14 Vector space classification

The document representation in Naive Bayes essentially is a vector of word counts. In this chapter, we adopt a different representation for classification, the vector space model, developed in Chapter 7. It represents each document as a vector with one real-valued component (e.g., a tf-idf weight) for each term. Thus, the instance space, the domain of the classification function $\gamma$, is $\mathbb{R}^{|V|}$ instead of $\mathbb{N}^{|V|}$. This chapter introduces a number of classification methods that operate on real-valued vectors.

The basic hypothesis in using the vector space model for classification is the contiguity hypothesis: documents in the same class form a contiguous region and regions of different classes don’t overlap. There are many classification tasks, in particular the type of topic 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 the 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 a line that separates them and classifies new documents. How exactly this is done is the main topic of this chapter.

The contiguity hypothesis is a fair characterization of the distribution of China and UK documents. It is easy to come up with examples, however, where the contiguity hypothesis does not hold. If the task is to decide whether the author of a text is male or female, then the distribution of documents corresponding to the two classes may well be indistinguishable.

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 etc. To see that the document representation is crucial, consider the two classes written by a group vs. written by a single person. Frequent occurrence of the first person pronoun I is evidence for the single-person class. But that information is deleted from the document representation if we use a stop list.
The similarity measure we employ also influences classification decisions. Consider a representation of web pages that consists of ordinary word features as well as hyperlink features. Hyperlink features (e.g., http://www.michaeljackson.com) are often better descriptors of content than word features (e.g., the words Michael Jackson). Thus, we prefer similarity measures that weight hyperlink features higher than word features. In this chapter, we use inner product similarity, the most commonly used similarity measure in vector space classification and weight all dimensions of the space equally.

This chapter introduces two vector space classification methods, Rocchio and kNN. Rocchio classification (Section 14.1) 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.

kNN or k nearest neighbor classification (Section 14.2) assigns the majority class of the k nearest neighbors to a test document. kNN requires no training in the sense of parameter estimation. If there exists a classifier with an error rate of 0, then 1NN will asymptotically approach that error rate as the training set increases. In practice, it is difficult to determine how close a given kNN classifier is to this optimum.

A large number of binary text classifiers can be viewed as simple linear classifiers – classifiers that compute a linear combination of the features and compare it to a threshold. Because of the bias-variance tradeoff (Section 14.3) 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 therefore more likely to make mistakes for small and noisy data.

When applying binary classifiers to problems with more than two classes, we distinguish one-of tasks – mutually exclusive classes – and any-of tasks – a document can be assigned to any number of classes (Section 14.3.1). Binary classifiers solve any-of problems and can be combined to solve one-of problems.

### 14.1 Rocchio classification

Figure 14.1 shows three classes, China, UK and Kenya, in a two-dimensional (2D) space. Documents are shown as circles, diamonds and X’s. The boundaries in the figure 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. Our task in vector space classification is to devise algorithms that compute good
14.1 Rocchio classification

Figure 14.1  Vector space classification into three classes. Documents are classified as being relevant to China (circles), UK (diamonds), and Kenya (X's). Regions are demarcated by lines. A new document (shown as a star) is assigned the class of the region it occurs in, the class China in this case.

boundaries where “good” means high classification accuracy on data unseen during training.

In Figure 14.1 and the other illustrations in this chapter, we generally show points and distances between points instead of vectors and similarities between vectors. But as we know from Chapter 7 (page 100), Euclidean distance and cosine similarity are equivalent for proximity ranking of normalized vectors, so there is no harm in explaining vector space classification in terms of distances. Also, normalized vectors live on the unit sphere, a complication that the figures do not show. But unit normalization just means that there is one effective dimension less, e.g., unit vectors in 3D only vary on two dimensions – the 2D surface of the 3D sphere; and unit vectors in 1000D only vary on 999 dimensions – the 999D surface of the 1000D hypersphere. For our purposes, it does not matter whether a set of documents with effective dimensionality $n - 1$ is represented as unit vectors in $\mathbb{R}^n$ or as unnormalized vectors in $\mathbb{R}^{n-1}$.

The main work we must do in vector space classification is to define the boundary lines between classes since they determine the classification decision. Perhaps the simplest way of doing this is to use centroids. The centroid
Figure 14.2 Rocchio classification. Centroids or prototypes (shown as solid circles) are computed as the average (or center of mass) of all points in a class. Boundaries between two classes are points of equal distance to the two centroids. For example, we have $a_1 = a_2, b_1 = b_2, c_1 = c_2$. The square is misclassified by Rocchio. It is a better fit for UK based on the overall distribution of points, but is closer to the centroid of Kenya.

Figure 14.2

A class $c$ is computed as the average or center of mass of its members:

$$\mu(c) = \frac{1}{|c|} \sum_{x \in c} x$$

(14.1)

Three example centroids are shown in Figure 14.2. The boundary between two classes is then the set of points with equal distance from the two centroids. We again classify points in accordance with the region they fall into. Equivalently, we determine which centroid the point is closest to and assign it to the class of that centroid. The algorithm is summarized in Figure 14.3.

The assignment criterion in Figure 14.3 is inner product similarity. An alternative is cosine similarity:

$$\text{Assign the document to class } c = \arg \max_{c_j} \cos(\mu(c_j), \tilde{d})$$

The two assignment criteria are different since the centroid is in general
14.1 Rocchio classification

**Training**
For each class $c_j$
- Compute centroid $\vec{\mu}(c_j)$

**Testing:** Classify test document $\vec{d}$
Assign the document to class $c = \arg \max_j (\vec{\mu}(c_j) \cdot \vec{d})$

Figure 14.3 Rocchio classification: Training and testing.

not a vector that has unit length. However, centroids in most applications in information retrieval are computed for small regions of the surface of a unit hypersphere, so the length of the centroid is approximately one. But if we compute centroids for vectors distributed over large regions of the hypersphere, then their lengths can be very different from 1.0. In particular, the length of the centroid of two diametrically opposed vectors is zero. We present the inner product version of Rocchio classification here because it is simpler and more efficient. We need not compute the length of the Rocchio vectors as would be required by length normalization. We will also use centroids for clustering in Chapter 17. There, inner product similarity allows a cleaner interpretation of a number of hierarchical clustering algorithms than cosine similarity.

This centroid-based classification algorithm is a form of Rocchio relevance feedback (Section 9.1.1, page 136). It is therefore called **Rocchio classification**. The average of the relevant documents, corresponding to the most important component of the Rocchio vector in relevance feedback (Equation (9.3), page 139), is the centroid of the “class” of relevant documents. Centroids are also called *prototypes* in this context. Rocchio classification can be applied to $J > 2$ classes whereas Rocchio relevance feedback is designed to distinguish only two classes, relevant and non-relevant.

In addition to respecting contiguity, the classes in Rocchio classification must also be of approximately spherical shape. In Figure 14.2, the solid square just below the boundary between UK and Kenya should intuitively be part of UK since UK is more scattered than Kenya. But the Rocchio classifier will put it in the Kenya category because details of the distribution of points in a class are ignored. Only the centroid is used for classification.

An even worse case for the assumption of sphericity is shown in Figure 14.4. The X class cannot be represented well with a single prototype because it has two centers. This type of *polymorphic class* will often be misclassified by Rocchio. A text classification example for polymorphism 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 encountered the same problem with polymorphism in relevance
feedback (Chapter 9, page 141).

The Rocchio classifier can even misclassify documents in the training set because of its strict assumptions about how documents in a class are distributed (Exercise 14.5). In the next section, we will introduce a vector space classifier, kNN, that deals better with classes that have non-spherical, disconnected or other irregular shapes.

Table 14.1 gives the time complexity of Rocchio classification. Computing either the sum or the inner product of a sparse vector with a non-sparse vector (such as a centroid) takes time proportional to the number of non-zero entries in the sparse vector. Adding one document to a centroid is therefore $O(L_d)$ and computing one inner product is $O(L_t)$. Overall, training time is linear in the size of the collection. Rocchio classification and Naive Bayes
14.2 k nearest neighbor

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. Decision boundaries are concatenated segments of the Voronoi tessellation as shown in Figure 14.5. 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 cell boundary of a Voronoi cell is a sequence of linear segments (or hyperplanar segments in higher dimensions).

1NN is not very robust. The classification decision of each document relies on the class of a single document, which may be incorrectly labeled or atypical. For improved robustness, we classify based on a neighborhood of the \( k > 1 \) closest neighbors, assigning documents to the majority class of their \( k \) closest neighbors in its categorical form, with ties broken randomly. In the probabilistic version of the method, the probability of membership in the
14 Vector space classification

Training
Preprocess documents in training set
Determine $k$ (e.g., optimal $k$ on held-out data)

Testing: Classify test document $\vec{d}$
Compute the similarity of all training documents with $\vec{d}$
Identify the set $S_k$ of the $k$ most similar training documents
For each class $c$
Compute $N(S_k, c)$, the number of members of $S_k$ in $c$
Estimate $\hat{P}(c|\vec{d})$ as $N(S_k, c)/|S_k|$

$\Rightarrow$ Figure 14.6 kNN training and testing.

<table>
<thead>
<tr>
<th>$D$</th>
<th>training set</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_d$</td>
<td>average length of a training document</td>
</tr>
<tr>
<td>$O(1)$</td>
<td>training time (without preprocessing)</td>
</tr>
<tr>
<td>$O(</td>
<td>D</td>
</tr>
</tbody>
</table>

| $L_t$ | average length of a test document |
| $m_t$ | average vocabulary size of a test document |
| $O(L_t + |D|L_d)$ | test time (docs were not preprocessed) |
| $O(L_t + |D|m_t)$ | test time (docs were preprocessed) |

$\Rightarrow$ Table 14.2 Training and test time for kNN classification.

class is estimated as the proportion of the $k$ nearest neighbors in the class. Figure 14.5 gives an example of kNN classification for $k = 3$. Class membership probabilities of the star are estimated as $\hat{P}$(circle class|star) = 1/3, $\hat{P}$(X class|star) = 2/3, and $\hat{P}$(diamond class|star) = 0.

To determine the parameter $k$ in kNN we can select the $k$ that maximizes effectiveness on a held-out subset of the training set. But often $k$ is chosen based on experience or knowledge about the classification problem at hand. In that case, $k$ will usually be odd to make ties less likely. $k = 3$ and $k = 5$ are common choices.

Figure 14.6 summarizes the kNN algorithm and Table 14.2 gives its complexity. The time complexity of kNN has properties that are quite different from most other classification algorithms. First, test time is independent of the number of classes $J$. kNN has therefore a potential advantage for problems with large $J$. Secondly, test time is $O(|D|)$ – linear in the size of the training set as we need to compute the similarity of each training document with the test document. Training a kNN classifier simply consists of determining $k$ and document preprocessing. 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 at some point. It
makes more sense to preprocess once as part of the training phase rather than repeatedly every time we classify a document.

Testing in kNN classification consists of estimating class probabilities based on the $k$-neighborhood $S_k$ of the test document. Time complexity is independent of $k$.$^1$ We rely solely on “memorizing” all examples in the training set and comparing 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.

Can kNN be made more efficient? There are fast kNN algorithms for small $k$ (Exercise 14.10). And there are approximations for large $k$ that give error bounds for specific efficiency gains (see Section 14.4). These approximations have not been extensively tested for text classification applications, so it is not clear whether they can achieve much better efficiency than $O(|D| m)$ without a significant loss of accuracy.

The attentive reader may have noticed the similarity between the problem of finding nearest neighbors of a test document and adhoc retrieval, where we search for the documents with the highest similarity to the query (Section 7.1.2, page 100). In fact, the two problems are both $k$ nearest neighbor problems and only differ in the sparseness of the test document in kNN (10s or 100s of non-zero entries) and the query in adhoc retrieval (usually fewer than 10 non-zero entries). We introduced the inverted index for efficient adhoc retrieval in Chapter 1 (page 4). Is the inverted index also the solution for efficient kNN?

Searching for the best matches in the inverted index is restricted to those documents that have at least one term in common with the query. Thus, the inverted index will be efficient if the test document has no term overlap with a large number of training documents. How often 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.

It is easy to construct synthetic data sets with sublinear search times. For example, search time is constant in the size of the collection if, as we generate new documents in the collection, terms are used for a period of time and then replaced by the next generation of terms. In this case, the lengths of inverted lists and hence search times are bounded by a constant. However, this type of word distribution is very improbable. In practice, kNN search is probably always $O(|D|)$ when implemented with an inverted index.

As we will see in the next chapter, kNN effectiveness is often close to the results for the best-performing classifiers in text classification (Table 15.2,

---

1. Strictly speaking, classification requires an $O(k \log k)$ sorting step. This component is so small compared to $O(|D|)$ that we ignore it here.
Figure 14.7 There is an infinite number of hyperplanes that separate two linearly separable classes.

page 248). However, the theoretical bounds for kNN are not as favorable for problems with an inherent error rate. The error of 1NN is asymptotically bounded by twice the Bayes’ error, that is, if the optimal classifier has an error rate of $x$, then 1NN has an asymptotic error rate of less than $2x$. Intuitively, this is the case because 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 0, the error rate of 1NN will approach 0 as the size of the training set increases.

14.3 Linear vs. nonlinear classifiers and the bias-variance tradeoff

The kNN classifier is nonlinear – it can model an arbitrarily complex decision boundary provided the training set is large enough. There is a tradeoff between the expressive power of such a complex nonlinear model and its accuracy on new data. In contrast, Rocchio and Naive Bayes are linear classifiers as we will show below. They decide class membership based on a linear combination of features. Their expressive power is low, but they are less sensitive to noise in small training sets if the linearity assumption is correct.

Rocchio, Naive Bayes and kNN exemplify the bias-variance tradeoff. We can think of bias as a kind of domain knowledge that we build into a classifier. If we know that the decision boundary is linear, then the linear bias will improve generalization. If the decision boundary is not linear and we incorrectly bias the classifier, then classification accuracy will be low.
14.3 Linear vs. nonlinear classifiers and the bias-variance tradeoff

<table>
<thead>
<tr>
<th>( w_i )</th>
<th>( x_i )</th>
<th>( w_i )</th>
<th>( x_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.70</td>
<td>prime</td>
<td>-0.71</td>
<td>dlr</td>
</tr>
<tr>
<td>0.67</td>
<td>rate</td>
<td>-0.35</td>
<td>world</td>
</tr>
<tr>
<td>0.63</td>
<td>interest</td>
<td>-0.33</td>
<td>sees</td>
</tr>
<tr>
<td>0.60</td>
<td>rates</td>
<td>-0.25</td>
<td>year</td>
</tr>
<tr>
<td>0.46</td>
<td>discount</td>
<td>-0.24</td>
<td>group</td>
</tr>
<tr>
<td>0.43</td>
<td>bundesbank</td>
<td>-0.24</td>
<td>dlr</td>
</tr>
</tbody>
</table>

Table 14.3 A linear classifier. The variables \( x_i \) and parameters \( w_i \) of a linear classifier for the class interest (as in interest rate) in Reuters-21578. The threshold is \( b = 0 \). Words like dlr and world have negative parameter weights because they are indicators for the competing class currency.

Rocchio and Naive Bayes are just two members of a large group of linear classification methods. In this section, we only consider binary classifiers, which decide between two classes (e.g., China and non-China). Linear classifiers are the simplest binary classifiers. In two dimensions, a linear classifier is a line. Five example lines are shown in Figure 14.7.

The lines in Figure 14.7 have the functional form \( w_1 x_1 + w_2 x_2 = b \). The classification rule of a linear classifier is to assign a document to class 1 if \( w_1 x_1 + w_2 x_2 > b \) and to class 2 if \( w_1 x_1 + w_2 x_2 \leq b \). Here, \( (x_1, x_2) \) is the two-dimensional vector representation of the document and \( (w_1, w_2) \) is the parameter vector defining (together with \( b \)) the decision boundary.

A hyperplane is given by:

\[
\mathbf{w}^T \mathbf{x} = b
\]

where \( \mathbf{w} \) is referred to as the normal vector and the criteria for assignment to classes 1 and 2 are \( \mathbf{w}^T \mathbf{x} > b \) and \( \mathbf{w}^T \mathbf{x} \leq b \), respectively. This definition of hyperplanes includes lines and planes. We call a hyperplane that is used for classification a decision hyperplane.

We now show that Rocchio and Naive Bayes are linear. To see this for Rocchio, observe that a document is on the decision boundary if its two inner products with the class centroids are equal:

\[
\mu(c_1) \cdot \bar{d} = \mu(c_2) \cdot \bar{d}
\]

\[
\Leftrightarrow \ (\mu(c_1) - \mu(c_2))^T \cdot \bar{d} = 0
\]

This corresponds to a linear classifier with normal vector \( \mathbf{w} = \mu(c_1) - \mu(c_2) \) and \( b = 0 \).
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 13.3, page 201) where:

\[
\hat{P}(c|d) = \hat{P}(c) \prod_{i \in S} \hat{P}(x_i|c)
\]

and \( S \) is the set of positions in the document. Denoting the complement category as \( \bar{c} \), we obtain for the log odds:

\[
\log \frac{\hat{P}(c|d)}{\hat{P}(\bar{c}|d)} = \log \frac{\hat{P}(c)}{\hat{P}(\bar{c})} + \sum_{k \in S} \log \frac{\hat{P}(x_i|c)}{\hat{P}(x_i|\bar{c})}
\]

(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 immediately obvious that Equation (14.5) is an instance of Equation (14.2) for \( w = \log [\hat{P}(x_i|c)/\hat{P}(x_i|\bar{c})] \), \( x_i = \) number of occurrences of \( w_i \) in \( d \), and \( b = -\log [\hat{P}(c)/\hat{P}(\bar{c})] \). So in log space, Naive Bayes is indeed a linear classifier.

Table 14.3 gives an example of a linear classifier for the category interest in Reuters-21578 (see Chapter 13, page 210). The document \( \vec{x}_1 \) “rate discount dlrs world” will be assigned to interest since \( \vec{w}^T \vec{x}_1 = 0.67 \cdot 1 + 0.46 \cdot 1 + (-0.71) \cdot 1 + (-0.35) \cdot 1 = 0.05 > 0 = b \). For simplicity, we assume a simple binary vector representation in this example: 1 for occurring words, 0 for non-occurring words. Document \( \vec{x}_2 \) “prime dlrs” would be assigned to the other class (not in interest) since \( \vec{w}^T \vec{x}_2 = -0.01 \leq b \).

Training a linear classifier amounts to identifying a separating hyperplane between the two classes. This is only possible if the two classes are linearly separable, that is, at least one separating hyperplane exists. In fact, if linear separability holds, then there is an infinite number of linear separators (Exercise 14.12). Five linear separators are shown in the example in Figure 14.7.

There are simple and efficient algorithms for finding a separating hyperplane, such as the perceptron algorithm (see Section 14.4), but they do not necessarily produce good classifiers. The challenge is to choose a hyperplane that has good classification effectiveness on the test set according to \( F_1 \), misclassification rate or a similar criterion. There are two broad classes of methods for making this selection:

- Methods that rely on the global distribution of documents. These methods are robust since atypical and noisy data are less likely to affect them. Naive Bayes, Rocchio and linear regression (see Section 14.4) are methods of this type.
- Methods that give more weight to documents close to the decision boundary. These methods are more accurate if enough training data of sufficient quality is available. Support vector machines (with linear kernel, see Chapter 15) and logistic regression (see Section 14.4) belong to this category.
14.3 Linear vs. nonlinear classifiers and the bias-variance tradeoff

It is perhaps surprising that so many of the best-known text classification algorithms are linear: perceptron, Naive Bayes, Rocchio, linear regression, logistic regression, and linear SVM. The simple neural network, a neural network without hidden layers, is also linear (see Section 14.4). Some of these methods, in particular regularized logistic regression and regularized linear regression, are among the best-performing text classifiers. Why is it that more complex models, models that can learn nonlinear decision boundaries, are not clearly superior to linear models?

The reason is the bias-variance tradeoff. Linear classifiers are high-bias and low-variance, nonlinear classifiers are low-bias and high-variance. Bias is the restrictiveness of the model of a classifier. Linear classifiers can only model one type of decision boundary, a hyperplane. The high-variance classifier kNN can – given enough training data – model an arbitrarily complex decision boundary.

Variance refers to the variability of classification behavior when we train on different randomly generated training sets. NB’s decision boundary is always a hyperplane and hence less variable. In contrast, kNN’s more complex boundary will vary much more depending on the sample we are training on – especially if the distribution is noisy and the sample is small.

But variance also corresponds to capacity – the number of independent parameters available to fit the data or, loosely speaking, the memory available to store information about 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 14.5. Thus, kNN’s capacity is unlimited in principle if enough training data is available. The capacity of NB is fixed and independent of the training set. It can only model a hyperplane. If the decision boundary is not linear and NB therefore incorrectly biased, then the greater capacity of kNN will carry the day (assuming a sufficiently large training set). Asymptotically, it will perfectly model the nonlinear decision boundary, something NB cannot do.

Figure 14.8 provides a (somewhat contrived) illustration of 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. In Figure 14.8, we see three classifiers:

- A high-bias low-variance classifier that only uses one feature, in this case the number of Roman alphabet characters. This classifier does not fit the distribution of the data and will not generalize well to new data, in par-
Figure 14.8 The bias-variance tradeoff. In this hypothetical web page classification scenario, we want to discriminate between Chinese-only web pages (circles) and mixed Chinese-English web pages (squares). The linear classifier (linear combination of two features, dashed line) offers the best tradeoff, with better generalization than the high-bias low-variance classifier (restricted to using one feature, dotted line) and the low-bias high-variance classifier (arbitrary decision boundary, solid line).

- A medium-bias medium-variance linear classifier using a linear combination of the two features. This classifier makes some mistakes on the training set, but captures the underlying generalization best: Pages with roughly more than 10% Roman characters are mixed pages.

- A low-bias high-variance classifier that selects a decision boundary that separates the two classes perfectly. Even though its accuracy on the training data is optimal, it is influenced by outliers and will not generalize well to new data.
14.3 Linear vs. nonlinear classifiers and the bias-variance tradeoff

The low-bias high-variance classifier *overfits* 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. In overfitting, the classifier also learns from the noise present in the training set. In Figure 14.8, this noise consists of the three outliers that do not fit into the general pattern of their respective classes. Overfitting decreases accuracy on new data and is a typical problem for high-variance classifiers.

The bias-variance tradeoff provides insight into why linear classifiers often perform best on text classification problems. Typical classes in text classification are complex and seem unlikely to be modelled well linearly. However, our intuitions are misleading for the high-dimensional spaces that we typically encounter in text applications. With increased dimensionality, the likelihood of linear separability increases rapidly (Exercise 14.14). Thus, linear models in high-dimensional spaces are quite powerful despite their linearity. More powerful nonlinear classifiers can model decision boundaries more complex than a hyperplane, but they are more sensitive to noise in the training data. Nonlinear classifiers may perform better if the training set is large, but by no means in all cases.

14.3.1 More than two classes

Linear classifiers are binary, that is, they distinguish two classes. What do we do if there are $J > 2$ different classes? This depends on whether the classes are mutually exclusive or not.

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 since the classes are rarely statistically independent. In terms of the formal definition of the classification problem in Equation (13.1) (page 195), we define $J$ different classification functions $\gamma_j$ for any-of classification, each returning either yes or no for its class: $\gamma_j(\vec{d}) \in \{0, 1\}$.

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

- 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).

- Given the test document, apply each classifier separately. The decision of one classifier has no influence on the decisions of the other classifiers.

The second type of classification with more than two classes is *one-of clas-
Figure 14.9  Linear classification for one-of classification with $J > 2$. A combination method has to be applied in this case since $J$ hyperplanes do not divide the space into $J$ disjoint regions.

sification. Here, the classes are mutually exclusive. Each document must belong to exactly one of the classes. One-of classification is also called multiclass classification. Formally, there is a single classification function $\gamma$ in one-of classification whose range is $C$, i.e., $\gamma(d) \in \{c_1, \ldots, c_J\}$. Naïve Bayes, Rocchio and kNN are one-of classifiers.

True one-of problems are less common in text classification than any-of problems. In topic classification (with classes like UK, China, poultry, or coffee), a document can obviously be relevant to any of the topics – 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 language identification the one-of assumption is also approximately true as most text is written in only one language. If the one-of assumption holds to an approximation, then it simplifies the classification problem without a large decrease in classification accuracy.

Linear classifiers must be used with a combination method for one-of classification as illustrated in Figure 14.9. $J$ hyperplanes do not divide $\mathbb{R}^{|V|}$ into the $J$ different regions we would need for one-of classification.

We need a combination method that performs some kind of category ranking and then selects the top-ranked class. Geometrically, the ranking can be with respect to the distances from the linear separators (one for each class).

2. A synonym of polytomous is polychotomous.
14.3 Linear vs. nonlinear classifiers and the bias-variance tradeoff

### Table 14.4

- A confusion matrix for Reuters-21578. Example: 14 documents from grain were incorrectly assigned to wheat. Adapted from Picca et al. (2006).

<table>
<thead>
<tr>
<th>true class</th>
<th>assigned class</th>
<th>money-fx</th>
<th>trade</th>
<th>interest</th>
<th>wheat</th>
<th>corn</th>
<th>grain</th>
</tr>
</thead>
<tbody>
<tr>
<td>money-fx</td>
<td>95 0 10 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trade</td>
<td>1 1 90 0 1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>interest</td>
<td>13 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wheat</td>
<td>0 0 1 34 3 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>corn</td>
<td>1 0 2 13 26 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>grain</td>
<td>0 0 2 14 5 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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, a direct measure of confidence can be used to rank classes, e.g., probability of class membership. The algorithm for one-of classification with linear classifiers can be summarized as follows:

- 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).
- Given the test document, apply each classifier separately.
- Assign the document to the class with
  - the maximum score,
  - the maximum confidence value,
  - or the maximum probability.

An important tool for analyzing the performance of a one-of classification system is the confusion matrix. The confusion matrix shows for each pair of classes \( c_1, c_2 \), how many documents from \( c_1 \) were incorrectly assigned to \( c_2 \) and vice versa. In Table 14.4, the classifiers manage to distinguish the three financial classes money-fx, trade, and interest from the three agricultural classes wheat, corn, and grain, but make 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 largest error in Table 14.4, one could attempt to introduce features that distinguish wheat documents from grain documents.
14.4 References and further reading

As discussed in Chapter 9, Rocchio relevance feedback is due to Rocchio (Rocchio 1971). It was widely used as a classification method in the TREC competition in the 1990s (Buckley et al. 1994a, Voorhees and Harman 2005). Initially, Rocchio classification was 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).

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. 2007). Kleinberg (1997) presents an approximate $O((M \log^2 M)(M + \log N))$ 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: $O((N \log M)^{2M})$. Yang (1994) uses an inverted index to speed up kNN classification. The optimality result for 1NN (twice the Bayes error asymptotically) is due to Cover and Hart (1967).


We have only presented the simplest method for combining binary classifiers into a multiclass classifier. Another important method is error-correcting codes, where a vector of decisions of different binary classifiers is constructed for each document. A test document’s vector is then “corrected” based on the distribution of decision vectors in the training set, a procedure that incorporates information from all binary classifiers and their correlations into the final classification decision (Dietterich and Bakiri 1995). Allwein et al. (2000) propose a general framework for combining binary classifiers.

14.5 Exercises

Exercise 14.1

In Figure 14.10, which of the three vectors $\vec{a}$, $\vec{b}$, and $\vec{c}$ is (i) most similar to $\vec{x}$ according to inner product similarity, (ii) most similar to $\vec{x}$ according to cosine similarity, (iii) closest to $\vec{x}$ according to Euclidean distance?
Figure 14.10 Example for differences between Euclidean distance, inner product similarity and cosine similarity. The coordinates of the vectors are (0.1,0.15) (a), (0.2,0.2) (x), (0.4,0.4) (b), and (0.8,0.6) (c).

Exercise 14.2
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 14.3
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 14.4
Create a training set of 300 documents, 100 each from three different languages (e.g., English, French, Spanish). Create a test set by the same procedure and then add 100 documents from a fourth language. (i) Build a one-of classifier that identifies the language of a document and evaluate it on the test set. (ii) Build an any-of classifier that identifies the language of a document and evaluate it on the test set.

Exercise 14.5
Show that Rocchio classification can assign a label to a document that is different from its training set label.

Exercise 14.6
Show that the decision boundaries in Rocchio classification are, as in kNN, a subset of the Voronoi tessellation.

Exercise 14.7
kNN decision boundaries are concatenations of linear segments for $k = 1$. Is this also true for $k > 1$?
Exercise 14.8
Why are ties unlikely for odd $k$ in kNN?

Exercise 14.9
Explain why kNN handles polymorphic classes better than Rocchio.

Exercise 14.10
Design an algorithm that performs an efficient kNN search in (i) 1 dimension, (ii) 2 dimensions.

Exercise 14.11
Can one design an exact efficient algorithm for kNN for very large $k$ along the ideas you used to solve the last exercise?

Exercise 14.12
Prove that the number of linear separators of two classes is either infinite or zero.

Exercise 14.13
We can easily construct non-separable data sets in high dimensions by embedding a non-separable set like the one shown in Figure 14.11. Explain why such an embedded configuration is likely to become separable in high dimensions after adding noise.

Exercise 14.14
Make plausible that non-separable data sets become unlikely with increasing dimensionality by showing that the percentage of non-separable assignments of the vertices of a hypercube to two classes decreases with dimensionality for $d > 1$. For example, for $d = 1$ the number of non-separable assignments is 0, for $d = 2$, it is 2/16. One of the two non-separable cases for $d = 2$ is shown in Figure 14.11, the other is its mirror image. Solve the exercise either analytically or by simulation.

Exercise 14.15
Although we point out the similarities of Naive Bayes with linear vector space classifiers, it does not really 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 KL-divergence to the document (represented as a count vector, normalized to be a probability distribution) is smallest.