] D. Hull, J. Pedersen, and H. Schutze. Method combination for document filtering, *ACM SIGIR Conference*, pages 279–287, 1996.

[71] R. Iyer, D. Lewis, R. Schapire, Y. Singer, and A. Singhal. Boosting for document routing, *CIKM Conference*, pages 70–77, 2000.

[72] M. James. *Classification Algorithms*, Wiley Interscience, 1985.

[73] D. Jensen, J. Neville, and B. Gallagher. Why collective inference improves relational classification, *ACM KDD Conference*, page 593–598, 2004.

[74] T. Joachims. A probabilistic analysis of the rocchio algorithm with TFIDF for text categorization, *ICML Conference*, pages 143–151, 1997.

[75] T. Joachims. Text categorization with support vector machines: Learning with many relevant features, *ECML Conference*, pages 137–142, 1998.

[76] T. Joachims. Transductive inference for text classification using support vector machines, *ICML Conference*, pages 200–209, 1999.

[77] T. Joachims. A statistical learning model of text classification for support vector machines, *ACM SIGIR Conference*, pages 128–136, 2001.

[78] T. Joachims. Making large scale SVMs practical, In *Advances in Kernel Methods, Support Vector Learning*, pages 169–184, MIT Press, Cambridge, MA, 1998.

[79] T. Joachims. Training linear SVMs in linear time, *KDD*, pages 217–226, 2006.

[80] D. Johnson, F. Oles, T. Zhang and T. Goetz. A decision tree-based symbolic rule induction system for text categorization, *IBM Systems Journal*, 41(3):428–437, 2002.

[81] I. T. Jolliffee. *Principal Component Analysis*, Springer, 2002.

[82] T. Kalt and W. B. Croft. A new probabilistic model of text classification and retrieval, *Technical Report IR-78, University of Massachusetts Center for Intelligent Information Retrieval*, 1996. `http://ciir.cs.umass.edu/publications/index.shtml`

[83] G. Karypis and E.-H. Han. Fast supervised dimensionality reduction with applications to document categorization and retrieval, *ACM CIKM Conference*, pages 12–19, 2000.

[84] T. Kawatani. Topic difference factor extraction between two document sets and its application to text categorization, *ACM SIGIR Conference*, pages 137–144, 2002.

[85] Y.-H. Kim, S.-Y. Hahn, and B.-T. Zhang. Text filtering by boosting naive Bayes classifiers, *ACM SIGIR Conference*, pages 168–175, 2000.

[86] D. Koller and M. Sahami. Hierarchically classifying documents using very few words, *ICML Conference*, pages 170–178, 2007.

[87] S. Lam and D. Lee. Feature reduction for neural network based text categorization, *DASFAA Conference*, pages 195–202, 1999.

[88] W. Lam and C. Y. Ho. Using a generalized instance set for automatic text categorization, *ACM SIGIR Conference*, pages 81–89, 1998.

[89] W. Lam and K.-Y. Lai. A meta-learning approach for text categorization, *ACM SIGIR Conference*, pages 303–309, 2001.

[90] K. Lang. Newsweeder: Learning to filter netnews, *ICML Conference*, page 331–339, 1995.

[91] L. S. Larkey and W. B. Croft. Combining classifiers in text categorization, *ACM SIGIR Conference*, pages 289–297, 1996.

[92] D. Lewis and J. Catlett. Heterogeneous uncertainty sampling for supervised learning, *ICML Conference*, pages 148–156, 1994.

[93] D. Lewis and M. Ringuette. A comparison of two learning algorithms for text categorization, *SDAIR*, pages 83–91, 1994.

[94] D. Lewis. Naive (Bayes) at forty: The independence assumption in information retrieval, *ECML Conference*, pages 4–15, 1998.

[95] D. Lewis. An evaluation of phrasal and clustered representations on a text categorization task, *ACM SIGIR Conference*, pages 37–50, 1992.

[96] D. Lewis and W. Gale. A sequential algorithm for training text classifiers, *SIGIR Conference*, pages 3–12, 1994.

[97] D. Lewis and K. Knowles. Threading electronic mail: A preliminary study, *Information Processing and Management*, 33(2):209–217, 1997.

[98] H. Li and K. Yamanishi. Document classification using a finite mixture model, *Annual Meeting of the Association for Computational Linguistics*, pages 39–47, 1997.

[99] Y. Li and A. Jain. Classification of text documents, *The Computer Journal*, 41(8):537–546, 1998.

[100] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining, *ACM KDD Conference*, pages 80–86, 1998.

[101] B. Liu and L. Zhang. A survey of opinion mining and sentiment analysis. In *Mining Text Data*, Ed. C. Aggarwal, C. Zhai, Springer, 2011.

[102] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 285–318, 1988.

[103] P. Long and R. Servedio. Random classification noise defeats all convex potential boosters, *ICML Conference*, pages 287–304, 2008.

[104] Y. Lu, Q. Mei, and C. Zhai. Investigating task performance of probabilistic topic models: an empirical study of PLSA and LDA, *Information Retrieval*, 14(2):178-203.

[105] S. J. Pan and Q. Yang. A survey on transfer learning, *IEEE Transactons on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[106] S. A. Macskassy and F. Provost. Classification in networked data: A toolkit and a univariate case study, *Journal of Machine Learning Research*, 8(May):935–983, 2007.

[107] A. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering and `http://www.cs.cmu.edu/~mccallum/bow`, 1996.

[108] A. McCallum, K. Nigam. A comparison of event models for naive Bayes text classification, *AAAI Workshop on Learning for Text Categorization*, 1998.

[109] A. McCallum, R. Rosenfeld, and T. Mitchell, A. Ng. Improving text classification by shrinkage in a hierarchy of classes, *ICML Conference*, pages 359–367, 1998.

[110] A.K. McCallum. "MALLET: A Machine Learning for Language Toolkit," `http://mallet.cs.umass.edu`, 2002.

[111] T. M. Mitchell. *Machine Learning*, WCB/McGraw-Hill, 1997.

[112] T. M. Mitchell. The role of unlabeled data in supervised learning, *Proceedings of the Sixth International Colloquium on Cognitive Science*, 1999.

[113] D. Mladenic, J. Brank, M. Grobelnik, and N. Milic-Frayling. Feature selection using linear classifier weights: Interaction with classification models, *ACM SIGIR Conference*, pages 234–241, 2004.

[114] K. Myers, M. Kearns, S. Singh, and M. Walker. A boosting approach to topic spotting on subdialogues, *ICML Conference*, 2000.

[115] H. T. Ng, W. Goh, and K. Low. Feature selection, perceptron learning, and a usability case study for text categorization, *ACM SIGIR Conference*, pages 67–73, 1997.

[116] A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes, *NIPS*. pages 841–848, 2001.

[117] K. Nigam, A. McCallum, S. Thrun, and T. Mitchell. Learning to classify text from labeled and unlabeled documents, *AAAI Conference*, pages 792–799, 1998.

[118] H.-J. Oh, S.-H. Myaeng, and M.-H. Lee. A practical hypertext categorization method using links and incrementally available class information, pages 264–271, *ACM SIGIR Conference*, 2000.

[119] G. Qi, C. Aggarwal, and T. Huang. Towards semantic knowledge propagation from text corpus to web images, *WWW Conference*, pages 297–306, 2011.

[120] G. Qi, C. Aggarwal, Y. Rui, Q. Tian, S. Chang, and T. Huang. Towards cross-category knowledge propagation for learning visual concepts, *CVPR Conference*, pages 897–904, 2011.

[121] X. Qi and B. Davison. Classifiers without borders: incorporating fielded text from neighboring web pages, *ACM SIGIR Conference*, pages 643–650, 2008.

[122] J. R. Quinlan, Induction of decision trees, *Machine Learning*, 1(1):81–106, 1986.

[123] H. Raghavan and J. Allan. An interactive algorithm for asking and incorporating feature feedback into support vector machines, *ACM SIGIR Conference*, pages 79–86, 2007.

[124] S. E. Robertson and K. Sparck-Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27(3):129–146, 1976.

[125] J. Rocchio. Relevance feedback information retrieval. In *The Smart Retrieval System- Experiments in Automatic Document Processing*, G. Salton, Ed., Prentice Hall, Englewood Cliffs, NJ, pages 313–323, 1971.

[126] M. Ruiz and P. Srinivasan. Hierarchical neural networks for text categorization, *ACM SIGIR Conference*, pages 281–282, 1999.

[127] F. Sebastiani. Machine learning in automated text categorization, *ACM Computing Surveys*, 34(1):1–47, 2002.

[128] M. Sahami. Learning limited dependence Bayesian classifiers, *ACM KDD Conference*, pages 335–338, 1996.

[129] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian approach to filtering junk e-mail, *AAAI Workshop on Learning for Text Categorization. Tech. Rep. WS-98-05, AAAI Press.* http://robotics.stanford.edu/users/sahami/papers.html

[130] T. Salles, L. Rocha, G. Pappa, G. Mourao, W. Meira Jr., and M. Goncalves. Temporally-aware algorithms for document classification, *ACM SIGIR Conference*, pages 307–314, 2010.

[131] G. Salton. *An Introduction to Modern Information Retrieval*, Mc Graw Hill, 1983.

[132] B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, Cambridge University Press, 2001.

[133] I. Steinwart and A. Christmann. *Support Vector Machines*, Springer, 2008.

[134] R. Schapire and Y. Singer. BOOSTEXTER: A Boosting-based system for text categorization, *Machine Learning*, 39(2/3):135–168, 2000.

[135] H. Schutze, D. Hull, and J. Pedersen. A comparison of classifiers and document representations for the routing problem, *ACM SIGIR Conference*, pages 229–237, 1995.

[136] B. Settles. *Active Learning*, Morgan and Claypool, 2012.

[137] R. Shapire, Y. Singer, and A. Singhal. Boosting and Rocchio applied to text filtering, *ACM SIGIR Conference*, pages 215–223, 1998.

[138] J. Shavlik and T. Eliassi-Rad. Intelligent agents for web-based tasks: An advice-taking approach, *AAAI-98 Workshop on Learning for Text Categorization. Tech. Rep. WS-98-05, AAAI Press*, 1998. http://www.cs.wisc.edu/~shavlik/mlrg/publications.html

[139] V. Sindhwani and S. S. Keerthi. Large scale semi-supervised linear SVMs, *ACM SIGIR Conference*, pages 477–484, 2006.

[140] N. Slonim and N. Tishby. The power of word clusters for text classification, *European Colloquium on Information Retrieval Research (ECIR)*, 2001.

[141] N. Slonim, N. Friedman, and N. Tishby. Unsupervised document classification using sequential information maximization, *ACM SIGIR Conference*, pages 129–136, 2002.

[142] J.-T. Sun, Z. Chen, H.-J. Zeng, Y. Lu, C.-Y. Shi, and W.-Y. Ma. Supervised latent semantic indexing for document categorization, *ICDM Conference*, pages 535–538, 2004.

[143] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining.* Pearson, 2005.

[144] V. Vapnik. *Estimations of dependencies based on statistical data*, Springer, 1982.

[145] V. Vapnik. *The Nature of Statistical Learning Theory*, Springer, New York, 1995.

[146] A. Weigand, E. Weiner, and J. Pedersen. Exploiting hierarchy in text catagorization. *Information Retrieval*, 1(3):193–216, 1999.

[147] S. M. Weiss, C. Apte, F. Damerau, D. Johnson, F. Oles, T. Goetz, and T. Hampp. Maximizing text-mining performance, *IEEE Intelligent Systems*, 14(4):63–69, 1999.

[148] S. M. Weiss and N. Indurkhya. Optimized rule induction, *IEEE Expert*, 8(6):61–69, 1993.

[149] E. Wiener, J. O. Pedersen, and A. S. Weigend. A neural network approach to topic spotting, *SDAIR*, pages 317–332, 1995.

[150] J. Xu and B. W. Croft, Improving the effectiveness of information retrieval with local context analysis, *ACM Transactions on Information Systems*, 18(1), Jan, 2000. pp. 79–112.

[151] G.-R. Xue, D. Xing, Q. Yang, Y. Yu. Deep classification in large-scale text hierarchies, *ACM SIGIR Conference*, 2008.

[152] J. Yan, N. Liu, B. Zhang, S. Yan, Z. Chen, Q. Cheng, W. Fan, W.-Y. Ma. OCFS: optimal orthogonal centroid feature selection for text categorization, *ACM SIGIR Conference*, 2005.

[153] Y. Yang, L. Liu. A re-examination of text categorization methods, *ACM SIGIR Conference*, 1999.

[154] Y. Yang, J. O. Pederson. A comparative study on feature selection in text categorization, *ACM SIGIR Conference*, 1995.

[155] Y. Yang, C.G. Chute. An example-based mapping method for text categorization and retrieval, *ACM Transactions on Information Systems*, 12(3), 1994.

[156] Y. Yang. Noise Reduction in a Statistical Approach to Text Categorization, *ACM SIGIR Conference*, 1995.

[157] Y. Yang. A Study on Thresholding Strategies for Text Categorization, *ACM SIGIR Conference*, 2001.

[158] Y. Yang, T. Ault, T. Pierce. Combining multiple learning strategies for effective cross-validation, *ICML Conference*, 2000.

[159] J. Zhang, Y. Yang. Robustness of regularized linear classification methods in text categorization, *ACM SIGIR Conference*, 2003.

[160] T. Zhang, A. Popescul, B. Dom. Linear prediction models with graph regularization for web-page categorization, *ACM KDD Conference*, 2006.

[161] C. Zhai, *Statistical Language Models for Information Retrieval (Synthesis Lectures on Human Language Technologies)*, Morgan and Claypool Publishers, 2008.

[162] S. Zhu, K. Yu, Y. Chi, Y. Gong. Combining content and link for classification using matrix factorization, *ACM SIGIR Conference*, 2007.

[163] X. Zhu and A. Goldberg. *Introduction to Semi-Supervised Learning*, Morgan and Claypool, 2009.