3

Content-based recommendation

From our discussion so far we see that for applying collaborative filtering techniques, except for the user ratings, nothing has to be known about the items to be recommended. The main advantage of this is, of course, that the costly task of providing detailed and up-to-date item descriptions to the system is avoided. The other side of the coin, however, is that with a pure collaborative filtering approach, a very intuitive way of selecting recommendable products based on their characteristics and the specific preferences of a user is not possible: in the real world, it would be straightforward to recommend the new *Harry Potter* book to Alice, if we know that (a) this book is a fantasy novel and (b) Alice has always liked fantasy novels. An electronic recommender system can accomplish this task only if two pieces of information are available: a description of the item characteristics and a *user profile* that somehow describes the (past) interests of a user, maybe in terms of preferred item characteristics. The recommendation task then consists of determining the items that match the user's preferences best. This process is commonly called *content-based recommendation*. Although such an approach must rely on additional information about items and user preferences, it does not require the existence of a large user community or a rating history – that is, recommendation lists can be generated even if there is only one single user.

In practical settings, *technical* descriptions of the features and characteristics of an item – such as the genre of a book or the list of actors in a movie – are more often available in electronic form, as they are partially already provided by the providers or manufacturers of the goods. What remains challenging, however, is the acquisition of subjective, *qualitative* features. In domains of quality and taste, for example, the reasons that someone likes something are not always related to certain product characteristics and may be based on a subjective impression of the item's exterior design. One notable and exceptional

endeavor in that context is the "Music Genome Project"[1], whose data are used by the music recommender on the popular Internet radio and music discovery and commercial recommendation site Pandora.com. In that project, songs are manually annotated by musicians with up to several hundred features such as instrumentation, influences, or instruments. Such a manual acquisition process – annotating a song takes about twenty to thirty minutes, as stated by the service providers – is, however, often not affordable.

We will refer to the descriptions of the item characteristics as "content" in this chapter, because most techniques described in the following sections were originally developed to be applied to recommending interesting text documents, such as newsgroup messages or web pages. In addition, in most of these approaches the basic assumption is that the characteristics of the items can be automatically extracted from the document content itself or from unstructured textual descriptions. Typical examples for content-based recommenders are, therefore, systems that recommend news articles by comparing the main keywords of an article in question with the keywords that appeared in other articles that the user has rated highly in the past. Correspondingly, the recommendable items will be often referred to as "documents".

There is no exact border between content-based and knowledge-based systems in the literature; some authors even see content-based approaches as a subset of knowledge-based approaches. In this book, we follow the traditional classification scheme, in which content-based systems are characterized by their focus on exploiting the information in the item descriptions, whereas in knowledge-based systems there typically exists some sort of additional means–end knowledge, such as a utility function, for producing recommendations.

In this chapter we discuss content-based recommendation, focusing particularly on algorithms that have been developed for recommending textually described items and for "learning" the user profile automatically (instead of explicitly asking the user for his or her interests, which is more common in conversational, knowledge-based systems).

3.1 Content representation and content similarity

The simplest way to describe catalog items is to maintain an explicit list of *features* for each item (also often called *attributes*, *characteristics*, or *item profiles*). For a book recommender, one could, for instance, use the genre, the author's name, the publisher, or anything else that describes the item and store

[1] *http://www.pandora.com/mgp.shtml*.

Table 3.1. *Book knowledge base.*

Title	Genre	Author	Type	Price	Keywords
The Night of the Gun	Memoir	David Carr	Paperback	29.90	press and journalism, drug addiction, personal memoirs, New York
The Lace Reader	Fiction, Mystery	Brunonia Barry	Hardcover	49.90	American contemporary fiction, detective, historical
Into the Fire	Romance, Suspense	Suzanne Brockmann	Hardcover	45.90	American fiction, murder, neo-Nazism
...					

this information in a relational database system. When the user's preferences are described in terms of his or her interests using exactly this set of features, the recommendation task consists of matching item characteristics and user preferences.

Consider the example in Table 3.1, in which books are described by characteristics such as title, genre, author, type, price, or keywords. Let us further assume that Alice's preferences are captured in exactly the same dimensions (Table 3.2).

A book recommender system can construct Alice's profile in different ways. The straightforward way is to explicitly ask Alice, for instance, for a desired price range or a set of preferred genres. The other way is to ask Alice to rate a set of items, either as a whole or along different dimensions. In the example, the set of preferred genres could be defined manually by Alice, whereas the system may automatically derive a set of keywords from those books that Alice liked, along with their average price. In the simplest case, the set of keywords corresponds to the set of all terms that appear in the document.

Table 3.2. *Preference profile.*

Title	Genre	Author	Type	Price	Keywords
...	Fiction, Suspense	Brunonia Barry, Ken Follett	Paperback	25.65	detective, murder, New York

https://doi.org/10.1017/CBO9780511763113.005 Published online by Cambridge University Press

To make recommendations, content-based systems typically work by evaluating how strongly a not-yet-seen item is "similar" to items the active user has liked in the past. Similarity can be measured in different ways in the example. Given an unseen book B, the system could simply check whether the genre of the book at hand is in the list of Alice's preferred genres. Similarity in this case is either 0 or 1. Another option is to calculate the similarity or overlap of the involved keywords. As a typical similarity metric that is suitable for multi-valued characteristics, we could, for example, rely on the Dice coefficient[2] as follows: If every book B_i is described by a set of keywords $keywords(B_i)$, the Dice coefficient measures the similarity between books b_i and b_j as

$$\frac{2 \times |keywords(b_i) \cap keywords(b_j)|}{|keywords(b_i)| + |keywords(b_j)|} \tag{3.1}$$

In principle, depending on the problem at hand, various similarity measures are possible. For instance, in Zanker et al. (2006) an approach is proposed in which several similarity functions for the different item characteristics are used. These functions are combined and weighted to calculate an overall similarity measure for cases in which both structured and unstructured item descriptions are available.

3.1.1 The vector space model and TF-IDF

Strictly speaking, the information about the publisher and the author are actually not the "content" of a book, but rather additional knowledge about it. However, content-based systems have historically been developed to filter and recommend text-based items such as e-mail messages or news. The standard approach in content-based recommendation is, therefore, not to maintain a list of "meta-information" features, as in the previous example, but to use a list of relevant keywords that appear within the document. The main idea, of course, is that such a list can be generated automatically from the document content itself or from a free-text description thereof.

The content of a document can be encoded in such a keyword list in different ways. In a first, and very naïve, approach, one could set up a list of all words that appear in all documents and describe each document by a Boolean vector, where a 1 indicates that a word appears in a document and a 0 that the word does not appear. If the user profile is described by a similar list (1 denoting interest

[2] For other measures, see, e.g., Maimon and Rokach (2005) or Baeza-Yaks and Ribaro-Nato (1999).

https://doi.org/10.1017/CBO9780511763113.005 Published online by Cambridge University Press

in a keyword), document matching can be done by measuring the overlap of interest and document content.

The problems with such a simple approach are obvious. First, the simple encoding is based on the assumption that every word has the same importance within a document, although it seems intuitive that a word that appears more often is better suited for characterizing the document. In addition, a larger overlap of the user profile and a document will naturally be found when the documents are longer. As a result, the recommender will tend to propose long documents.

To solve the shortcomings of the simple Boolean approach, documents are typically described using the TF-IDF encoding format (Salton et al. 1975). TF-IDF is an established technique from the field of information retrieval and stands for *term frequency-inverse document frequency*. Text documents can be TF-IDF encoded as vectors in a multidimensional Euclidian space. The space dimensions correspond to the keywords (also called *terms* or *tokens*) appearing in the documents. The coordinates of a given document in each dimension (i.e., for each term) are calculated as a product of two submeasures: term frequency and inverse document frequency.

Term frequency describes how often a certain term appears in a document (assuming that important words appear more often). To take the document length into account and to prevent longer documents from getting a higher relevance weight, some normalization of the document length should be done. Several schemes are possible (see Chakrabarti 2002, Pazzani and Billsus 2007, or Salton and Buckley 1988). A relatively simple one relates the actual number of term occurrences to the maximum frequency of the other keywords of the document as follows (see also Adomavicius and Tuzhilin 2005).

We search for the normalized term frequency value $TF(i, j)$ of keyword i in document j. Let $freq(i, j)$ be the absolute number of occurrences of i in j. Given a keyword i, let $OtherKeywords(i, j)$ denote the set of the other keywords appearing in j. Compute the maximum frequency $maxOthers(i, j)$ as $max(freq(z, j)), z \in OtherKeywords(i, j)$. Finally, calculate $TF(i, j)$ as in Chakrabarti (2002):

$$TF(i, j) = \frac{freq(i, j)}{maxOthers(i, j)} \tag{3.2}$$

Inverse document frequency is the second measure that is combined with term frequency. It aims at reducing the weight of keywords that appear very often in all documents. The idea is that those generally frequent words are not very helpful to discriminate among documents, and more weight should

https://doi.org/10.1017/CBO9780511763113.005 Published online by Cambridge University Press

therefore be given to words that appear in only a few documents. Let N be the number of all recommendable documents and $n(i)$ be the number of documents from N in which keyword i appears. The inverse document frequency for i is typically calculated as

$$IDF(i) = log \frac{N}{n(i)} \tag{3.3}$$

The combined TF-IDF weight for a keyword i in document j is computed as the product of these two measures:

$$TF\text{-}IDF(i, j) = TF(i, j) * IDF(i) \tag{3.4}$$

In the TF-IDF model, the document is, therefore, represented not as a vector of Boolean values for each keyword but as a vector of the computed TF-IDF weights.

3.1.2 Improving the vector space model/limitations

TF-IDF vectors are typically large and very sparse. To make them more compact and to remove irrelevant information from the vector, additional techniques can be applied.

Stop words and stemming. A straightforward method is to remove so-called stop words. In the English language these are, for instance, prepositions and articles such as "a", "the", or "on", which can be removed from the document vectors because they will appear in nearly all documents. Another commonly used technique is called *stemming* or *conflation*, which aims to replace variants of the same word by their common stem (root word). The word "stemming" would, for instance, be replaced by "stem", "went" by "go", and so forth.

These techniques further reduce the vector size and at the same time, help to improve the matching process in cases in which the word stems are also used in the user profile. Stemming procedures are commonly implemented as a combination of morphological analysis using, for instance, Porter's suffix-stripping algorithm (Porter 1980) and word lookup in dictionaries such as WordNet.[3] Although this technique is a powerful one in principle, there are some pitfalls, as stemming may also increase the danger of matching the profile with irrelevant documents when purely syntactic suffix stripping is used. For example, both the terms *university* and *universal* are stemmed to *univers*, which may lead to an unintended match of a document with the user profile (Chakrabarti 2002). Other

[3] http://wordnet.princeton.edu.

https://doi.org/10.1017/CBO9780511763113.005 Published online by Cambridge University Press

problems arise, in particular, when technical documents with many abbreviations are analyzed or when homonymous words (having multiple meanings) are in the text.

Size cutoffs. Another straightforward method to reduce the size of the document representation and hopefully remove "noise" from the data is to use only the n most informative words. In the Syskill & Webert system (Pazzani and Billsus 1997), for instance, the 128 most informative words (with respect to the expected information gain) were chosen. Similarly, Fab (Balabanović and Shoham 1997) used the top 100 words. The optimal number of words to be used was determined experimentally for the Syskill & Webert system for several domains. The evaluation showed that if too few keywords (say, fewer than 50) were selected, some possibly important document features were not covered. On the other hand, when including too many features (e.g., more than 300), keywords are used in the document model that have only limited importance and therefore represent noise that actually worsens the recommendation accuracy. In principle, complex techniques for "feature selection" can also be applied for determining the most *informative* keywords. However, besides an increase in model complexity, it is argued that learning-based approaches will tend to overfit the example representation to the training data (Pazzani and Billsus 1997). Instead, the usage of external lexical knowledge is proposed to remove words that are not relevant in the domain. Experiments show a consistent accuracy gain when such lexical knowledge is used, in particular when few training examples are available.

Phrases. A further possible improvement with respect to representation accuracy is to use "phrases as terms", which are more descriptive for a text than single words alone. Phrases, or composed words such as "United Nations", can be encoded as additional dimensions in the vector space. Detection of phrases can be done by looking up manually defined lists or by applying statistical analysis techniques (see Chakrabarti 2002 for more details).

Limitations. The described approach of extracting and weighting individual keywords from the text has another important limitation: it does not take into account the context of the keyword and, in some cases, may not capture the "meaning" of the description correctly. Consider the following simple example from Pazzani and Billsus (2007). A free-text description of a steakhouse used in a corresponding recommender system might state that "there is nothing on the menu that a vegetarian would like". In this case, in an automatically generated feature vector, the word *vegetarian* will most probably receive a higher weight

than desired and produce an unintended match with a user interested in vegetarian restaurants. Note, however, that in general we may assume that terms appearing in a document are usually well suited for characterizing documents and that a "negative context" – as shown in the example – is less likely to be encountered in documents.

3.2 Similarity-based retrieval

When the item selection problem in collaborative filtering can be described as "recommend items that similar users liked", content-based recommendation is commonly described as "recommend items that are similar to those the user liked in the past". Therefore, the task for a recommender system is again – based on a user profile – to predict, for the items that the current user has not seen, whether he or she will like them. The most common techniques that rely on the vector-space document representation model will be described in this section.

3.2.1 Nearest neighbors

A first, straightforward, method for estimating to what extent a certain document will be of interest to a user is simply to check whether the user liked similar documents in the past. To do this, two pieces of information are required. First, we need some history of "like/dislike" statements made by the user about previous items. Similar to collaborative approaches, these ratings can be provided either explicitly via the user interface or implicitly by monitoring the user's behavior. Second, a measure is needed that captures the similarity of two documents. In most reported approaches, the cosine similarity measure (already described in Section 2.2.1) is used to evaluate whether two document vectors are similar.

The prediction for a not-yet-seen item d is based on letting the k most similar items for which a rating exists "vote" for n (Allan et al. 1998). If, for instance, four out of $k = 5$ of the most similar items were liked by the current user, the system may guess that the chance that d will also be liked is relatively high. Besides varying the neighborhood size k, several other variations are possible, such as binarization of ratings, using a minimum similarity threshold, or weighting of the votes based on the degree of similarity.

Such a k-nearest-neighbor method (kNN) has been implemented, for instance, in the personalized and mobile news access system described by Billsus et al. (2000). In this system, the kNN method was used to model the short-term

interests of the users, which is of particular importance in the application domain of news recommendation. On arrival of a new message, the system looks for similar items in the set of stories that were recently rated by the user. By taking into account the last ratings only, the method is thus capable of adapting quickly and focusing on the user's short-term interests, which might be to read follow-up stories to recent events. At the same time, when relying on nearest neighbors, it is also possible to set an upper threshold for item similarity to prevent the system from recommending items that the user most probably has already seen.

In the described system, the kNN method was implemented as part of a multistrategy user profile technique. The system maintained profiles of short-term (ephemeral) and long-term interests. The short-term profile, as described earlier, allows the system to provide the user with information on topics of recent interest. For the long-term model, the system described by Billsus et al. (2000) therefore collects information over a longer period of time (e.g., several months) and also seeks to identify the most *informative* words in the documents by determining the terms that consistently receive high TF-IDF scores in a larger document collection. The prediction of whether an item is of interest with respect to the long-term user model is based on a probabilistic classification technique, which we describe in Section 3.3.1. Details on the threshold values and algorithms used in the experimental systems are described by Billsus and Pazzani (1999).

Given the interest predictions and recommendations for the short-term and the long-term user models, the question remains how to combine them. In the described system, a rather simple strategy is chosen. Neighbors in the short-term model are searched; if no such neighbors exist, the long-term user model is used. Other combinations are also possible. One option would be to acquire the short-term preferences online – for example, by questioning topics of interest and then sorting the matching items based on the information from the long-term preferences.

In summary, kNN-based methods have the advantage of being relatively simple to implement[4], adapt quickly to recent changes, and have the advantage that, when compared with other learning approaches, a relatively small number of ratings is sufficient to make a prediction of reasonable quality. However, as experiments show, the prediction accuracy of pure kNN methods can be lower than those of other more sophisticated techniques.

[4] Naive implementations of nearest-neighbor methods may, however, quickly become computationally intensive, so more advanced neighbor search methods may be required; see, e.g., Zezula et al. (2006).

3.2.2 Relevance feedback – Rocchio's method

Another method that is based on the vector-space model and was developed in the context of the pioneering information retrieval (IR) system SMART (Salton 1971) in the late 1960s is Rocchio's *relevance feedback* method. A particular aspect of SMART was that users could not only send (keyword-based) queries to the system but could also give feedback on whether the retrieved items were relevant. With the help of this feedback, the system could then internally extend the query to improve retrieval results in the next round of retrieval.

The SMART system for information retrieval did not exploit such additional information but rather allowed the user to interactively and *explicitly* rate the retrieved documents – that is, to tell the system whether they were relevant. This information is subsequently used to further refine the retrieval results. The rationale of following this approach is that with pure query- and similarity-based methods that do not provide any feedback mechanisms, the retrieval quality depends too strongly on the individual user's capability to formulate queries that contain the right keywords. User-defined queries often consist only of very few and probably polysemous words; a typical query on the web, for instance, consists of only two keywords on average (Chakrabarti 2002).

The relevance feedback loop used in this method will help the system improve and automatically extend the query as follows. The main idea is to first split the already rated documents into two groups, D^+ and D^-, of liked (interesting/relevant) and disliked documents and calculate a *prototype* (or average) vector for these categories. This prototype can also be seen as a sort of centroid of a cluster for relevant and nonrelevant document sets; see Figure 3.1.

The current query Q_i, which is represented as a multidimensional term vector just like the documents, is then repeatedly refined to Q_{i+1} by a weighted addition of the prototype vector of the relevant documents and weighted substraction of the vector representing the nonrelevant documents. As an effect, the query vector should consistently move toward the set of relevant documents as depicted schematically in Figure 3.2.

The proposed formula for computing the modified query Q_{i+1} from Q_i is defined as follows:

$$Q_{i+1} = \alpha * Q_i + \beta \left(\frac{1}{|D^+|} \sum_{d^+ \in D^+} d^+ \right) - \gamma \left(\frac{1}{|D^-|} \sum_{d^- \in D^-} d^- \right) \qquad (3.5)$$

The variables α, β, and γ are used to fine-tune the behavior of the "move" toward the more relevant documents. The value of α describes how strongly the last (or original) query should be weighted, and β and γ correspondingly capture how strongly positive and negative feedback should be taken into

https://doi.org/10.1017/CBO9780511763113.005 Published online by Cambridge University Press

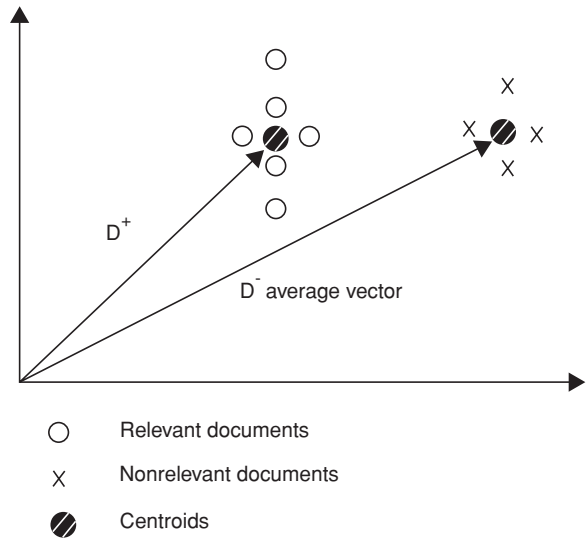

Figure 3.1. Average vectors for relevant and nonrelevant documents.

account in the improvement step. According to the analysis by Buckley et al. (1994), suitable parameter values are, for instance, 8, 16, and 4 (or 1, 2, and 0.25, respectively). These findings indicate that positive feedback is more valuable than negative feedback and it can be even better to take only positive feedback into account.

At first sight, this formula seems intuitive and the algorithm is very simple, but as stated by Pazzani and Billsus (2007), there is no theoretically motivated basis for Formula (3.5), nor can performance or convergence be guaranteed.

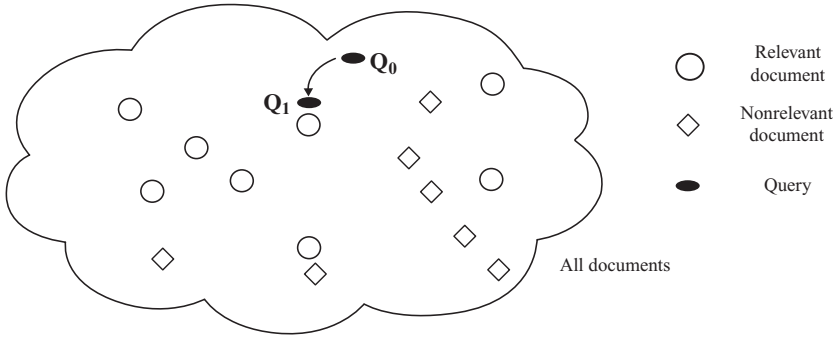

Figure 3.2. Relevance feedback. After feedback, the original query is moved toward the cluster of the relevant documents; see also Manning et al. (2008).

https://doi.org/10.1017/CBO9780511763113.005 Published online by Cambridge University Press

However, empirical evaluations with various document sets showed that useful retrieval performance can be improved, based on the feedback mechanism, already after the first iteration. More feedback iterations show only marginal improvements. An experimental evaluation using variations of this relevance feedback scheme, including an analysis of the effects of different settings, can be found in Salton and Buckley (1997) and Buckley et al. (1994). In practical settings it is also a good idea not to include *all* terms from D^+ and D^- to compute the new query (Chakrabarti 2002), as "one bad word may offset the benefits of many good words", but rather to use only the first 10 or 20 of the most relevant words in terms of the IDF measure.

Overall, the relevance feedback retrieval method and its variations are used in many application domains. It has been shown that the method, despite its simplicity, can lead to good retrieval improvements in real-world settings; see Koenemann and Belkin (1996) for a more detailed study. The main practical challenges – as with most content-based methods – are (a) a certain number of previous item ratings is needed to build a reasonable user model and (b) user interaction is required during the retrieval phase.

The first point can be partially automated by trying to capture the user ratings implicitly – for instance, by interpreting a click on a proposed document as a positive rating. The question of whether such assumptions hold in general – what to do when a user has read an article but was disappointed and what other techniques for gathering implicit feedback can be used – remains open (compare also the discussion on implicit ratings in Section 2.3.1).

Another technique for circumventing the problem of acquiring explicit user feedback is to rely on *pseudorelevance feedback* (blind feedback). The basic idea is to assume that the first n (say, 10) documents that match the query best with respect to the vector similarity measure are considered relevant. The set D^- is not used (γ is set to 0) unless an explicit negative feedback exists.

The second point – that user interaction is required during the proposal generation phase – at first glance appears to be a disadvantage over the fully automated proposal generation process of CF approaches. In fact, interactive query refinement also opens new opportunities for gathering information about the user's *real* preferences and may help the user to "learn" which vocabulary to use to retrieve documents that satisfy his or her information needs. The main assumption, of course, is that the user is capable of formulating a proper initial query, an assumption that might not always hold if we think of terminology aspects, multilanguage translation problems, or simply word misspellings (Manning et al. 2008). Further aspects of interactivity in recommender systems will be covered in more detail in Chapter 4.

Today's web search engines do not provide mechanisms for explicit feedback, although, as various evaluations show, they can lead to improved retrieval

performance. Chakrabarti (2002) mentions two reasons for that. First, he argues that today's web users are impatient and are not willing to give explicit feedback on the proposals. Second, second-round queries that include many more terms than the initial query are more problematic from a performance perspective and cannot be answered as quickly as the initial "two-term" queries.

In general, however, query-based retrieval approaches are quite obviously similar to modern web search engines, and the question may arise whether a search engine is also a "content-based recommender". Although until recently popular search engine providers such as Google or Yahoo! did not personalize their search results to particular users, it can be seen also from our news personalization example from the previous section (Das et al. 2007) that a trend toward increased personalization of search results can now be observed. Today we also see that the major players in the field have started to provide more features on their service platforms, which typically include personalized start pages, access to an e-mail service, online document manipulation, document management, and so forth. As users access these features with the same identity, a broad opportunity arises to also personalize the search results more precisely. However, reports on how personalized document rankings can be computed based on these kinds of information and, in particular, how the different relevance metrics, such as PageRank and document-query similarity, can be combined are not yet available.

3.3 Other text classification methods

Another way of deciding whether or not a document will be of interest to a user is to view the problem as a *classification* task, in which the possible classes are "like" and "dislike". Once the content-based recommendation task has been formulated as a classification problem, various standard (supervised) machine learning techniques can, in principle, be applied such that an intelligent system can automatically decide whether a user will be interested in a certain document. *Supervised learning* means that the algorithm relies on the existence of training data, in our case a set of (manually labeled) document-class pairs.

3.3.1 Probabilistic methods

The most prominent classification methods developed in early text classification systems are probabilistic ones. These approaches are based on the naive Bayes

https://doi.org/10.1017/CBO9780511763113.005 Published online by Cambridge University Press

Table 3.3. *Classification based on Boolean feature vector.*

Doc-ID	recommender	intelligent	learning	school	**Label**
1	1	1	1	0	**1**
2	0	0	1	1	**0**
3	1	1	0	0	**1**
4	1	0	1	1	**1**
5	0	0	0	1	**0**
6	1	1	0	0	**?**

assumption of conditional independence (with respect to term occurrences) and have also been successfully deployed in content-based recommenders.

Remember the basic formula to compute the posterior probability for document classification from Section 2.4.3:

$$P(Y|X) = \frac{\prod_{i=1}^{d} P(X_i|Y) \times P(Y)}{P(X)} \tag{3.6}$$

A straightforward application of this model to the classification task is described by Pazzani and Billsus (1997). The possible classes are, of course, "like" and "dislike" (named *hot* and *cold* in some articles). Documents are represented by Boolean feature vectors that describe whether a certain term appeared in a document; the feature vectors are limited to the 128 most informative words, as already mentioned.

Thus, in that model, $P(v_i|C = c)$ expresses the probability of term v_i appearing in a document labeled with class c. The conditional probabilities are again estimated by using the observations in the training data.

Table 3.3 depicts a simple example setting. The training data consist of five manually labeled training documents. Document 6 is a still-unlabeled document. The problem is to decide whether the current user will be interested – that is, whether to recommend the item. To determine the correct class, we can compute the class-conditional probabilities for the feature vector X of Document 6 again as follows:

$$\begin{aligned} \text{P(X|Label=1)} &= \text{P(recommender=1|Label=1)} \times \\ &\quad \text{P(intelligent=1|Label=1)} \times \\ &\quad \text{P(learning=0|Label=1)} \times \text{P(school=0|Label=1)} \\ &= 3/3 \times 2/3 \times 1/3 \times 2/3 \\ &\approx 0.149 \end{aligned}$$

https://doi.org/10.1017/CBO9780511763113.005 Published online by Cambridge University Press

The same can be done for the case $Label = 0$, and we see in the simple example that it is more probable that the user is more interested in documents (for instance, web pages) about intelligent recommender systems than in documents about learning in school. In real applications some sort of smoothing must be done for sparse datasets such that individual components of the calculation do not zero out the probability values. Of course, the resulting probability values can be used not only to decide whether a newly arriving document – in, for instance, a news filtering system – is relevant but also to rank a set of not-yet-seen documents. Remember that we also mentioned probabilistic techniques as possible recommendation methods in CF in the previous chapter. In CF, however, the classifier is commonly used to determine the membership of the active user in a cluster of users with similar preferences (by means of a latent class variable), whereas in content-based recommendation the classifier can also be directly used to determine the interestingness of a document.

Obviously, the core assumption of the naive Bayes model that the individual events are conditionally independent does not hold because there exist many term co-occurrences that are far more likely than others – such as the terms *Hong* and *Kong* or *New* and *York*. Nonetheless, the Bayes classifier has been shown to lead to surprisingly good results and is broadly used for text classification. An analysis of the reasons for this somewhat counterintuitive evidence can be found in Domingos and Pazzani (1996, 1997), or Friedman (1997). McCallum and Nigam (1998) summarize the findings as follows: "The paradox is explained by the fact that classification estimation is only a function of the sign (in binary case) of the function estimation; the function approximation can still be poor while classification accuracy remains high."

Besides the good accuracy that can be achieved with the naive Bayes classifier, a further advantage of the method – and, in particular, of the conditional independence assumption – is that the components of the classifier can be easily updated when new data are available and the learning time complexity remains linear to the number of examples; the prediction time is independent of the number of examples (Pazzani and Billsus 1997). However, as with most learning techniques, to provide reasonably precise recommendations, a certain amount of training data (past ratings) is required. The "cold-start" problem also exists for content-based recommenders that require some sort of relevance feedback. Possible ways of dealing with this are, for instance, to let the user manually label a set of documents – although this cannot be done for hundreds of documents – or to ask the user to provide a list of interesting words for each topic category (Pazzani and Billsus 1997).

The Boolean representation of document features has the advantage of simplicity but, of course, the possibly important information on how many times

Table 3.4. *Classification example with term counts.*

DocID	Words	**Label**
1	recommender intelligent recommender	**1**
2	recommender recommender learning	**1**
3	recommender school	**1**
4	teacher homework recommender	**0**
5	recommender recommender recommender teacher homework	**?**

a term occurred in the document is lost at this point. In the Syskill & Webert system, which relies on such a Boolean classifier for each topic category, the relevance of words is taken into account only when the initial set of appropriate keywords is determined. Afterward, the system cannot differentiate anymore whether a keyword appeared only once or very often in the document. In addition, this model also assumes *positional* independence – that is, it does not take into account *where* the term appeared in the document.

Other probabilistic modeling approaches overcome such limitations. Consider for instance, the classification method (example adapted from Manning et al. 2008) in Table 3.4, in which the number of term appearances shall also be taken into account.

The conditional probability of a term v_i appearing in a document of class C shall be estimated by the relative frequency of v_i in all documents of this class:

$$P(v_i|C = c) = \frac{CountTerms(v_i, docs(c))}{AllTerms(docs(c))} \tag{3.7}$$

where $CountTerms(v_i, docs(c))$ returns the number of appearances of term v_i in documents labeled with c and $AllTerms(docs(c))$ returns the number of all terms in these documents. To prevent zeros in the probabilities, Laplace (add-one) smoothing shall be applied in the example:

$$\hat{P}(v_i|C = c) = \frac{CountTerms(v_i, docs(c)) + 1}{AllTerms(docs(c)) + |V|} \tag{3.8}$$

where $|V|$ is the number of different terms appearing in all documents (called the "vocabulary"). We calculate the conditional probabilities for the relevant terms appearing in the new document as follows: the total length of the documents classified as "1" is 8, and the length of document 4 classified as "0" is 3. The

https://doi.org/10.1017/CBO9780511763113.005 Published online by Cambridge University Press

size of the vocabulary is 6.

$$\hat{P}(recommender|Label = 1) = (5 + 1)/(8 + 6) = 6/14$$
$$\hat{P}(homework|Label = 1) = (0 + 1)/(8 + 6) = 1/14$$
$$\hat{P}(teacher|Label = 1) = (0 + 1)/(8 + 6) = 1/14$$
$$\hat{P}(recommender|Label = 0) = (1 + 1)/(3 + 6) = 2/9$$
$$\hat{P}(homework|Label = 0) = (1 + 1)/(3 + 6) = 2/9$$
$$\hat{P}(teacher|Label = 0) = (1 + 1)/(3 + 6) = 2/9$$

The prior probabilities of a document falling into class 1 or class 0 are $3/4$ and $1/4$, respectively. The classifier would therefore calculate the posterior probabilities as

$$\hat{P}(Label = 1|v_1 \dots v_n) = 3/4 \times (3/7)^3 \times 1/14 \times 1/14 \approx 0.0003$$
$$\hat{P}(Label = 0|v_1 \dots v_n) = 1/4 \times (2/9)^3 \times 2/9 \times 2/9 \approx 0.0001$$

and therefore classify the unlabeled document as being relevant for the user. The classifier has taken the multiple evidence of the term "recommender" into account appropriately. If only the Boolean representation had been used, the classifier would have rejected the document, because two other terms that appear in the document ("homework", "teacher") suggest that it is not relevant, as they also appear in the rejected document 4.

Relation to text classification. The problem of labeling a document as relevant or irrelevant in our document recommendation scenarios can be seen as a special case of the more broader and older *text classification* (text categorization or topic spotting) problem, which consists of assigning a document to a set of predefined classes. Applications of these methods can be found in information retrieval for solving problems such as personal e-mail sorting, detection of spam pages, or sentiment detection (Manning et al. 2008). Different techniques of "supervised learning", such as the probabilistic one described previously, have been proposed. The basis for all the learning techniques is a set of manually annotated training documents and the assumption that the unclassified (new) documents are somehow similar to the manually classified ones. When compared with the described "like/dislike" document recommendation problem, general text classification problems are not limited to only two classes. Moreover, in some applications it is also desirable to assign one document to more than one individual class.

https://doi.org/10.1017/CBO9780511763113.005 Published online by Cambridge University Press

As noted earlier, probabilistic methods that are based on the naive Bayes assumption have been shown to be particularly useful for text classification problems. The idea is that both the training documents and the still unclassified documents are generated by the probability distributions. Basically, two different ways of modeling the documents and their features have been proposed: the multinomial model and the Bernoulli model. The main differences between these models are the "event model" and, accordingly, how the probabilities are estimated from the training data (see McCallum and Nigam 1998, Manning et al. 2008, or Pazzani and Billsus 2007 for a detailed discussion). In the multivariate Bernoulli model, a document is treated as a binary vector that describes whether a certain term is contained in the document. In the multinomial model the number of times a term occurred in a document is also taken into account, as in our earlier example. In both cases, the position of the terms in the document is ignored. Empirical evaluations show that the multinomial model leads to significantly better classification results than does the Bernoulli model (McCallum and Nigam 1998), in particular when it comes to longer documents and classification settings with a higher number of features. An illustrative example for both approaches can be found in Manning et al. (2008).

Finally, another interesting finding in probabilistic text classification is that not only can the manually labeled documents can be used to train the classifier, but still-unlabeled documents can also help to improve classification (Nigam et al. 1998). In the context of content-based recommendation this can be of particular importance, as the training set of manually or implicitly labeled documents is typically very small because every user has his or her personal set of training examples.

3.3.2 Other linear classifiers and machine learning

When viewing the content-based recommendation problem as a classification problem, various other machine learning techniques can be employed. At a more abstract level, most learning methods aim to find coefficients of a linear model to discriminate between relevant and nonrelevant documents.

Figure 3.3 sketches the basic idea in a simplified setting in which the available documents are characterized by only two dimensions. If there are only two dimensions, the classifier can be represented by a line. The idea can, however, also easily be generalized to the multidimensional space in which a two-class classifier then corresponds to a hyperplane that represents the decision boundary.

In two-dimensional space, the line that we search for has the form $w_1x_1 + w_2x_2 = b$ where x_1 and x_2 correspond to the vector representation of a document

https://doi.org/10.1017/CBO9780511763113.005 Published online by Cambridge University Press

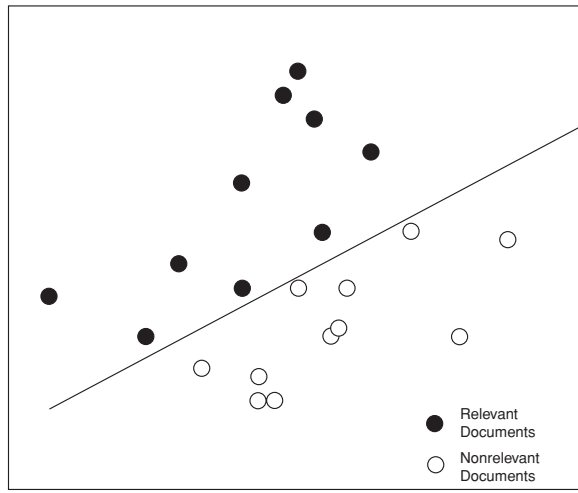

Figure 3.3. A linear classifier in two-dimensional space.

(using, e.g., TF-IDF weights) and w_1, w_2, and b are the parameters to be learned. The classification of an individual document is based on checking whether for a certain document $w_1x_1 + w_2x_2 > b$, which can be done very efficiently. In n-dimensional space, a generalized equation using weight and feature vectors instead of only two values is used, so the classification function is $\vec{w}^T\vec{x} = b$.

Many text classification algorithms are actually linear classifiers, and it can easily be shown that both the naive Bayes classifier and the Rocchio method fall into this category (Manning et al. 2008). Other methods for learning linear classifiers are, for instance, the Widrow-Hoff algorithm (see Widrow and Stearns 1985) or support vector machines (SVM; Joachims 1998). The kNN nearest-neighbor method, on the other hand, is not a linear classifier. In general, infinitely many hyperplanes (or lines in Figure 3.3) exist that can be used to separate the document space. The aforementioned learning methods will typically identify different hyperplanes, which may in turn lead to differences in classification accuracy. In other words, although all classifiers may separate the training data perfectly, they may show differences in their error rates for additional test data. Implementations based on SVM, for instance, try to identify decision boundaries that maximize the distance (called *margin*) to the existing datapoints, which leads to very good classification accuracy when compared with other approaches.

Another challenge when using a linear classifier is to deal with noise in the data. There can be noisy features that mislead the classifier if they are included in the document representation. In addition, there might also be *noise*

https://doi.org/10.1017/CBO9780511763113.005 Published online by Cambridge University Press

documents that, for whatever reason, are not near the cluster where they belong. The identification of such noise in the data is, however, not trivial.

A comparative evaluation of different training techniques for text classifiers can be found in Lewis et al. (1996) and in Yang and Liu (1999). Despite the fact that in these experiments some algorithms, and in particular SVM-based ones, performed better than others, there exists no strict guideline as to which technique performs best in every situation. Moreover, it is not always clear whether using a linear classifier is the right choice at all, as there are, of course, many problem settings in which the classification borders cannot be reasonably approximated by a line or hyperplane. Overall, "selecting an appropriate learning method is therefore an unavoidable part of solving a text classification problem" (Manning et al. 2008).

3.3.3 Explicit decision models

Two other learning techniques that have been used for building content-based recommender systems are based on decision trees and rule induction. They differ from the others insofar as they generate an explicit decision model in the training phase.

Decision tree learning based on ID3 or the later C4.5 algorithms (see Quinlan 1993 for an overview) has been successfully applied to many practical problems, such as data mining problems. When applied to the recommendation problem, the inner nodes of the tree are labeled with item features (keywords), and these nodes are used to partition the test examples based, for instance, simply on the existence or nonexistence of a keyword in the document. In a basic setting only two classes, interesting or not, might appear at the leaf nodes. Figure 3.4 depicts an example of such a decision tree.

Determining whether a new document is relevant can be done very efficiently with such a prebuilt classification tree, which can be automatically constructed (learned) from training data without the need for formalizing domain knowledge. Further general advantages of decision trees are that they are well understood, have been successfully applied in many domains, and represent a model that can be interpreted relatively easily.

The main issue in the content-based recommendation problem setting is that we have to work on relatively large feature sets using, for instance, a TF-IDF document representation. Decision tree learners, however, work best when a relatively small number of features exist, which would be the case if we do not use a TF-IDF representation of a document but rather a list of "meta"-features such as author name, genre, and so forth. An experimental evaluation actually shows that decision trees can lead to comparably poor classification

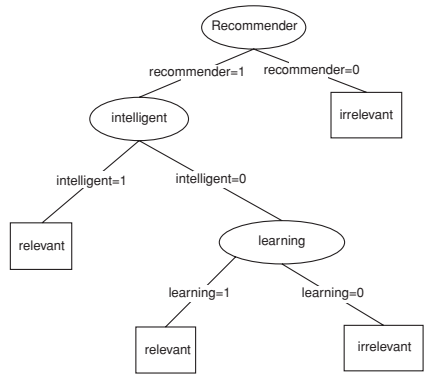

Figure 3.4. Decision tree example.

performance (Pazzani and Billsus 1997). The main reason for this limited performance on large feature sets lies in the typical splitting strategy based on the information gain, which leads to a bias toward small decision trees (Pazzani and Billsus 2007).

For these reasons, decision trees are seldom used for classical content-based recommendation scenarios. One of the few exceptions is the work of Kim et al. (2001), in which decision trees were used for personalizing the set of advertisements appearing on a web page. Still, even though decision trees might not be used directly as the core recommendation technique, they can be used in recommender systems in combination with other techniques to improve recommendation efficiency or accuracy. In Nikovski and Kulev (2006), for example, decision trees are used to compress in-memory data structures for a recommender system based on frequent itemset mining; Bellogín et al. (2010) propose to use decision trees to determine which user model features are the most relevant ones for providing accurate recommendations in a content-based collaborative hybrid news recommender system. Thus, the learning task in this work is to improve the recommendation model itself.

Rule induction is a similar method that is used to extract decision rules from training data. Methods built on the RIPPER algorithm (Cohen 1995, 1996) have been applied with some success for e-mail classification, which is, however, not a core application area of recommender systems. As mentioned by Pazzani and Billsus (2007), the relatively good performance when compared with other classification methods can be partially explained by the elaborate postpruning techniques of RIPPER itself and a particular extension that was made for e-mail classification that takes the specific document structure of e-mails with a subject line and a document body into account. A more recent evaluation and comparison of

https://doi.org/10.1017/CBO9780511763113.005 Published online by Cambridge University Press

e-mail classification techniques can be found in Koprinska et al. (2007), which shows that "random forests" (instead of simple trees) perform particularly well on this problem.

In summary, both decision tree learning and rule induction have been successfully applied to specific subproblems such as e-mail classification, advertisement personalization, or cases in which small feature sets are used to describe the items (Bouza et al. 2008), which is a common situation in knowledge-based recommenders. In these settings, two of the main advantages of these learning techniques are that (a) the inferred decision rules can serve as a basis for generating explanations for the system's recommendations and (b) existing prior domain knowledge can be incorporated in the models.

3.3.4 On feature selection

All the techniques described so far rely on the vector representation of documents and on TF-IDF weights. When used in a straightforward way, such document vectors tend to be very long (there are typically thousands of words appearing in the corpus) and very sparse (in every document only a fraction of the words is used), even if stop words are removed and stemming is applied. In practical applications, such long and sparse vectors not only cause problems with respect to performance and memory requirements, but also lead to an effect called *overfitting*. Consider an example in which a very rare word appears by pure chance only in documents that have been labeled as "hot". In the training phase, a classifier could therefore be misled in the direction that this word (which can, in fact, be seen as some sort of noise) is a good indicator of the interestingness of some document. Such overfitting can easily appear when only a limited number of training documents is available.

Therefore, it is desirable to use only a subset of all the terms of the corpus for classification. This process of choosing a subset of the available terms is called *feature selection*. Different strategies for deciding which features to use are possible. Feature selection in the Syskill & Webert recommender system mentioned earlier (Pazzani and Billsus 1997), for instance, is based on domain knowledge and lexical information from WordNet. The evaluation reported by Pazzani and Billsus (1997) shows not only that the recommendation accuracy is improved when irrelevant features are removed, but also that using around 100 "informative" features leads to the best results.

Another option is to apply frequency-based feature selection and use domain- or task-specific heuristics to remove words that are "too rare" or appear "too often" based on empirically chosen thresholds (Chakrabarti 2002).

Table 3.5. χ^2 *contingency table.*

	Term t appeared	Term t missing
Class "relevant"	A	B
Class "irrelevant"	C	D

For larger text corpora, such heuristics may not be appropriate, however, and more elaborate, statistics-based methods are typically employed. In theory, one could find the optimal feature subset by training a classifier on every possible subset of features and evaluate its accuracy. Because such an approach is computationally infeasible, the value of individual features (keywords) is rather evaluated independently and a ranked list of "good" keywords, according to some utility function, is constructed. The typical measures for determining the utility of a keyword are the χ^2 test, the mutual information measure, or Fisher's discrimination index (see Chakrabarti 2002).

Consider, for example, the χ^2 test, which is a standard statistical method to check whether two events are independent. The idea in the context of feature selection is to analyze, based on training data, whether certain classification outcomes are connected to a specific term occurrence. When such a statistically significant dependency for a term can be identified, we should include this term in the feature vector used for classification.

In our problem setting, a 2×2 contingency table of classification outcomes and occurrence of term t can be set up for every term as in Table 3.5 when we assume a binary document model in which the actual number of occurrences of a term in a document is not relevant.

The symbols A to D in the table can be directly taken from the training data: Symbol A stands for the number of documents that contained term t and were classified as relevant, and B is the number of documents that were classified as relevant but did not contain the term. Symmetrically, C and D count the documents that were classified as irrelevant. Based on these numbers, the χ^2 test measures the deviation of the given counts from those that we would statistically expect when conditional independence is given. The χ^2 value is calculated as follows:

$$\chi^2 = \frac{(A + B + C + D)(AD - BC)^2}{(A + B)(A + C)(B + D)(C + D)} \tag{3.9}$$

Higher values for χ^2 indicate that the events of term occurrence and membership in a class are not independent.

https://doi.org/10.1017/CBO9780511763113.005 Published online by Cambridge University Press

To select features based on the χ^2 test, the terms are first ranked by decreasing order of their χ^2 values. The logic behind that is that we want to include those features that help us to determine class membership (or nonmembership) first – that is, those for which class membership and term occurrence are correlated. After sorting the terms, according to the proposal by Chakrabarti (2002), a number of experiments should be made to determine the optimal number of features to use for the classifier.

As mentioned previously, other techniques for feature selection, such as mutual information or Fisher's discriminant, have also been proposed for use in information retrieval scenarios. In many cases, however, these techniques result more or less in the same set of keywords (maybe in different order) as long as different document lengths are taken into account (Chakrabarti 2002, Manning et al. 2008).

3.4 Discussion

3.4.1 Comparative evaluation

Pazzani and Billsus (1997) present a comparison of several learning-based techniques for content-based recommendation. Experiments were made for several relatively small and manually annotated document collections in different domains. The experiments made with the Syskill & Webert system were set up in a way in which a subset of documents was used to learn the user profile, which was then used to predict whether the user would be interested in the unseen documents.

The percentage of correctly classified documents was taken as an accuracy measure. The accuracy of the different recommenders varied relatively strongly in these experiments (from 60 percent to 80 percent). As with most learning algorithms, the most important factor was the size of the training set (up to fifty documents in these tests). For some example sets, the improvements were substantial and an accuracy of more than 80 percent could be achieved. In some domains, however, the classifier never significantly exceeded chance levels.

Overall, the detailed comparison of the algorithms (using twenty training examples in each method) brought no clear winner. What could be seen is that decision-tree learning algorithms, which we did not cover in detail, did not perform particularly well in the given setting and that the "nearest neighbors" method performed poorly in some domains. The Bayesian and Rocchio methods performed consistently well in all domains, and no significant differences could be found. In the experiments, a neural net method with a nonlinear activation

function was also evaluated but did not lead to improvements in classification accuracy.

In the Syskill & Webert system, a decision for a Bayes classifier was finally chosen, as it not only worked well in all tested domains (even if the assumption of conditional independence does not hold) but it also is relatively fast with respect to learning and predicting. It also seemed that using only Boolean document representation in the classifier – as opposed to TF-IDF weights – does not significantly affect the recommendation accuracy (Pazzani and Billsus 1997).

Finally, Manning et al. (2008) also mention that Bayes classifiers seem to work well in many domains and summarize several techniques that have been developed to improve classifier performance, such as *feature engineering* (the manual or automatic selection of "good" features), *hierarchical classification* for large category taxonomies, or taking into account that different feature sets could be used for the different zones of a document.

3.4.2 Limitations

Pure content-based recommender systems have known limitations, which rather soon led to the development of hybrid systems that combine the advantages of different recommendation techniques. The Fab system is an early example of such a hybrid system; Balabanović and Shoham (1997) mention the following limitations of content-based recommenders.

Shallow content analysis. Particularly when web pages are the items to be recommended, capturing the quality or interestingness of a web page by looking at the textual contents alone may not be enough. Other aspects, such as aesthetics, usability, timeliness, or correctness of hyperlinks, also determine the quality of a page. Shardanand and Maes (1995) also mention that when keywords are used to characterize documents, a recommender cannot differentiate between well-written articles and comparably poor papers that, naturally, use the same set of keywords. Furthermore, in some application domains the text items to be recommended may not be long enough to extract a good set of discriminating features. A typical example is the recommendation of jokes (Pazzani and Billsus 2007): Learning a good preference profile from a very small set of features may be difficult by itself; at the same time it is nearly impossible to distinguish, for instance, good lawyer jokes from bad ones.

Information in hypertext documents is also more and more contained in multimedia elements, such as images, as well as audio and video sequences. These contents are also not taken into account when only a shallow text analysis

is done. Although some recent advances have been made in the area of feature extraction from text documents, research in the extraction of features from multimedia content is still at an early stage. Early results in the music domain have been reported, for instance, by (Li et al. 2003; automated genre detection) or (Shen et al. 2006; singer identification). More research can be expected in that direction in the near future, as the web already now is established as a major distribution channel for digital music, in which personalized music recommendations play an important role. Similar things happen in the video domain, where, in particular, the new opportunities of semantic annotation based on the MPEG-7 standard (ISO/IEC 15938) also allow enhanced annotation capabilities.

If no automatic extraction of descriptive features is possible, manual annotation is a theoretical option. Many authors agree that in most domains manual annotation is too costly. However, new opportunities arise in light of what is called Web 2.0, in which Internet users more and more play the role of content providers. It can already be observed that today's web users actively and voluntarily annotate content such as images or videos on popular web portals (collaborative tagging). Although these tags are mostly not taken from limited-size ontologies and may be inconsistent, they could serve as a valuable resource for determining further features of a resource. How such user-provided tags can be exploited to recommend resources to users in social web platforms will be discussed in more detail in Chapter 11.

Overspecialization. Learning-based methods quickly tend to propose *more of the same* – that is, such recommenders can propose only items that are somehow similar to the ones the current user has already (positively) rated. This can lead to the undesirable effect that *obvious* recommendations are made and the system, for instance, recommends items that are too similar to those the user already knows. A typical example is a news filtering recommender that proposes a newspaper article that covers the same story that the user has already seen in another context. The system described by Billsus and Pazzani (1999) therefore defines a threshold to filter out not only items that are too different from the profile but also those that are too similar. A set of more elaborate metrics for measuring novelty and redundancy has been analyzed by Zhang et al. (2002).

A general goal therefore is to increase the *serendipity* of the recommendation lists – that is, to include "unexpected" items in which the user might be interested, because expected items are of little value for the user. A simple way of avoiding monotonous lists is to "inject a note of randomness" (Shardanand and Maes 1995).

A discussion of this additional aspect of recommender system quality, which also applies to systems that are based on other prediction techniques, can be found, for instance, in McNee et al. (2006). Ziegler et al. (2005) propose a technique for generating more diverse recommendation lists ("topic diversification"). A recent proposal for a metric to measure the serendipity of the recommendation lists can be found in Satoh et al. (2007).

Acquiring ratings. The cold-start problem, which we discussed for collaborative systems, also exists in a slightly different form for content-based recommendation methods. Although content-based techniques do not require a large user community, they require at least an initial set of ratings from the user, typically a set of explicit "like" and "dislike" statements. In all described filtering techniques, recommendation accuracy improves with the number of ratings; significant performance increases for the learning algorithms were reported by Pazzani and Billsus (1997) when the number of ratings was between twenty and fifty. However, in many domains, users might not be willing to rate so many items before the recommender service can be used. In the initial phase, it could be an option to ask the user to provide a list of keywords, either by selecting from a list of topics or by entering free-text input.

Again, in the context of Web 2.0, it might be an option to "reuse" information that the user may have provided or that was collected in the context of another personalized (web) application and take such information as a starting point to incrementally improve the user profile.

3.5 Summary

In this chapter we have discussed different methods that are commonly referred to as content-based recommendation techniques. The roots of most approaches can be found in the field of information retrieval (IR), as the typical IR tasks of information filtering or text classification can be seen as a sort of recommendation exercise. The presented approaches have in common that they aim to learn a model of the user's interest preferences based on explicit or implicit feedback. Practical evaluations show that a good recommendation accuracy can be achieved with the help of various machine learning techniques. In contrast to collaborative approaches, these techniques do not require a user community in order to work.

However, challenges exist. The first one concerns user preference elicitation and new users. Giving explicit feedback is onerous for the user, and deriving implicit feedback from user behavior (such as viewing item details for a certain

period of time) can be problematic. All learning techniques require a certain amount of training data to achieve good results; some learning methods tend to overfit the training data so the danger exists that the recommendation lists contain too many similar items.

The border between content-based recommenders and other systems is not strictly defined. Automatic text classification or information filtering are classical IR methods. In the context of recommender systems, perhaps the main difference is that these classical IR tasks are personalized – in other words, a general spam e-mail detector or a web search engine should not be viewed as a recommender system. If we think of personal e-mail sorting (according to different automatically detected document categories) or personalization of search results, however, the border is no longer clear.

Another fuzzy border is between content-based recommenders and knowledge-based ones. A typical difference between them is that content-based recommenders generally work on text documents or other items for which features can be automatically extracted and for which some sort of learning technique is employed. In contrast, knowledge-based systems rely mainly on externally provided information about the available items.

With respect to industrial adoption of content-based recommendation, one can observe that pure content-based systems are rarely found in commercial environments. Among the academic systems that were developed in the mid-1990s, the following works are commonly cited as successful examples demonstrating the general applicability of the proposed techniques.

Syskill & Webert (Pazzani et al. 1996, Pazzani and Billsus 1997) is probably the most-cited system here and falls into the category of web browsing assistants that use past ratings to predict whether the user will be interested in the links on a web page (using "thumbs up" and "thumbs down" annotations). Personal Web Watcher (Mladenic 1996) is a similar system and browsing assistant, which, however, exploits document information in a slightly different way than does Syskill & Webert; see also Mladenic 1999. The Information Finder system (Krulwich and Burkey 1997) aims to achieve similar goals but is based on a special phrase extraction technique and Bayesian networks. Newsweeder (Lang 1995) is an early news filtering system based on a probabilistic method. NewsRec (Bomhardt 2004) is a more recent system that is not limited to a specific document type, such as news or web pages, and is based on SVM as a learning technique.

As the aforementioned limitations of pure content-based recommenders are critical in many domains, researchers relatively quickly began to combine them with other techniques into hybrid systems. Fab (Balabanović and Shoham

1997) is an early example of a collaborative/content-based hybrid that tries to combine the advantages of both techniques. Many other hybrid approaches have been proposed since then and will be discussed in Chapter 5. Reports on pure content-based systems are relatively rare today. Examples of newer systems can be found in domains in which recent advances have been made with respect to automated feature extraction, such as music recommendation. Even there, however, hybrids using collaborative filtering techniques are also common; see, for instance, Logan (2004) or Yoshii et al. (2006).

3.6 Bibliographical notes

The roots of several techniques that are used in content-based recommenders are in the fields of information retrieval and information filtering. An up-to-date introduction on IR and its methods is given in the text book by Manning et al. (2008). The work covers several techniques discussed in this chapter, such as TF-IDF weighting, the vector-space document model, feature selection, relevance feedback, and naive Bayes text classification. It describes additional classification techniques based on SVM, linear regression, and clustering; it also covers further specific information retrieval techniques, such as LSI, which were also applied in the recommendation domain as described in Chapter 2, and, finally, it discusses aspects of performance evaluation for retrieval and filtering systems.

A similar array of methods is discussed in the textbook by Chakrabarti (2002), which has a strong focus on web mining and practical aspects of developing the technical infrastructure that is needed to, for instance, crawl the web.

IR methods have a long history. A 1992 review of information filtering techniques and, in particular, on the then newly developed LSI method can be found in Foltz and Dumais (1992). Housman and Kaskela (1970) is an overview paper on methods of "selective dissemination of information", a concept that early on implemented some of the features of modern filtering methods.

A recent overview of content-based recommendation techniques (in the context of adaptive web applications and personalization) is given by Pazzani and Billsus (2007). More details about the influential Syskill & Webert recommender system can be found in the original paper (Pazzani et al. 1996) by the same authors. In Pazzani and Billsus (1997), a comparative evaluation of different classification techniques from nearest neighbors over decision trees, Bayes classifiers, and neural nets is given.

Adomavicius and Tuzhilin (2005) give a compact overview on content-based methods and show how such approaches fit into a more general framework of recommendation methods. They also provide an extensive literature review that can serve as a good starting point for further reading. The structure of this chapter mainly follows the standard schemes developed by Adomavicius and Tuzhilin (2005) and Pazzani and Billsus (2007).