14 *Vector space classification*

The document representation in Naive Bayes is a sequence of terms or a binary vector $\langle e_1, \ldots, e_M \rangle \in \{0, 1\}^{|V|}$. In this chapter, we adopt a different representation for text classification, the vector space model, developed in Chapter 6. It represents each document as a vector with one real-valued component, usually a tf–idf weight, for each term. Thus, the document space $\mathbb{X}$, the domain of the classification function γ, is $\mathbb{R}^{|V|}$. This chapter introduces a number of classification methods that operate on real-valued vectors.

CONTIGUITY HYPOTHESIS

The basic hypothesis in using the vector space model for classification is the *contiguity hypothesis*.

Contiguity hypothesis. Documents in the same class form a contiguous region and regions of different classes do not overlap.

There are many classification tasks, in particular the type of text classification that we encountered in Chapter 13, where classes can be distinguished by word patterns. For example, documents in the class *China* tend to have high values on dimensions like Chinese, Beijing, and Mao, whereas documents in the class *UK* tend to have high values for London, British, and Queen. Documents of the two classes therefore form distinct contiguous regions as shown in Figure 14.1 and we can draw boundaries that separate them and classify new documents. How exactly this is done is the topic of this chapter.

Whether or not a set of documents is mapped into a contiguous region depends on the particular choices we make for the document representation: type of weighting, stop list, and so on. To see that the document representation is crucial, consider the two classes *written by a group* versus *written by a single person*. Frequent occurrence of the first person pronoun I is evidence for the single-person class. But that information is likely deleted from the document representation if we use a stop list. If the document representation chosen is unfavorable, the contiguity hypothesis will not hold and successful vector space classification is not possible.

The same considerations that led us to prefer weighted representations, in particular length-normalized tf–idf representations, in Chapters 6 and 7 also apply here. For example, a term with five occurrences in a document should get a higher weight than a term with one occurrence, but a weight five times larger would give too much emphasis to the term. Unweighted and unnormalized counts should not be used in vector space classification.

We introduce two vector space classification methods in this chapter, Rocchio and kNN. Rocchio classification (Section 14.2) divides the vector space into regions centered on centroids or *prototypes*, one for each class, computed as the center of mass of all documents in the class. Rocchio classification is simple and efficient, but inaccurate if classes are not approximately spheres with similar radii.

PROTOTYPE

kNN or k nearest neighbor classification (Section 14.3) assigns the majority class of the k nearest neighbors to a test document. kNN requires no explicit training and can use the unprocessed training set directly in classification. It is less efficient than other classification methods in classifying documents. If the training set is large, then kNN can handle nonspherical and other complex classes better than Rocchio.

A large number of text classifiers can be viewed as linear classifiers – classifiers that classify based on a simple linear combination of the features (Section 14.4). Such classifiers partition the space of features into regions separated by linear *decision hyperplanes*, in a manner to be detailed below. Because of the bias-variance tradeoff (Section 14.6) more complex nonlinear models are not systematically better than linear models. Nonlinear models have more parameters to fit on a limited amount of training data and are more likely to make mistakes for small and noisy data sets.

When applying two-class classifiers to problems with more than two classes, there are *one-of* tasks – a document must be assigned to exactly one of several mutually exclusive classes – and *any-of* tasks – a document can be assigned to any number of classes as we will explain in Section 14.5. Two-class classifiers solve any-of problems and can be combined to solve one-of problems.

14.1 Document representations and measures of relatedness in vector spaces

As in Chapter 6, we represent documents as vectors in $\mathbb{R}^{|V|}$ in this chapter. To illustrate properties of document vectors in vector classification, we will render these vectors as points in a plane as in the example in Figure 14.1. In reality, document vectors are length-normalized unit vectors that point to the surface of a hypersphere. We can view the two-dimensional (2D) planes in our figures as projections onto a plane of the surface of a (hyper-)sphere

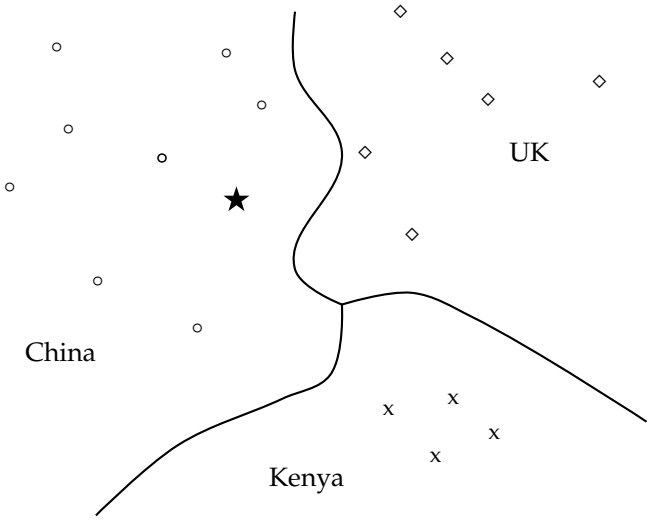

Figure 14.1 Vector space classification into three classes.

as shown in Figure 14.2. Distances on the surface of the sphere and on the projection plane are approximately the same as long as we restrict ourselves to small areas of the surface and choose an appropriate projection (Exercise 14.1).

Decisions of many vector space classifiers are based on a notion of distance, such as when computing the nearest neighbors in kNN classification. We will use Euclidean distance in this chapter as the underlying distance measure. We observed earlier (Exercise 6.18, page 121) that there is a direct correspondence between cosine similarity and Euclidean distance for length-normalized vectors. In vector space classification, it rarely matters whether the relatedness of two documents is expressed in terms of similarity or distance.

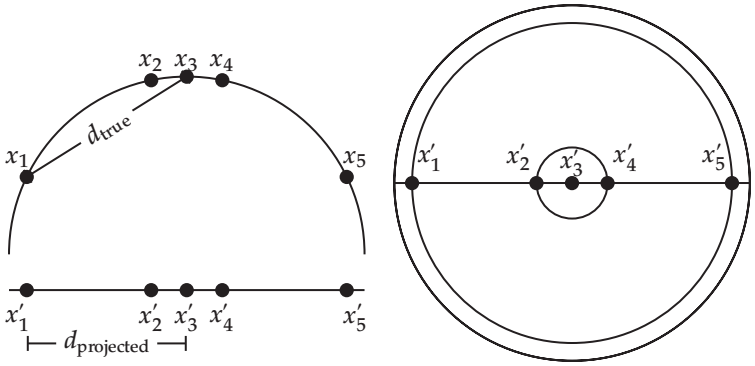

Figure 14.2 Projections of small areas of the unit sphere preserve distances. *Left:* A projection of the 2D semicircle to 1D. For the points x_1, x_2, x_3, x_4, x_5 at x coordinates -0.9, -0.2, 0, 0.2, 0.9 the distance $|x_2x_3| \approx 0.201$ only differs by 0.5% from $|x'_2x'_3| = 0.2$; but $|x_1x_3|/|x'_1x'_3| = d_{\text{true}}/d_{\text{projected}} \approx 1.06/0.9 \approx 1.18$ is an example of a large distortion (18%) when projecting a large area. *Right:* The corresponding projection of the 3D hemisphere to 2D.

However, in addition to documents, centroids or averages of vectors also play an important role in vector space classification. Centroids are not length normalized. For unnormalized vectors, dot product, cosine similarity, and Euclidean distance all have different behaviors in general (Exercise 14.6). We are mostly concerned with small local regions when computing the similarity between a document and a centroid, and the smaller the region the more similar the behavior of the three measures is.

? **Exercise 14.1** For small areas, distances on the surface of the hypersphere are approximated well by distances on its projection (Figure 14.2) because $\alpha \approx \sin \alpha$ for small angles. For what size angle is the distortion $\alpha / \sin(\alpha)$ (i) 1.01, (ii) 1.05 and (iii) 1.1?

14.2 Rocchio classification

Figure 14.1 shows three classes, *China*, *UK*, and *Kenya*, in a 2D space. Documents are shown as circles, diamonds, and Xs. The boundaries in the figure, which we call *decision boundaries*, are chosen to separate the three classes, but are otherwise arbitrary. To classify a new document, depicted as a star in the figure, we determine the region it occurs in and assign it the class of that region – *China* in this case. Our task in vector space classification is to devise algorithms that compute good boundaries where "good" means high classification accuracy on data unseen during training.

DECISION BOUNDARY

The main work we must do in vector space classification is to define the boundaries between classes because they determine the classification decision. Perhaps the best-known way of doing this is *Rocchio classification*, which uses *centroids* to define the boundaries. The centroid of a class c is computed as the vector average or center of mass of its members:

ROCCHIO CLASSIFICATION

CENTROID

$$\vec{\mu}(c) = \frac{1}{|D_c|} \sum_{d \in D_c} \vec{v}(d), \tag{14.1}$$

where D_c is the set of documents in $\mathbb{D}$ whose class is c: $D_c = \{d : \langle d, c \rangle \in \mathbb{D}\}$. We denote the normalized vector of d by $\vec{v}(d)$ (Equation (6.11), page 111). Three example centroids are shown as solid circles in Figure 14.3.

The boundary between two classes in Rocchio classification is the set of points with equal distance from the two centroids. For example, $|a_1| = |a_2|$, $|b_1| = |b_2|$, and $|c_1| = |c_2|$ in the figure. This set of points is always a line. The generalization of a line in M-dimensional space is a hyperplane, which we define as the set of points $\vec{x}$ that satisfy:

$$\vec{w}^T \vec{x} = b \tag{14.2}$$

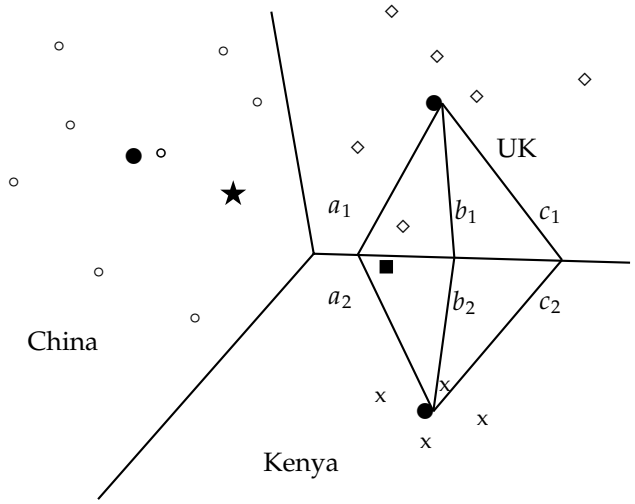

Figure 14.3 Rocchio classification.

NORMAL VECTOR

where $\vec{w}$ is the M-dimensional *normal vector*[1] of the hyperplane and b is a constant. This definition of hyperplanes includes lines (any line in 2D can be defined by $w_1x_1 + w_2x_2 = b$) and 2-dimensional planes (any plane in three dimensions (3D) can be defined by $w_1x_1 + w_2x_2 + w_3x_3 = b$). A line divides a plane in two, a plane divides 3D space in two, and hyperplanes divide higher-dimensional spaces in two.

Thus, the boundaries of class regions in Rocchio classification are hyperplanes. The classification rule in Rocchio is to classify a point in accordance with the region it falls into. Equivalently, we determine the centroid $\vec{\mu}(c)$ that the point is closest to and then assign it to c. As an example, consider the star in Figure 14.3. It is located in the *China* region of the space and Rocchio therefore assigns it to *China*. We show the Rocchio algorithm in pseudocode in Figure 14.4.

Example 14.1: Table 14.1 shows the tf–idf vector representations of the five documents in Table 13.1 (page 241), using the formula $(1 + \log_{10} \mathrm{tf}_{t,d}) \log_{10}(4/\mathrm{df}_t)$ if $\mathrm{tf}_{t,d} > 0$ (Equation (6.14), page 117). The two class centroids are $\mu_c = 1/3 \cdot (\vec{d}_1 + \vec{d}_2 + \vec{d}_3)$ and $\mu_{\overline{c}} = 1/1 \cdot (\vec{d}_4)$. The distances of the test document from the centroids are $|\mu_c - \vec{d}_5| \approx 1.15$ and $|\mu_{\overline{c}} - \vec{d}_5| = 0.0$. Thus, Rocchio assigns d_5 to $\overline{c}$.

The separating hyperplane in this case has the following parameters:

$$\vec{w} \approx (0 \ -0.71 \ -0.71 \ 1/3 \ 1/3 \ 1/3)^T$$
$$b = -1/3$$

See Exercise 14.15 for how to compute $\vec{w}$ and b. We can easily verify that this hyperplane separates the documents as desired:

[1] Recall from basic linear algebra that $\vec{v} \cdot \vec{w} = \vec{v}^T \vec{w}$, that is, the dot product of $\vec{v}$ and $\vec{w}$ equals the product by matrix multiplication of the transpose of $\vec{v}$ and $\vec{w}$.

TRAINROCCHIO($\mathbb{C}$, $\mathbb{D}$)
1 **for each** $c_j \in \mathbb{C}$
2 **do** $D_j \leftarrow \{d : \langle d, c_j \rangle \in \mathbb{D}\}$
3 $\quad \vec{\mu}_j \leftarrow \frac{1}{|D_j|} \sum_{d \in D_j} \vec{v}(d)$
4 **return** $\{\vec{\mu}_1, \ldots, \vec{\mu}_J\}$

APPLYROCCHIO($\{\vec{\mu}_1, \ldots, \vec{\mu}_J\}, d$)
1 **return** $\arg\min_j |\vec{\mu}_j - \vec{v}(d)|$

Figure 14.4 Rocchio classification: Training and testing.

$\vec{w}^T \vec{d}_1 \approx 0 \cdot 0 + -0.71 \cdot 0 + -0.71 \cdot 0 + 1/3 \cdot 0 + 1/3 \cdot 1.0 + 1/3 \cdot 0 = 1/3 > b$ (and, similarly, $\vec{w}^T \vec{d}_i > b$ for $2 \leq i \leq 3$) and $\vec{w}^T \vec{d}_4 = -1 < b$. Thus, documents in c are above the hyperplane ($\vec{w}^T \vec{d} > b$) and documents in $\bar{c}$ are below the hyperplane ($\vec{w}^T \vec{d} < b$).

The assignment criterion in Figure 14.4 is Euclidean distance. An alternative is cosine similarity:

$$\text{Assign } d \text{ to class } c = \arg\max_{c'} \cos(\vec{\mu}(c'), \vec{v}(d)).$$

As discussed in Section 14.1, the two assignment criteria will sometimes make different classification decisions. We present the Euclidean distance variant of Rocchio classification here because it emphasizes Rocchio's close correspondence to K-means clustering (Section 16.4, page 331).

Rocchio classification is a form of Rocchio relevance feedback (Section 9.1.1, page 163). The average of the relevant documents, corresponding to the most important component of the Rocchio vector in relevance feedback (Equation (9.3), page 166), is the centroid of the "class" of relevant documents. We omit the query component of the Rocchio formula in Rocchio classification because there is no query in text classification. Rocchio classification can be applied to $J > 2$ classes, whereas Rocchio relevance feedback is designed to distinguish only two classes, relevant and nonrelevant.

Table 14.1 Vectors and class centroids for the data in Table 13.1.

	term weights					
vector	Chinese	Japan	Tokyo	Macao	Beijing	Shanghai
$\vec{d}_1$	0	0	0	0	1.0	0
$\vec{d}_2$	0	0	0	0	0	1.0
$\vec{d}_3$	0	0	0	1.0	0	0
$\vec{d}_4$	0	0.71	0.71	0	0	0
$\vec{d}_5$	0	0.71	0.71	0	0	0
$\vec{\mu}_c$	0	0	0	0.33	0.33	0.33
$\vec{\mu}_{\bar{c}}$	0	0.71	0.71	0	0	0

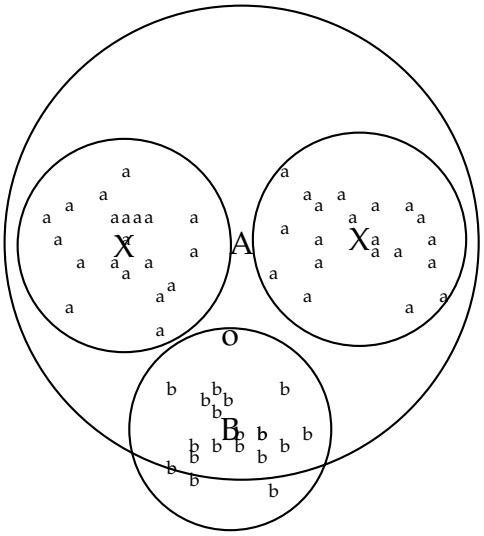

Figure 14.5 The multimodal class "a" consists of two different clusters (small upper circles centered on Xs). Rocchio classification will misclassify "o" as "a" because it is closer to the centroid A of the "a" class than to the centroid B of the "b" class.

In addition to respecting contiguity, the classes in Rocchio classification must be approximate spheres with similar radii. In Figure 14.3, the solid square just below the boundary between *UK* and *Kenya* is a better fit for the class *UK* because *UK* is more scattered than *Kenya*. But Rocchio assigns it to *Kenya* because it ignores details of the distribution of points in a class and only uses distance from the centroid for classification.

The assumption of sphericity also does not hold in Figure 14.5. We cannot represent the "a" class well with a single prototype because it has two clusters. Rocchio often misclassifies this type of *multimodal class*. A text classification example for multimodality is a country like Burma, which changed its name to Myanmar in 1989. The two clusters before and after the name change need not be close to each other in space. We also encountered the problem of multimodality in relevance feedback (Section 9.1.2, page 169).

MULTIMODAL CLASS

Two-class classification is another case where classes are rarely distributed like spheres with similar radii. Most two-class classifiers distinguish between a class like *China* that occupies a small region of the space and its widely scattered complement. Assuming equal radii will result in a large number of false positives. Most two-class classification problems therefore require a modified decision rule of the form:

$$\text{Assign } d \text{ to class } c \text{ iff } |\vec{\mu}(c) - \vec{v}(d)| < |\vec{\mu}(\overline{c}) - \vec{v}(d)| - b$$

for a positive constant b. As in Rocchio relevance feedback, the centroid of the negative documents is often not used at all, so that the decision criterion simplifies to $|\vec{\mu}(c) - \vec{v}(d)| < b'$ for a positive constant b'.

Table 14.2 Training and test times for Rocchio classification. L_{ave} is the average number of tokens per document. L_a and M_a are the numbers of tokens and types, respectively, in the test document. Computing Euclidean distance between the class centroids and a document is $\Theta(|\mathbb{C}|M_a)$.

mode	time complexity
training	$\Theta(\|\mathbb{D}\|L_{ave} + \|\mathbb{C}\|\|V\|)$
testing	$\Theta(L_a + \|\mathbb{C}\|M_a) = \Theta(\|\mathbb{C}\|M_a)$

Table 14.2 gives the time complexity of Rocchio classification.[2] Adding all documents to their respective vector sum is $\Theta(|\mathbb{D}|L_{ave})$ (as opposed to $\Theta(|\mathbb{D}|V|)$) because we need only consider nonzero entries. Dividing each vector sum by the size of its class to compute the centroid is $\Theta(|V|)$. Overall, training time is linear in the size of the collection (cf. Exercise 13.1). Thus, Rocchio classification and Naive Bayes have the same linear training time complexity.

In the next section, we will introduce another vector space classification method, kNN, that deals better with classes that have non-spherical, disconnected or other irregular shapes.

? **Exercise 14.2** [⋆] Show that Rocchio classification can assign a label to a document that is different from its training set label.

14.3 *k* nearest neighbor

k NEAREST NEIGHBOR CLASSIFICATION

Unlike Rocchio, *k nearest neighbor* or *kNN classification* determines the decision boundary locally. For 1NN we assign each document to the class of its closest neighbor. For kNN we assign each document to the majority class of its k closest neighbors where k is a parameter. The rationale of kNN classification is that, based on the contiguity hypothesis, we expect a test document d to have the same label as the training documents located in the local region surrounding d.

VORONOI TESSELLATION

Decision boundaries in 1NN are concatenated segments of the *Voronoi tessellation* as shown in Figure 14.6. The Voronoi tessellation of a set of objects decomposes space into Voronoi cells, where each object's cell consists of all points that are closer to the object than to other objects. In our case, the objects are documents. The Voronoi tessellation then partitions the plane into $|\mathbb{D}|$ convex polygons, each containing its corresponding document (and no other) as shown in Figure 14.6, where a convex polygon is a convex region in 2D space bounded by lines.

[2] We write $\Theta(|\mathbb{D}|L_{ave})$ for $\Theta(T)$ and assume that the length of test documents is bounded as we did on page 242.

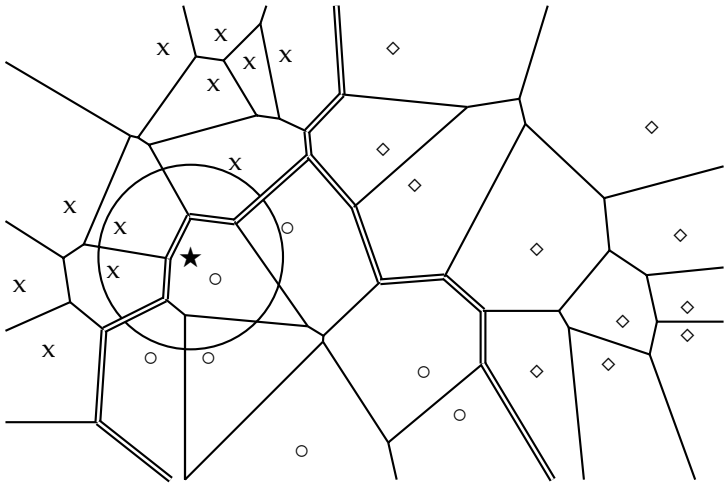

Figure 14.6 Voronoi tessellation and decision boundaries (double lines) in 1NN classification. The three classes are: X, circle and diamond.

For general $k \in \mathbb{N}$ in kNN, consider the region in the space for which the set of k nearest neighbors is the same. This again is a convex polygon and the space is partitioned into convex polygons, within each of which the set of k nearest neighbors is invariant (Exercise 14.11).[3]

1NN is not very robust. The classification decision of each test document relies on the class of a single training document, which may be incorrectly labeled or atypical. kNN for $k > 1$ is more robust. It assigns documents to the majority class of their k closest neighbors, with ties broken randomly.

There is a probabilistic version of this kNN classification algorithm. We can estimate the probability of membership in class c as the proportion of the k nearest neighbors in c. Figure 14.6 gives an example for $k = 3$. Probability estimates for class membership of the star are $\hat{P}(\text{circle class}|\text{star}) = 1/3$, $\hat{P}(\text{X class}|\text{star}) = 2/3$, and $\hat{P}(\text{diamond class}|\text{star}) = 0$. The 3NN estimate ($\hat{P}_1(\text{circle class}|\text{star}) = 1/3$) and the 1NN estimate ($\hat{P}_1(\text{circle class}|\text{star}) = 1$) differ with 3NN preferring the X class and 1NN preferring the circle class.

The parameter k in kNN is often chosen based on experience or knowledge about the classification problem at hand. It is desirable for k to be odd to make ties less likely. $k = 3$ and $k = 5$ are common choices, but much larger values, between 50 and 100, are also used. An alternative way of setting the parameter is to select the k that gives best results on a held-out portion of the training set.

[3] The generalization of a polygon to higher dimensions is a polytope. A *polytope* is a region in M-dimensional space bounded by $(M-1)$-dimensional hyperplanes. In M dimensions, the decision boundaries for kNN consist of segments of $(M-1)$-dimensional hyperplanes that form the Voronoi tessellation into convex polytopes for the training set of documents. The decision criterion of assigning a document to the majority class of its k nearest neighbors applies equally to $M = 2$ (tessellation into polygons) and $M > 2$ (tessellation into polytopes).

TRAIN-KNN($\mathbb{C}$, $\mathbb{D}$)
1 $\mathbb{D}' \leftarrow$ PREPROCESS($\mathbb{D}$)
2 $k \leftarrow$ SELECT-K($\mathbb{C}$, $\mathbb{D}'$)
3 **return** $\mathbb{D}'$, k

APPLY-KNN($\mathbb{C}$, $\mathbb{D}'$, k, d)
1 $S_k \leftarrow$ COMPUTENEARESTNEIGHBORS($\mathbb{D}'$, k, d)
2 **for each** $c_j \in \mathbb{C}$
3 **do** $p_j \leftarrow |S_k \cap c_j|/k$
4 **return** $\arg\max_j p_j$

Figure 14.7 kNN training (with preprocessing) and testing. p_j is an estimate for $P(c_j|S_k) = P(c_j|d)$. c_j denotes the set of all documents in the class c_j.

We can also weight the "votes" of the k nearest neighbors by their cosine similarity. In this scheme, a class's score is computed as:

$$\text{score}(c, d) = \sum_{d' \in S_k} I_c(d') \cos(\vec{v}(d'), \vec{v}(d))$$

where S_k is the set of d's k nearest neighbors and $I_c(d') = 1$ iff d' is in class c and 0 otherwise. We then assign the document to the class with the highest score. Weighting by similarities is often more accurate than simple voting. For example, if two classes have the same number of neighbors in the top k, the class with the more similar neighbors wins.

Figure 14.7 summarizes the kNN algorithm.

Example 14.2: The distances of the test document from the four training documents in Table 14.1 are $|\vec{d}_1 - \vec{d}_5| = |\vec{d}_2 - \vec{d}_5| = |\vec{d}_3 - \vec{d}_5| \approx 1.41$ and $|\vec{d}_4 - \vec{d}_5| = 0.0$. d_5's nearest neighbor is therefore d_4 and 1NN assigns d_5 to d_4's class, $\overline{c}$.

14.3.1 Time complexity and optimality of k nearest neighbor

Table 14.3 gives the time complexity of kNN. kNN has properties that are quite different from most other classification algorithms. Training a kNN classifier simply consists of determining k and preprocessing documents. In fact, if we preselect a value for k and do not preprocess, then kNN requires no training at all. In practice, we have to perform preprocessing steps like tokenization. It makes more sense to preprocess training documents once as part of the training phase rather than repeatedly every time we classify a new test document.

Test time is $\Theta(|\mathbb{D}|M_{\text{ave}}M_{\text{a}})$ for kNN. It is linear in the size of the training set; we need to compute the distance of each training document from the

Table 14.3 Training and test times for kNN classification. M_{ave} is the average size of the vocabulary of documents in the collection.

kNN with preprocessing of training set	
training	$\Theta(\|\mathbb{D}\|L_{\text{ave}})$
testing	$\Theta(L_{\text{a}} + \|\mathbb{D}\|M_{\text{ave}}M_{\text{a}}) = \Theta(\|\mathbb{D}\|M_{\text{ave}}M_{\text{a}})$
kNN without preprocessing of training set	
training	$\Theta(1)$
testing	$\Theta(L_{\text{a}} + \|\mathbb{D}\|L_{\text{ave}}M_{\text{a}}) = \Theta(\|\mathbb{D}\|L_{\text{ave}}M_{\text{a}})$

test document. Test time is independent of the number of classes J. kNN therefore has a potential advantage for problems with large J.

In kNN classification, we do not perform any estimation of parameters as we do in Rocchio classification (centroids) or in Naive Bayes (priors and conditional probabilities). kNN simply memorizes all examples in the training set and then compares the test document to them. For this reason, kNN is also called *memory-based learning* or *instance-based learning*. It is usually desirable to have as much training data as possible in machine learning. But in kNN, large training sets come with a severe efficiency penalty in classification.

MEMORY-BASED LEARNING

Can kNN testing be made more efficient than $\Theta(|\mathbb{D}|M_{\text{ave}}M_{\text{a}})$ or, ignoring the length of documents, more efficient than $\Theta(|\mathbb{D}|)$? There are fast kNN algorithms for small dimensionality M (Exercise 14.12). There are also approximations for large M that give error bounds for specific efficiency gains (see Section 14.7). These approximations have not been extensively tested for text classification applications, so it is not clear whether they can achieve much better efficiency than $\Theta(|\mathbb{D}|)$ without a significant loss of accuracy.

The reader may have noticed the similarity between the problem of finding nearest neighbors of a test document and ad hoc retrieval, where we search for the documents with the highest similarity to the query (Section 6.3.2, page 113). In fact, the two problems are both k nearest neighbor problems and only differ in the relative density of (the vector of) the test document in kNN (10s or 100s of non-zero entries) versus the sparseness of (the vector of) the query in ad hoc retrieval (usually fewer than ten nonzero entries). We introduced the inverted index for efficient ad hoc retrieval in Section 1.1 (page 5). Is the inverted index also the solution for efficient kNN?

An inverted index restricts a search to those documents that have at least one term in common with the query. Thus, in the context of kNN, the inverted index will be efficient if the test document has no term overlap with a large number of training documents. Whether this is the case depends on the classification problem. If documents are long and no stop list is used, then less time will be saved. But with short documents and a large stop list, an inverted index may well cut the average test time by a factor of 10 or more.

The search time in an inverted index is a function of the length of the postings lists of the terms in the query. Postings lists grow sublinearly with the length of the collection since the vocabulary increases according to Heaps'

law – if the probability of occurrence of some terms increases, then the probability of occurrence of others must decrease. However, most new terms are infrequent. We therefore take the complexity of inverted index search to be $\Theta(T)$ (as discussed in Section 2.4.2, page 38) and, assuming average document length does not change over time, $\Theta(T) = \Theta(|\mathbb{D}|)$.

As we will see in Chapter 15, kNN's effectiveness is close to that of the most accurate learning methods in text classification (Table 15.2, page 307). A measure of the quality of a learning method is its *Bayes error rate*, the average error rate of classifiers learned by it for a particular problem. kNN is not optimal for problems with a nonzero Bayes error rate, that is, for problems where even the best possible classifier has a nonzero classification error. The error of 1NN is asymptotically (as the training set increases) bounded by twice the Bayes error rate. That is, if the optimal classifier has an error rate of x, then 1NN has an asymptotic error rate of $2x$. This is due to the effect of noise – we already saw one example of noise in the form of noisy features in Section 13.5 (page 251), but noise can also take other forms as we will discuss in the next section. Noise affects two components of kNN: the test document and the closest training document. The two sources of noise are additive, so the overall error of 1NN is twice the optimal error rate. For problems with Bayes error rate 0, the error rate of 1NN will approach 0 as the size of the training set increases.

BAYES ERROR RATE

? **Exercise 14.3** Explain why kNN handles multimodal classes better than Rocchio.

14.4 Linear versus nonlinear classifiers

In this section, we show that the two learning methods – Naive Bayes and Rocchio – are instances of linear classifiers, the perhaps most important group of text classifiers, and contrast them with nonlinear classifiers. To simplify the discussion, we will only consider two-class classifiers in this section and define a *linear classifier* as a two-class classifier that decides class membership by comparing a linear combination of the features to a threshold.

LINEAR CLASSIFIER

In two dimensions, a linear classifier is a line. Five examples are shown in Figure 14.8. These lines have the functional form $w_1x_1 + w_2x_2 = b$. The classification rule of a linear classifier is to assign a document to c if $w_1x_1 + w_2x_2 > b$ and to $\overline{c}$ if $w_1x_1 + w_2x_2 \leq b$. Here, $(x_1, x_2)^T$ is the 2D vector representation of the document and $(w_1, w_2)^T$ is the parameter vector that defines (together with b) the decision boundary. An alternative geometric interpretation of a linear classifier is provided in Figure 15.7 (page 316).

We can generalize this 2D linear classifier to higher dimensions by defining a hyperplane as we did in Equation (14.2), repeated here as Equation (14.3):

$$\vec{w}^T\vec{x} = b. \tag{14.3}$$

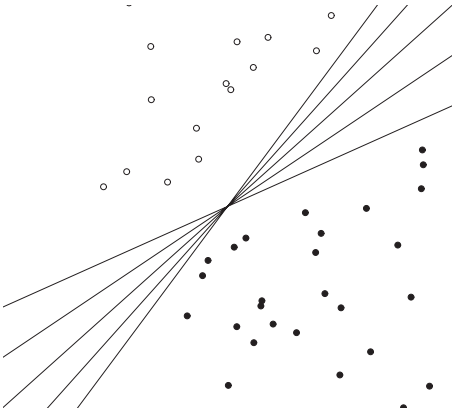

Figure 14.8 There are an infinite number of hyperplanes that separate two linearly separable classes.

The assignment criterion then is: assign to c if $\vec{w}^T\vec{x} > b$ and to $\overline{c}$ if $\vec{w}^T\vec{x} \leq b$.

DECISION HYPERPLANE

We call a hyperplane that we use as a linear classifier a *decision hyperplane*.

The corresponding algorithm for linear classification in M dimensions is shown in Figure 14.9. Linear classification at first seems trivial given the simplicity of this algorithm. However, the difficulty is in training the linear classifier, that is, in determining the parameters $\vec{w}$ and b based on the training set. In general, some learning methods compute much better parameters than others where our criterion for evaluating the quality of a learning method is the effectiveness of the learned linear classifier on new data.

We now show that Rocchio and Naive Bayes are linear classifiers. To see this for Rocchio, observe that a vector $\vec{x}$ is on the decision boundary if it has equal distance to the two class centroids:

$$|\vec{\mu}(c_1) - \vec{x}| = |\vec{\mu}(c_2) - \vec{x}|. \tag{14.4}$$

Some basic arithmetic shows that this corresponds to a linear classifier with normal vector $\vec{w} = \vec{\mu}(c_1) - \vec{\mu}(c_2)$ and $b = 0.5 * (|\vec{\mu}(c_1)|^2 - |\vec{\mu}(c_2)|^2)$ (Exercise 14.15).

We can derive the linearity of Naive Bayes from its decision rule, which chooses the category c with the largest $\hat{P}(c|d)$ (Figure 13.2, page 241) where:

$$\hat{P}(c|d) \propto \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c)$$

APPLYLINEARCLASSIFIER($\vec{w}$, b, $\vec{x}$)
1 $score \leftarrow \sum_{i=1}^{M} w_i x_i$
2 **if** $score > b$
3 **then return** 1
4 **else return** 0

Figure 14.9 Linear classification algorithm.

Table 14.4 A linear classifier. The dimensions t_i and parameters w_i of a linear classifier for the class *interest* (as in interest rate) in Reuters-21578. The threshold is $b = 0$. Terms like dlr and world have negative weights because they are indicators for the competing class *currency*.

t_i	w_i	d_{1i}	d_{2i}	t_i	w_i	d_{1i}	d_{2i}
prime	0.70	0	1	dlrs	-0.71	1	1
rate	0.67	1	0	world	-0.35	1	0
interest	0.63	0	0	sees	-0.33	0	0
rates	0.60	0	0	year	-0.25	0	0
discount	0.46	1	0	group	-0.24	0	0
bundesbank	0.43	0	0	dlr	-0.24	0	0

and n_d is the number of tokens in the document that are part of the vocabulary. Denoting the complement category as $\bar{c}$, we obtain for the log odds:

$$\log \frac{\hat{P}(c|d)}{\hat{P}(\bar{c}|d)} = \log \frac{\hat{P}(c)}{\hat{P}(\bar{c})} + \sum_{1 \le k \le n_d} \log \frac{\hat{P}(t_k|c)}{\hat{P}(t_k|\bar{c})}. \tag{14.5}$$

We choose class c if the odds are greater than 1 or, equivalently, if the log odds are greater than 0. It is easy to see that Equation (14.5) is an instance of Equation (14.3) for $w_i = \log[\hat{P}(t_i|c)/\hat{P}(t_i|\bar{c})]$, $x_i =$ number of occurrences of t_i in d, and $b = -\log[\hat{P}(c)/\hat{P}(\bar{c})]$. Here, the index i, $1 \le i \le M$, refers to terms of the vocabulary (not to positions in d as k does, cf. Section 13.4.1, page 250) and $\vec{x}$ and $\vec{w}$ are M-dimensional vectors. So in log space, Naive Bayes is a linear classifier.

Example 14.3: Table 14.4 defines a linear classifier for the category *interest* in Reuters-21578 (see Section 13.6, page 258). We assign document $\vec{d}_1$ "rate discount dlrs world" to *interest* since $\vec{w}^T\vec{d}_1 = 0.67 \cdot 1 + 0.46 \cdot 1 + (-0.71) \cdot 1 + (-0.35) \cdot 1 = 0.07 > 0 = b$. We assign $\vec{d}_2$ "prime dlrs" to the complement class (not in *interest*) because $\vec{w}^T\vec{d}_2 = -0.01 \le b$. For simplicity, we assume a simple binary vector representation in this example: 1 for occurring terms, 0 for nonoccurring terms.

Figure 14.10 is a graphical example of a *linear problem*, which we define to mean that the underlying distributions $P(d|c)$ and $P(d|\bar{c})$ of the two classes are separated by a line. We call this separating line the *class boundary*. It is the "true" boundary of the two classes and we distinguish it from the decision boundary that the learning method computes to approximate the class boundary.

CLASS BOUNDARY

As is typical in text classification, there are some *noise documents* in Figure 14.10 (marked with arrows) that do not fit well into the overall distribution of the classes. In Section 13.5 (page 251), we defined a noise feature as a misleading feature that, when included in the document representation, on average increases the classification error. Analogously, a noise document is a document that, when included in the training set, misleads the learning

NOISE DOCUMENT

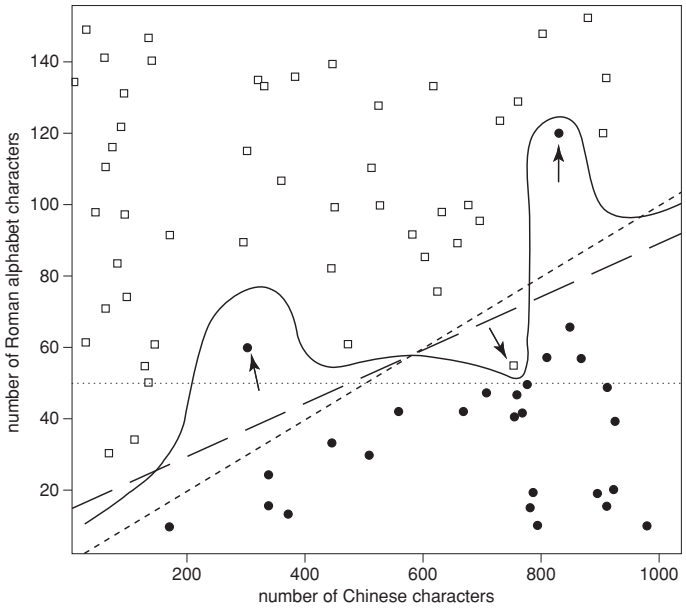

Figure 14.10 A linear problem with noise. In this hypothetical web page classification scenario, Chinese–only web pages are solid circles and mixed Chinese–English web pages are squares. The two classes are separated by a linear class boundary (*dashed line*, *short dashes*), except for three noise documents (*marked with arrows*).

method and increases classification error. Intuitively, the underlying distribution partitions the representation space into areas with mostly homogeneous class assignments. A document that does not conform with the dominant class in its area is a noise document.

Noise documents are one reason why training a linear classifier is hard. If we pay too much attention to noise documents when choosing the decision hyperplane of the classifier, then it will be inaccurate on new data. More fundamentally, it is usually difficult to determine which documents are noise documents and therefore potentially misleading.

LINEAR SEPARABILITY

If there exists a hyperplane that perfectly separates the two classes, then we call the two classes *linearly separable*. In fact, if linear separability holds, then there is an infinite number of linear separators (Exercise 14.4) as illustrated by Figure 14.8, where the number of possible separating hyperplanes is infinite.

Figure 14.8 illustrates another challenge in training a linear classifier. If we are dealing with a linearly separable problem, then we need a criterion for selecting among all decision hyperplanes that perfectly separate the training data. In general, some of these hyperplanes will do well on new data, some will not.

NONLINEAR CLASSIFIER

An example of a *nonlinear classifier* is kNN. The nonlinearity of kNN is intuitively clear when looking at examples like Figure 14.6. The decision boundary of kNN consists of locally linear segments, but in general has a complex shape that is not equivalent to a line in 2D or a hyperplane in higher dimensions.

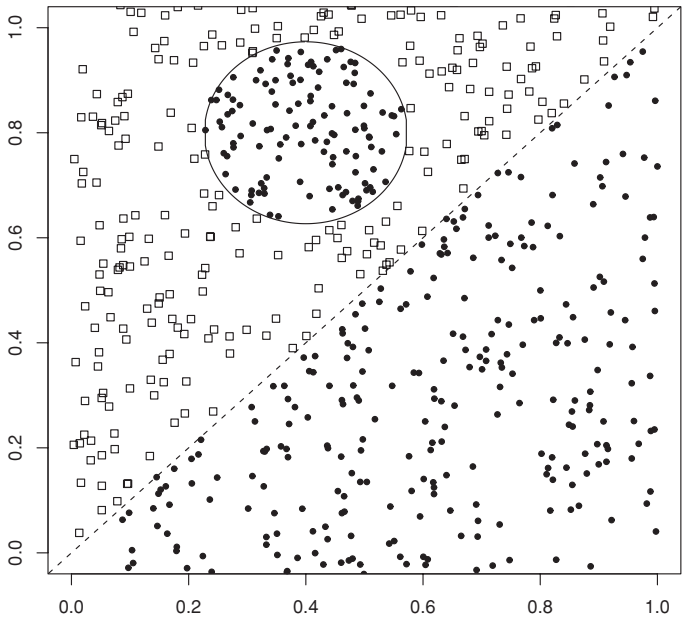

Figure 14.11 A nonlinear problem.

Figure 14.11 is another example of a nonlinear problem: There is no good linear separator between the distributions $P(d|c)$ and $P(d|\overline{c})$ because of the circular "enclave" in the upper left part of the graph. Linear classifiers misclassify the enclave, whereas a nonlinear classifier like kNN will be highly accurate for this type of problem if the training set is large enough.

If a problem is nonlinear and its class boundaries cannot be approximated well with linear hyperplanes, then nonlinear classifiers are often more accurate than linear classifiers. If a problem is linear, it is best to use a simpler linear classifier.

? **Exercise 14.4** Prove that the number of linear separators of two classes is either infinite or zero.

14.5 Classification with more than two classes

We can extend two-class linear classifiers to $J > 2$ classes. The method to use depends on whether the classes are mutually exclusive or not.

ANY-OF CLASSIFICATION

Classification for classes that are not mutually exclusive is called *any-of*, *multilabel*, or *multivalue classification*. In this case, a document can belong to several classes simultaneously, or to a single class, or to none of the classes. A decision on one class leaves all options open for the others. It is sometimes said that the classes are *independent* of each other, but this is misleading; the classes are rarely statistically independent in the sense defined on page 255. In terms of the formal definition of the classification problem in

Equation (13.1) (page 237), we learn J different classifiers γ_j in any-of classification, each returning either c_j or $\overline{c}_j$: $\gamma_j(d) \in \{c_j, \overline{c}_j\}$.

Solving an any-of classification task with linear classifiers is straightforward:

1. Build a classifier for each class, where the training set consists of the set of documents in the class (positive labels) and its complement (negative labels).
2. Given the test document, apply each classifier separately. The decision of one classifier has no influence on the decisions of the other classifiers.

ONE-OF CLASSIFICATION

The second type of classification with more than two classes is *one-of classification*. Here, the classes are mutually exclusive. Each document must belong to exactly one of the classes. One-of classification is also called *multinomial*, *polytomous*,[4] *multiclass*, or *single-label classification*. Formally, there is a single classification function γ in one-of classification whose range is $\mathbb{C}$, i.e., $\gamma(d) \in \{c_1, \ldots, c_J\}$. kNN is a (nonlinear) one-of classifier.

True one-of problems are less common in text classification than any-of problems. With classes like *UK*, *China*, *poultry*, or *coffee*, a document can be relevant to many topics simultaneously – as when the prime minister of the UK visits China to talk about the coffee and poultry trade.

Nevertheless, we will often make a one-of assumption, as we did in Figure 14.1, even if classes are not really mutually exclusive. For the classification problem of identifying the language of a document, the one-of assumption is a good approximation as most text is written in only one language. In such cases, imposing a one-of constraint can increase the classifier's effectiveness because errors that are due to the fact that the any-of classifiers assigned a document to either no class or more than one class are eliminated.

J hyperplanes do not divide $\mathbb{R}^{|V|}$ into J distinct regions as illustrated in Figure 14.12. Thus, we must use a combination method when using two-class linear classifiers for one-of classification. The simplest method is to rank classes and then select the top-ranked class. Geometrically, the ranking can be with respect to the distances from the J linear separators. Documents close to a class's separator are more likely to be misclassified, so the greater the distance from the separator, the more plausible it is that a positive classification decision is correct. Alternatively, we can use a direct measure of confidence to rank classes, for example, probability of class membership. We can state this algorithm for one-of classification with linear classifiers as follows:

1. Build a classifier for each class, where the training set consists of the set of documents in the class (positive labels) and its complement (negative labels).
2. Given the test document, apply each classifier separately.

[4] A synonym of polytomous is polychotomous.

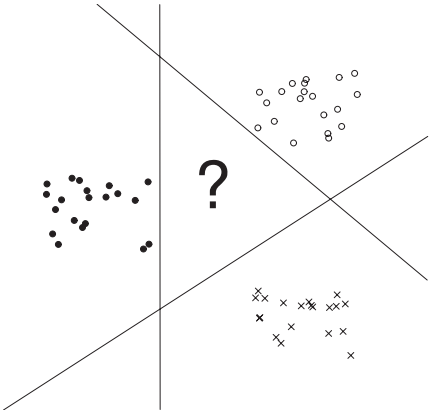

Figure 14.12 J hyperplanes do not divide space into J disjoint regions.

3. Assign the document to the class with
   - the maximum score,
   - the maximum confidence value,
   - or the maximum probability.

CONFUSION MATRIX

An important tool for analyzing the performance of a classifier for $J > 2$ classes is the *confusion matrix*. The confusion matrix shows for each pair of classes $\langle c_1, c_2 \rangle$, how many documents from c_1 were incorrectly assigned to c_2. In Table 14.5, the classifier manages to distinguish the three financial classes *money-fx*, *trade*, and *interest* from the three agricultural classes *wheat*, *corn*, and *grain*, but makes many errors within these two groups. The confusion matrix can help pinpoint opportunities for improving the accuracy of the system. For example, to address the second largest error in Table 14.5, one could attempt to introduce features that distinguish *wheat* documents from *grain* documents.

? **Exercise 14.5** Create a training set of 300 documents, 100 each from three different languages (e.g., English, French, and Spanish). Create a test set by the same procedure, but also add 100 documents from a fourth language. Train (i) a one-of classifier and (ii) an any-of classifier on this training set and evaluate it on the test set. (iii) Are there any interesting differences in how the two classifiers behave on this task?

Table 14.5 A confusion matrix for Reuters-21578. For example, fourteen documents from *grain* were incorrectly assigned to *wheat*. Adapted from Picca et al. (2006).

true class \ assigned class	*money-fx*	*trade*	*interest*	*wheat*	*corn*	*grain*
money-fx	95	0	10	0	0	0
trade	1	1	90	0	1	0
interest	13	0	0	0	0	0
wheat	0	0	1	34	3	7
corn	1	0	2	13	26	5
grain	0	0	2	14	5	10

14.6 The bias–variance tradeoff

Nonlinear classifiers are more powerful than linear classifiers. For some problems, there exists a nonlinear classifier with zero classification error, but no such linear classifier. Does that mean that we should always use nonlinear classifiers for optimal effectiveness in statistical text classification?

To answer this question, we introduce the bias–variance tradeoff in this section, one of the most important concepts in machine learning. The tradeoff helps to explain why there is no universally optimal learning method. Selecting an appropriate learning method is therefore an unavoidable part of solving a text classification problem.

Throughout this section, we use linear and nonlinear classifiers as prototypical examples of "less powerful" and "more powerful" learning, respectively. This is a simplification for a number of reasons. First, many nonlinear models subsume linear models as a special case. For instance, a nonlinear learning method like kNN will in some cases produce a linear classifier. Second, there are nonlinear models that are less complex than linear models. For instance, a quadratic polynomial with two parameters is less powerful than a 10,000-dimensional linear classifier. Third, the complexity of learning is not really a property of the classifier because there are many aspects of learning (such as feature selection, cf. Section 13.5, page 251 regularization, and constraints such as margin maximization in Chapter 15) that make a learning method either more powerful or less powerful without affecting the type of classifier that is the final result of learning-regardless of whether that classifier is linear or nonlinear. We refer the reader to the publications listed in Section 14.7 for a treatment of the bias–variance tradeoff that takes into account these complexities. In this section, linear and nonlinear classifiers will simply serve as proxies for weaker and stronger learning methods in text classification.

We first need to state our objective in text classification more precisely. In Section 13.1 (page 237), we said that we want to minimize classification error on the test set. The implicit assumption was that training documents and test documents are generated according to the same underlying distribution. We will denote this distribution by $P(\langle d, c\rangle)$ where d is the document and c its label or class. Figures 13.4 and 13.5 were examples of generative models that decompose $P(\langle d, c\rangle)$ into the product of $P(c)$ and $P(d|c)$. Figures 14.10 and 14.11 depict generative models for $\langle d, c\rangle$ with $d \in \mathbb{R}^2$ and $c \in$ {square, solid circle}.

In this section, instead of using the number of correctly classified test documents (or, equivalently, the error rate on test documents) as evaluation measure, we adopt an evaluation measure that addresses the inherent uncertainty of labeling. In many text classification problems, a given document representation can arise from documents belonging to different classes. This is because documents from different classes can be mapped to the

same document representation. For example, the one-sentence documents *China sues France* and *France sues China* are mapped to the same document representation $d' = \{\text{China, France, sues}\}$ in a bag-of-words model. But only the latter document is relevant to the class $c' =$ *legal actions brought by France* (which might be defined, for example, as a standing query by an international trade lawyer).

To simplify the calculations in this section, we do not count the number of errors on the test set when evaluating a classifier, but instead look at how well the classifier estimates the conditional probability $P(c|d)$ of a document being in a class. In the above example, we might have $P(c'|d') = 0.5$.

Our goal in text classification then is to find a classifier γ such that, averaged over documents d, $\gamma(d)$ is as close as possible to the true probability $P(c|d)$. We measure this using mean squared error:

$$\text{MSE}(\gamma) = E_d[\gamma(d) - P(c|d)]^2 \tag{14.6}$$

where E_d is the expectation with respect to $P(d)$. The mean squared error term gives partial credit for decisions by γ that are close if not completely right.

OPTIMAL CLASSIFIER

We define a classifier γ to be *optimal* for a distribution $P(\langle d, c\rangle)$ if it minimizes $\text{MSE}(\gamma)$.

Minimizing MSE is a desideratum for *classifiers*. We also need a criterion for *learning methods*. Recall that we defined a learning method Γ as a function that takes a labeled training set $\mathbb{D}$ as input and returns a classifier γ.

For learning methods, we adopt as our goal to find a Γ that, averaged over training sets, learns classifiers γ with minimal MSE. We can formalize this as minimizing *learning error*:

LEARNING ERROR

$$\text{learning-error}(\Gamma) = E_{\mathbb{D}}[\text{MSE}(\Gamma(\mathbb{D}))] \tag{14.7}$$

where $E_{\mathbb{D}}$ is the expectation over labeled training sets. To keep things simple, we can assume that training sets have a fixed size – the distribution $P(\langle d, c\rangle)$ then defines a distribution $P(\mathbb{D})$ over training sets.

We can use learning error as a criterion for selecting a learning method in statistical text classification. A learning method Γ is *optimal* for a distribution $P(\mathbb{D})$ if it minimizes the learning error.

OPTIMAL LEARNING METHOD

Writing $\Gamma_{\mathbb{D}}$ for $\Gamma(\mathbb{D})$ for better readability, we can transform Equation (14.7) as follows:

$$\begin{aligned}
\text{learning-error}(\Gamma) &= E_{\mathbb{D}}[\text{MSE}(\Gamma_{\mathbb{D}})] \\
&= E_{\mathbb{D}}E_d[\Gamma_{\mathbb{D}}(d) - P(c|d)]^2 && (14.10)\\
&= E_d[\text{bias}(\Gamma, d) + \text{variance}(\Gamma, d)] && (14.11)\\
\text{bias}(\Gamma, d) &= [P(c|d) - E_{\mathbb{D}}\Gamma_{\mathbb{D}}(d)]^2 && (14.12)\\
\text{variance}(\Gamma, d) &= E_{\mathbb{D}}[\Gamma_{\mathbb{D}}(d) - E_{\mathbb{D}}\Gamma_{\mathbb{D}}(d)]^2 && (14.13)
\end{aligned}$$

$$
\begin{aligned}
E[x-\alpha]^2 &= Ex^2 - 2Ex\alpha + \alpha^2 \\
&= (Ex)^2 - 2Ex\alpha + \alpha^2 \\
&\quad + Ex^2 - 2(Ex)^2 + (Ex)^2 \\
&= [Ex-\alpha]^2 \\
&\quad + Ex^2 - E2x(Ex) + E(Ex)^2 \\
&= [Ex-\alpha]^2 + E[x-Ex]^2
\end{aligned} \tag{14.8}
$$

$$
\begin{aligned}
E_{\mathbb{D}}E_d[\Gamma_{\mathbb{D}}(d) - P(c|d)]^2 &= E_d E_{\mathbb{D}}[\Gamma_{\mathbb{D}}(d) - P(c|d)]^2 \\
&= E_d[\ [E_{\mathbb{D}}\Gamma_{\mathbb{D}}(d) - P(c|d)]^2 \\
&\quad + E_{\mathbb{D}}[\Gamma_{\mathbb{D}}(d) - E_{\mathbb{D}}\Gamma_{\mathbb{D}}(d)]^2\ ]
\end{aligned} \tag{14.9}
$$

Figure 14.13 Arithmetic transformations for the bias–variance decomposition. For the derivation of Equation (14.9), we set $\alpha = P(c|d)$ and $x = \Gamma_{\mathbb{D}}(d)$ in Equation (14.8).

where the equivalence between Equations (14.10) and (14.11) is shown in Equation (14.9) in Figure 14.13. Note that d and $\mathbb{D}$ are independent of each other. In general, for a random document d and a random training set $\mathbb{D}$, $\mathbb{D}$ does not contain a labeled instance of d.

BIAS *Bias* is the squared difference between $P(c|d)$, the true conditional probability of d being in c, and $\Gamma_{\mathbb{D}}(d)$, the prediction of the learned classifier, averaged over training sets. Bias is large if the learning method produces classifiers that are consistently wrong. Bias is small if (i) the classifiers are consistently right or (ii) different training sets cause errors on different documents or (iii) different training sets cause positive and negative errors on the same documents, but that average out to close to 0. If one of these three conditions holds, then $E_{\mathbb{D}}\Gamma_{\mathbb{D}}(d)$, the expectation over all training sets, is close to $P(c|d)$.

Linear methods like Rocchio and Naive Bayes have a high bias for nonlinear problems because they can only model one type of class boundary, a linear hyperplane. If the generative model $P(\langle d, c\rangle)$ has a complex nonlinear class boundary, the bias term in Equation (14.11) will be high because a large number of points will be consistently misclassified. For example, the circular enclave in Figure 14.11 does not fit a linear model and will be misclassified consistently by linear classifiers.

We can think of bias as resulting from our domain knowledge (or lack thereof) that we build into the classifier. If we know that the true boundary between the two classes is linear, then a learning method that produces linear classifiers is more likely to succeed than a nonlinear method. But if the true class boundary is not linear and we incorrectly bias the classifier to be linear, then classification accuracy will be low on average.

Nonlinear methods like kNN have low bias. We can see in Figure 14.6 that the decision boundaries of kNN are variable – depending on the distribution

of documents in the training set, learned decision boundaries can vary greatly. As a result, each document has a chance of being classified correctly for some training sets. The average prediction $E_{\mathbb{D}}\Gamma_{\mathbb{D}}(d)$ is therefore closer to $P(c|d)$ and bias is smaller than for a linear learning method.

VARIANCE *Variance* is the variation of the prediction of learned classifiers: the average squared difference between $\Gamma_{\mathbb{D}}(d)$ and its average $E_{\mathbb{D}}\Gamma_{\mathbb{D}}(d)$. Variance is large if different training sets $\mathbb{D}$ give rise to very different classifiers $\Gamma_{\mathbb{D}}$. It is small if the training set has a minor effect on the classification decisions $\Gamma_{\mathbb{D}}$ makes, be they correct or incorrect. Variance measures how inconsistent the decisions are, not whether they are correct or incorrect.

Linear learning methods have low variance because most randomly drawn training sets produce similar decision hyperplanes. The decision lines produced by linear learning methods in Figures 14.10 and 14.11 will deviate slightly from the main class boundaries, depending on the training set, but the class assignment for the vast majority of documents (with the exception of those close to the main boundary) will not be affected. The circular enclave in Figure 14.11 will be consistently misclassified.

Nonlinear methods like kNN have high variance. It is apparent from Figure 14.6 that kNN can model very complex boundaries between two classes. It is therefore sensitive to noise documents of the sort depicted in Figure 14.10. As a result the variance term in Equation (14.11) is large for kNN: Test documents are sometimes misclassified – if they happen to be close to a noise document in the training set – and sometimes correctly classified – if there are no noise documents in the training set near them. This results in high variation from training set to training set.

OVERFITTING High-variance learning methods are prone to *overfitting* the training data. The goal in classification is to fit the training data to the extent that we capture true properties of the underlying distribution $P(\langle d, c\rangle)$. In overfitting, the learning method also learns from noise. Overfitting increases MSE and frequently is a problem for high-variance learning methods.

MEMORY CAPACITY We can also think of variance as the *model complexity* or, equivalently, *memory capacity* of the learning method – how detailed a characterization of the training set it can remember and then apply to new data. This capacity corresponds to the number of independent parameters available to fit the training set. Each kNN neighborhood S_k makes an independent classification decision. The parameter in this case is the estimate $\hat{P}(c|S_k)$ from Figure 14.7. Thus, kNN's capacity is only limited by the size of the training set. It can memorize arbitrarily large training sets. In contrast, the number of parameters of Rocchio is fixed – J parameters per dimension, one for each centroid – and independent of the size of the training set. The Rocchio classifier (in form of the centroids defining it) cannot "remember" fine-grained details of the distribution of the documents in the training set.

According to Equation (14.7), our goal in selecting a learning method is to minimize learning error. The fundamental insight captured by

Equation (14.11), which we can succinctly state as: learning error = bias + variance, is that the learning error has two components, bias and variance, which in general cannot be minimized simultaneously. When comparing two learning methods Γ_1 and Γ_2, in most cases the comparison comes down to one method having higher bias and lower variance and the other lower bias and higher variance. The decision for one learning method versus another is then not simply a matter of selecting the one that reliably produces good classifiers across training sets (small variance) or the one that can learn classification problems with very difficult decision boundaries (small bias). Instead, we have to weigh the respective merits of bias and variance in our application and choose accordingly. This tradeoff is called the *bias–variance tradeoff*.

BIAS–VARIANCE TRADEOFF

Figure 14.10 provides an illustration, which is somewhat contrived, but will be useful as an example for the tradeoff. Some Chinese text contains English words written in the Roman alphabet like CPU, ONLINE, and GPS. Consider the task of distinguishing Chinese–only web pages from mixed Chinese–English web pages. A search engine might offer Chinese users without knowledge of English (but who understand loanwords like CPU) the option of filtering out mixed pages. We use two features for this classification task: number of Roman alphabet characters and number of Chinese characters on the web page. As stated earlier, the distribution $P(\langle d, c\rangle)$ of the generative model generates most mixed (respectively, Chinese) documents above (respectively, below) the short-dashed line, but there are a few noise documents. In Figure 14.10, we see three classifiers.

- **One-feature classifier.** Shown as a dotted horizontal line. This classifier uses only one feature, the number of Roman alphabet characters. Assuming a learning method that minimizes the number of misclassifications in the training set, the position of the horizontal decision boundary is not greatly affected by differences in the training set (e.g., noise documents). So a learning method producing this type of classifier has low variance, but its bias is high because it will consistently misclassify squares in the lower left corner and "solid circle" documents with more than fifty Roman characters.
- **Linear classifier.** Shown as a dashed line with long dashes. Learning linear classifiers has less bias; only noise documents and possibly a few documents close to the boundary between the two classes are misclassified. The variance is higher than for the one-feature classifiers, but still small: The dashed line with long dashes deviates only slightly from the true boundary between the two classes, and so will almost all linear decision boundaries learned from training sets. Thus, very few documents (documents close to the class boundary) will be inconsistently classified.
- **"Fit-training-set-perfectly" classifier.** Shown as a solid line. Here, the learning method constructs a decision boundary that perfectly separates

the classes in the training set. This method has the lowest bias because there is no document that is consistently misclassified – the classifiers sometimes even get noise documents in the test set right. But the variance of this learning method is high. Because noise documents can move the decision boundary arbitrarily, test documents close to noise documents in the training set will be misclassified – something that a linear learning method is unlikely to do.

It is perhaps surprising that so many of the best-known text classification algorithms are linear. Some of these methods, in particular linear SVMs, regularized logistic regression and regularized linear regression, are among the most effective known methods. The bias–variance tradeoff provides insight into their success. Typical classes in text classification are complex and seem unlikely to be modeled well linearly. However, this intuition is misleading for the high-dimensional spaces that we typically encounter in text applications. With increased dimensionality, the likelihood of linear separability increases rapidly (Exercise 14.17). Thus, linear models in high-dimensional spaces are quite powerful despite their linearity. Even more powerful nonlinear learning methods can model decision boundaries that are more complex than a hyperplane, but they are also more sensitive to noise in the training data. Nonlinear learning methods sometimes perform better if the training set is large, but by no means in all cases.

? **Exercise 14.6** In Figure 14.14, which of the three vectors $\vec{a}$, $\vec{b}$, and $\vec{c}$ is (i) most similar to $\vec{x}$ according to dot product similarity, (ii) most similar to $\vec{x}$ according to cosine similarity, (iii) closest to $\vec{x}$ according to Euclidean distance?

Exercise 14.7 Download Reuters-21578 and train and test Rocchio and kNN classifiers for the classes *acquisitions*, *corn*, *crude*, *earn*, *grain*, *interest*, *money-fx*, *ship*, *trade*, and *wheat*. Use the ModApte split. You may want to use one of a number of software packages that implement Rocchio classification and kNN classification, for example, the Bow toolkit (McCallum 1996).

Exercise 14.8 Download 20 Newgroups (page 142) and train and test Rocchio and kNN classifiers for its twenty classes.

Exercise 14.9 Show that the decision boundaries in Rocchio classification are, as in kNN, given by the Voronoi tessellation.

Exercise 14.10 [⋆] Computing the distance between a dense centroid and a sparse vector is $\Theta(M)$ for a naive implementation that iterates over all M dimensions. Based on the equality $\sum(x_i - \mu_i)^2 = 1.0 + \sum \mu_i^2 - 2\sum x_i \mu_i$ and assuming that $\sum \mu_i^2$ has been precomputed, write down an algorithm that is $\Theta(M_a)$ instead, where M_a is the number of distinct terms in the test document.

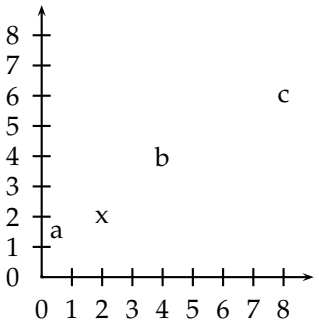

Figure 14.14 Example for differences between Euclidean distance, dot product similarity and cosine similarity. The vectors are $\vec{a} = (0.5\ \ 1.5)^T$, $\vec{x} = (2\ \ 2)^T$, $\vec{b} = (4\ \ 4)^T$, and $\vec{c} = (8\ \ 6)^T$.

Exercise 14.11 [⋆ ⋆ ⋆] Prove that the region of the plane consisting of all points with the same k nearest neighbors is a convex polygon.

Exercise 14.12 Design an algorithm that performs an efficient 1NN search in one dimension (where efficiency is with respect to the number of documents N). What is the time complexity of the algorithm?

Exercise 14.13 [⋆ ⋆ ⋆] Design an algorithm that performs an efficient 1NN search in two dimensions with at most polynomial (in N) preprocessing time.

Exercise 14.14 [⋆ ⋆ ⋆] Can one design an exact efficient algorithm for 1NN for very large M along the ideas you used to solve the last exercise?

Exercise 14.15 Show that Equation (14.4) defines a hyperplane with $\vec{w} = \vec{\mu}(c_1) - \vec{\mu}(c_2)$ and $b = 0.5 * (|\vec{\mu}(c_1)|^2 - |\vec{\mu}(c_2)|^2)$.

Exercise 14.16 We can easily construct nonseparable data sets in high dimensions by embedding a nonseparable set like the one shown in Figure 14.15. Consider embedding Figure 14.15 in 3D and then perturbing the four points slightly (i.e., moving them a small distance in a random direction). Why would you expect the resulting configuration to be linearly separable? How likely is then a non-separable set of $m \ll M$ points in M-dimensional space?

Exercise 14.17 Assuming two classes, show that the percentage of nonseparable assignments of the vertices of a hypercube decreases with dimensionality M for $M > 1$. For example, for $M = 1$ the proportion of nonseparable

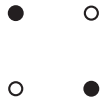

Figure 14.15 A simple nonseparable set of points.

assignments is 0, for $M = 2$, it is 2/16. One of the two nonseparable cases for $M = 2$ is shown in Figure 14.15, the other is its mirror image. Solve the exercise either analytically or by simulation.

Exercise 14.18 Although we point out the similarities of Naive Bayes with linear vector space classifiers, it does not make sense to represent count vectors (the document representations in NB) in a continuous vector space. There is however a formalization of NB that is analogous to Rocchio. Show that NB assigns a document to the class (represented as a parameter vector) whose Kullback-Leibler (KL) divergence (Section 12.4, page 231) to the document (represented as a count vector as in Section 13.4.1 (page 250), normalized to sum to 1) is smallest.

14.7 References and further reading

As discussed in Chapter 9, Rocchio relevance feedback is due to Rocchio (1971). Joachims (1997) presents a probabilistic analysis of the method. Rocchio classification was widely used as a classification method in TREC in the 1990s (Buckley et al. 1994a,b; Voorhees and Harman 2005). Initially, it was used as a form of *routing*. Routing merely ranks documents according to relevance to a class without assigning them. Early work on *filtering*, a true classification approach that makes an assignment decision on each document, was published by Ittner et al. (1995) and Schapire et al. (1998). The definition of routing we use here should not be confused with another sense. *Routing* can also refer to the electronic distribution of documents to subscribers, the so-called *push model* of document distribution. In a *pull model*, each transfer of a document to the user is initiated by the user, for example, by means of search or by selecting it from a list of documents on a news aggregation website.

ROUTING
FILTERING
PUSH MODEL
PULL MODEL

Some authors restrict the name *Roccchio classification* to two-class problems and use the terms *cluster-based* (Iwayama and Tokunaga 1995) and *centroid-based classification* (Han and Karypis 2000; Tan and Cheng 2007) for Rocchio classification with $J > 2$.

CENTROID-BASED CLASSIFICATION

A more detailed treatment of kNN can be found in (Hastie et al. 2001), including methods for tuning the parameter k. An example of an approximate fast kNN algorithm is locality-based hashing (Andoni et al. 2006). Kleinberg (1997) presents an approximate $\Theta((M \log^2 M)(M + \log N))$ kNN algorithm (where M is the dimensionality of the space and N the number of data points), but at the cost of exponential storage requirements: $\Theta((N \log M)^{2M})$. Indyk (2004) surveys nearest neighbor methods in high-dimensional spaces. Early work on kNN in text classification was motivated by the availability of massively parallel hardware architectures (Creecy et al. 1992). Yang (1994) uses an inverted index to speed up kNN classification. The optimality result

for 1NN (twice the Bayes error rate asymptotically) is due to Cover and Hart (1967).

The effectiveness of Rocchio classification and kNN is highly dependent on careful parameter tuning (in particular, the parameters b' for Rocchio on page 272 and k for kNN), feature engineering (Section 15.3, page 307) and feature selection (Section 13.5, page 251). Buckley and Salton (1995), Schapire et al. (1998), Yang and Kisiel (2003), and Moschitti (2003) address these issues for Rocchio and Yang (2001) and Ault and Yang (2002) for kNN. Zavrel et al. (2000) compare feature selection methods for kNN.

The bias–variance tradeoff was introduced by Geman et al. (1992). The derivation in Section 14.6 is for MSE(γ), but the tradeoff applies to many loss functions (cf. Friedman (1997), Domingos (2000)). Schütze et al. (1995) and Lewis et al. (1996) discuss linear classifiers for text and Hastie et al. (2001) linear classifiers in general. Readers interested in the algorithms mentioned, but not described in this chapter, may wish to consult Bishop (2006) for neural networks, Hastie et al. (2001) for linear and logistic regression, and Minsky and Papert (1988) for the perceptron algorithm. Anagnostopoulos et al. (2006) show that an inverted index can be used for highly efficient document classification with any linear classifier, provided that the classifier is still effective when trained on a modest number of features via feature selection.

We have only presented the simplest method for combining two-class classifiers into a one-of classifier. Another important method is the use of error-correcting codes, where a vector of decisions of different two-class classifiers is constructed for each document. A test document's decision vector is then "corrected" based on the distribution of decision vectors in the training set, a procedure that incorporates information from all two-class classifiers and their correlations into the final classification decision (Dietterich and Bakiri 1995). Ghamrawi and McCallum (2005) also exploit dependencies between classes in any-of classification. Allwein et al. (2000) propose a general framework for combining two-class classifiers.