IV

Categorization

Probably the most common theme in analyzing complex data is the classification, or categorization, of elements. Described abstractly, the task is to classify a given data instance into a prespecified set of categories. Applied to the domain of document management, the task is known as *text categorization* (TC) – given a set of categories (subjects, topics) and a collection of text documents, the process of finding the correct topic (or topics) for each document.

The study of automated text categorization dates back to the early 1960s (Maron 1961). Then, its main projected use was for indexing scientific literature by means of controlled vocabulary. It was only in the 1990s that the field fully developed with the availability of ever increasing numbers of text documents in digital form and the necessity to organize them for easier use. Nowadays automated TC is applied in a variety of contexts – from the classical automatic or semiautomatic (interactive) indexing of texts to personalized commercials delivery, spam filtering, Web page categorization under hierarchical catalogues, automatic generation of metadata, detection of text genre, and many others.

As with many other artificial intelligence (AI) tasks, there are two main approaches to text categorization. The first is the *knowledge engineering* approach in which the expert's knowledge about the categories is directly encoded into the system either declaratively or in the form of procedural classification rules. The other is the *machine learning* (ML) approach in which a general inductive process builds a classifier by learning from a set of preclassified examples. In the document management domain, the knowledge engineering systems usually outperform the ML systems, although the gap in performance steadily shrinks. The main drawback of the knowledge engineering approach is what might be called the *knowledge acquisition bottleneck* – the huge amount of highly skilled labor and expert knowledge required to create and maintain the knowledge-encoding rules. Therefore, most of the recent work on categorization is concentrated on the ML approach, which requires only a set of manually classified training instances that are much less costly to produce.

This chapter is organized as follows. We start with the description of several common applications of text categorization. Then the formal framework and the issues of problem representation are described. Next we survey the most commonly used algorithms solving the TC problem and wrap up with the issues of experimental evaluation and a comparison between the different algorithms.

IV.1 APPLICATIONS OF TEXT CATEGORIZATION

Three common TC applications are text indexing, document sorting and text filtering, and Web page categorization. These are only a small set of possible applications, but they demonstrate the diversity of the domain and the variety of the TC subcases.

IV.1.1 Indexing of Texts Using Controlled Vocabulary

The topic of most of the early research in the TC field is text indexing. In Boolean information retrieval (IR) systems, each document in a big collection is assigned one or more key terms describing its content. Then, the IR system is able to retrieve the documents according to the user queries, which are based on the key terms. The key terms all belong to a finite set called *controlled vocabulary*, which is often a thematic hierarchical thesaurus such as the NASA aerospace thesaurus or the MESH thesaurus for medicine.

The task of assigning keywords from a controlled vocabulary to text documents is called *text indexing*. If the keywords are viewed as categories, then text indexing is an instance of the general TC problem and can be addressed by the automatic techniques described in this chapter.

Typically, each document should receive at least one, and not more than *k*, keywords. Also, the task can be solved either fully automatically or semiautomatically, in which case the user selects a set of keywords from a ranked list supplied by a TC system.

Automatic indexing can be a part of *automated extraction of metadata*. The metadata describe a document in a variety of aspects, some of which are thematic – related to the contents of the document – the bibliographic codes, key terms, and so on. Extraction of this metadata can be viewed as a document indexing problem, which can be tackled by TC techniques.

IV.1.2 Document Sorting and Text Filtering

Another common problem related but distinct from document indexing is sorting the given collection of documents into several "bins." For instance, in a newspaper, the classified ads may need to be categorized into "Personal," "Car Sale," "Real Estate," and so on. Another example is e-mail coming into an organization, which may need to be sorted into categories such as "Complaints," "Deals," "Job applications," and others.

The document sorting problem has several features that distinguish it from the related tasks. The main difference is the requirement that each document belong to exactly one category. Other typical features are relatively small numbers of categories and the "online" nature of the task: The documents to be categorized are usually presented to the classifier one by one, not as a single batch.

Text filtering activity can be seen as document sorting with only two bins – the "relevant" and "irrelevant" documents. Examples of text filtering abound. A sports-related online magazine should filter out all nonsport stories it receives from the news feed. An e-mail client should filter away spam. A personalized ad filtering system should block any ads that are uninteresting to the particular user.

For most of the TC systems, recall errors (which arise when a category is missing some document that should have been assigned to it) and precision errors (which occur when a category includes documents that should not belong to it) are considered to have about the same cost. For many of the filtering tasks, however, the recall errors (e.g., an important letter is considered spam and hence is missing from the "good documents" category) are much more costly than precision errors (some of the spam still passes through, and thus the "good documents" category contains some extra letters).

For personalized filtering systems it is common for the user to provide the feedback to the system – by marking received documents as relevant or irrelevant. Because it is usually computationally unfeasible to fully retrain the system after each document, *adaptive* learning techniques are required (see Bibliography).

IV.1.3 Hierarchical Web Page Categorization

A common use of TC is the automatic classification of Web pages under the hierarchical catalogues posted by popular Internet portals such as Yahoo. Such catalogues are very useful for direct browsing and for restricting the query-based search to pages belonging to a particular topic.

The other applications described in this section usually constrain the number of categories to which a document may belong. Hierarchical Web page categorization, however, constrains the number of documents belonging to a particular category to prevent the categories from becoming excessively large. Whenever the number of documents in a category exceeds k, it should be split into two or more subcategories. Thus, the categorization system must support adding new categories and deleting obsolete ones.

Another feature of the problem is the hypertextual nature of the documents. The Web documents contain links, which may be important sources of information for the classifier because linked documents often share semantics.

The hierarchical structure of the set of categories is also uncommon. It can be dealt with by using a separate classifier at every branching point of the hierarchy.

IV.2 DEFINITION OF THE PROBLEM

The general text categorization task can be formally defined as the task of approximating an unknown category assignment function $F : D \times C \to \{0, 1\}$, where D is the set of all possible documents and C is the set of predefined categories. The value of $F(d, c)$ is 1 if the document d belongs to the category c and 0 otherwise. The approximating function $M : D \times C \to \{0, 1\}$ is called a *classifier*, and the task is to build a classifier that produces results as "close" as possible to the true category assignment function F.

IV.2.1 Single-Label versus Multilabel Categorization

Depending on the properties of F, we can distinguish between *single-label* and *multilabel* categorization. In multilabel categorization the categories overlap, and a document may belong to any number of categories. In single-label categorization, each document belongs to exactly one category. *Binary* categorization is a special case of single-label categorization in which the number of categories is two. The binary case is the most important because it is the simplest, most common, and most often used for the demonstration of categorization techniques. Also, the general single-label case is frequently a simple generalization of the binary case. The multilabel case can be solved by $|C|$ binary classifiers (|C| is the number of categories), one for each category, provided the decisions to assign a document to different categories are independent from each other.

IV.2.2 Document-Pivoted versus Category-Pivoted Categorization

Usually, the classifiers are used in the following way: Given a document, the classifier finds all categories to which the document belongs. This is called a *document-pivoted categorization*. Alternatively, we might need to find all documents that should be filed under a given category. This is called a *category-pivoted categorization*. The difference is significant only in the case in which not all documents or not all categories are immediately available. For instance, in "online" categorization, the documents come in one-by-one, and thus only the document-pivoted categorization is possible. On the other hand, if the categories set is not fixed, and if the documents need to be reclassified with respect to the newly appearing categories, then category-pivoted categorization is appropriate. However, most of the techniques described in this chapter allow both.

IV.2.3 Hard versus Soft Categorization

A fully automated categorization system makes a binary decision on each document-category pair. Such a system is said to be doing the *hard* categorization. The level of performance currently achieved by fully automatic systems, however, may be insufficient for some applications. Then, a semiautomated approach is appropriate in which the decision to assign a document to a category is made by a human for whom the TC system provides a list of categories arranged by the system's estimated appropriateness of the category for the document. In this case, the system is said to be doing the *soft* or *ranking* categorization. Many classifiers described in this chapter actually have the whole segment [0, 1] as their range – that is, they produce a real value between zero and one for each document-category pair. This value is called a *categorization status value* (CSV). Such "continuous" classifiers naturally perform ranking categorization, but if a binary decision is needed it can be produced by checking the CSV against a specific threshold.

Various possible policies exist for setting the threshold. For some types of classifiers it is possible to calculate the thresholds analytically, using decision-theoretic measures such as utility. There are also general classifier-independent methods. *Fixed thresholding* assigns exactly k top-ranking categories to each document. *Proportional thresholding* sets the threshold in such a way that the same fraction of the test set belongs to a category as to the corresponding fraction of the training set. Finally, the most common method is to set the value of the threshold in such a way as to maximize the performance of the classifier on a validation set. The validation set is some portion of the training set that is not used for creating the model. The sole purpose of the validation set is to optimize some of the parameters of the classifier (such as the threshold). Experiments suggest that the latter method is usually superior to the others in performance (Lewis 1992a, 1992b; Yang 1999).

IV.3 DOCUMENT REPRESENTATION

The common classifiers and learning algorithms cannot directly process the text documents in their original form. Therefore, during a preprocessing step, the documents are converted into a more manageable representation. Typically, the documents are represented by *feature vectors*. A feature is simply an entity without internal structure – a dimension in the feature space. A document is represented as a vector in this space – a sequence of features and their weights.

The most common *bag-of-words* model simply uses all words in a document as the features, and thus the dimension of the feature space is equal to the number of different words in all of the documents. The methods of giving weights to the features may vary. The simplest is the *binary* in which the feature weight is either one – if the corresponding word is present in the document – or zero otherwise. More complex weighting schemes are possible that take into account the frequencies of the word in the document, in the category, and in the whole collection. The most common TF-IDF scheme gives the word w in the document d the weight

$$TF\text{-}IDF\_Weight\,(w, d) = TermFreq(w, d) \cdot \log\,(N \,/\, DocFreq(w)),$$

where $TermFreq(w, d)$ is the frequency of the word in the document, N is the number of all documents, and $DocFreq(w)$ is the number of documents containing the word w.

IV.3.1 Feature Selection

The number of different words is large even in relatively small documents such as short news articles or paper abstracts. The number of different words in big document collections can be huge. The dimension of the bag-of-words feature space for a big collection can reach hundreds of thousands; moreover, the document representation vectors, although sparse, may still have hundreds and thousands of nonzero components.

Most of those words are irrelevant to the categorization task and can be dropped with no harm to the classifier performance and may even result in improvement owing to noise reduction. The preprocessing step that removes the irrelevant words is called *feature selection*. Most TC systems at least remove the *stop words* – the function words and in general the common words of the language that usually do not contribute to the semantics of the documents and have no real added value. Many systems, however, perform a much more aggressive filtering, removing 90 to 99 percent of all features.

In order to perform the filtering, a measure of the relevance of each feature needs to be defined. Probably the simplest such measure is the document frequency *DocFreq*(*w*). Experimental evidence suggests that using only the top 10 percent of the most frequent words does not reduce the performance of classifiers. This seems to contradict the well-known “law” of IR, according to which the terms with low-to-medium document frequency are the most informative. There is no contradiction, however, because the large majority of all words have a *very* low document frequency, and the top 10 percent do contain all low-to-medium frequency words.

More sophisticated measures of feature relevance exist that take into account the relations between features and the categories. For instance, the *information gain*

$$IG(w) = \sum_{c \in C \cup \overline{C}} \sum_{f \in \{w, \overline{w}\}} P(f, c) \cdot \log \frac{P(c \mid f)}{P(c)}$$

measures the number of bits of information obtained for the prediction of categories by knowing the presence or absence in a document of the feature *f*. The probabilities are computed as ratios of frequencies in the training data. Another good measure is the chi-square

$$\chi^2_{\max}(f) = \max_{c \in C} \frac{|Tr| \cdot (P(f, c) \cdot P(\bar{f}, \bar{c}) - P(f, \bar{c}) \cdot P(\bar{f}, c))^2}{P(f) \cdot P(\bar{f}) \cdot P(c) \cdot P(\bar{c})},$$

which measures the maximal strength of dependence between the feature and the categories. Experiments show that both measures (and several other measures) can reduce the dimensionality by a factor of 100 without loss of categorization quality – or even with a small improvement (Yang and Pedersen 1997).

IV.3.2 Dimensionality Reduction by Feature Extraction

Another way of reducing the number of dimensions is to create a new, much smaller set of synthetic features from the original feature set. In effect, this amounts to creating a transformation from the original feature space to another space of much lower dimension. The rationale for using synthetic features rather than naturally occurring words (as the simpler feature filtering method does) is that, owing to polysemy, homonymy, and synonymy, the words may not be the optimal features. By transforming the set of features it may be possible to create document representations that do not suffer from the problems inherent in those properties of natural language.

Term clustering addresses the problem of synonymy by grouping together words with a high degree of semantic relatedness. These word groups are then used as features instead of individual words. Experiments conducted by several groups of researchers showed a potential in this technique only when the background information about categories was used for clustering (Baker and McCallum 1998; Slonim and Tishby 2001). With unsupervised clustering, the results are inferior (Lewis 1992a, 1992b; Li and Jain 1998).

A more systematic approach is latent semantic indexing (LSI). The details of this method are described in Chapter V. For the TC problem, the performance of the LSI also improves if the categories information is used. Several LSI representations, one for each category, outperform a single global LSI representation. The experiments also show that LSI usually performs better than the chi-square filtering scheme.

IV.4 KNOWLEDGE ENGINEERING APPROACH TO TC

The knowledge engineering approach to TC is focused around manual development of classification rules. A domain expert defines a set of sufficient conditions for a document to be labeled with a given category. The development of the classification rules can be quite labor intensive and tedious.

We mention only a single example of the knowledge engineering approach to the TC – the well-known CONSTRUE system (Hayes, Knecht, and Cellio 1988; Hayes et al. 1990; Hayes and Weinstein 1990; Hayes 1992) built by the Carnegie group for Reuters. A typical rule in the CONSTRUE system is as follows:

if DNF (disjunction of conjunctive clauses) formula ***then*** category ***else*** ¬category

Such rule may look like the following:

If ((wheat & farm) or
(wheat & commodity) or
(bushels & export) or
(wheat & tonnes) or
(wheat & winter & ¬soft))
then Wheat
else ¬Wheat

The system was reported to produce a 90-percent breakeven between precision and recall on a small subset of the Reuters collection (723 documents). It is unclear whether the particular chosen test collection influenced the results and whether the system would scale up, but such excellent performance has not yet been unattained by machine learning systems.

However, the knowledge acquisition bottleneck that plagues such expert systems (it took several man-years to develop and fine-tune the CONSTRUE system for Reuters) makes the ML approach attractive despite possibly somewhat lower quality results.

IV.5 MACHINE LEARNING APPROACH TO TC

In the ML approach, the classifier is built automatically by learning the properties of categories from a set of preclassified training documents. In the ML terminology, the learning process is an instance of *supervised* learning because the process is guided by applying the known true category assignment function on the training set. The unsupervised version of the classification task, called *clustering*, is described in Chapter V. There are many approaches to classifier learning; some of them are variants of more general ML algorithms, and others have been created specifically for categorization.

Four main issues need to be considered when using machine learning techniques to develop an application based on text categorization. First, we need to decide on the categories that will be used to classify the instances. Second, we need to provide a training set for each of the categories. As a rule of thumb, about 30 examples are needed for each category. Third, we need to decide on the features that represent each of the instances. Usually, it is better to generate as many features as possible because most of the algorithms will be able to focus just on the relevant features. Finally, we need to decide on the algorithm to be used for the categorization.

IV.5.1 Probabilistic Classifiers

Probabilistic classifiers view the categorization status value CSV(d, c) as the probability $P(c \mid d)$ that the document d belongs to the category c and compute this probability by an application of Bayes' theorem:

$$P(c \mid d) = \frac{P(d \mid c)P(c)}{P(d)}.$$

The marginal probability $P(d)$ need not ever be computed because it is constant for all categories. To calculate $P(d \mid c)$, however, we need to make some assumptions about the structure of the document d. With the document representation as a feature vector $\mathbf{d} = (w_1, w_2, \ldots)$, the most common assumption is that all coordinates are independent, and thus

$$P(d \mid c) = \prod_i P(w_i \mid c).$$

The classifiers resulting from this assumption are called *Naïve Bayes* (NB) classifiers. They are called "naïve" because the assumption is never verified and often is quite obviously false. However, the attempts to relax the naïve assumption and to use the probabilistic models with dependence so far have not produced any significant improvement in performance. Some theoretic justification to this unexpected robustness of the Naïve Bayes classifiers is given in Domingos and Pazzani (1997).

IV.5.2 Bayesian Logistic Regression

It is possible to model the conditional probability $P(c \mid d)$ directly. Bayesian logistic regression (BLR) is an old statistical approach that was only recently applied to the TC problem and is quickly gaining popularity owing to its apparently very high performance.

Assuming the categorization is binary, we find that the logistic regression model has the form

$$P(c \mid \mathbf{d}) = \varphi(\boldsymbol{\beta} \cdot \mathbf{d}) = \varphi\left(\sum_i \beta_i d_i\right),$$

where $c = \pm 1$ is the category membership value (± 1 is used instead of $\{0, 1\}$ for simpler notation), $\mathbf{d} = (d_1, d_2, \ldots)$ is the document representation in the feature space, $\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots)$ is the model parameters vector, and φ is the *logistic link* function

$$\varphi(x) = \frac{\exp(x)}{1 + \exp(x)} = \frac{1}{1 + \exp(-x)}.$$

Care must be taken in order for a logistic regression model not to overfit the training data. The Bayesian approach is to use a prior distribution for the parameter vector $\boldsymbol{\beta}$ that assigns a high probability to each β_i's being at or near zero. Different priors are possible, and the commonly used are Gaussian and Laplace priors.

The simplest is the Gaussian prior with zero mean and variance τ:

$$p(\beta_i \mid \tau) = N(0, \tau) = \frac{1}{\sqrt{2\pi\tau}} \exp\left(\frac{-\beta_i^2}{2\tau}\right).$$

If the a priori independence of the components of $\boldsymbol{\beta}$ and the equality of variances τ for all components are assumed, the overall prior for $\boldsymbol{\beta}$ is the product of the priors for β_i. With this prior, the maximum a posteriori (MAP) estimate of $\boldsymbol{\beta}$ is equivalent to ridge regression for the logistic model.

The disadvantage of the Gaussian prior in the TC problem is that, although it favors the parameter values' being close to zero, the MAP estimates of the parameters will rarely be exactly zero; thus, the model will not be sparse. The alternative Laplace prior does achieve sparseness:

$$p(\beta_i \mid \lambda) = \frac{\lambda}{2} \exp(-\lambda\, |\beta_i|).$$

Using this kind of prior represents a belief that a small portion of the input variables has a substantial effect on the outcome, whereas most of the other variables are unimportant. This belief is certainly justifiable for the TC task. The particular value of the hyperparameter λ (and τ for the Gaussian prior) can be chosen a priori or optimized using a validation set.

In an "ideal" setting, the posterior distribution of $\boldsymbol{\beta}$ would be used for the actual prediction. Owing to computational cost constraints, however, it is common to use a point estimate of $\boldsymbol{\beta}$, of which the *posterior mode* (any value of $\boldsymbol{\beta}$ at which the posterior distribution of $\boldsymbol{\beta}$ takes on its maximal value) is the most common.

The log-posterior distribution of $\boldsymbol{\beta}$ is

$$l(\boldsymbol{\beta}) = p(\boldsymbol{\beta} \mid D) = -\left(\sum_{(\mathbf{d},c)\in D} \ln(\exp(-c\,\boldsymbol{\beta}\cdot\mathbf{d}) + 1)\right) + \ln p(\boldsymbol{\beta}),$$

where $D = \{(\mathbf{d}_1, c_1), (\mathbf{d}_2, c_2)\ldots\}$ is the set of training documents $\mathbf{d}_i$ and their true category membership values $c_i = \pm 1$, and $p(\boldsymbol{\beta})$ is the chosen prior:

$$\ln p(\boldsymbol{\beta}) = -\left(\sum_i \left(\ln\sqrt{\tau} + \frac{\ln 2\pi}{2} + \frac{\beta_i^2}{\tau}\right)\right),\ \text{for Gaussian prior, and}$$

$$\ln p(\boldsymbol{\beta}) = -\left(\sum_i (\ln 2 - \ln\lambda + \lambda\,|\,\beta_i\,|)\right),\ \text{for Laplace prior.}$$

The MAP estimate of $\boldsymbol{\beta}$ is then simply $\arg\max_{\boldsymbol{\beta}} l(\boldsymbol{\beta})$, which can be computed by any convex optimization algorithm.

IV.5.3 Decision Tree Classifiers

Many categorization methods share a certain drawback: The classifiers cannot be easily understood by humans. The *symbolic* classifiers, of which the decision tree classifiers are the most prominent example, do not suffer from this problem.

A decision tree (DT) classifier is a tree in which the internal nodes are labeled by the features, the edges leaving a node are labeled by tests on the feature's weight, and the leaves are labeled by categories. A DT categorizes a document by starting at the root of the tree and moving successively downward via the branches whose

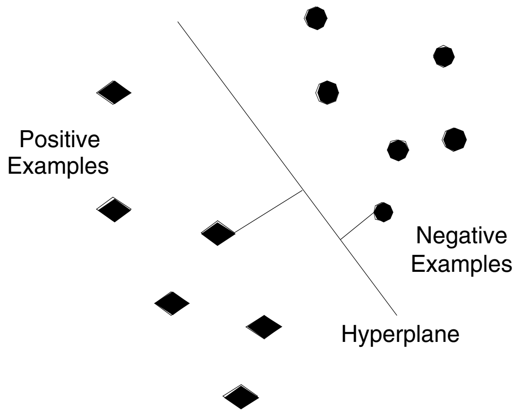

Figure IV.1. A Decision Tree classifier.

conditions are satisfied by the document until a leaf node is reached. The document is then assigned to the category that labels the leaf node. Most of the DT classifiers use a binary document representation, and thus the trees are binary. For example, the tree that corresponds to the CONSTRUE rule mentioned in Section IV.4 may look like Figure IV.1.

Most of the DT-based systems use some form of general procedure for a DT induction such as ID3, C4.5, and CART. Typically, the tree is built recursively by picking a feature f at each step and dividing the training collection into two subcollections, one containing f and another not containing f, until only documents of a single category remain – at which point a leaf node is generated. The choice of a feature at each step is made by some information-theoretic measure such as information gain or entropy. However, the trees generated in such a way are prone to overfit the training collection, and so most methods also include *pruning* – that is, removing the too specific branches.

The performance of a DT classifier is mixed but is inferior to the top-ranking classifiers. Thus it is rarely used alone in tasks for which the human understanding of the classifier is not essential. DT classifiers, however, are often used as a baseline for comparison with other classifiers and as members of classifier committees.

IV.5.4 Decision Rule Classifiers

Decision rule (DR) classifiers are also symbolic like decision trees. The rules look very much like the disjunctive normal form (DNF) rules of the CONSTRUE system but are built from the training collection using *inductive rule learning*. Typically, the rule learning methods attempt to select the best rule from the set of all possible covering rules (i.e., rules that correctly classify all training examples) according to some optimality criterion. DNF rules are often built in a bottom-up fashion. The initial most specific classifier is built from the training set by viewing each training document as a clause

$$d_1 \wedge d_2 \wedge \ldots \wedge d_n \rightarrow c,$$

where d_i are the features of the document and c its category. The learner then applies a series of generalizations (e.g., by removing terms from the clauses and by merging rules), thus maximizing the compactness of the rules while keeping the *covering* property. At the end of the process, a *pruning* step similar to the DT pruning is applied that trades covering for more generality.

Rule learners vary widely in their specific methods, heuristics, and optimality criteria. One of the prominent examples of this family of algorithms is RIPPER (repeated incremental pruning to produce error reduction) (Cohen 1995a; Cohen 1995b; Cohen and Singer 1996). Ripper builds a rule set by first adding new rules until all positive category instances are covered and then adding conditions to the rules until no negative instance is covered. One of the attractive features of Ripper is its ability to bias the performance toward higher precision or higher recall as determined by the setting of the *loss ratio* parameter, which measures the relative cost of "false negative" and "false positive" errors.

IV.5.5 Regression Methods

Regression is a technique for approximating a real-valued function using the knowledge of its values on a set of points. It can be applied to TC, which is the problem of approximating the category assignment function. For this method to work, the assignment function must be considered a member of a suitable family of continuous real-valued functions. Then the regression techniques can be applied to generate the (real-valued) classifier.

One regression method is the linear least-square fit (LLSF), which was first applied to TC in Yang and Chute (1994). In this method, the category assignment function is viewed as a $|C| \times |F|$ matrix, which describes some linear transformation from the feature space to the space of all possible category assignments. To build a classifier, we create a matrix that best accounts for the training data. The LLSF model computes the matrix by minimizing the error on the training collection according to the formula

$$M = \text{arg min}_M ||MD - O||_F,$$

where D is the $|F| \times |TrainingCollection|$ matrix of the training document representations, O is the $|C| \times |TrainingCollection|$ matrix of the true category assignments, and the $|| \cdot ||_F$ is the *Frobenius norm*

$$||A||_F = \sqrt{\sum A_{ij}^2}.$$

The matrix M can be computed by performing singular value decomposition on the training data. The matrix element m_{ij} represents the degree of association between the ith feature and the jth category.

IV.5.6 The Rocchio Methods

The Rocchio classifier categorizes a document by computing its distance to the prototypical examples of the categories. A prototypical example for the category c is a vector $(w_1, w_2, \ldots)$ in the feature space computed by

$$w_i = \frac{\alpha}{|\text{POS}(c)|} \sum_{d \in \text{POS}(c)} w_{di} - \frac{\beta}{|\text{NEG}(c)|} \sum_{d \in \text{NEG}(c)} w_{di},$$

where POS(c) and NEG(c) are the sets of all training documents that belong and do not belong to the category c, respectively, and w_{di} is the weight of ith feature in the document d. Usually, the positive examples are much more important than the negative ones, and so $\alpha >> \beta$. If $\beta = 0$, then the prototypical example for a category is simply the centroid of all documents belonging to the category.

The Rocchio method is very easy to implement, and it is cheap computationally. Its performance, however, is usually mediocre – especially with categories that are unions of disjoint clusters and in, general, with the categories that are not linearly separable.

IV.5.7 Neural Networks

Neural network (NN) can be built to perform text categorization. Usually, the input nodes of the network receive the feature values, the output nodes produce the categorization status values, and the link weights represent dependence relations. For classifying a document, its feature weights are loaded into the input nodes; the activation of the nodes is propagated forward through the network, and the final values on output nodes determine the categorization decisions.

The neural networks are trained by *back propagation*, whereby the training documents are loaded into the input nodes. If a misclassification occurs, the error is propagated back through the network, modifying the link weights in order to minimize the error.

The simplest kind of a neural network is a perceptron. It has only two layers – the input and the output nodes. Such network is equivalent to a linear classifier. More complex networks contain one or more *hidden* layers between the input and output layers. However, the experiments have shown very small – or no – improvement of nonlinear networks over their linear counterparts in the text categorization task (Schutze, Hull, and Pederson 1995; Wiener 1995).

IV.5.8 Example-Based Classifiers

Example-based classifiers do not build explicit declarative representations of categories but instead rely on directly computing the similarity between the document to be classified and the training documents. Those methods have thus been called lazy learners because they defer the decision on how to generalize beyond the training data until each new query instance is encountered. "Training" for such classifiers consists of simply storing the representations of the training documents together with their category labels.

The most prominent example of an example-based classifier is kNN (k-nearest neighbor). To decide whether a document d belongs to the category c, kNN checks whether the k training documents most similar to d belong to c. If the answer is positive for a sufficiently large proportion of them, a positive decision is made; otherwise, the decision is negative. The distance-weighted version of kNN is a variation that weighs the contribution of each neighbor by its similarity to the test document.

In order to use the algorithm, one must choose the value of k. It can be optimized using a validation set, but it is probable that a good value can be picked a priori.

Larkey and Croft (Larkey and Croft 1996) use $k = 20$, whereas Yang (Yang 2001) has found $30 \leq k \leq 45$ to yield the best effectiveness. Various experiments have shown that increasing the value of k does not significantly degrade the performance.

The kNN is one of the best-performing text classifiers to this day. It is robust in the sense of not requiring the categories to be linearly separated. Its only drawback is the relatively high computational cost of classification – that is, for each test document, its similarity to all of the training documents must be computed.

IV.5.9 Support Vector Machines

The support vector machine (SVM) algorithm is very fast and effective for text classification problems.

In geometrical terms, a binary SVM classifier can be seen as a hyperplane in the feature space separating the points that represent the positive instances of the category from the points that represent the negative instances. The classifying hyperplane is chosen during training as the unique hyperplane that separates the known positive instances from the known negative instances with the maximal margin. The margin is the distance from the hyperplane to the nearest point from the positive and negative sets. The diagram shown in Figure IV.2 is an example of a maximal margin hyperplane in two dimensions.

It is interesting to note that SVM hyperplanes are fully determined by a relatively small subset of the training instances, which are called the *support vectors*. The rest of the training data have no influence on the trained classifier. In this respect, the SVM algorithm appears to be unique among the different categorization algorithms.

The SVM classifier has an important advantage in its theoretically justified approach to the overfitting problem, which allows it to perform well irrespective of the dimensionality of the feature space. Also, it needs no parameter adjustment

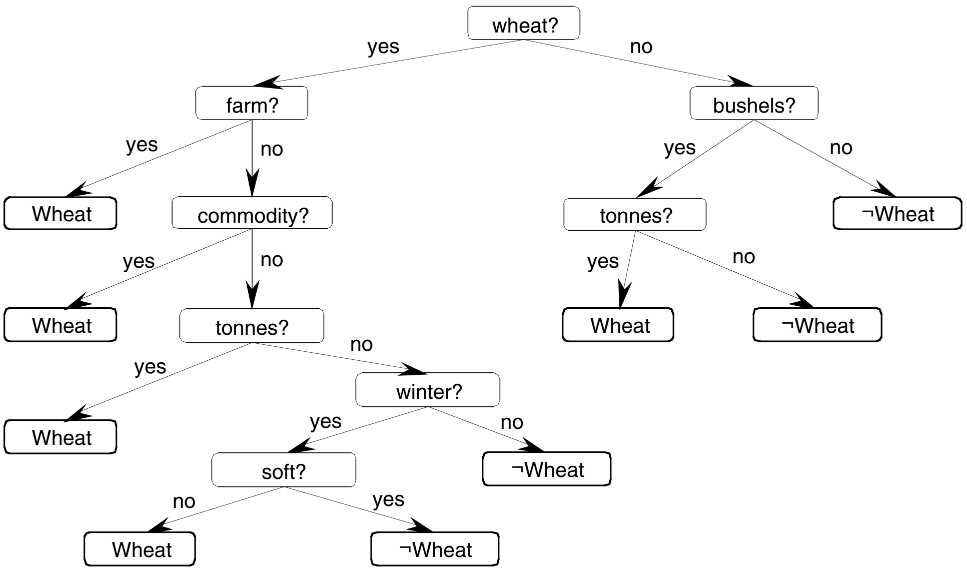

Figure IV.2. Diagram of a 2-D Linear SVM.

because there is a theoretically motivated "default" choice of parameters that has also been shown experimentally to provide the best performance.

IV.5.10 Classifier Committees: Bagging and Boosting

The idea of using committees of classifiers stems from the intuition that a team of experts, by combining their knowledge, may produce better results than a single expert alone. In the *bagging* method of building committees, the individual classifiers are trained in parallel on the same training collection. In order for the committee to work, the classifiers must differ significantly from each other – either in their document representation or in their learning methods. In text categorization, the latter method is usually chosen. As this chapter suggests, there is certainly no shortage of widely different learning methods.

Assume there are k different classifiers. To build a single committee classifier, one must choose the method of combining their results. The simplest method is the *majority vote* in which a category is assigned to a document iff at least $(k+1)/2$ classifiers decide this way (k must be an odd number, obviously). Another possibility, suited for continuous output, is the *weighted linear combination*, whereby the final CSV is given by a weighted sum of the CSVs of the k classifiers. The weights can be estimated on a validation dataset. Other methods of combining classifiers are also possible.

Boosting is another method of improving the quality of categorization by using several classifiers. Unlike the bagging method, in boosting the classifiers are trained sequentially. Before training the ith classifier, the training set is reweighed with greater weight given to the documents that were misclassified by the previous classifiers. The AdaBoost algorithm is the best known example of this approach. It is defined as follows:

Let X be the feature space, and let $D = \{(d_1, c_1), (d_2, c_3), \ldots\}$ be the training data, where $d_i \in X$ are the training document representations and $c_i \in \{+1, -1\}$ the category assignment (binary). A *weak learner* is some algorithm that is able to produce a *weak hypothesis* (classifier) $h : X \to \{\pm 1\}$ given the training data D together with a weight distribution W upon it. The "goodness" of a hypothesis is measured by its *error*

$$\varepsilon(h, W) = \sum_{i : h(d_i) \neq c_i} W(i),$$

which is the sum of weights of misclassified documents.

The AdaBoost algorithm

- Initializes weights distribution $W_1(i) = 1/|\mathrm{D}|$for all i, and
- Repeats for $t = 1, \ldots, k$.
  Train a weak classifier h_t using the current weights W_t.
  Let $\alpha_t = \frac{1}{2} \ln \frac{1-\varepsilon(h_t, W_t)}{\varepsilon(h_t, W_t)}$.

Update the weights: $W_{t+1}(i) = Z_t \cdot W_t(i) \cdot$

$$\begin{cases} \exp(-\alpha_t), & \text{if } h_t(d_i) = c_i, \\ \exp(\alpha_t), & \text{otherwise.} \end{cases}$$

(Z_t is the normalization factor chosen so that $\sum_i W_{t+1}(i) = 1$).

- The final classifier is $H(d) = \text{sign}\left(\sum_{t=1..k} \alpha_t h_t(d)\right)$.

It can be proved that, if the weak learner is able to generate classifiers with error $\varepsilon < \frac{1}{2} - \lambda$ for any fixed $\lambda > 0$ (which means, if the weak classifiers are any better than random), then the training error for the final classifier drops exponentially fast with the number k of algorithm steps. It can also be shown that AdaBoost has close relations with SVM, for it also maximizes the margin between training instances. Because of this, AdaBoost also has a similar resistance to overfitting.

IV.6 USING UNLABELED DATA TO IMPROVE CLASSIFICATION

All of the ML classifiers require fairly large training collections of preclassified documents. The task of manually labeling a large number of documents, although much less costly than manually creating a classification knowledge base, is still usually quite a chore. On the other hand, unlabeled documents usually exist in abundance, and any amount of them can be acquired with little cost. Therefore, the ability to improve the classifier performance by augmenting a relatively small number of labeled documents with a large number of unlabeled ones is very useful for applications. The two common ways of incorporating knowledge from unlabeled documents are expectation maximization (EM) and cotraining.

EM works with probabilistic generative classifiers such as Naïve Bayes. The idea is to find the most probable model given both labeled and unlabeled documents. The EM algorithm performs the optimization in a simple and appealing way:

- First, the model is trained over the labeled documents.
- Then the following steps are iterated until convergence in a local maximum occurs:

  E-step: the unlabeled documents are classified by the current model.

  M-step: the model is trained over the combined corpus.

In the M-step, the category assignments of the unlabeled documents are assumed to be fractional according to the probabilities produced by the E-step.

Cotraining works with the documents, for which two *views* are available, providing two different document representations, both of which are sufficient for classification. For example, a Web page may have its content as one view and the anchor text appearing in the hyperlinks to the page as another. In the domain of MedLine papers, the abstract may be one view and the whole text another. The cotraining is a bootstrapping strategy in which the unlabeled documents classified by means of one of the views are then used for training the classifier using the other view, and vice versa.

Both EM and cotraining strategies have experimentally shown a significant reduction (up to 60%) in the amount of labeled training data required to produce the same classifier performance.

IV.7 EVALUATION OF TEXT CLASSIFIERS

Because the text categorization problem is not sufficiently well-defined, the performance of classifiers can be evaluated only experimentally.

Any TC experiment requires a document collection labeled with a set of categories. This collection is divided into two parts: the *training* and *test* document sets. The training set, as the name suggests, is used for training the classifier, and the test set is the one on which the performance measures are calculated. Usually, the test set is the smaller of the two. It is very important not to use the test set in any way during the classifier training and fine-tuning. When there is a need to optimize some classifier parameters experimentally, the training set is further divided into two parts – the training set proper and a *validation* set, which is used for the parameter optimizations.

A commonly used method to smooth out the variations in the corpus is the *n-fold cross-validation*. In this method, the whole document collection is divided into n equal parts, and then the training-and-testing process is run n times, each time using a different part of the collection as the test set. Then the results for n folds are averaged.

IV.7.1 Performance Measures

The most common performance measures are the classic IR measures of *recall* and *precision*. A recall for a category is defined as the percentage of correctly classified documents among all documents belonging to that category, and precision is the percentage of correctly classified documents among all documents that were assigned to the category by the classifier.

Many classifiers allow trading recall for precision or vice versa by raising or lowering parameter settings or the output threshold. For such classifiers there is a convenient measure, called the *breakeven* point, which is the value of recall and precision at the point on the recall-versus-precision curve where they are equal. Alternatively, there is the F_1 measure, equal to 2/(1/recall + 1/precision), which combines the two measures in an ad hoc way.

IV.7.2 Benchmark Collections

The most known publicly available collection is the Reuters set of news stories, classified under economics-related categories. This collection accounts for most of the experimental work in TC so far. Unfortunately, this does not mean that the results produced by different researchers are directly comparable because of subtle differences in the experimental conditions.

In order for the results of two experiments to be directly comparable, the following conditions must be met:

(1) The experiments must be performed on exactly the same collection (meaning the same documents and same categories) using the same split between training and test sets.
(2) The same performance measure must be chosen.

(3) If a particular part of a system is compared, all other parts must be exactly the same. For instance, when comparing learning algorithms, the document representations must be the same, and when comparing the dimension reduction methods, the learning algorithms must be fixed together with their parameters.

These conditions are very difficult to meet – especially the last one. Thus, in practice, the only reliable comparisons are those done by the same researcher.

Other frequently used benchmark collections are the OHSUMED collection of titles and abstracts of papers from medical journals categorized with MESH thesaurus terms, 20 Newsgroups collection of messages posted to newsgroups with the newsgroups themselves as categories, and the TREC-AP collection of newswire stories.

IV.7.3 Comparison among Classifiers

Given the lack of a reliable way to compare classifiers across researchers, it is possible to draw only very general conclusions in reference to the question Which classifier is the best?

- According to most researchers, the top performers are SVM, AdaBoost, *k*NN, and Regression methods. Insufficient statistical evidence has been compiled to determine the best of these methods. Efficiency considerations, implementation complexity, and other application-related issues may assist in selecting from among these classifiers for specific problems.
- Rocchio and Naïve Bayes have the worst performance among the ML classifiers, but both are often used as baseline classifiers. Also, NB is very useful as a member of classifier committees.
- There are mixed results regarding the neural networks and decision tree classifiers. Some of the experiments have demonstrated rather poor performance, whereas in other experiments they performed nearly as well as SVM.

IV.8 CITATIONS AND NOTES

Section IV.1

Applications of text categorization are described in Hayes et al. (1988); Ittner, Lewis, and Ahn (1995); Larkey (1998); Lima, Laender, and Ribeiro-Neto (1998); Attardi, Gulli, and Sebastiani (1999); Drucker, Vapnik, and Wu (1999); Moens and Dumortier (2000); Yang, Ault, Pierce, and Lattimer (2000); Gentili et al. (2001); Krier and Zaccà (2002); Fall et al. (2003); and Giorgetti and Sebastiani (2003a, 2003b).

Section IV.2

For a general introduction to text categorization, refer to Sebastiani (2002) and Lewis (2000), which provides an excellent tutorial on the subject.

Section IV.3

Approaches that integrate linguistic and background knowledge into the categorization process can be found in Jacobs (1992); Rodriguez et al. (1997); Aizawa (2001); and Benkhalifa, Mouradi, and Bouyakhf (2001a, 2001b).

Section IV.5.3–IV.5.4

The following papers discuss how to use decision trees and decision lists for text categorization: Apte, Damerau, and Weiss (1994a, 1994b, 1994c); Li and Yamanishi (1999); Chen and Ho (2000); and Li and Yamanishi (2002).

Section IV.5.5

The use of regression for text categorization is discussed in Zhang and Oles (2001), Zhang et al. (2003), and Zhang and Yang (2003).

Section IV.5.8

The *k*NN algorithm is discussed and described in Yavuz and Guvenir (1998); Han, Karypis, and Kumar (2001); Soucy and Mineau (2001b); and Kwon and Lee (2003).

Section IV.5.9

The SVM algorithm is described and discussed in Vapnik (1995); Joachims (1998); Kwok (1998); Drucker, Vapnik, et al. (1999); Joachims (1999); Klinkenberg and Joachims (2000); Siolas and d'Alche-Buc (2000); Tong and Koller (2000); Joachims (2001); Brank et al. (2002); Joachims (2002); Leopold and Kindermann (2002); Diederich et al. (2003); Sun, Naing, et al. (2003); Xu et al. (2003); and Zhang and Lee (2003).

Section IV.5.10

Approaches that combine several algorithms by using committees of algorithms or by using boosting are described in Larkey and Croft (1996); Liere and Tadepalli (1997); Liere and Tadepalli (1998); Forsyth (1999); Ruiz and Srinivasan (1999a, 1999b); Schapire and Singer (2000); Sebastiani, Sperduti, and Valdambrini (2000); Al-Kofahi et al. (2001); Bao et al. (2001); Lam and Lai (2001); Taira and Haruno (2001); and Nardiello, Sebastiani, and Sperduti (2003).

Additional Algorithms

There are several *adaptive* (or *online*) algorithms that build classifiers incrementally without requiring the whole training set to be present at once. A simple *perceptron* is described in Schutze et al. (1995) and Wiener (1995). A WINNOW algorithm, which is a multiplicative variant of perceptron, is described in Dagan, Karov, and Roth (1997). Other online algorithms include WIDROW_HOFF, EXPONENTIATED GRADIENT (Lewis et al. 1996), and SLEEPING EXPERTS (Cohen and Singer 1999).

Relational and rule-based approaches to text categorization are discussed in Cohen (1992); Cohen (1995a, 1995b); and Cohen and Hirsh (1998).

Section IV.7

Comparisons between the categorization algorithms are discussed in Yang (1996) and Yang (1999).