16 *Text Categorization*

THIS CHAPTER both introduces an important NLP problem, text categorization, and provides a more general perspective on classification, including coverage of a number of important classification techniques that are not covered elsewhere in the book. *Classification* or *categorization* is the task of assigning objects from a universe to two or more *classes* or *categories*. Some examples are shown in table 16.1. Many of the tasks that we have already studied in detail, such as tagging, word sense disambiguation, and prepositional phrase attachment are classification tasks. In tagging and disambiguation, we look at a word in context and classify it as being an instance of one of its possible part of speech tags or an instance of one of its senses. In PP attachment, the two classes are the two different attachments. Two other NLP classification tasks are author and language identification. Determining whether a newly discovered poem was written by Shakespeare or by a different author is an example of author identification. A language identifier tries to pick the language that a document of unknown origin is written in (see exercise 16.6).

CLASSIFICATION
CATEGORIZATION
CLASSES
CATEGORIES

In this chapter, we will concentrate on another classification problem, *text categorization*. The goal in text categorization is to classify the topic or theme of a document. A typical set of topic categories is the one used in the Reuters text collection, which we will introduce shortly. Some of its topics are "mergers and acquisitions," "wheat," "crude oil," and "earnings reports." One application of text categorization is to filter a stream of news for a particular interest group. For example, a financial journalist may only want to see documents that have been assigned the category "mergers and acquisitions."

TEXT CATEGORIZATION

In general, the problem of statistical classification can be characterized as follows. We have a *training set* of objects, each labeled with one or

TRAINING SET

Problem	Object	Categories
tagging	context of a word	the word's tags
disambiguation	context of a word	the word's senses
PP attachment	sentence	parse trees
author identification	document	authors
language identification	document	languages
text categorization	document	topics

Table 16.1 Some examples of classification tasks in NLP. For each example, the table gives the type of object that is being classified and the set of possible categories.

DATA REPRESENTATION MODEL

more classes, which we encode via a *data representation model*. Typically each object in the training set is represented in the form $(\vec{x}, c)$, where $\vec{x} \in \mathbb{R}^n$ is a vector of measurements and c is the class label. For text categorization, the information retrieval vector space model is frequently used as the data representation. That is, each document is represented as a vector of (possibly weighted) word counts (see section 15.2). Finally, we define a *model class* and a *training procedure*. The model class is a parameterized family of classifiers and the training procedure selects one classifier from this family.[1] An example of such a family for binary classification is linear classifiers which take the following form:

MODEL CLASS

TRAINING PROCEDURE

$$g(\vec{x}) = \vec{w} \cdot \vec{x} + w_0$$

where we choose class c_1 for $g(\vec{x}) > 0$ and class c_2 for $g(\vec{x}) \leq 0$. This family is parameterized by the vector $\vec{w}$ and the threshold w_0.

We can think of training procedures as algorithms for function fitting, which search for a good set of parameter values, where 'goodness' is determined by an optimization criterion such as misclassification rate or entropy. Some training procedures are guaranteed to find the optimal set of parameters. However, many iterative training procedures are only guaranteed to find a better set in each iteration. If they start out in the wrong part of the search space, they may get stuck in a local optimum without ever finding the global optimum. An example of such a training procedure for linear classifiers is *gradient descent* or *hill climbing* which we will introduce below in the section on perceptrons.

GRADIENT DESCENT

HILL CLIMBING

1. Note however that some classifiers like nearest-neighbor classifiers are non-parametric and are harder to characterize in terms of a model class.

	YES is correct	NO is correct
YES was assigned	a	b
NO was assigned	c	d

Table 16.2 Contingency table for evaluating a binary classifier. For example, a is the number of objects in the category of interest that were correctly assigned to the category.

Once we have chosen the parameters of the classifier (or, as we usually say, *trained* the classifier), it is a good idea to see how well it is doing on a *test set*. This test set should consist of data that was not used during training. It is trivial to do well on data that the classifier was trained on. The real test is an evaluation on a representative sample of unseen data since that is the only measure that will tell us about actual performance in an application.

TEST SET

For binary classification, classifiers are typically evaluated using a table of counts like table 16.2. An important measure is classification *accuracy* which is defined as $\frac{a+d}{a+b+c+d}$, the proportion of correctly classified objects. Other measures are precision, $\frac{a}{a+b}$, recall, $\frac{a}{a+c}$, and fallout, $\frac{b}{b+d}$. See section 8.1.

ACCURACY

In classification tasks with more than two categories, one begins by making a 2×2 contingency table for each category c_i separately (evaluating c_i versus $\neg c_i$). There are then two ways to proceed. One can compute an evaluation measure like accuracy for each contingency table separately and then average the evaluation measure over categories to get an overall measure of performance. This process is called *macro-averaging*. Or one can do *micro-averaging*, in which one first makes a single contingency table for all the data by summing the scores in each cell for all categories. The evaluation measure is then computed for this large table. Macro-averaging gives equal weight to each category whereas micro-averaging gives equal weight to each object. The two types of averaging can give divergent results when precision is averaged over categories with different sizes. Micro-averaged precision is dominated by the large categories whereas macro-averaged precision will give a better sense of the quality of classification across all categories.

MACRO-AVERAGING

MICRO-AVERAGING

In this chapter we describe four classification techniques: decision trees, maximum entropy modeling, perceptrons, and k nearest neighbor classification. These are either important classification techniques

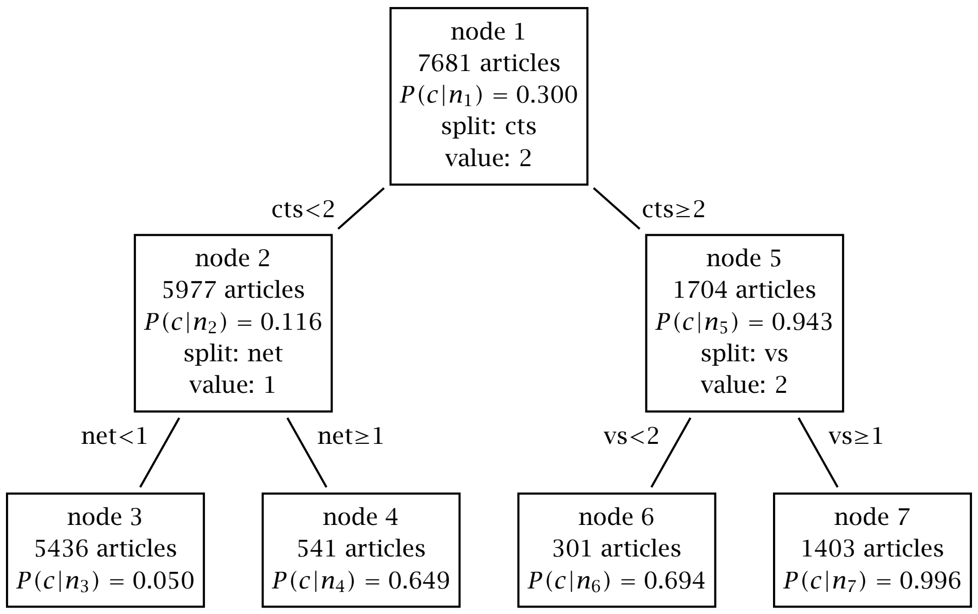

Figure 16.1 A decision tree. This tree determines whether a document is part of the topic category "earnings" or not. $P(c|n_i)$ is the probability of a document at node n_i to belong to the "earnings" category c.

in their own right, or, in the case of the perceptron, are the simplest example of an important class of techniques, neural networks. We conclude with some pointers to further reading.

16.1 Decision Trees

DECISION TREES

As the first class of classification models, we introduce *decision trees*. An example of a decision tree is shown in figure 16.1. This tree decides whether to assign documents to the Reuters category "earnings." We classify a document by starting at the top node, testing its question, branching to the appropriate node, and then repeating this process until we reach a leaf node. For example, a document with weight 1 for *cts* and weight 3 for *net* takes the left branch at the top node and then the right branch at the child node. Its probability $P(c|n_4)$ of being in the category "earnings" given that it belongs to node 4 is then estimated as 0.649. At each node, we show the number of articles in the training set that belong

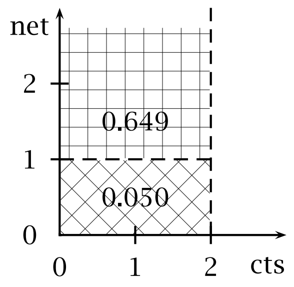

Figure 16.2 Geometric interpretation of part of the tree in figure 16.1.

to the node, the probability of a member of the node being in the category "earnings," the word (or dimension) we split on at this node, and the weight of the word we split on.

Another way to visualize the tree is shown in figure 16.2. The horizontal axis corresponds to the weight for *cts*, the vertical axis to the weight for *net*. Questions ask whether the value of some feature is less than some value or not. The top node in figure 16.1 defines a decision boundary corresponding to the vertical line "*cts* = 2" in figure 16.2. The left child node subdivides the left region into two regions above and below "*net* = 1." The upper subregion (marked with $P(c|n) = 0.649$) corresponds to node 4, the lower subregion to node 4. Note that the region to the right of the decision boundary "*cts* = 2" is not further subdivided because node 5 splits on *vs*, not on *net*. We would need a three-dimensional graph to also show the effect of node 5.

The text categorization task that we use as an example in this chapter is to build categorizers that distinguish the "earnings" category in the REUTERS *Reuters* collection. The Reuters collection is currently the most popular database for evaluating text categorization research. The version we use (based on the so-called Modified Apte Split, Apté et al. 1994) consists of 9603 training articles and 3299 test articles that were sent over the Reuters newswire in 1987. The articles are categorized with more than 100 topics such as "mergers and acquisitions" and "interest rates." An example of an article in this category is shown in figure 16.3. See the website for references to the Reuters collection.

The first task in text categorization is to find an appropriate data rep-

```
<REUTERS NEWID="11">
<DATE>26-FEB-1987 15:18:59.34</DATE>
<TOPICS><D>earn</D></TOPICS>
<TEXT>
<TITLE>COBANCO INC &lt;CBCO> YEAR NET</TITLE>
<DATELINE>    SANTA CRUZ, Calif., Feb 26 - </DATELINE>
<BODY>Shr 34 cts vs 1.19 dlrs
    Net 807,000 vs 2,858,000
    Assets 510.2 mln vs 479.7 mln
    Deposits 472.3 mln vs 440.3 mln
    Loans 299.2 mln vs 327.2 mln
    Note: 4th qtr not available. Year includes 1985
extraordinary gain from tax carry forward of 132,000 dlrs,
or five cts per shr.
 Reuter
</BODY></TEXT>
</REUTERS>
```

Figure 16.3 An example of a Reuters news story in the topic category "earnings." Parts of the original have been omitted for brevity.

resentation model. This is an art in itself, and usually depends on the particular categorization method used, but to simplify things, we will use a single data representation throughout this chapter. It is based on the 20 words whose X^2 score with the category "earnings" in the training set was highest (see section 5.3.3 for the X^2 measure). The words *loss*, *profit*, and *cts* (for "cents"), all three of which seem obvious as good indicators for an earnings report, are some of the 20 words that were selected. Each document was then represented as a vector of $K = 20$ integers, $\vec{x}_j = (s_{1j}, \ldots, s_{Kj})$, where s_{ij} was computed as the following quantity:

$$s_{ij} = \text{round}\left(10 \times \frac{1 + \log(tf_{ij})}{1 + \log(l_j)}\right) \tag{16.1}$$

Here, tf_{ij} is the number of occurrences of term i in document j and l_j is the length of document j. The score s_{ij} is set to 0 for no occurrences of the term. So for example, if *profit* occurs 6 times in a document of length 89 words, then the score for *profit* would be $s_{ij} = 10 \times \frac{1+\log(6)}{1+\log(89)} \approx 5.09$

Word w^j		Term weight s_{ij}	Classification
vs		5	
mln		5	
cts		3	
;		3	
&		3	
000		4	
loss		0	
’		0	
"		0	
3	$\vec{x} =$	4	$c = 1$
profit		0	
dlrs		3	
1		2	
pct		0	
is		0	
s		0	
that		0	
net		3	
lt		2	
at		0	

Table 16.3 The representation of document 11, shown in figure 16.3. This illustrates the data representation model which we use for classification in this chapter.

which would be rounded to 5. This weighting scheme does log weighting similar to the schemes discussed in chapter 15, while at the same time incorporating weight normalization. We round values to make it easier to present and inspect data for pedagogical reasons.

The representation of the document in figure 16.3 is shown in table 16.3. As tends to happen when using an automatic feature selection method, some of the selected words don’t seem promising as indicators of “earnings,” for example, *that* and *s*. The three symbols “&”, “lt”, and “;” were selected because of a formatting peculiarity in the publicly available Reuters collection: a large proportion of articles in the category “earnings” have a company tag like <CBCO> in the title line whose left angle bracket was converted to an SGML character entity. We can think of this

entropy at node 1, $P(C\|N) = 0.300$	0.611
entropy at node 2, $P(C\|N) = 0.116$	0.359
entropy at node 5, $P(C\|N) = 0.943$	0.219
weighted sum of 2 and 5	$\frac{5977}{7681} \times 0.359 + \frac{1704}{7681} \times 0.219 = 0.328$
information gain	$0.611 - 0.328 = 0.283$

Table 16.4 An example of information gain as a splitting criterion. The table shows the entropies for nodes 1, 2, and 5 in figure 16.1, the weighted sum of the child nodes and the information gain for splitting 1 into 2 and 5.

left angle bracket as indicating: "This document is about a particular company." We will see that this 'meta-tag' is very helpful for classification. The title line of the document in figure 16.3 has an example of the meta-tag.[2]

Now that we have a model class (decision trees) and a representation for the data (20-element vectors), we need to define the training procedure. Decision trees are usually built by first growing a large tree and then *pruning* it back to a reasonable size. The pruning step is necessary because very large trees *overfit* the training set. Overfitting occurs when classifiers make decisions based on accidental properties of the training set that will lead to errors on the test set (or any new data). For example, if there is only one document in the training set that contains both the words *dlrs* and *pct* (for "dollars" and "percent") and this document happens to be in the earnings category, then the training procedure may grow a large tree that categorizes all documents with this property as being in this category. But if there is only one such document in the training set, this is probably just a coincidence. When the tree is pruned back, then the part that makes the corresponding inference (assign to "earnings" if one finds both *dlrs* and *pct*), will be cut off, thus leading to better performance on the test set.

PRUNING

OVERFITTING

For growing the tree, we need a *splitting criterion* for finding the feature and its value that we will split on and a *stopping criterion* which determines when to stop splitting. The stopping criterion can trivially be that all elements at a node have an identical representation or the same category so that splitting would not further distinguish them.

SPLITTING CRITERION

STOPPING CRITERION

The splitting criterion which we will use here is to split the objects at a

2. The string "&lt;" should really have been tokenized as a unit, but it can serve as an example of the low-level data problems that occur frequently in text categorization.

node into two piles in the way that gives us maximum information gain. INFORMATION GAIN *Information gain* (Breiman et al. 1984: 25, Quinlan 1986: section 4, Quinlan 1993) is an information-theoretic measure defined as the difference of the entropy of the mother node and the weighted sum of the entropies of the child nodes:

$$G(a, y) = H(\mathrm{t}) - H(\mathrm{t}|a) = H(\mathrm{t}) - \left(\mathrm{p}_L H(\mathrm{t}_L) + \mathrm{p}_R H(\mathrm{t}_R)\right) \qquad (16.2)$$

where a is the attribute we split on, y is the value of a we split on, t is the distribution of the node that we split, p_L and p_R are the proportion of elements that are passed on to the left and right nodes, and t_L and t_R are the distributions of the left and right nodes. As an example, we show the values of these variables and the resulting information gain in table 16.4 for the top node of the decision tree in figure 16.1.

Information gain is intuitively appealing because it can be interpreted as measuring the reduction of uncertainty. If we make the split that maximizes information gain, then we reduce the uncertainty in the resulting classification as much as possible. There are no general algorithms for finding the optimal splitting value efficiently. In practice, one uses heuristic algorithms that find a near-optimal value.[3]

A node that is not split by the algorithm because of the stopping criterion is a LEAF NODE *leaf node*. The prediction we make at a leaf node is based on its members. We can compute maximum likelihood estimates, but smoothing is often appropriate. For example, if a leaf node has 6 members in the category "earnings" and 2 other members, then we would estimate the category membership probability of a new document d in the node as $P(\text{earnings}|d) = \frac{6+1}{2+6+1+1} = 0.7$ if we use add-one smoothing (section 6.2.2).

Once the tree has been fully grown, we prune it to avoid overfitting and to optimize performance on new data. At each step, we select the remaining leaf node that we expect by some criterion to be least helpful (or even harmful) for accurate classification. One common pruning criterion is to compute a measure of confidence that indicates how much evidence there is that the node is 'helpful' (Quinlan 1993). We repeat the pruning process until no node is left. Each step in the process (from the full tree to the empty tree) defines a classifier – the classifier that corresponds to the decision tree with the nodes remaining at this point. One way to se-

3. We could afford an exhaustive search for the optimal splitting value here because the s_{ij} are integers in a small interval.

lect the best of these n trees (where n is the number of internal nodes of the full tree) is by *validation* on held out data.

VALIDATION

VALIDATION SET

Validation evaluates a classifier on a held-out data set, the *validation set* to assess its accuracy. For the same reason as needing independent test data, in order to evaluate how much to prune a decision tree we need to look at a new set of data – which is what evaluation on the validation set does. (See section 6.2.3 for the same basic technique used in smoothing.)

An alternative to pruning a decision tree is to keep the whole tree, but to make the classifier probability estimates a function of internal as well as leaf nodes. This is a means of getting at the more reliable probability distributions of higher nodes without actually pruning away the nodes below them. For each leaf node, we can read out the sequence of nodes and associated probability distributions from that node to the root of the decision tree. Rather than using held out data for pruning, held out data can be used to train the parameters of a linear interpolation (section 6.3.1) of all these distributions for each leaf node, and these interpolated distributions can then be used as the final classification functions. Magerman (1994) argues that this gives superior performance to pruning, at least for the statistical parsing problem on which he was working (see section 12.2.2).

Figure 16.4 shows how performance of a decision tree depends on pruning. The x-axis corresponds to number of nodes pruned, the y-axis to classification accuracy. In order to produce the graph we grew the tree on 80% of the training set (7681 documents) and set 20% (1922 documents) aside as the validation set. The pruning of the top node is not shown in the graph.[4]

The pattern we find is standard: performance on the training set is maximum for the full tree and then falls off continuously. Since we optimize the initial construction of the full tree on the training set, larger trees will fit the properties of the training set better than pruned trees, hence the decrease in performance on the training set as we go from left to right.

Accuracy for validation and test sets peaks somewhere in the middle. When performance peaks we have reached the point where parts of the tree that fit accidental properties of the training set have been pruned

4. The pruning criterion was to select the leaf node with the lowest information gain on the validation set.

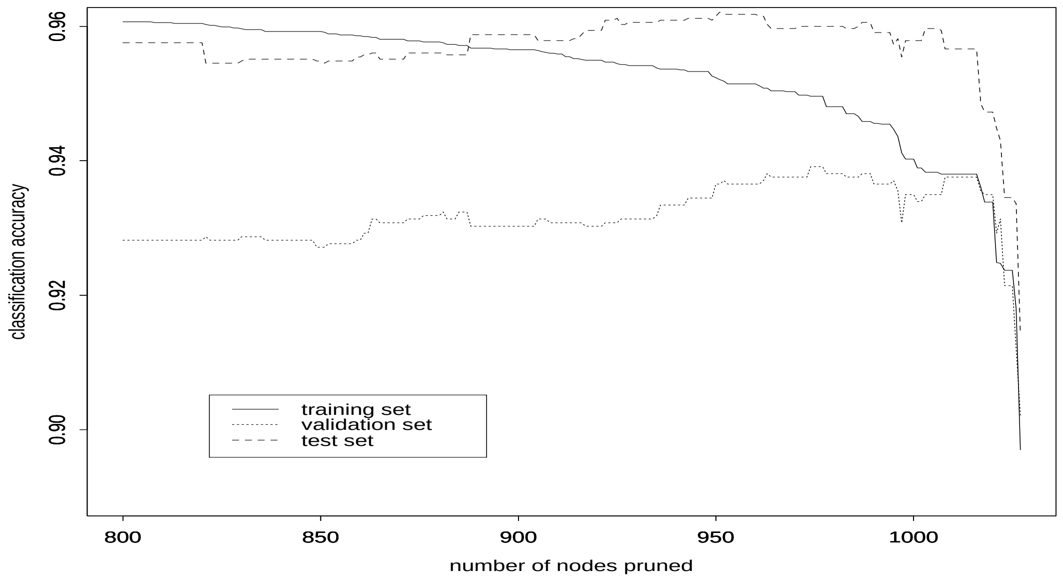

Figure 16.4 Pruning a decision tree. The graph shows how classification accuracy of a decision tree depends on pruning. Optimal performance on the test set (96.21% accuracy) is reached for 951 nodes pruned. Optimal performance on the validation set (93.91% accuracy) is reached for 974–977 nodes pruned. For these four pruned trees, performance on the test set is 96.00%, close to optimal performance. Performance on the training set is monotonically decreasing.

away. Oversimplifying a little bit, any further pruning will delete nodes that capture correct generalizations about the "earnings" category, hence the decrease in performance.

One strategy for selecting a tree is to pick the one that performs best for the validation set. As we can see in the picture it does not perfectly coincide with the peak for the test set, but it is close enough. The tree that performs best on the validation set will often be slightly overtrained or slightly undertrained, but usually it will be close to optimal performance.

Table 16.5 evaluates performance of the tree with 50 internal nodes on the test set. This is the smallest tree with the optimal performance of 93.91% accuracy on the validation set.

A problem with setting aside a validation set is that a relatively large part of the full training set is wasted. A better method is to use *n*-fold

CROSS-VALIDATION *cross-validation* (cf. section 6.2.4) to estimate a good size for the pruned

"earnings"	"earnings" correct?	
assigned?	YES	NO
YES	1024	69
NO	63	2143

Table 16.5 Contingency table for a decision tree for the Reuters category "earnings." Classification accuracy on the test set is 96.0%.

decision tree. For example, in 5-fold cross-validation we split the data into five parts. We reserve one part as a validation set, train the tree on the other four parts and then prune it back based on the held-out part. This process is repeated four times using each of the other four parts as a validation set. We then determine the average size of an optimally performing pruned tree. Finally, a new tree is grown for the *entire* training set and pruned back to what we have calculated to be the optimal size.

The interdependence of complexity of the learning device and accuracy on the training set is an important characteristic of many classification methods. If the device is too complex (or has too many parameters), then we risk overfitting and low accuracy on new data. If the device is not complex enough, it is not able to make maximum use of the training data, which again leads to lower than optimal accuracy on new data. The trick is to find just the right balance and cross-validation is one approach to doing that.

Another common property of classification methods is shown in figure 16.5, the dependence of classification accuracy on the amount of training data available. Not surprisingly, the more training data, the better – up to a point where performance improvement levels off. Sometimes one gets lucky with a small set (hence the fluctuations), but one cannot be sure that a tree trained on a small data set will perform well.

LEARNING CURVES

Computing *learning curves* like those in figure 16.5 is important to determine the size of an appropriate training set. Many training procedures are computationally expensive, so it is advantageous to avoid overly large training sets. But on the other hand, insufficient training data will result in suboptimal classification accuracy. Looking at the curve lets one decide how much data is enough for optimal performance. (Of course, there are many cases in which one has no control over the amount of training data available and has to live with a small training set even though a larger one would give much better performance.)

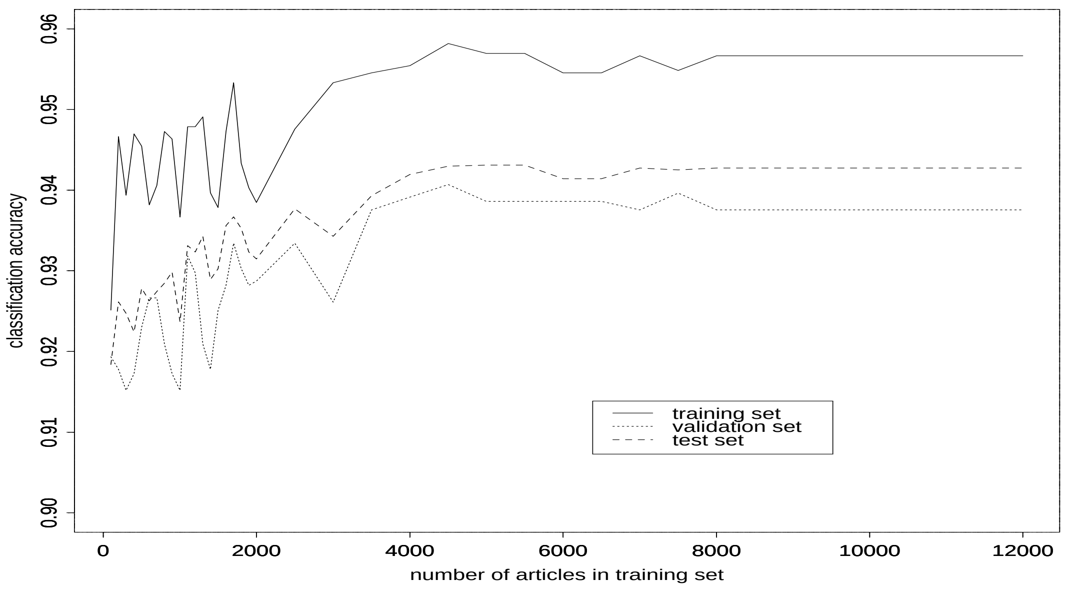

Figure 16.5 Classification accuracy depends on the amount of training data available. The *x* axis corresponds to the number of training documents the decision tree was trained on. The *y* axis corresponds to accuracy on the test set for a decision tree selected based on a constant size validation set. Classification accuracy is highly variable for small training set sizes and increases and levels off for larger sets.

When are decision trees appropriate for a classification task in NLP? Decision trees are more complex than classifiers like Naive Bayes (section 7.2.1), linear regression (section 15.4.1), and logistic regression. If the classification problem is simple (in particular, if it is *linearly separable*, see below), then these simpler methods are often preferable. Decision trees also split the training set into smaller and smaller subsets. This makes correct generalization harder, since there may not be enough data for reliable prediction, and incorrect generalization easier, since smaller sets have accidental regularities that don't generalize. Figure 16.6 gives a simple example of this problem from the domain of learning phonological rules. Pruning addresses this problem to some extent, but some learning problems are better handled by methods that look at all features simultaneously. The other three methods we introduce in this chapter all have this property.

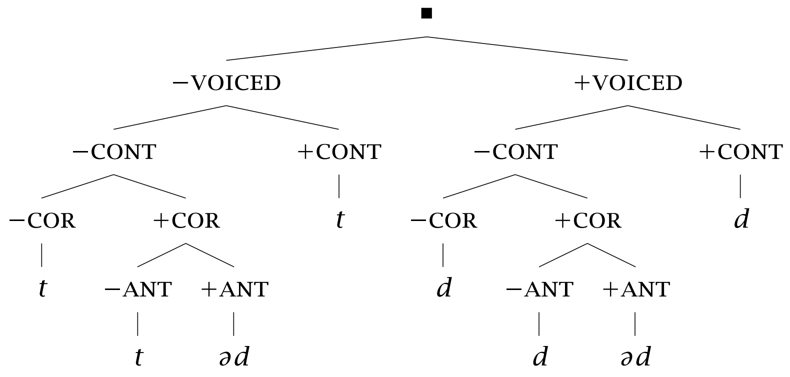

Figure 16.6 An example of how decision trees use data inefficiently from the domain of phonological rule learning. The regular rule for the English past tense is that one gets /t/ after a voiceless sound, and /d/ after a voiced sound, except after [−CONT, +COR, +ANT] sounds (i.e., /t/, /d/) where one gets /əd/. Because the voicing feature has the greatest information gain, the tree splits on that feature first, but that makes the remaining conditioning harder to learn because the relevant data has been subdivided into different bins within which learning is attempted independently.

The greatest advantage of decision trees is that they can be interpreted so easily. It is easy to trace the path from the root to a leaf node for a couple of articles and to develop an intuition as to how the decision tree works. This is not only invaluable in debugging one's own code and understanding a new problem domain, but it also allows one to explain the classifier to researchers and laymen alike, an important property in research collaboration and practical applications.

Exercise 16.1 [⋆]

What is the classification accuracy of the trivial tree with one leaf node for the "earnings" category?

Exercise 16.2 [⋆]

In section 7.1, we introduced upper and lower bounds as a way to assess how hard a particular classification problem is. What are upper and lower bounds for the 'earnings' category?

Exercise 16.3 [⋆⋆]

An important application of text categorization is the detection of spam (also known as unsolicited bulk email). Try to collect at least a hundred spam messages and non-spam messages, divide them into training and test sets and build a decision tree that detects spam. Finding the right features is paramount for this task, so design your feature set carefully.

Exercise 16.4 [⋆⋆]

Another important application of text categorization is the detection of 'adult' content, that is, content that is not appropriate for children because it is sexually explicit. Collect training and test sets of adult and non-adult material from the World Wide Web and build a decision tree that can block access to adult material.

Exercise 16.5 [⋆⋆]

Collect a reasonable amount of text written by yourself and by a friend. You may want to break up individual texts (e.g., term papers) into smaller pieces to get a large enough set. Build a decision tree that automatically determines whether you are the author of a piece of text. Note that it is often the 'little' words that give an author away (for example, the relative frequencies of words like *because* or *though*).

Exercise 16.6 [⋆⋆]

Download a set of English and non-English texts from the World Wide Web or use some other multilingual source. Build a decision tree that can distinguish between English and non-English texts. (See also exercise 6.10.)

16.2 Maximum Entropy Modeling

Maximum entropy modeling is a framework for integrating information from many heterogeneous information sources for classification. The data for a classification problem is described as a (potentially large) number of features. These features can be quite complex and allow the experimenter to make use of prior knowledge about what types of information are expected to be important for classification. Each feature corresponds to a constraint on the model. We then compute the *maximum entropy model*, the model with maximum entropy of all the models that satisfy the constraints. This term may initially seem perverse, since we have spent most of the book trying to minimize the (cross) entropy of models, but the idea is that we do not want to go beyond the data. If we chose a model with less entropy, we would add 'information' constraints to the model that are not justified by the empirical evidence available to us. Choosing the maximum entropy model is motivated by the desire to preserve as much uncertainty as possible.

We have simplified matters in this chapter by neglecting the problem of feature selection (we use the same 20 features throughout). In maximum entropy modeling, feature selection and training are usually integrated.

Ideally, this enables us to specify all potentially relevant information at the beginning, and then to let the training procedure worry about how to come up with the best model for classification. We will only introduce the basic method here and refer the reader to the Further Reading for feature selection.

For a given set of features, we first compute the *expectation* of each feature based on the training set. Each feature then defines the constraint that this *empirical expectation* be the same as the expectation the feature has in our final maximum entropy model. Of all probability distributions that obey these constraints, we attempt to find the *maximum entropy distribution*, the one with the highest entropy. One can show that there is a unique such maximum entropy distribution and there exists an algorithm, generalized iterative scaling, which is guaranteed to converge to it.

EMPIRICAL EXPECTATION

MAXIMUM ENTROPY DISTRIBUTION

The features f_i are binary functions that can be used to characterize any property of a pair $(\vec{x}, c)$, where $\vec{x}$ is a vector representing an input element (in our case the 20-dimensional vector of word weights representing an article as in table 16.3), and c is the class label (1 if the article is in the “earnings” category, 0 otherwise). For text categorization, we define features as follows:

$$f_i(\vec{x}_j, c) = \begin{cases} 1 & \text{if } s_{ij} > 0 \text{ and } c = 1 \\ 0 & \text{otherwise} \end{cases} \tag{16.3}$$

Recall that s_{ij} is the term weight for word i in Reuters article j. Note that the use of binary features is different from the rest of this chapter: The other classifiers use the magnitude of the weight, not just the presence or absence of a word.[5]

The model class for the particular variety of maximum entropy modeling that we introduce here is *loglinear models* of the following form:

LOGLINEAR MODELS

$$\mathrm{p}(\vec{x}, c) = \frac{1}{Z} \prod_{i=1}^{K} \alpha_i^{f_i(\vec{x}, c)} \tag{16.4}$$

where K is the number of features, α_i is the weight for feature f_i and Z is a normalizing constant, used to ensure that a probability distribution results. To use the model for text categorization, we compute $\mathrm{p}(\vec{x}, 0)$ and

5. While the maximum entropy approach is not in principle limited to binary features, known reasonably efficient solution procedures such as generalized iterative scaling, which we introduce below, do only work for binary features.

$p(\vec{x}, 1)$ and, in the simplest case, choose the class label with the greater probability.

FEATURES

Note that, in this section, *features* contain information about the *class* of the object in addition to the *'measurements'* of the object we want to classify. Here, we are following most publications on maximum entropy modeling in defining feature in this sense. The more common use of the term "feature" (which we have adopted for the rest of the book) is that it only refers to some characteristic of the object, independent of the class the object is a member of.

Equation (16.4) defines a loglinear model because, if we take logs on both sides, then $\log p$ is a linear combination of the logs of the weights:

$$\log p(\vec{x}, c) = -\log Z + \sum_{i=1}^{K} f_i(\vec{x}, c) \log \alpha_i \qquad (16.5)$$

Loglinear models are an important class of models for classification. Other examples of the class are logistic regression (McCullagh and Nelder 1989) and decomposable models (Bruce and Wiebe 1999). We introduce the maximum entropy modeling approach here because maximum entropy models have been the most widely used loglinear models in Statistical NLP and because it is an application of the important maximum entropy principle.

16.2.1 Generalized iterative scaling

GENERALIZED ITERATIVE SCALING

Generalized iterative scaling is a procedure for finding the maximum entropy distribution p^* of form (16.4) that obeys the following set of constraints:

$$E_{p^*} f_i = E_{\tilde{p}} f_i \qquad (16.6)$$

In other words, the expected value of f_i for p^* is the same as the expected value for the empirical distribution (in other words, the training set).

The algorithm requires that the sum of the features for each possible $(\vec{x}, c)$ be equal to a constant C:[6]

$$\forall \vec{x}, c \quad \sum_i f_i(\vec{x}, c) = C \qquad (16.7)$$

6. See Berger et al. (1996) for *Improved Iterative Scaling*, a variant of generalized iterative scaling that does not impose this constraint.

In order to fulfil this requirement, we define C as the greatest possible feature sum:

$$C \stackrel{\text{def}}{=} \max_{\vec{x},c} \sum_{i=1}^{K} f_i(\vec{x}, c)$$

and add a feature f_{K+1} that is defined as follows:

$$f_{K+1}(\vec{x}, c) = C - \sum_{i=1}^{K} f_i(\vec{x}, c)$$

Note that this feature is not binary, in contrast to the others.

$E_{\mathrm{p}} f_i$ is defined as follows:

(16.8) $$E_{\mathrm{p}} f_i = \sum_{\vec{x},c} \mathrm{p}(\vec{x}, c) f_i(\vec{x}, c)$$

where the sum is over the event space, that is, all possible vectors $\vec{x}$ and class labels c. The empirical expectation is easy to compute:

(16.9) $$E_{\tilde{\mathrm{p}}} f_i = \sum_{\vec{x},c} \tilde{\mathrm{p}}(\vec{x}, c) f_i(\vec{x}, c) = \frac{1}{N} \sum_{j=1}^{N} f_i(\vec{x}_j, c)$$

where N is the number of elements in the training set and we use the fact that the empirical probability for a pair that doesn't occur in the training set is 0.

In general, the maximum entropy distribution $E_{\mathrm{p}} f_i$ cannot be computed efficiently since it would involve summing over all possible combinations of $\vec{x}$ and c, a potentially infinite set. Instead, we use the following approximation (Lau 1994: 25):

(16.10) $$E_{\mathrm{p}} f_i = \frac{1}{N} \sum_{j=1}^{N} \sum_{c} \mathrm{p}(c|\vec{x}_j) f_i(\vec{x}_j, c)$$

where c ranges over all possible classes, in our case $c \in \{0, 1\}$.

Now we have all the pieces to state the generalized iterative scaling algorithm:

- Initialize $\{\alpha_i^{(1)}\}$. Any initialization will do, but usually we choose $\alpha_i^{(1)} = 1, \forall 1 \leq j \leq K+1$. Compute $E_{\tilde{\mathrm{p}}} f_i$ as shown above. Set $n = 1$.
- Compute $\mathrm{p}^{(n)}(\vec{x}, c)$ for the distribution $\mathrm{p}^{(n)}$ given by the $\{\alpha_i^{(n)}\}$ for each element $(\vec{x}, c)$ in the training set:

(16.11) $$\mathrm{p}^{(n)}(\vec{x}, c) = \frac{1}{Z} \prod_{i=1}^{K+1} (\alpha_i^{(n)})^{f_i(x,c)}$$

$\vec{x}$	c				
profit	"earnings"	f_1	f_2	$\beta = f_1 \log \alpha_1 + f_2 \log \alpha_2$	2^β
(0)	0	0	1	1	2
(0)	1	0	1	1	2
(1)	0	0	1	1	2
(1)	1	1	0	2	4

Table 16.6 An example of a maximum entropy distribution in the form of equation (16.4). The vector $\vec{x}$ consists of a single element, indicating the presence or absence of the word *profit* in the article. There are two classes (member of "earnings" or not). Feature f_1 is 1 if and only if the article is in "earnings" and *profit* occurs. f_2 is the "filler" feature f_{K+1}. For one particular choice of the parameters, namely $\log \alpha_1 = 2.0$ and $\log \alpha_2 = 1.0$, we get after normalization ($Z = 2 + 2 + 2 + 4 = 10$) the following maximum entropy distribution: $\mathrm{p}(0,0) = \mathrm{p}(0,1) = \mathrm{p}(1,0) = 2/Z = 0.2$ and $\mathrm{p}(1,1) = 4/Z = 0.4$. An example of a data set with the same empirical distribution is $((0,0),(0,1),(1,0),(1,1),(1,1))$.

- Compute $E_{\mathrm{p}^{(n)}} f_i$ for all $1 \leq i \leq K + 1$ according to equation (16.10).
- Update the parameters α_i:

(16.12)
$$\alpha_i^{(n+1)} = \alpha_i^{(n)} \left(\frac{E_{\tilde{\mathrm{p}}} f_i}{E_{\mathrm{p}^{(n)}} f_i} \right)^{\frac{1}{C}}$$

- If the parameters of the procedure have converged, stop, otherwise increment n and go to 2.

We present the algorithm in this form for readability. In an actual implementation, it is more convenient to do the computations using logarithms.

One can show that this procedure converges to a distribution p^* that obeys the constraints (16.6), and that of all such distributions it is the one that maximizes the entropy $H(\mathrm{p})$ and the likelihood of the data. Darroch and Ratcliff (1972) show that this distribution always exists and is unique.

A toy example of a maximum entropy distribution that generalized iterative scaling will converge to is shown in table 16.6.

Exercise 16.7 [⋆]

What are the classification decisions for the distribution in table 16.6? Compute $P(\text{"earnings"}|\textit{profit})$ and $P(\text{"earnings"}|\neg\textit{profit})$.

Does *profit* occur?	Is topic "earnings"? YES	NO
YES	20	9
NO	8	13

Table 16.7 An empirical distribution whose corresponding maximum entropy distribution is the one in table 16.6.

Exercise 16.8 [★]

Show that the distribution in table 16.6 is a fixed point for iterative generalized scaling. That is, computing one iteration should leave the distribution unchanged.

Exercise 16.9 [★]

Consider the distribution in table 16.7. Show that for the features defined in table 16.6, this distribution has the same feature expectations E_{p} as the one in table 16.6.

Exercise 16.10 [★]

Compute a number of iterations of generalized iterative scaling for the data in table 16.7 (using the features defined in table 16.6). The procedure should converge towards the distribution in table 16.6.

Exercise 16.11 [★★]

Select one of exercises 16.3 through 16.6 and build a maximum entropy model for the corresponding text categorization task.

16.2.2 Application to text categorization

We have already suggested how to define appropriate features for text categorization in equation (16.3). For the task of identifying Reuters "earnings" articles we end up with 20 features, each corresponding to one of the selected words, and the f_{K+1} feature introduced at the start of the last subsection.

Table 16.8 shows the weights found by generalized iterative scaling after convergence (500 iterations). We trained on the 9603 articles in the training set. The features with the highest weights are *vs*, *cts*, *profit* and *lt*. If we use $P(\text{“earnings”}|\vec{x}) > P(\neg\text{“earnings”}|\vec{x})$ as our decision rule, we get the classification results in table 16.9. Classification accuracy is 88.6%.

An important question in an implementation is when to stop the iteration. One way to test for convergence is to compute the log difference

Word w^i	Feature weight $\log \alpha_i$
vs	0.613
mln	−0.110
cts	1.298
;	−0.432
&	−0.429
000	−0.413
loss	−0.332
'	−0.085
"	0.202
3	−0.463
profit	0.360
dlrs	−0.202
1	−0.211
pct	−0.260
is	−0.546
s	−0.490
that	−0.285
net	−0.300
lt	1.016
at	−0.465
f_{K+1}	0.009

Table 16.8 Feature weights in maximum entropy modeling for the category "earnings" in Reuters.

"earnings"	"earnings" correct?	
assigned?	YES	NO
YES	735	24
NO	352	2188

Table 16.9 Classification results for the distribution corresponding to table 16.8 on the test set. Classification accuracy is 88.6%.

between empirical and estimated feature expectations ($\log E_{\tilde{p}} - \log E_{p^{(n)}}$), which should approach zero. Ristad (1996) recommends to also look at the largest α when doing iterative scaling. If the largest weight becomes too large, then this indicates a problem with either the data representation or the implementation.

When is the maximum entropy framework presented in this section appropriate as a classification method? The somewhat lower performance on the "earnings" task compared to some of the other methods indicates one characteristic that is a shortcoming in some situations: the restriction to binary features seems to have led to lower performance. In text categorization, we often need a notion of "strength of evidence" which goes beyond a simple binary feature recording presence or absence of evidence. The feature-based representation we use for maximum entropy modeling is not optimal for this purpose.

Generalized iterative scaling can also be computationally expensive due to slow convergence (but see (Lau 1994) for suggestions for speeding up convergence). For binary classification, the loglinear model defines a linear separator that is in principle no more powerful than *Naive Bayes* or *linear regression*, classifiers that can be trained more efficiently. However, it is important to stress that, apart from the theoretical power of a classification method, the training procedure is crucial. Generalized iterative scaling takes dependence between features into account in contrast to Naive Bayes and other linear classifiers. If feature dependence is not expected to a be a problem, then Naive Bayes is a better choice than maximum entropy modeling.

NAIVE BAYES
LINEAR REGRESSION

Finally, the lack of smoothing can also cause problems. For example, if we have a feature that always predicts a certain class, then this feature may get an excessively high weight. One way to deal with this is to 'smooth' the empirical data by adding events that did not occur. In practice, features that occur less than five times are usually eliminated.

One of the strengths of maximum entropy modeling is that it offers a framework for specifying all possibly relevant information. The attraction of the method lies in the fact that arbitrarily complex features can be defined if the experimenter believes that these features may contribute useful information for the classification decision. For example, Berger et al. (1996: 57) define a feature for the translation of the preposition *in* from English to French that is 1 if and only if *in* is translated as *pendant* and *in* is followed by the word *weeks* within three words. There is also no need to worry about heterogeneity of features or weighting features, two problems that often cause difficulty in other classification approaches. Model selection is well-founded in the maximum entropy principle: we should not add any information over and above what we find in the empirical evidence. Maximum entropy modeling thus provides a well-motivated probabilistic framework for integrating information from heterogeneous sources.

We could only allude here to another strength of the method: an integrated framework for feature selection and classification. (The fact that we have not done maximum entropy feature selection here is perhaps the main reason for the lower classification accuracy.) Most classification methods cannot deal with a very large number of features. If there are too many features initially, an (often ad-hoc) method has to be used to winnow the feature set down to a manageable size. This is what we have done here, using the X^2 test for words. In maximum entropy modeling one can instead specify all potentially relevant features and use extensions of the basic method such as those described by Berger et al. (1996) to simultaneously select features and fit the classification model. Since the two are integrated, there is a clear probabilistic interpretation (in terms of maximum entropy and maximum likelihood) of the resulting classification model. Such an interpretation is less clear for other methods such as perceptrons and k nearest neighbors.

In this section we have only attempted to give enough information to help the reader to decide whether maximum entropy modeling is an appropriate framework for their classification task when compared to other classification methods. More detailed treatments of maximum entropy modeling are mentioned in the Further Reading.

16.3 Perceptrons

GRADIENT DESCENT
HILL CLIMBING

We present perceptrons here as a simple example of a *gradient descent* (or reversing the direction of goodness, *hill climbing*) algorithm, an important class of iterative learning algorithms. In gradient descent, we attempt to optimize a function of the data that computes a goodness criterion like squared error or likelihood. In each step, we compute the derivative of the function and change the parameters of the model in the direction of the steepest gradient (steepest ascent or descent, depending on the optimality function). This is a good idea because the direction of

```
**comment**: Categorization Decision
**funct** decision($\vec{x}, \vec{w}, \theta$) $\equiv$
  **if** $\vec{w} \cdot \vec{x} > \theta$ **then**
        **return yes**
     **else**
        **return no**
  **fi.**
**comment**: Initialization
$\vec{w} = 0$
$\theta = 0$
**comment**: Perceptron Learning Algorithm
**while** not converged yet **do**
    **for** all elements $\vec{x}_j$ in the training set **do**
        $d$ = decision($\vec{x}_j, \vec{w}, \theta$)
        **if** class($\vec{x}_j$) $= d$ **then**
            **continue**
        **elsif** class($\vec{x}_j$) = **yes and** $d$ = **no then**
            $\theta = \theta - 1$
            $\vec{w} = \vec{w} + \vec{x}_j$
        **elsif** class($\vec{x}_j$) = **no and** $d$ = **yes then**
            $\theta = \theta + 1$
            $\vec{w} = \vec{w} - \vec{x}_j$
        **fi**
    **end**
**end**
```

Figure 16.7 The Perceptron Learning Algorithm. The perceptron decides "yes" if the inner product of weight vector and data vector is greater than θ and "no" otherwise. The learning algorithm cycles through all examples. If the current weight vector makes the right decision for an instance, it is left unchanged. Otherwise, the data vector is added to or subtracted from the weight vector, depending on the direction of the error.

steepest gradient is the direction where we can expect the most improvement in the goodness criterion.

Without further ado, we introduce the perceptron learning algorithm in figure 16.7. As before, text documents are represented as term vectors. Our goal is to learn a weight vector $\vec{w}$ and a threshold θ, such that comparing the dot product of the weight vector and the term vector against the threshold provides the categorization decision. We decide "yes" (the article is in the "earnings" category) if the inner product of weight vector and document vector is greater than the threshold and "no" otherwise:

$$\text{Decide "yes" iff} \quad \vec{w} \cdot \vec{x} = \sum_{i=1}^{K} w_i x_{ij} > \theta$$

where K is the number of features ($K = 20$ for our example as before) and x_{ij} is component i of vector $\vec{x}_j$.

The basic idea of the perceptron learning algorithm is simple. If the weight vector makes a mistake, we move it (and θ) in the direction of greatest change for our optimality criterion $\sum_{i=1}^{K} w_i x_{ij} - \theta$. To see that the changes to $\vec{w}$ and θ in figure 16.8 are made in the direction of greatest change, we first define an optimality criterion ϕ that incorporates θ into the weight vector:

$$\phi(\vec{w}\,') = \phi(\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \\ \theta \end{pmatrix}) = \vec{w}\,' \cdot \vec{x}\,' = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \\ \theta \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \\ -1 \end{pmatrix}$$

The gradient of ϕ (which is the direction of greatest change) is the vector $\vec{x}\,'$:

$$\nabla \phi(\vec{w}\,') = \vec{x}\,'$$

Of all vectors of a given length that we can add to $\vec{w}\,'$, $\vec{x}\,'$ is the one that will change ϕ the most. So that is the direction we want to take for gradient descent; and it is indeed the change that is implemented in figure 16.7.

Figure 16.8 shows one error-correcting step of the algorithm for a two-dimensional problem. The figure also illustrates the class of models that can be learned by perceptrons: linear separators. Each weight vector defines an orthogonal line (or a plane or hyperplane in higher dimensions) that separates the vector space into two halves, one with positive values,

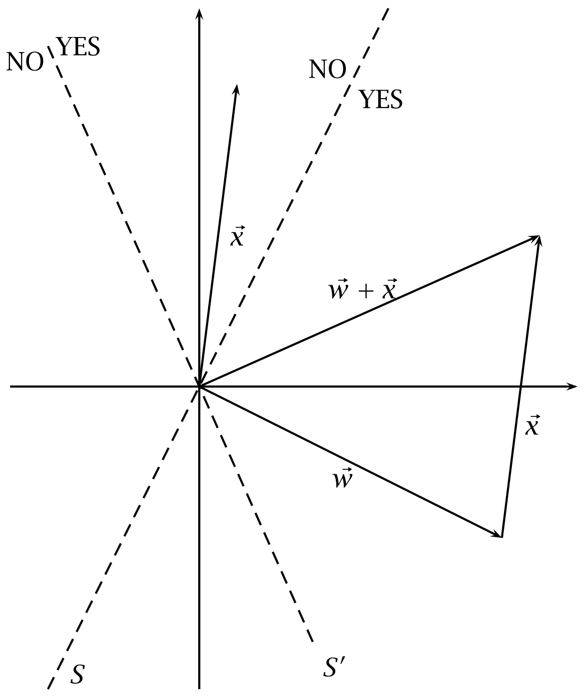

Figure 16.8 One error-correcting step of the perceptron learning algorithm. Data vector $\vec{x}$ is misclassified by the current weight vector $\vec{w}$ since it lies on the "no"-side of the decision boundary S. The correction step adds $\vec{x}$ to $\vec{w}$ and (in this case) corrects the decision since $\vec{x}$ now lies on the "yes"-side of S', the decision boundary of the new weight vector $\vec{w} + \vec{x}$.

one with negative values. In figure 16.8, the separator is S, defined by $\vec{w}$. Classification tasks in which the elements of two classes can be perfectly separated by such a hyperplane are called *linearly separable*.

LINEARLY SEPARABLE

One can show that the perceptron learning algorithm always converges towards a separating hyperplane when applied to a linearly separable problem. This is the *perceptron convergence theorem*. It might seem plausible that the perceptron learning algorithm will eventually find a solution if there is one, since it keeps adjusting the weights for misclassified elements, but since an adjustment for one element will often reverse classification decisions for others, the proof is not trivial.

PERCEPTRON CONVERGENCE THEOREM

Word w^i	Weight
vs	11
mln	6
cts	24
;	2
&	12
000	−4
loss	19
’	−2
"	7
3	−7
profit	31
dlrs	1
1	3
pct	−4
is	−8
s	−12
that	−1
net	8
lt	11
at	−6
θ	37

Table 16.10 Perceptron for the “earnings” category. The weight vector $\vec{w}$ and θ of a perceptron learned by the perceptron learning algorithm for the category “earnings” in Reuters.

Table 16.10 shows the weights learned by the perceptron learning algorithm for the “earnings” category after about 1000 iterations. As with the model parameters in maximum entropy modeling we get high weights for *cts*, *profit* and *lt*. Table 16.11 shows the classification results on the test set. Overall accuracy is 83%. This suggests that the problem is not linearly separable. See the exercises.

A perceptron in 20 dimensions is hard to visualize, so we reran the algorithm with just two dimensions, *mln* and *cts*. The weight vector and the linear separator defined by it are shown in figure 16.9.

The perceptron learning algorithm is guaranteed to learn a linearly sep-

“earnings”	“earnings” correct?	
assigned?	YES	NO
YES	1059	521
NO	28	1691

Table 16.11 Classification results for the perceptron in table 16.10 on the test set. Classification accuracy is 83.3%.

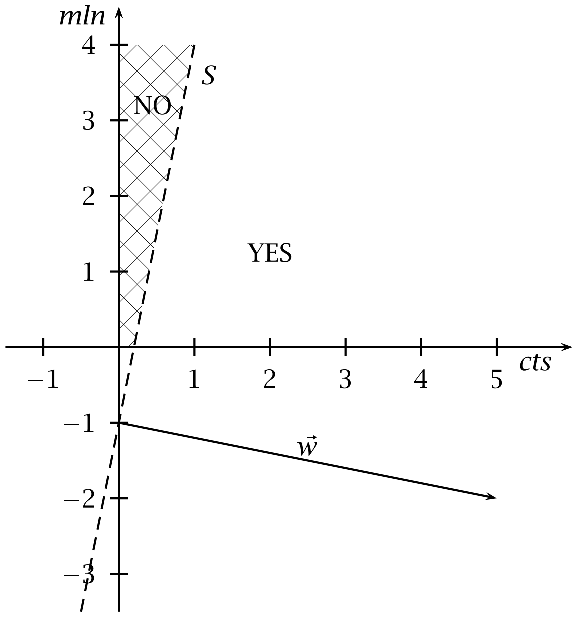

Figure 16.9 Geometric interpretation of a perceptron. The weight for *cts* is 5, the weight for *mln* is −1, and the threshold is 1. The linear separator *S* divides the upper right quadrant into a NO and a YES region. Only documents with many more occurrences of *mln* than *cts* are categorized as not belonging to “earnings.”

arable problem. There are similar convergence theorems for some other gradient descent algorithms, but in most cases convergence will only be to a *local optimum*, locations in the weight space that are locally optimal, but inferior to the globally optimal solution. Perceptrons converge to a *global optimum* because they select a classifier from a class of simple models, the linear separators. There are many important problems that are not linearly separable, the most famous being the XOR problem. The XOR (i.e., eXclusive OR) problem involves a classifier with two features C_1 and C_2 where the answer should be "yes" if C_1 is true and C_2 false or vice versa. A decision tree can easily learn such a problem, whereas a perceptron cannot. After some initial enthusiasm about perceptrons (Rosenblatt 1962), researchers realized these limitations. As a consequence, interest in perceptrons and related learning algorithms faded quickly and remained low for decades. The publication of (Minsky and Papert 1969) is often seen as the point at which the interest in this genre of learning algorithms started to wane. See Rumelhart and Zipser (1985) for a historical summary.

LOCAL OPTIMUM

GLOBAL OPTIMUM

If the perceptron learning algorithm is run on a problem that is not linearly separable, the linear separator will sometimes move back and forth erratically while the algorithm tries in vain to find a perfectly separating plane. This is in fact what happened when we ran the algorithm on the "earnings" data. Classification accuracy fluctuated between 72% and 93%. We picked a state that lies in the middle of this spectrum of accuracy for tables 16.10 and 16.11. Perceptrons have not been used much in NLP because most NLP problems are not linearly separable and the perceptron learning algorithm does not find a good approximate separator in such cases. However, in cases where a problem is linearly separable, perceptrons can be an appropriate classification method due to their simplicity and ease of implementation.

A resurgence of work on gradient descent learning algorithms occurred in the eighties when several learning algorithms were proposed that overcame the limitations of perceptrons, most notably the *backpropagation* algorithm which is used to train *multi-layer perceptrons* (MLPs), otherwise known as *neural networks* or *connectionist models*. Backpropagation applied to MLPs can in principle learn any classification function including XOR. But it converges more slowly than the perceptron learning algorithm and it can get caught in *local optima*. Pointers to uses of neural networks in NLP appear in the Further Reading.

BACKPROPAGATION

NEURAL NETWORKS

CONNECTIONIST MODELS

Exercise 16.12 [⋆⋆]

Build an animated visualization that shows how the perceptron's decision boundary moves during training and run it for a two-dimensional classification problem.

Exercise 16.13 [⋆⋆]

Select a subset of 10 "earnings" and 10 non-"earnings" documents. Select two words, one for the *x* axis, one for the *y* axis and plot the 20 documents with class labels. Is the set linearly separable?

Exercise 16.14 [⋆]

How can one show that a set of data points from two classes is not linearly separable?

Exercise 16.15 [⋆]

Show that the "earnings" data set is not linearly separable.

Exercise 16.16 [⋆]

Suppose a problem is linearly separable and we train a perceptron to convergence. In such a case, classification accuracy will often not be 100%. Why?

Exercise 16.17 [⋆⋆]

Select one of exercises 16.3 through 16.6 and build a perceptron for the corresponding text categorization task.

16.4 *k* Nearest Neighbor Classification

NEAREST NEIGHBOR CLASSIFICATION RULE

The rationale for the *nearest neighbor classification rule* is remarkably simple. To classify a new object, find the object in the training set that is most similar. Then assign the category of this nearest neighbor.

The basic idea is that if there is an identical article in the training set (or at least one with the same representation), then the obvious decision is to assign the same category. If there is no identical article, then the most similar one is our best bet.

k NEAREST NEIGHBOR

A generalization of the nearest neighbor rule is *k nearest neighbor* or *KNN* classification. Instead of using only one nearest neighbor as the basis for our decision, we consult *k* nearest neighbors. KNN for $k > 1$ is more robust than the '1 nearest neighbor' method.

The complexity of KNN is in finding a good measure of similarity. As an example of what can go wrong consider the task of deciding whether there is an eagle in an image. If we have a drawing of an eagle that we want to classify and all exemplars in the database are photographs of eagles, then KNN will classify the drawing as non-eagle because according to any low-level similarity metric based on image features drawings and photographs will come out as very different no matter what their content (and how to implement high-level similarity is an unsolved problem). If one doesn't have a good similarity metric, one can't use KNN.

Fortunately, many NLP tasks have simple similarity metrics that are quite effective. For the "earnings" data, we implemented cosine similarity (see section 8.5.1), and chose $k = 1$. This '1NN algorithm' for binary categorization can be stated as follows.

- Goal: categorize $\vec{y}$ based on the training set X.
- Determine the largest similarity with any element in the training set: $\text{sim}_{\max}(\vec{y}) = \max_{\vec{x} \in X} \text{sim}(\vec{x}, \vec{y})$
- Collect the subset of X that has highest similarity with $\vec{y}$:

  $A = \{\vec{x} \in X | \text{sim}(\vec{x}, \vec{y}) = \text{sim}_{\max}(\vec{y})\}$
- Let n_1 and n_2 be the number of elements in A that belong to the two classes c_1 and c_2, respectively. Then we estimate the conditional probabilities of membership as follows:

(16.13) $$P(c_1|\vec{y}) = \frac{n_1}{n_1 + n_2} \qquad P(c_2|\vec{y}) = \frac{n_2}{n_1 + n_2}$$

- Decide c_1 if $P(c_1|\vec{y}) > P(c_2|\vec{y})$, c_2 otherwise.

This version deals with the case where there is a single nearest neighbor (in which case we simply adopt its category) as well as with cases of ties. For the Reuters data, 2310 of the 3299 articles in the test set have one nearest neighbor. The other 989 have 1NN neighborhoods with more than 1 element (the largest neighborhood has 247 articles with identical representation). Also, 697 articles of the 3299 have a nearest neighbor with identical representation. The reason is not that there are that many duplicates in the test set. Rather, this is a consequence of the feature representation which can give identical representations to two different documents.

It should be obvious how this algorithm generalizes for the case $k > 1$: one simply chooses the k nearest neighbors, again with suitable provisions for ties, and decides based on the majority class of these k neighbors. It is often desirable to weight neighbors according to their similarity, so that the nearest neighbor gets more weight than the farthest.

"earnings"	"earnings" correct?	
assigned?	YES	NO
YES	1022	91
NO	65	2121

Table 16.12 Classification results for an 1NN categorizer for the "earnings" category. Classification accuracy is 95.3%.

BAYES ERROR RATE

One can show that for large enough training sets, the error rate of KNN approaches twice the Bayes error rate. The *Bayes error rate* is the optimal error rate that is achieved if the true distribution of the data is known and we use the decision rule "c_1 if $P(c_1|\vec{y}) > P(c_2|\vec{y})$, otherwise c_2" (Duda and Hart 1973: 98).

The results of applying 1NN to the "earnings" category in Reuters are shown in table 16.12. Overall accuracy is 95.3%.

As alluded to above, the main difficulty with KNN is that its performance is very dependent on the right similarity metric. Much work in implementing KNN for a problem often goes into tuning the similarity metric (and to a lesser extent k, the number of nearest neighbors used). Another potential problem is efficiency. Computing similarity with all training exemplars takes more time than computing a linear classification function or determining the appropriate path of a decision tree.

However, there are ways of implementing KNN search efficiently, and often there is an obvious choice for a similarity metric (as in our case). In such cases, KNN is a robust and conceptually simple method that often performs remarkably well.

Exercise 16.18 [$\star$]

Two of the classifiers we have introduced in this chapter have a linear decision boundary. Which?

Exercise 16.19 [$\star$]

If a classifier's decision boundary is linear, then it cannot achieve perfect accuracy on a problem that is not linearly separable. Does this necessarily mean that it will perform worse on a classification task than a classifier with a more complex decision boundary? Why not? (See Roth (1998) for discussion.)

Exercise 16.20 [$\star\star$]

Select one of exercises 16.3 through 16.6 and build a nearest neighbors classifier for the corresponding text categorization task.

16.5 Further Reading

The purpose of this chapter is to give the student interested in classification for NLP some orientation points. A recent in-depth introduction to machine learning is (Mitchell 1997). Comparisons of several learning algorithms applied to text categorization can be found in (Yang 1998), (Lewis et al. 1996), and (Schütze et al. 1995).

The features and the data representation based on the features used in this chapter can be downloaded from the book's website.

Some important classification techniques which we have not covered are: logistic regression and linear discriminant analysis (Schütze et al. 1995); decision lists, where an ordered list of rules that change the classification is learned (Yarowsky 1994); winnow, a mistake-driven online linear threshold learning algorithm (Dagan et al. 1997a); and the Rocchio algorithm (Rocchio 1971; Schapire et al. 1998).

NAIVE BAYES

Another important classification technique, *Naive Bayes*, was introduced in section 7.2.1. See (Domingos and Pazzani 1997) for a discussion of its properties, in particular the fact that it often does surprisingly well even when the feature independence assumed by Naive Bayes does not hold.

BAGGING
BOOSTING

Other examples of the application of decision trees to NLP tasks are parsing (Magerman 1994) and tagging (Schmid 1994). The idea of using held out training data to train a linear interpolation over all the distributions between a leaf node and the root was used both by Magerman (1994) and earlier work at IBM. Rather than simply using cross-validation to determine an optimal tree size, an alternative is to grow multiple decision trees and then to average the judgements of the individual trees. Such techniques go under names like *bagging* and *boosting*, and have recently been widely explored and found to be quite successful (Breiman 1994; Quinlan 1996). One of the first papers to apply decision trees to text categorization is (Lewis and Ringuette 1994).

MAXIMUM ENTROPY MODELING

Jelinek (1997: ch. 13–14) provides an in-depth introduction to maximum entropy modeling. See also (Lau 1994) and (Ratnaparkhi 1997b). Darroch and Ratcliff (1972) introduced the generalized iterative scaling procedure, and showed its convergence properties. Feature selection algorithms are described by Berger et al. (1996) and Della Pietra et al. (1997).

Maximum entropy modeling has been used for tagging (Ratnaparkhi 1996), text segmentation (Reynar and Ratnaparkhi 1997), prepositional phrase attachment (Ratnaparkhi 1998), sentence boundary detection (Mikheev 1998), determining coreference (Kehler 1997), named entity recognition (Borthwick et al. 1998) and partial parsing (Skut and Brants 1998). Another important application is language modeling for speech recognition (Lau et al. 1993; Rosenfeld 1994, 1996). Iterative proportional fitting, a technique related to generalized iterative scaling, was used by Franz (1996, 1997) to fit loglinear models for tagging and prepositional phrase attachment.

NEURAL NETWORKS

Neural networks or multi-layer perceptrons were one of the statistical techniques that revived interest in Statistical NLP in the eighties based on work by Rumelhart and McClelland (1986) on learning the past tense of English verbs and Elman's (1990) paper "Finding Structure in Time," an attempt to come up with an alternative framework for the conceptualization and acquisition of hierarchical structure in language. Introductions to neural networks and backpropagation are (Rumelhart et al. 1986), (McClelland et al. 1986), and (Hertz et al. 1991). Other neural network research on NLP problems includes tagging (Benello et al. 1989; Schütze 1993), sentence boundary detection (Palmer and Hearst 1997), and parsing (Henderson and Lane 1998). Examples of neural networks used for text categorization are (Wiener et al. 1995) and (Schütze et al. 1995). Miikkulainen (1993) develops a general neural network framework for NLP.

The Perceptron Learning Algorithm in figure 16.7 is adapted from (Littlestone 1995). A proof of the perceptron convergence theorem appears in (Minsky and Papert 1988) and (Duda and Hart 1973: 142).

KNN, or *memory-based learning* as it is sometimes called, has also been applied to a wide range of different NLP problems, including pronunciation (Daelemans and van den Bosch 1996), tagging (Daelemans et al. 1996; van Halteren et al. 1998), prepositional phrase attachment (Zavrel et al. 1997), shallow parsing (Argamon et al. 1998), word sense disambiguation (Ng and Lee 1996) and smoothing of estimates (Zavrel and Daelemans 1997). For KNN-based text categorization see (Yang 1994), (Yang 1995), (Stanfill and Waltz 1986; Masand et al. 1992), and (Hull et al. 1996). Yang (1994, 1995) suggests methods for weighting neighbors according to their similarity. We used cosine as the similarity measure. Other common metrics are Euclidean distance (which is different only if vectors are not normalized, as discussed in section 8.5.1) and the Value Difference Metric (Stanfill and Waltz 1986).