Chapter 4

Content-Based Recommender Systems

"*Form must have a content, and that content must be linked with nature.*"– Alvar Aalto

4.1 Introduction

The collaborative systems discussed in the previous chapters use the correlations in the ratings patterns across users to make recommendations. On the other hand, these methods do not use item attributes for computing predictions. This would seem rather wasteful; after all, if John likes the futuristic science fiction movie *Terminator*, then there is a very good chance that he might like a movie from a similar genre, such as *Aliens*. In such cases, the ratings of other users may not be required to make meaningful recommendations. Content-based systems are designed to exploit scenarios in which items can be described with descriptive sets of attributes. In such cases, a user's *own* ratings and actions on other movies are sufficient to discover meaningful recommendations. This approach is particularly useful when the item is new, and there are few ratings available for that item.

Content-based recommender systems try to match users to items that are similar to what they have liked in the past. This similarity is not necessarily based on rating correlations across users but on the basis of the *attributes* of the objects liked by the user. Unlike collaborative systems, which explicitly leverage the ratings of *other* users in addition to that of the target user, content-based systems largely focus on the target user's *own* ratings and the attributes of the items liked by the user. Therefore, the other users have little, if any, role to play in content-based systems. In other words, the content-based methodology leverages a different source of data for the recommendation process. As we will see in Chapter 6, many recommender systems leverage the power of both sources. Such recommender systems are referred to as *hybrid* recommender systems.

C.C. Aggarwal, *Recommender Systems: The Textbook*,
DOI 10.1007/978-3-319-29659-3_4

At the most basic level, content-based systems are dependent on two sources of data:

1. The first source of data is a description of various items in terms of content-centric attributes. An example of such a representation could be the text description of an item by the manufacturer.

2. The second source of data is a *user profile*, which is generated from user feedback about various items. The user feedback might be explicit or implicit. Explicit feedback may correspond to ratings, whereas implicit feedback may correspond to user actions. The ratings are collected in a way similar to collaborative systems.

   The user profile relates the attributes of the various items to user interests (ratings). A very basic example of a user profile might simply be a set of labeled training documents of item descriptions, the user ratings as the labels, and a classification or regression model relating the item attributes to the user ratings. The specific user profile is heavily dependent on the methodology at hand. For example, explicit ratings might be used in one setting, and implicit feedback might be used in another. It is also possible for the user to specify her own profile in terms of keywords of interest, and this approach shares some characteristics with *knowledge-based recommender systems*.

It is noteworthy that the ratings of the *other* users usually play no role in a content-based recommendation algorithm. This is both an advantage and a disadvantage, depending on the scenario at hand. On the one hand, in *cold-start scenarios*, where little information about the ratings of other users is available, such an approach can still be used as long as sufficient information about the user's own interests are available. This, at least, partially alleviates the cold-start problem when the number of other users in the recommender system is small. Furthermore, when an item is new, it is not possible to obtain the ratings of other users for that item. Content-based methods enable recommendations in such settings because they can extract the attributes from the new item, and use them to make predictions. On the other hand, the cold-start problem for new *users* cannot be addressed with content-based recommender systems. Furthermore, by not using the ratings of other users, one reduces the diversity and novelty of the recommended items. In many cases, the recommended items may be obvious items for the user, or they may be other items that the user has consumed before. This is because the content attributes will always recommend items with similar attributes to what the user has seen in the past. A recommended item with similar attributes often presents little surprise to the user. These advantages and disadvantages will be discussed in a later section of this chapter.

Content-based systems are largely used in scenarios in which a significant amount of attribute information is available at hand. In many cases, these attributes are keywords, which are extracted from the product descriptions. In fact, the vast majority of content-based systems extract text attributes from the underlying objects. Content-based systems are, therefore, particularly well suited to giving recommendations in text-rich and *unstructured* domains. A classical example of the use of such systems is in the recommendation of Web pages. For example, the previous browsing behavior of a user can be utilized to create a content-based recommender system. However, the use of such systems is not restricted only to the Web domain. Keywords from product descriptions are used to create item and user profiles for the purposes of recommendations in other e-commerce settings. In other settings, relational attributes such as manufacturer, genre, and price, may be used in addition to keywords. Such attributes can be used to create *structured* representations, which can be stored in a relational database. In these cases, it is necessary to combine the structured and unstructured attributes in a single structured representation. The basic principles of

content-based systems, however, remain invariant to whether a structured or unstructured representation is used. This is because most learning methods in the structured domain have direct analogs in the unstructured domain, and vice versa. To preserve uniformity in exposition, our discussion in this chapter will be focused on unstructured settings. However, most of these methods can be easily adapted to structured settings.

Content-based systems are closely related to knowledge-based recommender systems. A summary of the relationship between the various types of systems is provided in Table 1.2 of Chapter 1. Like content-based systems, knowledge-based recommender systems use the content attributes of the items to make recommendations. The main difference is that knowledge-based systems support the *explicit specification* of user requirements in conjunction with interactive interfaces between the user and the recommender systems. Knowledge bases are used in conjunction with this interactivity to match user requirements to items. On the other hand, content-based systems generally use a learning-based approach based on *historical ratings*. Therefore, knowledge-based systems provide better control to the user in the recommendation process, whereas content-based systems leverage past behavior more effectively. Nevertheless, these differences are not so significant, and some content-based methods also allow users to explicitly specify their interest profiles. A number of systems leverage both the learning and interactive aspects within a unified framework. Such systems are referred to as *hybrid recommender systems*. Knowledge-based recommender systems are discussed in Chapter 5, whereas hybrid recommender systems are discussed in Chapter 6.

This chapter is organized as follows. The next section provides an overview of the basic components of a content-based recommender system. Feature extraction and selection methods are discussed in section 4.3. The process of learning user profiles and leveraging them for recommendations is discussed in section 4.4. A comparison of the main properties of collaborative and content-based systems is provided in section 4.5. The connections between collaborative filtering and content-based methods are explored in section 4.6. A summary is given in section 4.7.

4.2 Basic Components of Content-Based Systems

Content-based systems have certain basic components, which remain invariant across different instantiations of such systems. Since content-based systems work with a wide variety of item descriptions and knowledge about users, one must convert these different types of unstructured data into standardized descriptions. In most cases, it is preferred to convert the item descriptions into keywords. Therefore, content-based systems largely, but not exclusively, operate in the text domain. Many natural applications of content-based systems are also text-centric. For example, news recommender systems are often content-based systems, and they are also text-centric systems. In general, text classification and regression modeling methods remain the most widely used tools for creating content-based recommender systems.

The main components of content-based systems include the (offline) preprocessing portion, the (offline) learning portion, and the online prediction portion. The offline portions are used to create a summarized model, which is often a classification or regression model. This model is then used for the online generation of recommendations for users. The various components of content-based systems are as follows:

1. *Preprocessing and feature extraction:* Content-based systems are used in a wide variety of domains, such as Web pages, product descriptions, news, music features, and so on. In most cases, features are extracted from these various sources to convert them into

a keyword-based vector-space representation. This is the first step of any content-based recommendation system, and it is highly domain-specific. However, the proper extraction of the most informative features is essential for the effective functioning of any content-based recommender system.

2. *Content-based learning of user profiles:* As discussed earlier, a content-based model is specific to a given user. Therefore, a user-specific *model* is constructed to predict user interests in items, based on their past history of either buying or rating items. In order to achieve this goal, user feedback is leveraged, which may be manifested in the form of previously specified ratings (explicit feedback) or user activity (implicit feedback). Such feedbacks are used in conjunction with the attributes of the items in order to construct the training data. A learning model is constructed on this training data. This stage is often not very different from classification or regression modeling, depending on whether the feedback is categorical (e.g., binary act of selecting an item), or whether the feedback is numerical (e.g., ratings or buying frequency). The resulting model is referred to as the *user profile* because it conceptually relates user interests (ratings) to item attributes.

3. *Filtering and recommendation:* In this step, the learned model from the previous step is used to make recommendations on items for specific users. It is important for this step to be very efficient because the predictions need to be performed in real time.

In the following sections, we will describe each of these phases in detail. The second phase of learning often utilizes off-the-shelf classification models. The field of data classification is a vast area in its own right, and it is not the goal of this book to discuss classification models in detail. Therefore, throughout this chapter, we will assume a working familiarity with classification models. The goal will be to show how a specific classification model can be used as a black-box in the recommender system and the kinds of classification models that are specially suited to content-based recommender systems. A brief description of two of the most commonly used models is included, but this is by no means an exhaustive description. For the reader who is unfamiliar with classification models, pointers to several useful resources are included in the bibliographic notes.

4.3 Preprocessing and Feature Extraction

The first phase in all content-based models is to extract discriminative features for representing the items. Discriminative features are those, which are highly predictive of user interests. This phase is highly dependent on the specific application at hand. For example, a Web page recommendation system will be very different from a product recommendation system.

4.3.1 Feature Extraction

In the feature extraction phase, the descriptions of various items are extracted. Although it is possible to use any kind of representation, such as a multidimensional data representation, the most common approach is to extract keywords from the underlying data. This choice is because unstructured text descriptions are often widely available in a variety of domains, and they remain the most natural representations for describing items. In many cases, the items may have multiple fields describing various aspects of the item. For example, a merchant selling books may have descriptions of the books and keywords describing the

content, title, and author. In some cases, these descriptions can be converted into a bag of keywords. In other cases, one might work directly with a multidimensional (structured) representation. The latter is necessary when the attributes contain numerical quantities (e.g., price) or fields that are drawn from a small universe of possibilities (e.g., color).

The various fields need to be weighted appropriately in order to facilitate their use in the classification process. *Feature weighting* is closely related to *feature selection*, in that the former is a soft version of the latter. In the latter case, attributes are either included or not included depending on their relevance, whereas in the former case, features are given differential weight depending on their importance. The issue of feature selection will be discussed in detail in section 4.3.4. Because the feature extraction phase is highly application-specific, we provide the reader with a flavor of the types of features that may need to be extracted in the context of various applications.

4.3.1.1 Example of Product Recommendation

Consider a movie recommendation site[1] such as IMDb [699], that provides personalized recommendations for movies. Each movie is usually associated with a description of the movie such as its synopsis, the director, actors, genre, and so on. A short description of *Shrek* at the IMDb Website is as follows:

> "After his swamp is filled with magical creatures, an ogre agrees to rescue a princess for a villainous lord in order to get his land back."

Many other attributes, such as user tags, are also available, which can be treated as content-centric keywords.

In the case of *Shrek*, one might simply concatenate all the keywords in the various fields to create a text description. The main problem is that the various keywords may not have equal importance in the recommendation process. For example, a particular actor might have greater importance in the recommendation than a word from the synopsis. This can be achieved in two ways:

1. Domain-specific knowledge can be used to decide the relative importance of keywords. For example, the title of the movie and the primary actor may be given more weight than the words in the description. In many cases, this process is done in a heuristic way with trial and error.

2. In many cases, it may be possible to *learn* the relative importance of various features in an automated way. This process is referred to as feature weighting, which is closely related to feature selection. Both feature weighting and feature selection are described in a later section.

4.3.1.2 Example of Web Page Recommendation

Web documents require specialized preprocessing techniques because of some common properties of their structure and the richness of the links inside them. Two major aspects of Web document preprocessing include the removal of specific parts of the documents (e.g., tags) that are not useful and the leveraging of the actual structure of the document.

All fields in a Web document are not equally important. HTML documents have numerous fields in them, such as the title, the meta-data, and the body of the document.

[1]The exact recommendation method used by IMDb is proprietary and not known to the author. The description here is intended only for illustrative purposes.

Typically, analytical algorithms treat these fields with different levels of importance, and therefore weight them differently. For example, the title of a document is considered more important than the body and is weighted more heavily. Another example of a specially processed portion of a Web document is anchor text. Anchor text contains a description of the Web page pointed to by a link. Because of its descriptive nature, it is considered important, but it is sometimes not relevant to the topic of the page itself. Therefore, it is often removed from the text of the document. In some cases, where possible, anchor text could even be added to the text of the document *to which it points*. This is because anchor text is often a summary description of the document to which it points. The learning of the importance of these various features can be done through automated methods, as discussed in section 4.3.4.

A Web page may often be organized into content blocks that are not related to the primary subject matter of the page. A typical Web page will have many irrelevant blocks, such as advertisements, disclaimers, or notices, which are not very helpful for mining. It has been shown that the quality of mining results improves when only the text in the main block is used. However, the (automated) determination of main blocks from Web-scale collections is itself a data mining problem of interest. While it is relatively easy to decompose the Web page into blocks, it is sometimes difficult to identify the main block. Most automated methods for determining main blocks rely on the fact that a particular site will typically utilize a similar layout for all its documents. Therefore, the structure of the layout is learned from the documents at the site by extracting *tag trees* from the site. Other main blocks are then extracted through the use of the *tree-matching algorithm* [364, 662]. Machine learning methods can also be used for this task. For example, the problem of labeling the main block in a page can be treated as a classification problem. The bibliographic notes contain pointers to methods for extracting the main block from a Web document.

4.3.1.3 Example of Music Recommendation

Pandora Internet radio [693] is a well-known music recommendation engine, which associates tracks with features extracted from the Music Genome Project [703]. Examples of such features of tracks could be "feature trance roots," "synth riffs," "tonal harmonies," "straight drum beats," and so on. Users can initially specify a single example of a track of their interest to create a "station." Starting with this single training example, similar songs are played for the user. The users can express their likes or dislikes to these songs.

The user feedback is used to build a more refined model for music recommendation. It is noteworthy, that even though the underlying features are quite different in this case, they can still be treated as keywords, and the "document" for a given song corresponds to the bag of keywords associated with it. Alternatively, one can associate specific attributes with these different keywords, which leads to a structural multidimensional representation.

The initial specification of a track of interest is more similar to a knowledge-based recommender system than a content-based recommender system. Such types of knowledge-based recommender systems are referred to as case-based recommender systems. However, when ratings are leveraged to make recommendations, the approach becomes more similar to that of a content-based recommender system. In many cases, Pandora also provides an explanation for the recommendations in terms of the item attributes.

4.3.2 Feature Representation and Cleaning

This process is particularly important when the unstructured format is used for representation. The feature extraction phase is able to determine bags of words from the unstructured descriptions of products or Web pages. However, these representations need to be cleaned and represented in a suitable format for processing. There are several steps in the cleaning process:

1. *Stop-word removal:* Much of the text that is extracted from free-form descriptions of items, will contain many words that are not specific to the item but that are a common part of English vocabulary. Such words are typically high-frequency words. For example, words such as '*a*," "*an*," and "*the*" will not be particularly specific to the item at hand. In the movie recommendation application, it is common to find such words in the synopsis. In general, articles, prepositions, conjunctions, and pronouns are treated as stop-words. In most cases, standardized lists of stop-words are available in various languages.

2. *Stemming:* In stemming, variations of the same word are consolidated. For example, singular and plural forms of a word or different tenses of the same word are consolidated. In some cases, common roots are extracted from various words. For example, words such as "hoping" and "hope" are consolidated into the common root "hop." Of course, stemming can sometimes have a detrimental effect, because a word such as "hop" has a different meaning of its own. Many off-the-shelf tools [710–712] are available for stemming.

3. *Phrase extraction:* The idea is to detect words that occur together in documents on a frequent basis. For example, a phrase such as "hot dog" means something different from its constituent words. Manually defined dictionaries are available for phrase extraction, although automated methods can also be used [144, 364, 400].

After executing these steps, the keywords are converted into a *vector-space representation*. Each word is also referred to as a *term*. In the vector-space representation, documents are represented as bags of words, together with their frequencies. Although it might be tempting to use the raw frequency of word occurrence, this is often not desirable. This is because commonly occurring words are often statistically less discriminative. Therefore, such words are often discounted by down-weighting. This is similar to the principle of stop-words, except that it is done in a soft way by discounting the word, rather than completely removing it.

How are words discounted? This is achieved by using the notion of *inverse document frequency*. The inverse document frequency id_i of the ith term is a decreasing function of the number of documents n_i in which it occurs.

$$id_i = \log(n/n_i) \tag{4.1}$$

Here, the number of documents in the collection is denoted by n.

Furthermore, care needs to be taken that the excessive occurrence of a single word in the collection is not given too much importance. For example, when item descriptions are collected from unreliable sources or open platforms, such as the Web, they are liable to contain a significant amount of spam. To achieve this goal, a damping function $f(\cdot)$, such as the square root or the logarithm, is optionally applied to the frequencies before similarity computation.

$$f(x_i) = \sqrt{x_i}$$
$$f(x_i) = \log(x_i)$$

Frequency damping is optional and is often omitted. Omitting the damping process is equivalent to setting $f(x_i)$ to x_i. The *normalized* frequency $h(x_i)$ for the ith word is defined by combining the inverse document frequency with the damping function:

$$h(x_i) = f(x_i)id_i \tag{4.2}$$

This model is popularly referred to as the tf-idf model, where tf represents the term frequency and idf represents the inverse document frequency.

4.3.3 Collecting User Likes and Dislikes

Aside from the content about the items, it is also necessary to collect data about the user likes and dislikes for the recommendation process. The data collection is done during the offline phase, whereas recommendations are determined during the online phase when a specific user is interacting with the system. The user for whom the prediction is performed at any given time is referred to as the *active user*. During the online phase, the user's own preferences are combined with the content to create the predictions. The data about user likes and dislikes can take on any of the following forms:

1. *Ratings:* In this case, users specify ratings indicating their preference for the item. Ratings can be binary, interval-based, or ordinal. In rare cases, ratings can even be real valued. The nature of the rating has a significant impact on the model used for learning the user profiles.

2. *Implicit feedback:* Implicit feedback refers to user actions, such as buying or browsing an item. In most cases, only the positive preferences of a user are captured with implicit feedback but not negative preferences.

3. *Text opinions:* In many cases, users may express their opinions in the form of text descriptions. In such cases, implicit ratings can be extracted from these opinions. This form of rating extraction is related to the field of *opinion mining* and *sentiment analysis*. This area is beyond the scope of this book. Interested readers are referred to [364].

4. *Cases:* Users might specify examples (or *cases*) of items that they are interested in. Such cases can be used as implicit feedback with nearest neighbor or Rocchio classifiers. However, when the similarity retrieval is used in conjunction with carefully designed utility functions, these methods are more closely related to case-based recommender systems. Case-based systems are a subclass of knowledge-based recommender systems in which domain knowledge is used to discover matching items, instead of learning algorithms (cf. section 5.3.1 of Chapter 5). It is often difficult to delineate where content-based recommender systems end and knowledge-based recommender systems begin. For example, Pandora Internet Radio often uses an initial case of an interesting music album to set up "radio stations" for users with similar music items. At a later stage, user feedback about likes and dislikes is utilized to refine the recommendations. Therefore, the first part of the approach can be viewed as a knowledge-based system, and the second part of the approach can be viewed as a content-based (or collaborative) system.

In all the aforementioned cases, the likes or dislikes of a user for an item are finally converted into a unary, binary, interval-based, or real rating. This rating can also be viewed as the extraction of a *class label* or *dependent variable*, which is eventually leveraged for learning purposes.

4.3.4 Supervised Feature Selection and Weighting

The goal of feature selection and weighting is to ensure that only the most informative words are retained in the vector-space representation. In fact, many well-known recommender systems [60, 476] explicitly advocate that a size cut-off should be used on the number of keywords. The experimental results in [476], which were performed in a number of domains, suggested that the number of extracted words should be somewhere between 50 and 300. The basic idea is that the noisy words often result in overfitting and should therefore be removed *a priori*. This is particularly important, considering the fact that the number of documents available to learn a particular user profile is often not very large. When the number of documents available for learning is small, the tendency of the model to overfit will be greater. Therefore, it is crucial to reduce the size of the feature space.

There are two distinct aspects to incorporating the feature informativeness in the document representation. One is feature *selection*, which corresponds to the removal of words. The second is feature *weighting*, which involves giving greater importance to words. Note that stop-word removal and the use of inverse-document frequency are examples of feature selection and weighting, respectively. However, these are unsupervised ways of feature selection and weighting, where user feedback is given no importance. In this section, we will study supervised methods for feature selection, which take the user ratings into account for evaluating feature informativeness. Most of these methods evaluate the sensitivity of the dependent variable to a feature in order to evaluate its informativeness.

The measures for computing feature informativeness can be used to either perform a hard selection of features or to heuristically weight the features with a function of the computed quantification of informativeness. The measures used for feature informativeness are also different, depending on whether the user rating is treated as a numeric or categorical value. For example, in the context of binary ratings (or ratings with a small number of discrete values), it makes sense to use categorical rather than numeric representations. We will also describe a few methods that are commonly used for feature weighting. In most of the following descriptions, we will assume an unstructured (textual) representation, although the methods can be generalized easily to structured (multidimensional) representations. This is because the vector-space representation of text can be viewed as a special case of the multidimensional representation. The bibliographic notes contain pointers to more details of feature selection methods.

4.3.4.1 Gini Index

The Gini index is one of the most commonly used measures for feature selection. It is a simple and intuitive measure, which is easy to understand. The Gini index is inherently suited to binary ratings, ordinal ratings, or ratings which are distributed into a small number of intervals. The latter case may sometimes be obtained by discretizing the ratings. The ordering among the ratings is ignored, and each possible value of the rating is treated as an instance of a categorical attribute value. This might seem like a disadvantage because it loses information about the relative ordering of the ratings. However, in practice, the number of possible ratings is usually small and therefore significant accuracy is not lost.

Let t be the total number of possible values of the rating. Among documents containing a particular word w, let $p_1(w) \ldots p_t(w)$ be the fraction of the items rated at each of these t possible values. Then, the Gini index of the word w is defined as follows:

$$\text{Gini}(w) = 1 - \sum_{i=1}^{t} p_i(w)^2 \tag{4.3}$$

The value of Gini(w) always lies in the range $(0, 1-1/t)$, with smaller values being indicative of greater discriminative power. For example, when the presence of the word w always results in the document being rated at the jth possible rating value (i.e., $p_j(w) = 1$), then such a word is very discriminative for rating predictions. Correspondingly, the value of the Gini index in such a case is $1 - 1^2 = 0$. When each value of $p_j(w)$ takes on the same value of $1/t$, the Gini index takes on its maximum value of $1 - \sum_{i=1}^{t}(1/t^2) = 1 - 1/t$.

4.3.4.2 Entropy

Entropy is very similar in principle to the Gini index except that information-theoretic principles are used to design the measure. As in the previous case, let t be the total number of possible values of the rating and $p_1(w) \ldots p_t(w)$ be the fraction of the documents containing a particular word w, which are rated at each of these t possible values. Then, the entropy of the word w is defined as follows:

$$\text{Entropy}(w) = -\sum_{i=1}^{t} p_i(w)\log(p_i(w)) \tag{4.4}$$

The value of Entropy(w) always lies in the range $(0, 1)$, with smaller values being more indicative of discriminative power. It is easy to see that entropy has similar characteristics with the Gini index. In fact, these two measures often yield very similar results although they have different probabilistic interpretations. The Gini index is easier to understand, whereas entropy measures are more firmly grounded in mathematical principles from information theory.

4.3.4.3 χ^2-Statistic

The χ^2-statistic can be computed by treating the co-occurrence between the word and class as a *contingency table*. For example, consider a scenario where we are trying to determine whether a particular word is relevant to a user's buying interests. Assume that the user has bought about 10% of the items in the collection, and the word w occurs in about 20% of the descriptions. Assume that the total number of items (and corresponding documents) in the collection is 1000. Then, the *expected* number of occurrences of each possible combination of word occurrence and class contingency is as follows:

	Term occurs in description	Term does not occur
User bought item	$1000 * 0.1 * 0.2 = 20$	$1000 * 0.1 * 0.8 = 80$
User did not buy item	$1000 * 0.9 * 0.2 = 180$	$1000 * 0.9 * 0.8 = 720$

The aforementioned expected values are computed under the assumption that the occurrence of the term in the description and the user interest in the corresponding item are independent of one another. If these two quantities are independent, then clearly the term will be irrelevant to the learning process. However, in practice, the item may be highly related to the item at hand. For example, consider a scenario where the contingency table deviates from expected values and the user is very likely to buy the item containing the term. In such a case, the contingency table may appear as follows:

	Term occurs in description	Term does not occur
User bought item	$O_1 = 60$	$O_2 = 40$
User did not buy item	$O_3 = 140$	$O_4 = 760$

The χ^2-statistic measures the normalized deviation between observed and expected values across the various cells of the contingency table. In this case, the contingency table contains $p = 2 \times 2 = 4$ cells. Let O_i be the observed value of the ith cell and E_i be the expected value of the ith cell. Then, the χ^2-statistic is computed as follows:

$$\chi^2 = \sum_{i=1}^{p} \frac{(O_i - E_i)^2}{E_i} \tag{4.5}$$

Therefore, in the particular example of this table, the χ^2-statistic evaluates to the following:

$$\begin{aligned}\chi^2 &= \frac{(60-20)^2}{20} + \frac{(40-80)^2}{80} + \frac{(140-180)^2}{180} + \frac{(760-720)^2}{720} \\ &= 80 + 20 + 8.89 + 2.22 \\ &= 111.11\end{aligned}$$

It is also possible to compute the χ^2-statistic as a function of the observed values in the contingency table without explicitly computing expected values. This is possible because the expected values are functions of the aggregate observed values across rows and columns. A simple arithmetic formula to compute the χ^2-statistic in a 2×2 contingency table is as follows (see Exercise 8):

$$\chi^2 = \frac{(O_1 + O_2 + O_3 + O_4) \cdot (O_1 O_4 - O_2 O_3)^2}{(O_1 + O_2) \cdot (O_3 + O_4) \cdot (O_1 + O_3) \cdot (O_2 + O_4)} \tag{4.6}$$

Here, $O_1 \ldots O_4$ are the observed frequencies according to the table above. It is easy to verify that this formula yields the same χ^2-statistic of 111.11. Note that the χ^2-test can also be interpreted in terms of the probabilistic level of significance with the use of a χ^2 distribution. However, for practical purposes, it is sufficient to know that larger values of the χ^2-statistic indicate that a particular term and item are related to a greater degree. Note that if the observed values are exactly equal to the expected values, then it implies that the corresponding term is irrelevant to the item at hand. In such a case, the χ^2-statistic will evaluate to its least possible value of 0. Therefore, the top-k features with the largest χ^2-statistic are retained.

4.3.4.4 Normalized Deviation

The problem with most of the aforementioned measures is that they lose information about the relative ordering of ratings. For cases in which the ratings have high granularity, the normalized deviation is an appropriate measure.

Let σ^2 be the variance of the ratings in all the documents. Furthermore, let $\mu^+(w)$ be the average rating of all documents containing the word w, and $\mu^-(w)$ be the average rating of all documents that do not contain the word w. Then, the normalized deviation of the word w is defined as follows:

$$\text{Dev}(w) = \frac{|\mu^+(w) - \mu^-(w)|}{\sigma} \tag{4.7}$$

Larger values of $\text{Dev}(w)$ are indicative of more discriminatory words.

The aforementioned quantification is based on the relative distribution of the ratings for documents containing a specific word with respect to the ratings distribution of all documents. Such an approach is particularly suitable when ratings are treated as numerical quantities. A related measure is the Fisher's discrimination index, which computes the ratio of the inter-class separation to the intra-class separation in the *feature space* (rather than on the ratings dimension). This measure is described in detail in [22]. Fisher's discriminant index is however, better suited to categorical dependent variables rather than numerical dependent variables, such as ratings.

4.3.4.5 Feature Weighting

Feature Weighting can be viewed as a soft version of feature selection. In the earlier section on feature representation in this chapter, it was already discussed how measures such as the inverse document frequency can be used to weight documents. However, the inverse document frequency is an unsupervised measure that does not depend on user likes or dislikes. A supervised measure can also be used to further weight the vector-space representation in order to yield differential importance to different words. For example, in a movie recommendation application, keywords describing a movie genre or actor name are more important than words selected from the synopsis of the movie. On the other hand, the words in the synopsis are also somewhat indicative of tastes. Therefore, they cannot be excluded either. Feature weighting is a more refined approach for discriminating between various words by using a weight rather than a hard binary decision. The simplest approach to feature weighting is to take any of the feature selection measures and use them to derive the weights. For example, the inverse of the Gini index or entropy could be used. In many cases, a heuristic function can be further applied on the selection measure to control the sensitivity of the weighting process. For example, consider the following weighting function $g(w)$ for word w, where a is a parameter greater than 1.

$$g(w) = a - \text{Gini}(w) \tag{4.8}$$

The resulting weight $g(w)$ will always lie in the range $(a-1, a)$. By varying the value of a, the sensitivity of the weighting process can be controlled. Smaller values of a will lead to greater sensitivity. The weight of each word w in the vector-space representation is then multiplied by $g(w)$. Similar weighting functions can be defined with respect to the entropy and the normalized deviation. The process of selecting an appropriate feature weighting is a highly heuristic process that varies significantly corresponding to the application at hand. The value of a can be viewed as a parameter of the weighting function. It is also possible to learn the optimal parameters of such a function using *cross-validation* techniques. Such techniques are discussed in Chapter 7.

4.4 Learning User Profiles and Filtering

The learning of user profiles is closely related to the classification and regression modeling problem. When the ratings are treated as discrete values (e.g., "thumbs up" or "thumbs down"), the problem is similar to that of text classification. On the other hand, when the ratings are treated as a set of numerical entities, the problem is similar to that of regression modeling. Furthermore, the learning problem can be posed in both structured and unstructured domains. For homogeneity in presentation, we will assume that the descriptions of

items are in the form of documents. However, the approach can easily be generalized to any type of multidimensional data because text is a special type of multidimensional data.

In each case, we assume that we have a set $\mathcal{D}_L$ of training documents, which are labeled by a specific user. This user is also referred to as the active user when that user obtains a recommendation from the system. The training documents correspond to the descriptions of items, which are extracted in the preprocessing and feature selection phases. Furthermore, the training data contain the ratings assigned by the active user to these documents. These documents are used to construct a training model. Note that the labels assigned by other users (than the active user) are not used in the training process. Therefore, the training models are specific to particular users, and they cannot be used for arbitrarily chosen users. This is different from traditional collaborative filtering, in which methods like matrix factorization build a single model across all users. The training model for a specific user represents the user profile.

The labels on the documents correspond to the numeric, binary, or unary ratings. Assume that the ith document in $\mathcal{D}_L$ has a rating denoted by c_i. We also have a set $\mathcal{D}_U$ of testing documents, which are unlabeled. Note that both $\mathcal{D}_L$ and $\mathcal{D}_U$ are specific to a particular (active) user. The testing documents might correspond to descriptions of items, which might be potentially recommended to the user but which have not yet been bought or rated by the user. In domains such as news recommendation the documents in $\mathcal{D}_U$ might correspond to candidate Web documents for recommendation to the active user. The precise definition of $\mathcal{D}_U$ depends on the domain at hand, but the individual documents in $\mathcal{D}_U$ are extracted in a similar way to those in $\mathcal{D}_L$. The training model on $\mathcal{D}_L$ is used to make recommendations from $\mathcal{D}_U$ to the active user. As in the case of collaborative filtering, the model can be used to provide either a predicted value of the rating or a ranked list of top-k recommendations.

It is immediately evident that this problem is similar to that of classification and regression modeling in the text domain. The reader is referred to a recent survey [21] for a detailed discussion of many of these techniques. In the following, we will discuss some of the common learning methods.

4.4.1 Nearest Neighbor Classification

The nearest neighbor classifier is one of the simplest classification techniques, and it can be implemented in a relatively straightforward way. The first step is to define a similarity function, which is used in the nearest neighbor classifier. The most commonly used similarity function is the cosine function. Let $\overline{X} = (x_1 \ldots x_d)$ and $\overline{Y} = (y_1 \ldots y_d)$ be a pair of documents, in which the normalized frequencies of the ith word are given by x_i and y_i, respectively, in the two documents. Note that these frequencies are normalized or weighted with the use of unsupervised tf-idf weighting or the supervised methods discussed in the previous section. Then, the cosine measure is defined using these normalized frequencies as follows:

$$\text{Cosine}(\overline{X}, \overline{Y}) = \frac{\sum_{i=1}^{d} x_i y_i}{\sqrt{\sum_{i=1}^{d} x_i^2} \sqrt{\sum_{i=1}^{d} y_i^2}} \tag{4.9}$$

The cosine similarity is frequently used in the text domain because of its ability to adjust to the varying lengths of the underlying documents. When this approach is used for other types of structured and multidimensional data, other similarity/distance functions, such as the Euclidean distance and Manhattan distance, are used. For relational data with categorical attributes, various match-based similarity measures are available [22].

This similarity function is useful in making predictions for items (documents) in which the user preference is unknown. For each document in $\mathcal{D}_U$, its k-nearest neighbors in $\mathcal{D}_L$ are determined using the cosine similarity function. The average value of the rating for the k neighbors of each item in $\mathcal{D}_U$ is determined. This average value is the predicted rating for the corresponding item in $\mathcal{D}_U$. An additional heuristic enhancement is that one can weight each rating with the similarity value. In cases where ratings are treated as categorical values, the number of votes for each value of the rating is determined, and the rating value with the largest frequency is predicted. The documents in $\mathcal{D}_U$ are then ranked based on the predicted value of the rating, and the top items are recommended to the user.

The main challenge with the use of this approach is its high computational complexity. Note that the nearest neighbor of each document in $\mathcal{D}_U$ needs to be determined, and the time required for each nearest neighbor determination is linear to the size of $\mathcal{D}_L$. Therefore, the computational complexity is equal to $|\mathcal{D}_L| \times |\mathcal{D}_U|$. One way of making the approach faster is to use clustering to reduce the number of training documents in $\mathcal{D}_L$. For each distinct value of the rating, the corresponding subset of documents in $\mathcal{D}_L$ are clustered into $p \ll |\mathcal{D}_L|$ groups. Therefore, if there are s distinct values of the ratings, then the total number of groups is $p \cdot s$. Typically, a fast centroid-based (i.e., k-means) clustering is used to create each group of p clusters. Note that the number of groups $p \cdot s$ is significantly smaller than the number of training documents. In such cases, each group is converted into a larger document corresponding to the concatenation[2] of the documents in that group. The vector-space representation of this larger document can be extracted by adding up the word frequencies of its constituents. The corresponding rating label associated with the document is equal to the rating of the constituent documents. For each target document T, the closest $k < p$ documents are found from this newly created set of p documents. The average rating of this set of k documents is returned as the label for the target. As in the previous case, the rating is predicted for each item in $\mathcal{D}_U$, and the top-ranked items are returned to the active user. This approach speeds up the classification process, because one must compute the similarity between the target document and a relatively small number of aggregated documents. Even though this approach incurs an additional preprocessing overhead of clustering, this overhead is generally small compared to the savings at recommendation time when the sizes of $\mathcal{D}_L$ and $\mathcal{D}_U$ are large.

A special case of this clustering-based approach is one in which all documents belonging to a particular value of the rating are aggregated into a single group. Thus, the value of p is set to 1. The vector-space representation of the resulting vector of each group is also referred to as the *prototype* vector. For a test document, the rating of the closest document is reported as the relevant one for the target. This approach is closely related to Rocchio classification, which also allows for the notion of *relevance feedback* from the active user. The Rocchio method was originally designed for binary classes, which, in our case, translate to binary ratings. The bibliographic notes contain pointers to the Rocchio method.

4.4.2 Connections with Case-Based Recommender Systems

Nearest neighbor methods are connected to knowledge-based recommender systems in general, and case-based recommender systems in particular. Knowledge-based recommender systems are discussed in detail in Chapter 5. The main difference is that in case-based recommender systems, the user interactively specifies a *single* example of interest, and the nearest neighbors of this example are retrieved as possible items of interest for the user.

[2]For structured data, the centroid of the group may be used.

Furthermore, a significant amount of domain knowledge is used in the design of the similarity function, because only a single example is available. This single example can be more appropriately viewed as a user requirement rather than a historical rating, because it is specified interactively. In knowledge-based systems, there is less emphasis on using historical data or ratings. Like the Rocchio method, such methods are also interactive, although the interactivity is far more sophisticated in case-based systems.

4.4.3 Bayes Classifier

The Bayes classifier is discussed in section 3.4 of Chapter 3 in collaborative filtering. However, the discussion in Chapter 3 is a non-standard use of the Bayes model in which the missing entries are predicted from the specified ones. In the context of content-based recommender systems, the problem translates to a more conventional use of the Bayes model for text classification. Therefore, we will revisit the Bayes model in the context of text classification.

In this case, we have a set $\mathcal{D}_L$ containing the training documents, and a set $\mathcal{D}_U$ containing the test documents. For ease in discussion, we will assume that the labels are binary in which users specify either a like or a dislike rating as $+1$ or -1, respectively for each of the training documents in $\mathcal{D}_L$. It is, however, relatively easy to generalize this classifier to the case where the ratings take on more than two values.

As before, assume that the rating of the ith document in $\mathcal{D}_L$ is denoted by $c_i \in \{-1, 1\}$. Therefore, this labeled set represents the user profile. There are two models that are commonly used in text data, which correspond to the Bernoulli and the multinomial models, respectively. In the following, we will discuss only the Bernoulli model. The multinomial model is discussed in detail in [22].

In the Bernoulli model, the frequencies of the words are ignored, and only the presence or absence of the word in the document is considered. Therefore, each document is treated as a binary vector of d words containing only values of 0 and 1. Consider a target document $\overline{X} \in \mathcal{D}_U$, which might correspond to the description of an item. Assume that the d binary features in $\overline{X}$ are denoted by $(x_1 \ldots x_d)$. Informally, we would like to determine $P(\text{Active user likes } \overline{X} | x_1 \ldots x_d)$. Here, each x_i is a 0-1 value, corresponding to whether or not the ith word is present in the document $\overline{X}$. Then, if the class (binary rating) of $\overline{X}$ is denoted by $c(\overline{X})$, this is equivalent to determining the value of $P(c(\overline{X}) = 1 | x_1 \ldots x_d)$. By determining both $P(c(\overline{X}) = 1 | x_1 \ldots x_d)$ and $P(c(\overline{X}) = -1 | x_1 \ldots x_d)$ and selecting the larger of the two, one can determine whether or not the active user likes $\overline{X}$. These expressions can be evaluated by using the Bayes rule and then applying a *naive assumption* as follows:

$$\begin{aligned} P(c(\overline{X}) = 1 | x_1 \ldots x_d) &= \frac{P(c(\overline{X}) = 1) \cdot P(x_1 \ldots x_d | c(\overline{X}) = 1)}{P(x_1 \ldots x_d)} \\ &\propto P(c(\overline{X}) = 1) \cdot P(x_1 \ldots x_d | c(\overline{X}) = 1) \\ &= P(c(\overline{X}) = 1) \cdot \prod_{i=1}^{d} P(x_i | c(\overline{X}) = 1) \quad \text{[Naive Assumption]} \end{aligned}$$

The naive assumption states that the occurrences of words in documents are conditionally independent events (on a specific class), and therefore one can replace $P(x_1 \ldots x_d | c(\overline{X}) = 1)$ with $\prod_{i=1}^{d} P(x_i | c(\overline{X}) = 1)$. Furthermore, the constant of proportionality is used in the first relationship because the denominator is independent of the class. Therefore, the denominator does not play any role in deciding between the relative order of the classes.

The denominator, however, does play a role in terms of ranking the propensity of *different items (documents)* to be liked by the user. This is relevant to the problem of *ranking* items for a specific user, in order of $P(c(\overline{X}) = 1|x_1 \ldots x_d)$.

In cases where such a ranking of the items is needed, the constant of proportionality is no longer irrelevant. This is particularly common in recommendation applications where it is not sufficient to determine the relative probabilities of items belonging to different rating values, but to actually rank them with respect to one another. In such cases, the constant of proportionality needs to be determined. Assume that the constant of proportionality in the relationship above is denoted by K. The constant of proportionality K can be obtained by using the fact that the sum of the probabilities of all possible instantiations of $c(\overline{X})$ should always be 1. Therefore, we have:

$$K \cdot \left[P(c(\overline{X}) = 1) \cdot \prod_{i=1}^{d} P(x_i|c(\overline{X}) = 1) + P(c(\overline{X}) = 1) \cdot \prod_{i=1}^{d} P(x_i|c(\overline{X}) = 1) \right] = 1$$

Therefore, we can derive the following value for K:

$$K = \frac{1}{P(c(\overline{X}) = 1) \cdot \prod_{i=1}^{d} P(x_i|c(\overline{X}) = 1) + P(c(\overline{X}) = -1) \cdot \prod_{i=1}^{d} P(x_i|c(\overline{X}) = -1)}$$

This approach is used to determine the probability of a user liking each possible item in $\mathcal{D}_U$. The items in $\mathcal{D}_U$ are then ranked according to this probability and presented to the user. These methods are particularly well suited to binary ratings. There are other ways of using the probability to estimate the predicted value of the ratings and rank the items when dealing with ratings that are not necessarily binary. Such methods are discussed in detail in section 3.4 of Chapter 3.

4.4.3.1 Estimating Intermediate Probabilities

The Bayes method requires the computation of intermediate probabilities such as $P(x_i|c(\overline{X}) = 1)$. So far, we have not yet discussed how these probabilities can be estimated in a data-driven manner. The main utility of the aforementioned Bayes rule is that it expresses the prediction probabilities in terms of other probabilities [e.g., $P(x_i|c(\overline{X}) = 1)$] that can be estimated more easily in a data-driven way. We reproduce the Bayes condition above:

$$P(c(\overline{X}) = 1|x_1 \ldots x_d) \propto P(c(\overline{X}) = 1) \cdot \prod_{i=1}^{d} P(x_i|c(\overline{X}) = 1)$$

$$P(c(\overline{X}) = -1|x_1 \ldots x_d) \propto P(c(\overline{X}) = -1) \cdot \prod_{i=1}^{d} P(x_i|c(\overline{X}) = -1)$$

In order to compute the Bayes probabilities, we need to estimate the probabilities on the right-hand side of the equations above. These include the prior class probabilities $P(c(\overline{X}) = 1)$ and $P(c(\overline{X}) = -1)$. Furthermore, the feature-wise conditional probabilities, such as $P(x_i|c(\overline{X}) = 1)$ and $P(x_i|c(\overline{X}) = -1)$, need to be estimated. The probability $P(c(\overline{X}) = 1)$ can be estimated as the fraction of positive training examples $\mathcal{D}_L^+$ in the labeled data $\mathcal{D}_L$. In order to reduce overfitting, Laplacian smoothing is performed by adding values proportional to a small parameter $\alpha > 0$ to the numerator and denominator.

$$P(c(\overline{X}) = 1) = \frac{|\mathcal{D}_L^+| + \alpha}{|\mathcal{D}_L| + 2 \cdot \alpha} \tag{4.10}$$

Table 4.1: Illustration of the Bayes method for a content-based system

Keyword ⇒ Song-Id ⇓	Drums	Guitar	Beat	Classical	Symphony	Orchestra	Like or Dislike
1	1	1	1	0	0	0	Dislike
2	1	1	0	0	0	1	Dislike
3	0	1	1	0	0	0	Dislike
4	0	0	0	1	1	1	Like
5	0	1	0	1	0	1	Like
6	0	0	0	1	1	0	Like
Test-1	0	0	0	1	0	0	?
Test-2	1	0	1	0	0	0	?

The value of $P(c(\overline{X}) = -1)$ is estimated in an exactly similar way. Furthermore, the conditional feature probability $P(x_i|c(\overline{X}) = 1)$ is estimated as the fraction of the instances in the positive class for which the ith feature takes on the value of x_i. Let $q^+(x_i)$ represent the number of instances in the positive class that take on the value of $x_i \in \{0, 1\}$ for the ith feature. Then, we can use a Laplacian smoothing parameter $\beta > 0$ to estimate the probability as follows:

$$P(x_i|c(\overline{X}) = 1) = \frac{q^+(x_i) + \beta}{|\mathcal{D}_L^+| + 2 \cdot \beta} \tag{4.11}$$

A similar approach can be used to estimate $P(x_i|c(\overline{X}) = -1)$. Note that the Laplacian smoothing is helpful for cases where little training data are available. In the extreme case, where $\mathcal{D}_L^+$ is empty, the probability $P(x_i|c(\overline{X}) = 1)$ would be (appropriately) estimated to be 0.5 as a kind of prior belief. Without smoothing, such an estimation would be indeterminate, because both the numerator and denominator of the ratio would be 0. Laplacian smoothing, like many regularization methods, can be interpreted in terms of the greater importance of prior beliefs when the amount of training data is limited. Although we have presented the aforementioned estimation for the case of binary ratings, it is relatively easy to generalize the estimation when there are k distinct values of the ratings. A similar type of estimation is discussed in the context of collaborative filtering in section 3.4 of Chapter 3.

4.4.3.2 Example of Bayes Model

We provide an example of the use of the Bayes model for a set of 6 training examples and two test examples. In Table 4.1, the columns correspond to features representing properties of various songs. The user like or dislike is illustrated in the final column of the table. Therefore, the final column can be viewed as the rating. The first 6 rows correspond to the training examples, which correspond to the user profile. The final pair of rows correspond to two candidate music tracks that need to be ranked for the specific user at hand. In machine learning parlance, these rows are also referred to as test instances. Note that the final (dependent variable) column is specified only for the training rows because the user like or dislike (ratings) are not known for the test rows. These values need to be predicted.

By examining the features in Table 4.1, it becomes immediately evident that the first three features (columns) might often occur in many popular music genres such as rock music, whereas the final three features typically occur in classical music. The user profile, represented by Table 4.1 clearly seems to suggest a preference for classical music over rock

music. Similarly, among the test examples, only the first of the two examples seems to match the user's interests. Let us examine how the Bayes approach is able to derive this fact in a data-driven way. For ease in computation, we will assume that Laplacian smoothing is not used, although it is important to use such smoothing methods in real applications.

By using the Bayes model, we can derive the conditional probabilities for likes and dislikes based on the observed features of the test examples:

$$\begin{aligned} P(\text{Like}|\,\textit{Test-1}) &\propto 0.5 \prod_{i=1}^{6} P(\text{Like}|x_i) \\ &= (0.5) \cdot \frac{3}{4} \cdot \frac{2}{2} \cdot \frac{3}{4} \cdot \frac{3}{3} \cdot \frac{1}{4} \cdot \frac{1}{3} \\ &= \frac{3}{128} \\ P(\text{Dislike}|\,\textit{Test-1}) &\propto 0.5 \prod_{i=1}^{6} P(\text{Dislike}|x_i) \\ &= (0.5) \cdot \frac{1}{4} \cdot \frac{0}{2} \cdot \frac{1}{4} \cdot \frac{0}{3} \cdot \frac{3}{4} \cdot \frac{2}{3} \\ &= 0 \end{aligned}$$

By normalizing the two probabilities to sum to 1, we obtain the result that $P(\text{Like}|\,\textit{Test-1})$ is 1 and $P(\text{Dislike}|\,\textit{Test-1})$ is 0. In the case of *Test-2*, exactly the opposite result is obtained where $P(\text{Like}|\,\textit{Test-2})$ is 0. Therefore, *Test-1* should be recommended to the active user over *Test-2*. This is the same result that we obtained on visual inspection of this example.

When Laplacian smoothing is used, we will not obtain such binary probability values for the various classes, although one of the classes will obtain a much higher probability than the other. In such cases, all the test examples can be ranked in order of their predicted probability of a "Like" and recommended to the user. Laplacian smoothing is advisable because a single 0-value in the product-wise form of the expression on the right-hand side of the Bayes rule can result in a conditional probability value of 0.

4.4.4 Rule-based Classifiers

Rule-based classifiers can be designed in a variety of ways, including leave-one-out methods, as well as associative methods. A detailed description of the various types of rule-based classifiers is provided in [18, 22]. In the following, we will discuss only associative classifiers because they are based on the simple principles of association rules. A discussion of rule-based methods is provided in section 3.3 of Chapter 3. Refer to that section for the basic definitions of association rules and their measures, such as *support* and *confidence*. The support of a rule defines the fraction of rows satisfying both the antecedent and the consequent of a rule. The confidence of a rule is the fraction of rows satisfying the consequent, from the rows already known to satisfy the antecedent. The concept of a row "satisfying" the antecedent or consequent is described in more detail below.

Rule-based classifiers in content-based systems are similar to rule-based classifiers in collaborative filtering. In the item-item rules of collaborative filtering, both the antecedents and consequents of rules correspond to ratings of items. The main difference is that the antecedents of the rules in collaborative filtering correspond[3] to the ratings of various items,

[3]A different approach in collaborative filtering is to leverage user-user rules. For user-user rules, the antecedents and consequents may both contain the ratings of specific users. Refer to section 3.3 of Chapter 3.

whereas the antecedents of the rules in content-based methods correspond to the presence of specific keywords in item descriptions. Therefore, the rules are of the following form:

$$\text{Item contains keyword set A} \Rightarrow \text{Rating= Like}$$
$$\text{Item contains keyword set B} \Rightarrow \text{Rating=Dislike}$$

Therefore, an antecedent of a rule is said to "satisfy" a particular row (keyword representation of item), if all keywords in the antecedent are contained in that row. The consequents correspond to the various ratings, which we have assumed to be binary likes or dislikes for simplicity. A row is said to satisfy the consequent of that rule if the rating value in the consequent matches the dependent variable (rating) of that row.

The first step is to leverage the active user profile (i.e., training documents) to mine all the rules at a desired level of support and confidence. As in all content-based methods, the rules are specific to the active user at hand. For example, in the case of Table 4.1, the active user seems to be interested in classical music. In this case, an example of a relevant rule, which has 33% support and 100% confidence, is as follows:

$$\{\text{Classical, Symphony}\} \Rightarrow \text{Like}$$

Therefore, the basic idea is to mine all such rules for a given active user. Then, for target items where the user's interests are unknown, it is determined which rules are *fired*. A rule is fired by a target item description if the former's antecedent keywords are included in the latter. Once all such fired rules have been determined for the active user, the average rating in the consequents of these rules is reported as the rating of the target item. Many different heuristics exist for combining the ratings of the consequents. For example, we can choose to weight the rating with the confidence of the rule while computing the average. In the event that no rule is fired, default heuristics need to be used. For example, one can determine the average rating of the active user over all items and also determine the average rating of the target item by all users. The average of these two quantities is reported. Therefore, the overall approach for rule-based classification can be described as follows:

1. **(Training phase:)** Determine all the relevant rules from the user profile at the desired level of minimum support and confidence from the training data set $\mathcal{D}_L$.

2. **(Testing phase)** For each item description in $\mathcal{D}_U$, determine the fired rules and an average rating. Rank the items in $\mathcal{D}_U$ on the basis of this average rating.

One advantage of rule-based systems is the high level of interpretability they provide. For example, for a recommended item, one can use the keywords in the antecedent of the fired rules to give a recommendation to the target user about why she might like a particular item.

4.4.4.1 Example of Rule-based Methods

In order to illustrate the use of rule-based methods, we will provide an example of the rules generated for the active user in Table 4.1. At a support level of 33% and confidence level of

75%, the following rules are generated along with their support-confidence values:

Rule 1: {Classical} ⇒ Like (50%, 100%)
Rule 2: {Symphony} ⇒ Like (33%, 100%)
Rule 3: {Classical, Symphony} ⇒ Like (33%, 100%)
Rule 4: {Drums, Guitar} ⇒ Dislike (33%, 100%)
Rule 5: {Drums} ⇒ Dislike (33%, 100%)
Rule 6: {Beat} ⇒ Dislike (33%, 100%)
Rule 7: {Guitar} ⇒ Dislike (50%, 75%)

The aforementioned rules are primarily sorted in order of decreasing confidence, with ties broken in order of decreasing support. It is evident that rule 2 is fired by *Test-1*, whereas rules 5 and 6 are fired by *Test-2*. Therefore, *Test-1* should be preferred over *Test-2* as a recommendation to the active user. Note that the rules fired by *Test-1* also provide an understanding of why it should be considered the best recommendation for the active user. Such explanations are often very useful in recommender systems both from the perspective of the customer and the perspective of the merchant.

4.4.5 Regression-Based Models

Regression-based models have the merit that they can be used for various types of ratings such as binary ratings, interval-based ratings, or numerical ratings. Large classes of regression models such as linear models, logistic regression models, and ordered probit models can be used to model various types of ratings. Here, we will describe the simplest model, which is referred to as *linear regression*. The bibliographic notes contain pointers to more sophisticated regression methods.

Let D_L be an $n \times d$ matrix representing the n documents in the labeled training set $\mathcal{D}_L$ on a lexicon of size d. Similarly, let $\overline{y}$ be an n-dimensional column vector containing the ratings of the active user for the n documents in the training set. The basic idea in linear regression is to assume that the ratings can be modeled as a linear function of the word frequencies. Let $\overline{W}$ be a d-dimensional row vector representing the coefficients of each word in the linear function relating word frequencies to the rating. Then, the linear regression model assumes that the word frequencies in the training matrix D_L are related to rating vectors as follows:

$$\overline{y} \approx D_L \overline{W}^T \tag{4.12}$$

Therefore, the vector $(D_L \overline{W}^T - \overline{y})$ is an n-dimensional vector of prediction errors. In order to maximize the quality of the prediction, one must minimize the squared norm of this vector. Furthermore, a regularization term $\lambda ||\overline{W}||^2$ may be added to the objective function in order to reduce overfitting. This form of regularization is also referred to as *Tikhonov regularization*. Here, $\lambda > 0$ is the regularization parameter. Therefore, the objective function O can be expressed as follows:

$$\text{Minimize } O = ||D_L \overline{W}^T - \overline{y}||^2 + \lambda ||\overline{W}||^2 \tag{4.13}$$

The problem can be solved by setting the gradient of this objective function with respect to $\overline{W}$ to 0. This results in the following condition:

$$D_L^T (D_L \overline{W}^T - \overline{y}) + \lambda \overline{W}^T = 0$$
$$(D_L^T D_L + \lambda I) \overline{W}^T = D_L^T \overline{y}$$

Table 4.2: The family of regression models and applicability to various types of ratings

Regression Model	Nature of Rating (Target Variable)
Linear Regression	Real
Polynomial Regression	Real
Kernel Regression	Real
Binary Logistic Regression	Unary, Binary
Multiway Logistic regression	Categorical, Ordinal
Probit	Unary, Binary
Multiway Probit	Categorical, Ordinal
Ordered Probit	Ordinal, Interval-based

The matrix $(D_L^T D_L + \lambda I)$ can be shown to be positive-definite, and therefore invertible (see Exercise 7). Therefore, we can directly solve for the weight vector $\overline{W}$ as follows:

$$\overline{W}^T = (D_L^T D_L + \lambda I)^{-1} D_L^T \overline{y} \tag{4.14}$$

Here, I is a $d \times d$ identity matrix. Therefore, a closed-form solution always exists for $\overline{W}^T$. For any given document vector (item description) $\overline{X}$ from the unlabeled set $\mathcal{D}_U$, its rating can be predicted as the dot product between $\overline{W}$ and $\overline{X}$. Tikhonov regularization uses the L_2-regularization term $\lambda \cdot ||W||^2$. It is also possible to use L_1-regularization, in which this term is replaced with $\lambda \cdot ||W||$. The resulting optimization problem does not have a closed-form solution, and gradient descent methods must be used. This form of regularization, also known as *Lasso* [242], can be used in the dual role of feature selection. This is because such methods have the tendency to select sparse coefficient vectors for $\overline{W}$, in which most components of $\overline{W}$ take on the value of 0. Such features can be discarded. Therefore, L_1-regularization methods provide highly interpretable insights about important subsets of features for the recommendation process. A detailed discussion of these models can be found in [22].

The linear model is one example of a regression model that is suitable for real-valued ratings. In practice, ratings might be unary, binary, interval-based, or categorical (small number of ordinal values). Various linear models have been designed for different types of target class variables. Some examples include logistic regression, probit regression, ordered probit regression, and nonlinear regression. Unary ratings are often treated as binary ratings, in which the unlabeled items are treated as negative instances. However, specialized positive-unlabeled (PU) models exist for such cases [364]. Ordered probit regression is especially useful for interval-based ratings. Furthermore, nonlinear regression models, such as polynomial regression and kernel regression, may be used in cases where the dependency between the features and target variables is nonlinear. When the number of features is large and the number of training samples is small, linear models usually perform quite well and may, in fact, outperform nonlinear models. This is because linear models are less prone to overfitting. Table 4.2 shows the mapping between the various regression models and the nature of the target variable (rating).

4.4.6 Other Learning Models and Comparative Overview

As the problem of content-based filtering is a direct application of classification and regression modeling, many other techniques can be used from the literature. A detailed discussion of various classification models can be found in [18, 86, 242, 436]. The decision-tree model

discussed in Chapter 3 can also be applied to content-based methods. However, for very high-dimensional data, such as text, decision trees often do not provide very effective results. Experimental results [477] have shown the poor performance of decision trees compared to other classification methods. Even though rule-based classifiers are closely related to decision trees, they can often provide superior results because they do not assume a strict partitioning of the feature space. Successful results have been obtained with rule-based classifiers for email classification [164, 165]. Among the various models, the Bayes approach has the advantage that it can handle all types of feature variables with the use of an appropriate model. Regression-based models are very robust, and they can handle all forms of target variables. Logistic regression and ordered probit regression are particularly useful for binary and interval-based ratings.

In the case of binary ratings, support vector machines [114] are a popular choice. Support vector machines are very similar to logistic regression; the main difference is that the loss is quantified as a *hinge-loss* rather than with the use of the *logit* function. Support vector machines are highly resistant to overfitting, and numerous off-the-shelf implementations exist. Both linear and kernel-based support vector machines have been used in the literature. For the case of high-dimensional data, such as text, it has been observed that linear support vector machines are sufficient. For such cases, specialized methods with linear performance [283] have been designed. Although neural networks [87] can be used for building arbitrarily complex models, they are not advisable when the amount of available data is small. This is because neural networks are sensitive to the noise in the underlying data, and they can overfit the training data when its size is small.

4.4.7 Explanations in Content-Based Systems

Since content-based systems extract models based on content features, they often provide highly interpretable insights for the recommendation process. For example, in a movie recommendation system, it is often useful to present the user with a reason as to why they might like a specific movie, such as the presence of a particular genre feature, actor feature, or an informative set of keywords. As a result, the active user will be able to make a more informed choice about whether they should watch that movie. Similarly, a descriptive set of keywords in a music recommendation system can provide a better understanding of why a user might like a specific track. As a specific example, Pandora Internet radio [693] provides explanations for recommended tracks, such as the following:

> "We are playing this track because it features trance roots, four-on-the-floor beats, disco influences, a knack for catchy hooks, beats made for dancing, straight drum beats, clear pronunciation, romantic lyrics, storytelling lyrics, subtle buildup/breakdown, a rhythmic intro, use of modal harmonies, the use of chordal patterning, light drum fills, emphasis on instrumental performance, a synth bass riff, synth riffs, subtle use of arpeggiatted synths, heavily effected synths, and synth swoops."

Each of these reported characteristics can be viewed as an important feature, which are responsible for the classification of the test instance as a "like." Note that such detailed explanations are often lacking in collaborative systems, where a recommendation can be explained only in terms of similar items, rather than in terms of detailed *characteristics* of these items. The nature and extent of the insights are, however, highly sensitive to the specific model used. For example, the Bayes model and rule-based systems are very highly

interpretable in terms of the specific causality of the classification. Consider the example of Table 4.1 in which the following rule is fired for the example *Test-1*:

$$\{\text{Symphony}\} \Rightarrow \text{Like}$$

It is evident that the user has been recommended the item described by *Test-1* because it is a symphony. Similarly, in the Bayes classification model, it is evident that the contribution of $P(\text{Symphony}|\text{Like})$ is largest in the multiplicative formula for classification. Other models, such as linear and nonlinear regression models, are harder to interpret. Nevertheless, certain instances of these models, such as *Lasso*, provide important insights about the most relevant features for the classification process.

4.5 Content-Based Versus Collaborative Recommendations

It is instructive to compare content-based methods with the collaborative methods discussed in Chapters 2 and 3. Content-based methods have several advantages and disadvantages as compared to collaborative methods. The advantages of content-based methods are as follows:

1. When a new item is added to a ratings matrix, it has no ratings from the various users. None of the memory-based and model-based collaborative filtering methods would recommend such an item, because sufficient ratings are not available for recommendation purposes. On the other hand, in the case of content-based methods, the previous items rated by a given user are leveraged to make recommendations. Therefore, as long as the *user* is not new, meaningful recommendations can always be made in a way that treats the new item in a fair way in comparison to other items. Collaborative systems have cold-start problems *both* for new users and new items, whereas content-based systems have cold-start problems only for new users.

2. As discussed in the previous section, content-based methods provide explanations in terms of the features of items. This is often not possible with collaborative recommendations.

3. Content-based methods can generally be used with off-the-shelf text classifiers. Furthermore, each user-specific classification problem is generally not very large, as in the case of collaborative systems. Therefore, they are particularly easy to use with relatively little engineering effort.

On the other hand, content-based methods also have several disadvantages that are not present in collaborative recommenders.

1. Content-based systems tend to find items that are similar to those the user has seen so far. This problem is referred to as *overspecialization*. It is always desirable to have a certain amount of novelty and serendipity in the recommendations. Novelty refers to the fact that the item is different from one the user has seen in the past. Similarly, serendipity implies that the user would like to discover *surprisingly relevant* items that they might otherwise not have found. This is a problem for content-based systems in which attribute-based classification models tend to recommend very similar items. For example, if a user has never listened to or rated classical music, a content-based

system will typically not recommend such an item to her because classical music will be described by very different attribute values than those that the user has rated so far. On the other hand, a collaborative system might recommend such items by leveraging the interests of her *peer* group. For example, a collaborative system might automatically infer a *surprising* association between certain pop songs and classical songs and recommend the corresponding classical songs to a user who is a pop music lover. Overspecialization and lack of serendipity are the two most significant challenges of content-based recommender systems.

2. Even though content-based systems help in resolving cold-start problems for new *items*, they do not help in resolving these problems for new *users*. In fact, for new users, the problem in content-based systems may be more severe because a text classification model usually requires a sufficient number of training documents to avoid overfitting. It would seem rather wasteful that the training data for all the other users is discarded and only the (small) training data set specific to a single user is leveraged.

In spite of these disadvantages, content-based systems often complement collaborative systems quite well because of their ability to leverage content-based knowledge in the recommendation process. This complementary behavior is often leveraged in *hybrid recommender systems* (cf. Chapter 6), in which the goal is to combine the best of both worlds to create an even more robust recommender system. In general, content-based systems are rarely used in isolation, and they are generally used in combination with other types of recommender systems.

4.6 Using Content-Based Models for Collaborative Filtering

There is an interesting connection between collaborative filtering models and content-based methods. It turns out that content-based methods can be directly used for collaborative filtering. Although the content description of an item refers to its descriptive keywords, it is possible to envision scenarios, where the ratings of users are leveraged to define content-based descriptions. For each item, one can concatenate the user name (or identifier) of a user who has rated the item with the value of this rating to create a new "keyword." Therefore, each item would be described in terms of as many keywords as the number of ratings of that item. For example, consider a scenario where the descriptions of various movies are as follows:

Terminator: John#Like, Alice#Dislike, Tom#Like
Aliens: John#Like, Peter#Dislike, Alice#Dislike, Sayani#Like
Gladiator: Jack#Like, Mary#Like, Alice#Like

The "#" symbol is used to denote the demarcation of the concatenation and ensure a unique keyword for each user-rating combination. This approach is generally more effective, when the number of possible ratings is small (e.g., unary or binary ratings). After such a content-based description has been constructed, it can be used in conjunction with an off-the-shelf content-based algorithm. There is almost a one-to-one mapping between the resulting methods and various collaborative filtering models, depending on the base method used for classification. Although each such technique maps to a collaborative filtering model,

the converse is not true because many collaborative filtering methods cannot be captured by this approach. Nevertheless, we provide some examples of the mapping:

1. A nearest neighbor classifier on this representation approximately maps to an item-based neighborhood model for collaborative filtering (cf. section 2.3.2 of Chapter 2).
2. A regression model on the content approximately maps to a user-wise regression model for collaborative filtering (cf. section 2.6.1 of Chapter 2).
3. A rule-based classifier on the content approximately maps to an user-wise rule-based classifier for collaborative filtering (cf. section 3.3.2 of Chapter 3).
4. A Bayes classifier on the content approximately maps to a user-wise Bayes model for collaborative filtering (cf. Exercise 4 of Chapter 3).

Therefore, many methods for collaborative filtering can be captured by defining an appropriate content representation and directly using off-the-shelf content-based methods. These observations are important because they open up numerous opportunities for hybridization. For example, one can combine the ratings-based keywords with actual descriptive keywords to obtain an even more robust model. In fact, this approach is often used in some hybrid recommendation systems. Such an approach no longer wastes the available ratings data from other users, and it combines the power of content-based and collaborative models within a unified framework.

4.6.1 Leveraging User Profiles

Another case in which collaborative filtering-like models can be created with content attributes is when user profiles are available in the form of specified keywords. For example, users may choose to specify their particular interests in the form of keywords. In such cases, instead of creating a local classification model for each user, one can create a global classification model over all users by using the user features. For each user-item *combination*, a content-centric representation can be created by using the Kronecker-product of the attribute vectors of the corresponding user and item [50]. A classification or regression model is constructed on this representation to map user-item *combinations* to ratings. Such an approach is described in detail in section 8.5.3 of Chapter 8.

4.7 Summary

This chapter introduces the methodology of content-based recommender systems in which user-specific training models are created for the recommendation process. The content attributes in item descriptions are combined with user ratings to create user profiles. Classification models are created on the basis of these models. These models are then used to classify item descriptions that have as of yet not been rated by the user. Numerous classification and regression models are used by such systems, such as nearest-neighbor classifiers, rule-based methods, the Bayes method, and linear models. The Bayes method has been used with great success in a variety of scenarios because of its ability to handle various types of content. Content-based systems have the advantage that they can handle cold-start problems with respect to new items, although they cannot handle cold-start problems with respect to new users. The serendipity of content-based systems is relatively low because content-based recommendations are based on the content of the items previously rated by the user.

4.8 Bibliographic Notes

The earliest content-based systems were attributed to the work in [60] and the *Syskill & Webert* [82, 476–478] systems. *Fab*, however, uses a partial hybridization design in which the peer group is determined using content-based methods, but the ratings of other users are leveraged in the recommendation process. The works in [5, 376, 477] provide excellent overview articles on content-based recommender systems. The latter work was designed for finding interesting Websites, and therefore numerous text classifiers were tested for their effectiveness. In particular, the work in [82] provides a number of useful pointers about the relative performance of various content-based systems. Probabilistic methods for user modeling are discussed in [83]. The work in [163, 164] is notable for its use of rule-based systems in e-mail classification. Rocchio's relevance feedback [511] was also used during the early years, although the work does not have theoretical underpinnings, and it can often perform poorly in many scenarios. Numerous text classification methods, which can be used for content-based recommendations, are discussed in [21, 22, 400]. A discussion of the notion of serendipity in the context of information retrieval is provided in [599]. Some content-based systems explicitly filter out very similar items in order to improve serendipity [85]. The work in [418] discusses how one can go beyond accuracy metrics to measure the quality of a recommender system.

Methods for feature extraction, cleaning, and feature selection in text classification are discussed in [21, 364, 400]. The extraction of the main content block from a Web page containing multiple blocks is achieved with the help of the tree-matching algorithm can be found in [364, 662]. The use of visual representations for extracting content structure from Web pages is described in [126]. A detailed discussion of feature selection measures for classification may be found in [18]. A recent text classification survey [21] discusses feature selection algorithms for the specific case of text data.

Numerous real-world systems have been designed with the use of content-based systems. Some of the earliest are *Fab* [60] and *Syskill & Webert* [477]. An early system, referred to as *Personal WebWatcher* [438, 439], makes recommendations by learning the interests of users from the Web pages that they visit. In addition, the Web pages that are linked to by the visited page are used in the recommendation process. The *Letizia* system [356] uses a Web-browser extension to track the user's browsing behavior, and uses it to make recommendations. A system known as *Dynamic-Profiler* uses a pre-defined taxonomy of categories to make news recommendations to users in real time [636]. In this case, user Web logs are used to learn the preferences and make personalized recommendations. The *IfWeb* system [55] represents the user interests in the form of a semantic network. The *WebMate* system [150] learns user profiles in the form of keyword vectors. This system is designed for keeping track of positive user interests rather than negative ones. The general principles in Web recommendations are not very different from those of news filtering. Methods for performing news recommendations are discussed in [41, 84, 85, 392, 543, 561]. Some of these methods use enhanced representations, such as *WordNet*, to improve the modeling process. Web recommender systems are generally more challenging than news recommender systems because the underlying text is often of lower quality. The *Citeseer* system [91] is able to discover interesting publications in a bibliographic database by identifying the common citations among the papers. Thus, it explicitly uses citations as a content mechanism for determination of similarity.

Content-based systems have also been used in other domains such as books, music, and movies. Content-based methods for book recommendations are discussed in [448]. The main challenge in music recommendations is the semantic gap between easily available features

and the likelihood of a user appreciating the music. This is a common characteristic between the music and the image domains. Some progress in bridging the semantic gap has been made in [138, 139]. *Pandora* [693] uses the features extracted in the Music Genome Project to make recommendations. The *ITR* system discusses how one might use text descriptions [178] of items (e.g., book descriptions or movie plots) to make recommendations. Further work [179] shows how one might integrate tags in a content-based recommender. The approach uses linguistic tools such as *WordNet* to extract knowledge for the recommendation process. A movie recommendation system that uses text categorization is the *INTIMATE* system [391]. A method that combines content-based and collaborative recommender systems is discussed in [520]. A broader overview of hybrid recommender systems is provided in [117]. A potential direction of work, mentioned in [376], is to enhance content-based recommender systems with encyclopedic knowledge [174, 210, 211], such as that gained from Wikipedia. A few methods have been designed that use Wikipedia for movie recommendation [341]. Interestingly, this approach does not improve the accuracy of the recommender system. The application of advanced semantic knowledge in content-based recommendations has been mentioned as a direction of future work in [376].

4.9 Exercises

1. Consider a scenario in which a user provides like/dislike ratings of a set of 20 items, in which she rates 9 items as a "like" and the remaining as a "dislike." Suppose that 7 item descriptions contain the word "thriller," and the user dislikes 5 of these items. Compute the Gini index with respect to the original data distribution, and with respect to the subset of items containing the word "thriller." Should feature selection algorithms retain this word in the item descriptions?

2. Implement a rule-based classifier with the use of association pattern mining.

3. Consider a movie recommender system in which movies belong to one or more of the genres illustrated in the table, and a particular user provides the following set of ratings to each of the movies.

Genre ⇒ / Movie-Id ⇓	Comedy	Drama	Romance	Thriller	Action	Horror	Like or Dislike
1	1	0	1	0	0	0	Dislike
2	1	1	1	0	1	0	Dislike
3	1	1	0	0	0	0	Dislike
4	0	0	0	1	1	0	Like
5	0	1	0	1	1	1	Like
6	0	0	0	0	1	1	Like
Test-1	0	0	0	1	0	1	?
Test-2	0	1	1	0	0	0	?

Mine all the rules with at least 33% support and 75% confidence. Based on these rules, would you recommend the item *Test-1* or *Test-2* to the user?

4. Implement a Bayes classifier with Laplacian smoothing.

5. Repeat Exercise 3 with the use of a Bayes classifier. Do not use Laplacian smoothing. Explain why Laplacian smoothing is important in this case.

6. Repeat Exercise 3 with the use of a 1-nearest neighbor classifier.

7. For a training data matrix D, regularized least-squares regression requires the inversion of the matrix $(D^T D + \lambda I)$, where $\lambda > 0$. Show that this matrix is always invertible.

8. The χ^2 distribution is defined by the following formula, as discussed in the chapter:

$$\chi^2 = \sum_{i=1}^{p} \frac{(O_i - E_i)^2}{E_i}$$

Show that for a 2×2 contingency table, the aforementioned formula can be rewritten as follows:

$$\chi^2 = \frac{(O_1 + O_2 + O_3 + O_4) \cdot (O_1 O_4 - O_2 O_3)^2}{(O_1 + O_2) \cdot (O_3 + O_4) \cdot (O_1 + O_3) \cdot (O_2 + O_4)}$$

Here, $O_1 \ldots O_4$ are defined in the same way as in the tabular example in the text.