Chapter 7
Data Mining Methods for Recommender Systems

Xavier Amatriain and Josep M. Pujol

7.1 Introduction

Recommender Systems (RS) typically apply techniques and methodologies from other neighboring areas such as Human Computer Interaction (HCI) or Information Retrieval (IR). Most of these systems also bear in their core an algorithm that can be understood as a particular instance of a Data Mining (DM) process [5].

The data mining process typically consists of 3 steps, carried out in succession: *Data Preprocessing* [78], *Model Learning*, and *Result Interpretation* (see Fig. 7.1). We will analyze some of the most important methods for data preprocessing in Sect. 7.2. In particular, we will focus on sampling, dimensionality reduction, and the use of distance functions because of their significance and their role in RS. In Sects. 7.3 and 7.4, we provide an overview introduction to the machine learning methods that are most commonly used in RS: classification, clustering and association rule discovery (see Fig. 7.1 for a detailed view of the different topics covered in the chapter).

This chapter does not intend to give a thorough review of Data Mining methods, but rather to highlight the impact that DM algorithms have in the RS field, and to provide an overview of the key DM techniques that have been successfully used. We direct the interested reader to Data Mining and Machine Learning textbooks (see [14, 19, 39, 70, 93], for example) or the more focused references that are provided throughout the chapter.

X. Amatriain (✉)
Netflix, 100 Winchester Cr., Los Gatos, CA 95032, USA

Quora, 150 Castro St., Mountain View, USA
e-mail: xavier@amatriain.net

J.M. Pujol
Cliqz, Rosenkavalierplatz 10, 81925 Munich, Germany
e-mail: josep@cliqz.com

F. Ricci et al. (eds.), *Recommender Systems Handbook*,
DOI 10.1007/978-1-4899-7637-6_7

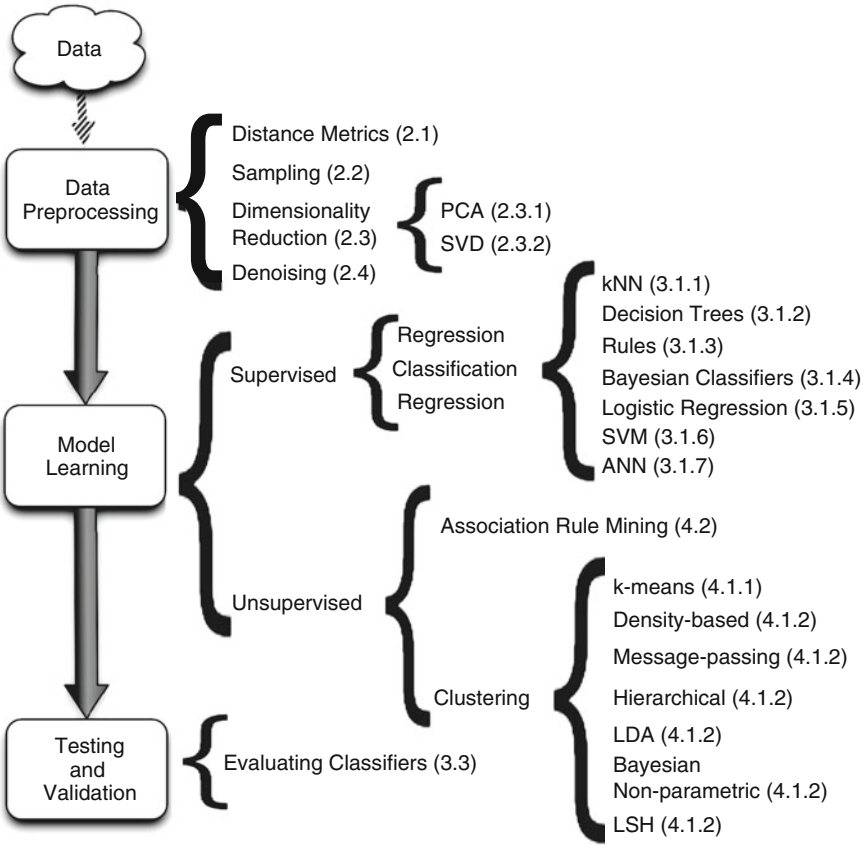

Fig. 7.1 Main steps and methods in a Data Mining process, with their correspondence to chapter sections

7.2 Data Preprocessing

We define *data* as a collection of *objects* and their *attributes*, where an attribute is defined as a property or characteristic of an object. Other names for object include *record*, *item*, *point*, *sample*, *observation*, or *instance*. An attribute might be also be referred to as a *variable*, *field*, *characteristic*, or *feature*. In the context of a typical collaborative filtering setting the objects in our dataset might be each of the ratings we have captured from the users. For each of them we will have typical attributes such as the user and item the rating refers to or the value of the rating itself. We can also add many other features such as the time or the location the rating occurred, or any other characteristic of the item or user such as item popularity, user age, or even the location of the item in the page at the time we received the rating [75].

Real-life data typically needs to be *preprocessed* (e.g. cleansed, filtered, transformed) in order to be used by the machine learning techniques in the model learning step. In this section, we focus on three issues that are of particular importance when designing a RS. First, we review different similarity or distance measures. Next, we discuss the issue of sampling as a way to reduce the number of items in very large collections while preserving its main characteristics. Finally, we describe the most common techniques to reduce dimensionality.

7.2.1 Similarity Measures

One of the preferred approaches to collaborative filtering (CF) recommenders is to use the *k*NN classifier that will be described in Sect. 7.3.1.1. This classification method—as most classifiers and clustering techniques—is highly dependent on defining an appropriate similarity or distance measure.[1]

The simplest and most common example of a distance measure is the Euclidean distance or the *L2 Norm*:

$$d(x, y) = \sqrt{\sum_{k=1}^{n}(x_k - y_k)^2} \tag{7.1}$$

where n is the number of dimensions (attributes) and x_k and y_k are the kth attributes (components) of data objects x and y, respectively.

The Minkowski Distance is a generalization of Euclidean Distance:

$$d(x, y) = \left(\sum_{k=1}^{n} |x_k - y_k|^r\right)^{\frac{1}{r}} \tag{7.2}$$

where r is the degree of the distance. Depending on the value of r, the generic Minkowski distance is known with specific names: For $r = 1$, the *city block*, (*Manhattan*, *taxicab* or *L1 norm*) distance; For $r = 2$, the *Euclidean* distance; For $r \to \infty$, the *supremum* (L_{max} *norm* or L_∞ *norm*) distance, which corresponds to computing the maximum difference between any dimension of the data objects.

The Mahalanobis distance is defined as:

$$d(x, y) = \sqrt{(x - y)\sigma^{-1}(x - y)^T} \tag{7.3}$$

where σ is the covariance matrix of the data.

[1]Note that a similarity measure is not a preprocessing step in itself but rather a prerequisite for being able to execute other data mining processes.

Another very common approach is to consider items as document vectors of an n-dimensional space and compute their similarity as the cosine of the angle that they form:

$$cos(x, y) = \frac{(x \bullet y)}{||x||||y||} \tag{7.4}$$

where $\bullet$ indicates vector dot product and $||x||$ is the norm of vector x. This similarity is known as the *cosine similarity*.

The similarity between items can also be given by their *correlation* which measures the linear relationship between objects. While there are several correlation coefficients that may be applied, the *Pearson correlation* is the most commonly used. Given the covariance of data points x and y Σ, and their standard deviation σ, we compute the Pearson correlation using:

$$Pearson(x, y) = \frac{\Sigma(x, y)}{\sigma_x \times \sigma_y} \tag{7.5}$$

Several similarity measures have been proposed in the case of items that only have binary attributes. First, the *M01*, *M10*, *M11*, and *M00* quantities are computed, where *M01* = the number of attributes where x was 0 and y was 1, *M10* = the number of attributes where x was 1 and y was 0, and so on. From those quantities we can compute: The *Simple Matching* coefficient $SMC = \frac{numberofmatches}{numberofattributes}$ $= \frac{M11+M00}{M01+M10+M00+M11}$; the *Jaccard* coefficient $JC = \frac{M11}{M01+M10+M11}$. The *Extended Jaccard* (*Tanimoto*) coefficient is a variation of JC for continuous or count attributes that is computed by $d = \frac{x \bullet y}{\|x\|^2+\|x\|^2-x \bullet y}$.

RS have traditionally used either the cosine similarity (Eq. (7.4)) or the Pearson correlation (Eq. (7.5))—or one of their many variations through, for instance, weighting schemes—Chap. 2 details the use of different distance functions for CF. Most of the other distance measures previously reviewed are possible. Spertus et al. [88] did a large-scale study to evaluate six different similarity measures in the context of the Orkut social network. Although their results might be biased by the particular setting of their experiment, it is interesting to note that the best response to recommendations were to those generated using the cosine similarity. Lathia et al. [64] also carried out a study of several similarity measures where they concluded that, in the general case, the prediction accuracy of a RS was *not* affected by the choice of the similarity measure. As a matter of fact and in the context of their work, using a random similarity measure sometimes yielded better results than using any of the well-known approaches.

7.2.2 Sampling

Sampling is the main technique used in DM for selecting a subset of relevant data from a large data set. It is used both in the preprocessing and final data interpretation steps. Sampling may be used because processing the entire data set is computationally too expensive. It can also be used to create *training* and *testing* datasets. In this case, the training dataset is used to learn the parameters or configure the algorithms used in the analysis step, while the testing dataset is used to evaluate the model or configuration obtained in the training phase, making sure that it performs well with previously unseen data. As a matter of fact, in most cases we not only need training and testing, but we also need to think about creating a third validation dataset. The training set is used for model fitting, the validation one for learning hyperparameters, and the testing to see how the model generalizes.

The key issue to sampling is finding a subset of the original data set that is *representative*—i.e. it has approximately the same property of interest—of the entire set. The simplest sampling technique is *random sampling*, where there is an equal probability of selecting any item. However, more sophisticated approaches are possible. For instance, in *stratified sampling* the data is split into several partitions based on a particular feature, followed by random sampling on each partition independently.

The most common approach to sampling consists of using sampling *without replacement*: When an item is selected, it is removed from the population. However, it is also possible to perform sampling *with replacement*, where items are not removed from the population once they have been selected, allowing for the same sample to be selected more than once.

It is common practice to use standard random sampling without replacement with an $80/20$ proportion when separating the training and testing data sets. This means that we use random sampling without replacement to select $20\,\%$ of the instances for the testing set and leave the remaining $80\,\%$ for training. The $80/20$ proportion should be taken as a rule of thumb as, in general, any value over $2/3$ for the training set is appropriate.

Sampling can lead to an over-specialization to the particular division of the training and testing data sets. For this reason, the training process may be repeated several times. The training and test sets are created from the original data set, the model is trained using the training data and tested with the examples in the test set. Next, different training/test data sets are selected to start the training/testing process again that is repeated K times. Finally, the *average* performance of the K learned models is reported. This process is known as cross-validation. There are several cross-validation techniques. In *repeated random sampling*, a standard random sampling process is carried out K times. In *n-Fold cross validation*, the data set is divided into n folds. One of the folds is used for testing the model and the remaining $n - 1$ folds are used for training. The cross validation process is then repeated n times with each of the n subsamples used exactly once as validation data. Finally, the *leave-one-out (LOO)* approach can be seen as an extreme case of n-Fold

cross validation where n is set to the number of items in the data set. Therefore, the algorithms are run as many times as data points using only one of them as a test each time. It should be noted, though, that as Isaksson et al. discuss in [57], cross-validation may be unreliable unless the data set is sufficiently large.

A common approach in RS is to sample the available feedback from the users—e.g. in the form of ratings—to separate it into training and testing. Cross-validation is also common. Although a standard random sampling is acceptable in the general case, in others we might need to bias our sampling for the test set in different ways. We might, for instance, decide to sample only from most recent ratings—since those are the ones we would be predicting in a real-world situation. We might also be interested in ensuring that the proportion of ratings per user is preserved in the test set and therefore impose that the random sampling is done on a per user basis. However, all these issues relate are beyond the scope of this chapter.

7.2.3 Reducing Dimensionality

It is common in RS to have not only a data set with features that define a high-dimensional space, but also very sparse information in that space—i.e. there are values for a limited number of features per object. The notions of density and distance between points, which are critical for clustering and outlier detection, become less meaningful in highly dimensional spaces. This is known as the Curse of Dimensionality. Dimensionality reduction techniques help overcome this problem by transforming the original high-dimensional space into a lower-dimensionality.

Sparsity and the *curse of dimensionality* are recurring problems in RS. Even in the simplest setting, we are likely to have a sparse matrix with thousands of rows and columns (i.e. users and items), most of which are zeros. Therefore, dimensionality reduction comes in naturally. Applying dimensionality reduction makes such a difference and its results are so directly applicable to the computation of the predicted value, that these methods are in fact considered an approach to building a RS, rather than a preprocessing technique. In this case we speak of these techniques as *Matrix Completion Methods*.

In the following paragraphs, we summarize the two most relevant dimensionality reduction algorithms in the context of RS: *Principal Component Analysis (PCA)* and *Singular Value Decomposition (SVD)*.

7.2.3.1 Principal Component Analysis

Principal Component Analysis [59] is a classical statistical method to find patterns in high dimensionality data sets. PCA allows to obtain an ordered list of components that account for the largest amount of the variance from the data in terms of least square errors: The amount of variance captured by the first component is larger

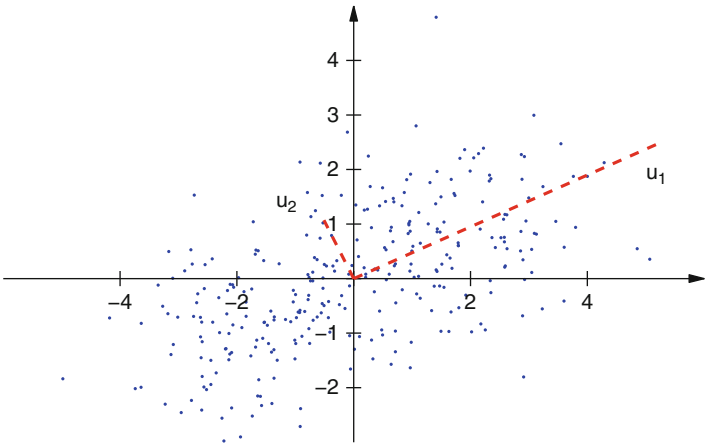

Fig. 7.2 PCA analysis of a two-dimensional point cloud from a combination of Gaussians. The principal components derived using PCS are u_1 and u_2, whose length is relative to the energy contained in the components

than the amount of variance on the second component and so on. We can reduce the dimensionality of the data by neglecting those components with a small contribution to the variance.

Figure 7.2 shows the PCA analysis to a two-dimensional point cloud generated by a combination of Gaussians. After the data is centered, the principal components are obtained and denoted by u_1 and u_2. Note that the length of the new coordinates is relative to the energy contained in their eigenvectors. Therefore, for the particular example depicted in Fig. 7.2, the first component u_1 accounts for 83.5 % of the energy, which means that removing the second component u_2 would imply losing only 16.5 % of the information. The rule of thumb is to choose the target dimensionality m' so that the cumulative energy is above a certain threshold, typically 90 %. PCA allows us to retrieve the original data matrix by projecting the data onto the new coordinate system $X'_{n\times m'} = X_{n\times m}W'_{m\times m'}$. The new data matrix X' contains most of the information of the original X with a dimensionality reduction of $m - m'$.

PCA is a powerful technique, but it does have important limitations. PCA relies on the empirical data set to be a linear combination of a certain basis—although generalizations of PCA for non-linear data have been proposed. Another important assumption of PCA is that the original data set has been drawn from a Gaussian distribution. When this assumption does not hold true, there is no warranty that the principal components are meaningful.

Although current trends seem to indicate that other matrix factorizations techniques such as SVD or Non-Negative Matrix Factorization are preferred for RS, earlier works used PCA. Goldberg et al. proposed an approach to use PCA in the context of an online joke recommendation system [50]. Their system, known as

Eigentaste,[2] starts from a standard matrix of user ratings to items. They then select their *gauge* set by choosing the subset of items for which all users had a rating. This new matrix is then used to compute the global correlation matrix where a standard two-dimensional PCA is applied.

7.2.3.2 Matrix Factorization and Singular Value Decomposition

Singular Value Decomposition [51] is a powerful technique for dimensionality reduction. It is a particular realization of the Matrix Factorization approach. The key issue in an SVD decomposition is to find a lower dimensional feature space where the new features represent "concepts" and the strength of each concept in the context of the collection is computable. Because SVD allows to automatically derive semantic "concepts" in a low dimensional space, it can be used as the basis of *latent-semantic analysis* [34], a very popular technique for text classification in Information Retrieval (IR).

The core of the SVD algorithm lies in the following theorem: It is always possible to decompose a given matrix A into $A = U\lambda V^T$. Given the $n \times m$ matrix data A (n items, m features), we can obtain an $n \times r$ matrix U (n items, r concepts), an $r \times r$ diagonal matrix λ (strength of each concept), and an $m \times r$ matrix V (m features, r concepts). Figure 7.3 illustrates this idea. The λ diagonal matrix contains the *singular values*, which will always be positive and sorted in decreasing order. The U matrix is interpreted as the "item-to-concept" similarity matrix, while the V matrix is the "term-to-concept" similarity matrix.

In order to compute the SVD of a rectangular matrix A, we consider AA^T and A^TA. The columns of U are the eigenvectors of AA^T, and the columns of V are the eigenvectors of A^TA. The singular values on the diagonal of λ are the positive square roots of the nonzero eigenvalues of both AA^T and A^TA. Therefore, in order

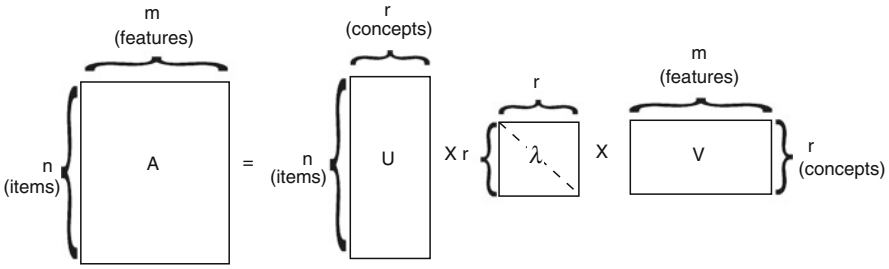

Fig. 7.3 Illustrating the basic Singular Value Decomposition Theorem: an item × features matrix can be decomposed into three different ones: an item × concepts, a concept strength, and a concept × features

[2] http://eigentaste.berkeley.edu.

to compute the SVD of matrix A we first compute T as AA^T and D as A^TA and then compute the eigenvectors and eigenvalues for T and D.

The r eigenvalues in λ are ordered in decreasing magnitude. Therefore, the original matrix A can be approximated by simply truncating the eigenvalues at a given k. The truncated SVD creates a rank-k approximation to A so that $A_k = U_k\lambda_k V_k^T$. A_k is the *closest* rank-k matrix to A. The term "closest" means that A_k minimizes the sum of the squares of the differences of the elements of A and A_k. The truncated SVD is a representation of the underlying latent structure in a reduced k-dimensional space, which generally means that the noise in the features is reduced.

The use of SVD as tool to improve collaborative filtering has been known for some time. Sarwar et al. [85] describe two different ways to use SVD in this context. First, SVD can be used to uncover latent relations between customers and products. In order to accomplish this goal, they first fill the zeros in the user-item matrix with the item average rating and then normalize by subtracting the user average. This matrix is then factored using SVD and the resulting decomposition can be used—after some trivial operations—directly to compute the predictions. The other approach is to use the low-dimensional space resulting from the SVD to improve neighborhood formation for later use in a kNN approach.

As described by Sarwar et al. [84], one of the big advantages of SVD is that there are incremental algorithms to compute an approximated decomposition. This allows to accept new users or ratings without having to recompute the model that had been built from previously existing data. The same idea was later extended and formalized by Brand [23] into an online SVD model. The use of incremental SVD methods has recently become a commonly accepted approach after its success in the Netflix Prize.[3] The publication of Simon Funk's simplified incremental SVD method [47] marked an inflection point in the contest. Since its publication, several improvements to SVD have been proposed in this same context (see Paterek's ensembles of SVD methods [74] or Kurucz et al. evaluation of SVD parameters [63]).

In that sense, Matrix Factorization approaches should be considered as more than a simple preprocessing or dimensionality reduction technique since the whole recommendation problem can be formalized as one of Matrix Completion. We can design a sparse matrix that represents users in rows and items in columns. Each known preference of a user for an item will represent a value in the matrix. All other positions will be unknown. It is in that setting, where coming up with a prediction of how much a user will like an item can be simplified to the task of completing missing values in the matrix (see Chap. 2 for more details on this usage).

It should be noted that different variants of Matrix Factorization (MF) methods such as the Non-negative Matrix Factorization (NNMF) have also been used [94]. These algorithms are, in essence, similar to SVD. The basic idea is to decompose the ratings matrix into two matrices, one of which contains features that describe the users and the other contains features describing the items. Matrix Factorization

[3] http://www.netflixprize.com.

methods can handle the missing values by introducing a bias term to the model. This can also be handled in the SVD preprocessing step by replacing zeros with the item average. MF is prone to overfitting. However, there exist MF variants, such as the Regularized Kernel Matrix Factorization, that can avoid the issue efficiently.

7.2.4 Denoising

Data collected for data-mining purposes might be subject to different kinds of noise such as missing values or outliers. Denoising is a very important preprocessing step that aims at removing any unwanted effect in the data while maximizing its information.

In a general sense we define noise as any unwanted artifact introduced in the data collection phase that might affect the result of our data analysis and interpretation. In the context of RS, we distinguish between *natural* and *malicious* noise [72]. The former refers to noise that is involuntarily introduced by users when giving feedback on their preferences. The latter refers to noise that is deliberately introduced in a system in order to bias the results.

It is clear that malicious noise can affect the output of a RS. But, also, we performed a study that concluded that the effects of natural noise on the performance of RS is far from being negligible [7]. In order to address this issue, we designed a denoising approach that is able to improve accuracy by asking some users to re-rate some items [8]. We concluded that accuracy improvements by investing in this pre-processing step could be larger than the ones obtained by complex algorithm optimizations.

7.3 Supervised Learning

7.3.1 Classification

A classifier is a mapping between a feature space and a label space, where the features represent characteristics of the elements to classify and the labels represent the classes. A restaurant RS, for example, can be implemented by a classifier that classifies restaurants into one of two categories (good, bad) based on a number of features that describe it.

There are many types of classifiers, but in general we will talk about either *supervised* or *unsupervised* classification. In supervised classification, a set of labels or categories is known in advance and we have a set of labeled examples which constitute a training set. In unsupervised classification, the labels or categories are unknown in advance and the task is to suitably (according to some criteria) organize the elements at hand. In this section we describe several algorithms to learn supervised classifiers and will be covering unsupervised classification (i.e. clustering) in Sect. 7.4.

7.3.1.1 Nearest Neighbors

Instance-based classifiers work by storing training records and using them to predict the class label of unseen cases. A trivial example is the so-called *rote-learner*. This classifier memorizes the entire training set and classifies only if the attributes of the new record match one of the training examples exactly. A more elaborate, and far more popular, instance-based classifier is the *Nearest neighbor classifier* (kNN) [32]. Given a point to be classified, the kNN classifier finds the k closest points (*nearest neighbors*) from the training records. It then assigns the class label according to the class labels of its *nearest-neighbors*. The underlying idea is that if a record falls in a particular neighborhood where a class label is predominant it is because the record is likely to belong to that very same class.

Given a query point q for which we want to know its class l, and a training set $X = \{\{x_1, l_1\} \ldots \{x_n\}\}$, where x_j is the j-th element and l_j is its class label, the k-nearest neighbors will find a subset $Y = \{\{y_1, l_1\} \ldots \{y_k\}\}$ such that $Y \in X$ and $\sum_1^k d(q, y_k)$ is minimal. Y contains the k points in X which are closest to the query point q. Then, the class label of q is $l = f(\{l_1 \ldots l_k\})$.

Perhaps the most challenging issue in kNN is how to choose the value of k. If k is too small, the classifier will be sensitive to noise points. But if k is too large, the neighborhood might include too many points from other classes. The right plot in Fig. 7.4 shows how different k yields different class label for the query point, if $k = 1$ the class label would be *circle* whereas $k = 7$ classifies it as *square*. Note that the query point from the example is on the boundary of two clusters, and therefore, it is difficult to classify.

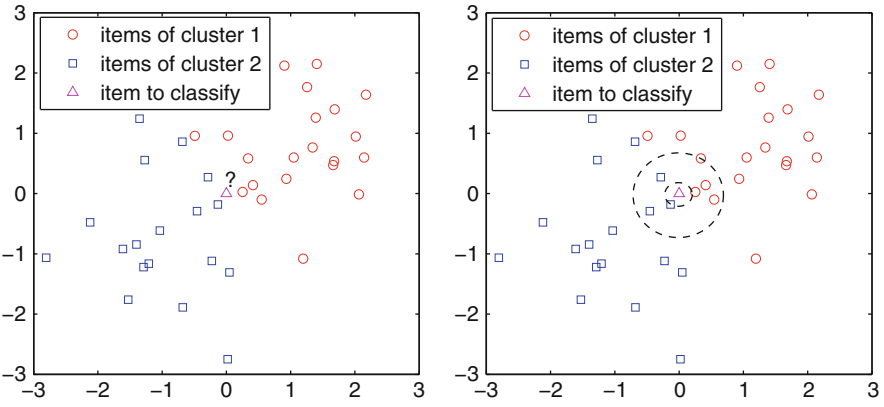

Fig. 7.4 Example of k-nearest neighbors. The *left subfigure* shows the training points with two class labels (*circles and squares*) and the query point (as a *triangle*). The *right sub-figure* illustrates closest neighborhood for $k = 1$ and $k = 7$. The query point would be classified as *square* for $k = 1$, and as a *circle* for $k = 5$ according to the simple majority vote rule. Note that the query points was just on the boundary between the two clusters

*k*NN classifiers are amongst the simplest of all machine learning algorithms. Since *k*NN does not build models explicitly it is considered a *lazy learner*. Unlike eager learners such as decision trees or rule-based systems (see Sects. 7.3.1.2 and 7.3.1.3, respectively), *k*NN classifiers leave many decisions to the classification step. Therefore, classifying unknown records is relatively expensive.

Nearest Neighbor is one of the most common approaches to CF—and therefore to designing a RS. As a matter of fact, any overview on RS—such as the one by Adomavicius and Tuzhilin [1]—will include an introduction to the use of nearest neighbors in this context. One of the advantages of this classifier is that it is conceptually very much related to the idea of CF: Finding like-minded users (or similar items) is essentially equivalent to finding neighbors for a given user or an item. The other advantage is that, being the *k*NN classifier a lazy learner, it does not require to learn and maintain a given model. Therefore, in principle, the system can adapt to rapid changes in the user ratings matrix. Unfortunately, this comes at the cost of recomputing the neighborhoods and therefore the similarity matrix. This is why we proposed a neighborhood model that uses a reduced set of experts as the source for selecting neighbors [6].

The *k*NN approach, although simple and intuitive, has shown good accuracy results and is very amenable to improvements. As a matter of fact, its supremacy as the de facto standard for CF recommendation has only been challenged recently by approaches based on Matrix Completion. That said, the traditional *k*NN approach to CF has experienced improvements in several directions. For instance, in the context of the Netflix Prize, Bell and Koren propose a method to remove *global effects* such as the fact that some items may attract users that consistently rate lower. They also propose an optimization method for computing interpolating weights once the neighborhood is created.

See Chap. 2 for more details on enhanced CF techniques based on the use of neighborhoods.

7.3.1.2 Decision Trees

Decision trees [80] are classifiers on a target attribute (or class) in the form of a tree structure. The observations (or items) to classify are composed of attributes and their target value. The nodes of the tree can be: (a) *decision nodes*, in these nodes a single attribute-value is tested to determine to which branch of the subtree applies. Or (b) *leaf nodes* which indicate the value of the target attribute.

There are many algorithms for decision tree induction: Hunt's Algorithm, CART, ID3, C4.5, SLIQ, SPRINT to mention the most common. The recursive Hunt algorithm, which is one of the earliest and easiest to understand, relies on the *test condition* applied to a given attribute that discriminates the observations by their target values. Once the partition induced by the test condition has been found, the algorithm is recursively repeated until a partition is empty or all the observations have the same target value.

Splits can be decided by maximizing the information gain, defined as follows,

$$\Delta_i = I(parent) - \sum_{j=1}^{k_i} \frac{N(v_j)I(v_j)}{N} \tag{7.6}$$

where k_i are values of the attribute i, N is the number of observations, v_j is the j-th partition of the observations according to the values of attribute i. Finally, I is a function that measures node *impurity*. There are different measures of impurity: Gini Index, Entropy and misclassification error are the most common in the literature.

Decision tree induction stops once all observations belong to the same class (or the same range in the case of continuous attributes). This implies that the impurity of the leaf nodes is zero. For practical reasons, however, most decision trees implementations use pruning by which a node is no further split if its impurity measure or the number of observations in the node are below a certain threshold.

The main advantages of building a classifier using a decision tree is that it is inexpensive to construct and it is extremely fast at classifying unknown instances. Another appreciated aspect of decision tree is that they can be used to produce a set of rules that are easy to interpret (see Sect. 7.3.1.3) while maintaining an accuracy comparable to other basic classification techniques.

Decision trees may be used in a model-based approach for a RS. One possibility is to use content features to build a decision tree that models all the variables involved in the user preferences. Bouza et al. [21] use this idea to construct a Decision Tree using semantic information available for the items. The tree is built after the user has rated only two items. The features for each of the items are used to build a model that explains the user ratings. They use the information gain of every feature as the splitting criteria. It should be noted that although this approach is interesting from a theoretical perspective, the precision they report on their system is worse than that of recommending the average rating.

As it could be expected, it is very difficult and unpractical to build a decision tree that tries to explain all the variables involved in the decision making process. Decision trees, however, may also be used in order to model a particular part of the system. Cho et al. [28], for instance, present a RS for online purchases that combines the use of Association Rules (see Sect. 7.4.2) and Decision Trees. The Decision Tree is used as a filter to select which users should be targeted with recommendations. In order to build the model they create a candidate user set by selecting those users that have chosen products from a given category during a given time frame. In their case, the dependent variable for building the decision tree is chosen as whether the customer is likely to buy new products in that same category. Nikovski and Kulev [71] follow a similar approach combining Decision Trees and Association Rules. In their approach, frequent itemsets are detected in the purchase dataset and then they apply standard tree-learning algorithms for simplifying the recommendations rules.

Another option to use Decision Trees in a RS is to use them as a tool for exploring the space of possible items to present to a user during the coldstarting phase. The basic idea of the approach is to maximize the amount of information obtained with

each item presented by considering it a node in a decision tree. Golbandi et al. [49], for instance, detail an efficient tree learning algorithm, specifically tailored to this application.

The use of Decision Trees for ranking has been studied in several settings and their use in a RS for this purpose is fairly straightforward [11, 27]. While it is possible to use individual trees for ranking, it is much more efficient to use ensembles of decision trees for this purpose. The two kinds of tree ensembles that are commonly used both for classification and ranking are Random Forests [25], and Gradient Boosted Decision Trees [45]. Both these techniques are used in collaborative filtering or personalized ranking applications (see [4, 12]).

Finally, trees or trees ensembles can be used as a way to combine different algorithms in an ensemble. The solution to the Netflix Prize, for example, used Gradient Boosted Decision Trees to combine the more than 100 methods that had been trained [61].

7.3.1.3 Ruled-Based Classifiers

Rule-based classifiers classify data by using a collection of "**if** ... **then** ..." rules. The rule *antecedent* or condition is an expression made of attribute conjunctions. The rule *consequent* is a positive or negative classification.

We say that a rule *r covers* a given instance *x* if the attributes of the instance satisfy the rule condition. We define the *coverage* of a rule as the fraction of records that satisfy its antecedent. On the other hand, we define its *accuracy* as the fraction of records that satisfy both the antecedent and the consequent. We say that a classifier contains *mutually exclusive rules* if the rules are independent of each other—i.e. every record is covered by at most one rule. Finally we say that the classifier has *exhaustive rules* if they account for every possible combination of attribute values—i.e. each record is covered by at least one rule.

In order to build a rule-based classifier we can follow a direct method to extract rules directly from data. Examples of such methods are RIPPER, or CN2. On the other hand, it is common to follow an indirect method and extract rules from other classification models such as decision trees or neural networks.

The advantages of rule-based classifiers are that they are extremely expressive since they are symbolic and operate with the attributes of the data without any transformation. Rule-based classifiers, and by extension decision trees, are easy to interpret, easy to generate and they can classify new instances efficiently.

In a similar way to Decision Tress, however, it is very difficult to build a complete recommender model based on rules. As a matter of fact, this method is not very popular in the context of RS because deriving a rule-based system means that we either have some explicit prior knowledge of the decision making process or that we derive the rules from another model such a decision tree. However a rule-based system can be used to improve the performance of a RS by injecting partial domain knowledge or business rules. Anderson et al. [9], for instance, implemented a CF music RS that improves its performance by applying a rule-based system to the

results of the CF process. If a user rates an album by a given artist high, for instance, predicted ratings for all other albums by this artist will be increased.

Gutta et al. [40] implemented a rule-based RS for TV content. In order to do, so they first derived a C4.5 Decision Tree that is then decomposed into rules for classifying the programs. Basu et al. [15] followed an inductive approach using the *Ripper* [30] system to learn rules from data. They report slightly better results when using hybrid content and collaborative data to learn rules than when following a pure CF approach.

7.3.1.4 Bayesian Classifiers

A Bayesian classifier [46] is a probabilistic framework for solving classification problems. It is based on the definition of conditional probability and the Bayes theorem. The Bayesian school of statistics uses probability to represent uncertainty about the relationships learned from the data. In addition, the concept of *priors* is very important as they represent our expectations or prior knowledge about what the true relationship might be. In particular, the probability of a model given the data (*posterior*) is proportional to the product of the *likelihood* times the *prior probability* (or prior). The likelihood component includes the effect of the data while the prior specifies the belief in the model before the data was observed.

Bayesian classifiers consider each attribute and class label as (continuous or discrete) random variables. Given a record with N attributes $(A_1, A_2, \ldots, A_N)$, the goal is to predict class C_k by finding the value of C_k that maximizes the posterior probability of the class given the data $P(C_k|A_1, A_2, \ldots, A_N)$. Applying Bayes' theorem, $P(C_k|A_1, A_2, \ldots, A_N) \propto P(A_1, A_2, \ldots, A_N|C_k)P(C_k)$.

A particular but very common Bayesian classifier is the *Naive Bayes Classifier*. In order to estimate the conditional probability, $P(A_1, A_2, \ldots, A_N|C_k)$, a Naive Bayes Classifier assumes the probabilistic *independence* of the attributes—i.e. the presence or absence of a particular attribute is unrelated to the presence or absence of any other. This assumption leads to $P(A_1, A_2, \ldots, A_N|C_k) = P(A_1|C_k)P(A_2|C_k) \ldots P(A_N|C_k)$.

The main benefits of Naive Bayes classifiers are that they are robust to isolated noise points and irrelevant attributes, and they handle missing values by ignoring the instance during probability estimate calculations. However, the independence assumption may not hold for some attributes as they might be correlated. In this case, the usual approach is to use the so-called *Bayesian Belief Networks (BBN)* (or Bayesian Networks, for short). BBN's use an acyclic graph to encode the dependence between attributes and a probability table that associates each node to its immediate parents. BBN's provide a way to capture prior knowledge in a domain using a graphical model. In a similar way to Naive Bayes classifiers, BBN's handle incomplete data well and they are quite robust to model overfitting.

Bayesian classifiers are particularly popular for model-based RS. They are often used to derive a model for content-based RS. Ghani and Fano [48], for instance, use a Naive Bayes classifier to implement a content-based RS. The use of this model

allows for recommending products from unrelated categories in the context of a department store.

Bayesian classifiers can also be used in a CF setting. Miyahara and Pazzani [68], for instance, implement a RS based on a Naive Bayes classifier. In order to do so, they define two classes: *like* and *don't like*. In this context they propose two ways of using the Naive Bayesian Classifier: The *Transformed Data Model* assumes that all features are completely independent, and feature selection is implemented as a preprocessing step. On the other hand, the *Sparse Data Model* assumes that only known features are informative for classification. Furthermore, it only makes use of data which both users rated in common when estimating probabilities. Experiments show both models to perform better than a correlation-based CF.

Pronk et al. [77] use a Bayesian Naive Classifier as the base for incorporating user control and improving performance, especially in cold-start situations. In order to do so they propose to maintain two profiles for each user: one learned from the rating history, and the other explicitly created by the user. The blending of both classifiers can be controlled in such a way that the user-defined profile is favored at early stages, when there is not too much rating history, and the learned classifier takes over at later stages.

In the previous section we mentioned that Gutta et al. [40] implemented a rule-based approach in a TV content RS. Another of the approaches they tested was a Bayesian classifier. They define a two-class classifier, where the classes are *watched/not watched*. The user profile is then a collection of attributes together with the number of times they occur in positive and negative examples. This is used to compute prior probabilities that a show belongs to a particular class and the conditional probability that a given feature will be present if a show is either positive or negative. It must be noted that features are, in this case, related to both content—i.e. genre—and contexts—i.e. time of the day. The posteriori probabilities for a new show are then computed from these.

Breese et al. [24] implement a Bayesian Network where each node corresponds to each item. The states correspond to each possible vote value. In the network, each item will have a set of parent items that are its best predictors. The conditional probability tables are represented by decision trees. The authors report better results for this model than for several nearest-neighbors implementations over several datasets.

Hierarchical Bayesian Networks have also been used in several settings as a way to add domain-knowledge for information filtering [98]. One of the issues with hierarchical Bayesian networks, however, is that it is very expensive to learn and update the model when there are many users in it. Zhang and Koren [99] propose a variation over the standard Expectation-Maximization (EM) model in order to speed up this process in the scenario of a content-based RS.

7.3.1.5 Logistic Regression

Logistic Regression (LR) is perhaps one the most basic probabilistic classification models. Although it is not widely spread in the Recommender Systems literature it is used in the industry, arguably because its simplicity and efficiency.

It is important to note that even though Logistic Regression has the term regression in its name it is not a regression model but a classifier [19]. The term regression is due to legacy since LR is a based on the basic Linear Regression.

In regression models the output is always a continuous value, e.g. the predicted audience of a movie based on a set of features such as production costs, marketing budget, cast, feedback of the preview, etc. On the other hand in a classifier the output is a class label. Following the same example, the output would be whether the movie will be a block-buster or not.

The Linear Regression model is defined by the following linear equation,

$$h_\theta(x) = \theta^\top x \tag{7.7}$$

once we have learned the parameters *theta* using the training set the hypothesis can take any continuous value. The Logistic Regression is similar, but there is an extra function $g(z)$ known as the logistic function,

$$h_\theta(x) = g(\theta^\top x) \tag{7.8}$$

$$g(z) = \frac{1}{1 + e^{-z}} \tag{7.9}$$

The logistic function yields $\frac{1}{2}$ when z is zero. For positive values of z it quickly goes to 1 and symmetrically it goes quickly to zero for negative values of z. Since it guarantees that $0 \leq h_\theta(x) \leq 1$ we can treat the output as a probability of belonging to a particular class. We can predict the class label 1 when $h_\theta(x) \geq \frac{1}{2}$ and class label 0 when $h_\theta(x) < \frac{1}{2}$.

Logistic Regression creates then a decision boundary defined by $\theta^\top x \geq 0$. This hyperplane (a line in the case of a single feature) separates data into two classes.

The concept of decision boundary is also present in other classification methods, notably in Support Vector Machines (see Sect. 7.3.1.6). Unlike in SVM's the decision boundary yield by Logistic Regression is not aware of margins between data, as a consequence, the decision boundary might less resilient to the presence of outliers. On the positive side Logistic Regression is easy to implement, and it is very efficient specially when there is a large number of features.

Zhang et al. [81] evaluated Logistic Regression, together with other probabilistic methodologies, for the case of online reviews with a very small set of users assessing the quality of these reviews. Logistic Regression has also been successfully tested in the context of tag recommendation by Montañés et al. [36]. A final use of both linear and (ordinal) logistic regression can be found in Parra et al.'s work [73]. In this case, regression is used as a way to convert implicit feedback into explicit ratings.

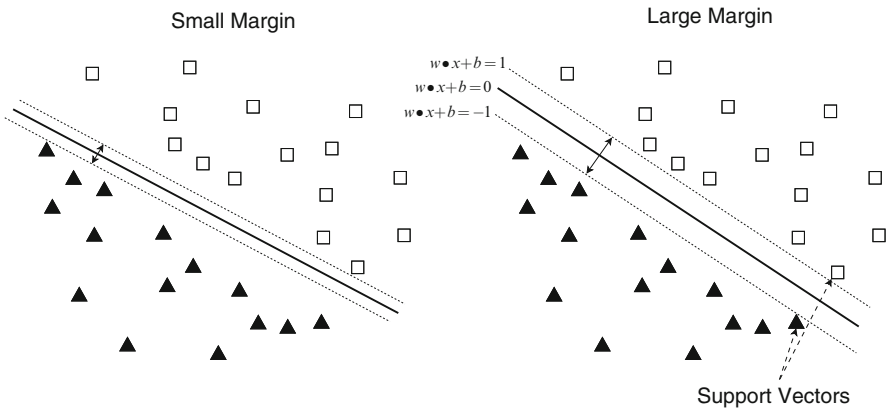

Fig. 7.5 Different boundary decisions are possible to separate two classes in two dimensions. Each boundary has an associated margin

7.3.1.6 Support Vector Machines

The goal of a Support Vector Machine (SVM) classifier [33] is to find a linear hyperplane (decision boundary) that separates the data in such a way that the margin is maximized. For instance, if we look at a two class separation problem in two dimensions like the one illustrated in Fig. 7.5, we can easily observe that there are many possible boundary lines to separate the two classes. Each boundary has an associated margin. The rationale behind SVM's is that if we choose the one that maximizes the margin we are less likely to misclassify unknown items in the future.

A linear separation between two classes is accomplished through the function $w \bullet x + b = 0$. We define a function that can classify items of being of class $+1$ or -1 as long as they are separated by some minimum distance from the class separation function. The function is given by Eq. (7.10)

$$f(x) = \begin{cases} 1, & \text{if } w \bullet x + b \geq 1 \\ -1, & \text{if } w \bullet x + b \leq -1 \end{cases} \tag{7.10}$$

$$Margin = \frac{2}{\|w\|^2} \tag{7.11}$$

Following the main rationale for SVM's, we would like to maximize the margin between the two classes, given by Eq. (7.11). This is in fact equivalent to minimizing the inverse value $L(w) = \frac{\|w\|^2}{2}$ but subjected to the constraints given by $f(x)$. This is a constrained optimization problem and there are numerical approaches to solve it (e.g., quadratic programming).

If the items are not linearly separable we can decide to turn the svm into a *soft margin* classifier by introducing a *slack variable*. In this case the formula to

minimize is given by Eq. (7.12) subject to the new definition of $f(x)$ in Eq. (7.13). On the other hand, if the decision boundary is not linear we need to transform data into a higher dimensional space . This is accomplished thanks to a mathematical transformation known as the *kernel trick*. The basic idea is to replace the dot products in Eq. (7.10) by a *kernel* function. There are many different possible choices for the kernel function such as Polynomial or Sigmoid. But the most common kernel functions are the family of Radial Basis Function (RBF).

$$L(w) = \frac{\|w\|^2}{2} + C\sum_{i=1}^{N}\epsilon \tag{7.12}$$

$$f(x) = \begin{cases} 1, & \text{if } w \bullet x + b \geq 1 - \epsilon \\ -1, & \text{if } w \bullet x + b \leq -1 + \epsilon \end{cases} \tag{7.13}$$

Support Vector Machines have recently gained popularity for their performance and efficiency in many settings. SVM's have also shown promising recent results in RS. Kang and Yoo [60], for instance, report on an experimental study that aims at selecting the best preprocessing technique for predicting missing values for an SVM-based RS. In particular, they use SVD and Support Vector Regression. The Support Vector Machine RS is built by first binarizing the 80 levels of available user preference data. They experiment with several settings and report best results for a threshold of 32—i.e. a value of 32 and less is classified as *prefer* and a higher value as *do not prefer*. The user id is used as the class label and the positive and negative values are expressed as preference values 1 and 2.

Xu and Araki [96] used SVM to build a TV program RS. They used information from the Electronic Program Guide (EPG) as features. But in order to reduce features they removed words with lowest frequencies. Furthermore, and in order to evaluate different approaches, they used both the Boolean and the *Term frequency—inverse document frequency* (TFIDF) weighting schemes for features. In the former, 0 and 1 are used to represent absence or presence of a term on the content. In the latter, this is turned into the TFIDF numerical value.

Xia et al. [95] present different approaches to using SVM's for RS in a CF setting. They explore the use of Smoothing Support Vector Machines (SSVM). They also introduce a SSVM-based heuristic (SSVMBH) to iteratively estimate missing elements in the user-item matrix. They compute predictions by creating a classifier for each user. Their experimental results report best results for the SSVMBH as compared to both SSVM's and traditional user-based and item-based CF. Finally, Oku et al. [38] propose the use of Context-Aware Vector Machines (C-SVM) for context-aware RS. They compare the use of standard SVM, C-SVM and an extension that uses CF as well as C-SVM. Their results show the effectiveness of the context-aware methods for restaurant recommendations.

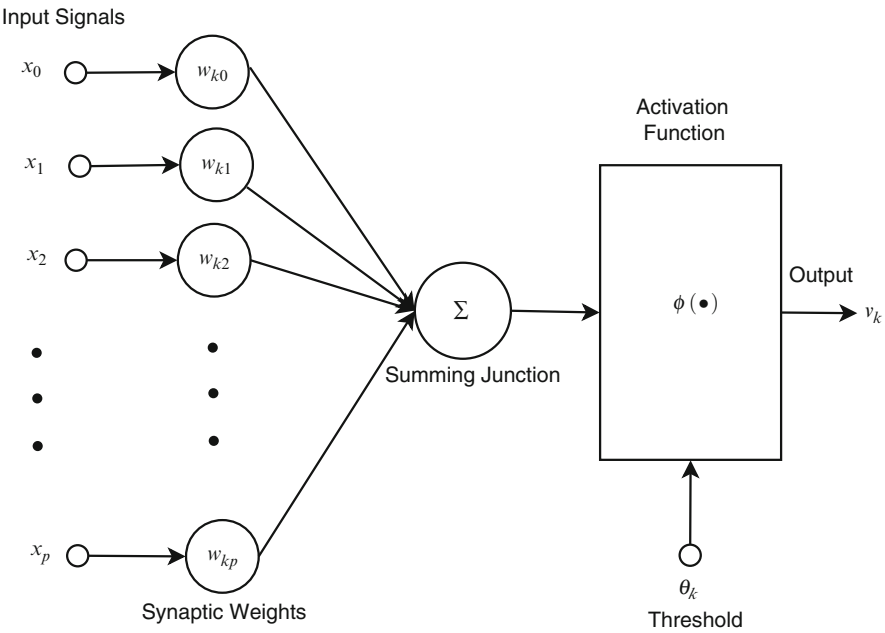

Fig. 7.6 Perceptron model

7.3.1.7 Artificial Neural Networks

An Artificial Neural Network (ANN) [101] is an assembly of inter-connected nodes and weighted links that is inspired in the architecture of the biological brain. Nodes in an ANN are called *neurons* as an analogy with biological neurons. These simple functional units are composed into networks that have the ability to learn a classification problem after they are trained with sufficient data.

The simplest case of an ANN is the *perceptron* model, illustrated in Fig. 7.6. If we particularize the *activation function* ϕ to be the simple Threshold Function, the output is obtained by summing up each of its input value according to the weights of its links and comparing its output against some threshold θ_k. The output function can be expressed using Eq. (7.14). The perceptron model is a linear classifier that has a simple and efficient learning algorithm. But, besides the simple Threshold Function used in the Perceptron model, there are several other common choices for the activation function such as sigmoid, tanh, or step functions.

$$y_k = \begin{cases} 1, & \text{if } \sum x_i w_{ki} \geq \theta_k \\ 0, & \text{if } \sum x_i w_{ki} < \theta_k \end{cases} \tag{7.14}$$

An ANN can have any number of layers. Layers in an ANN are classified into three types: input, hidden, and output. Units in the input layer respond to data that

is fed into the network. Hidden units receive the weighted output from the input units. And the output units respond to the weighted output from the hidden units and generate the final output of the network. Using neurons as atomic functional units, there are many possible architectures to put them together in a network. But, the most common approach is to use the *feed-forward ANN*. In this case, signals are strictly propagated in one way: from input to output.

The main advantages of ANN are that—depending on the activation function—they can perform non-linear classification tasks, and that, due to their parallel nature, they can be efficient and even operate if part of the network fails. The main disadvantage is that it is hard to come up with the ideal network topology for a given problem and once the topology is decided this will act as a lower bound for the classification error. ANN's belong to the class of *sub-symbolic* classifiers, which means that they provide no semantics for inferring knowledge—i.e. they promote a kind of *black-box* approach.

ANN's can be used in a similar way as Bayesian Networks to construct model-based RS's. However, there is no conclusive study to whether ANN introduce any performance gain. As a matter of fact, Pazzani and Billsus [76] did a comprehensive experimental study on the use of several machine learning algorithms for web site recommendation. Their main goal was to compare the simple naive Bayesian Classifier with computationally more expensive alternatives such as Decision Trees and Neural Networks. Their experimental results show that Decision Trees perform significantly worse. On the other hand ANN and the Bayesian classifier performed similarly. They conclude that there does not seem to be a need for nonlinear classifiers such as the ANN. Berka et al. [42] used ANN to build an URL RS for web navigation. They implemented a content-independent system based exclusively on *trails*—i.e. associating pairs of domain names with the number of people who traversed them. In order to do so they used feed-forward Multilayer Perceptrons trained with the Backpropagation algorithm.

ANN can be used to combine (or hybridize) the input from several recommendation modules or data sources. Hsu et al. [41], for instance, build a TV recommender by importing data from four different sources: user profiles and stereotypes; viewing communities; program metadata; and viewing context. They use the back-propagation algorithm to train a three-layered neural network. Christakou and Stafylopatis [29] also built a hybrid content-based CF RS. The content-based recommender is implemented using three neural networks per user, each of them corresponding to one of the following features: "kinds", "stars", and "synopsis". They trained the ANN using the Resilient Backpropagation method.

More recently, variations of NN have been used in different collaborative filtering settings. Salakhutdinov et al. used Restricted Boltzmann Machines in the context of the Netflix Prize [83] to predict ratings. This solution is actually part of the current Netflix production system (see Chap. 11).

7.3.2 Ensembles of Classifiers

The basic idea behind the use of *ensembles* of classifiers is to construct a set of classifiers from the training data and predict class labels by aggregating their predictions. Ensembles of classifiers work whenever we can assume that the classifiers are independent. In this case we can ensure that the ensemble will produce results that are in the worst case as bad as the worst classifier in the ensemble. Therefore, combining independent classifiers of a similar classification error will only improve results.

Several approaches are possible to generate ensembles. The two most common techniques are *Bagging* and *Boosting*. In Bagging, we perform sampling with replacement, building the classifier on each bootstrap sample. Each sample has probability $(1-\frac{1}{N})^N$ of being selected—note that if N is large enough, this converges to $1-\frac{1}{e} \approx 0.623$. In Boosting we use an iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records. Initially, all records are assigned equal weights. But, unlike bagging, weights may change at the end of each boosting round: Records that are wrongly classified will have their weights increased while records that are classified correctly will have their weights decreased. An example of boosting is the AdaBoost algorithm.

The use of ensembles of classifiers is common practice in the RS field. As a matter of fact, any *hybridation* technique [26] can be considered an ensemble as it combines in one way or another several classifiers. An explicit example of this is Tiemann and Pauws' music recommender, in which they use ensemble learning methods to combine a social and a content-base RS [90].

Experimental results show that ensembles can produce better results than any classifier in isolation. Bell et al. [17], for instance, used a combination of 107 different methods in their progress prize winning solution to the Netflix challenge. They state that their findings show that it pays off more to find substantially different approaches rather than focusing on refining a particular technique. In order to blend the results from the ensembles they use a linear regression approach and to derive weights for each classifier, they partition the test dataset into 15 different bins and derive unique coefficients for each of the bins. Different uses of ensembles in the context of the Netflix prize can be tracked in other approaches such as in Schclar et al.'s [86] or Toescher et al.'s [91].

The boosting approach has also been used in RS. Freund et al., for instance, present an algorithm called RankBoost to combine preferences [43]. They apply the algorithm to produce movie recommendations in a CF setting. The winning solution to the Netflix Prize [61] used Gradient Boosted Decision Trees, a tree-based ensemble technique that uses boosting, for the final combination of the individual predictors.

7.3.3 Evaluating Classifiers

The most commonly accepted evaluation measures for RS are the Mean Average Error (MAE) or Root Mean Squared Error (RMSE) between the predicted interest (or rating) and the measured one. These measures compute accuracy without any assumption on the purpose of the RS. However, as McNee et al. point out [67], there is much more than accuracy to deciding whether an item should be recommended. Herlocker et al. [55] provide a comprehensive review of algorithmic evaluation approaches to RS. They suggest that some measures could potentially be more appropriate for some tasks. However, they are not able to validate the measures when evaluating the different approaches empirically on a class of recommendation algorithms and a single set of data.

A step forward is to consider that the purpose of a "real" RS is to produce a top-N list of recommendations and evaluate RS depending on how well they can classify items as being *recommendable*. If we look at our recommendation as a classification problem, we can make use of well-known measures for classifier evaluation such as precision and recall. In the following paragraphs, we will review some of these measures and their application to RS evaluation. Note however that learning algorithms and classifiers can be evaluated by multiple criteria. This includes how accurately they perform the classification, their computational complexity during training , complexity during classification, their sensitivity to noisy data, their scalability, and so on. But in this section we will focus only on classification performance.

In order to evaluate a model we usually take into account the following measures: **True Positives** (*TP*): number of instances classified as belonging to class *A* that truly belong to class *A*; **True Negatives** (*TN*): number of instances classified as not belonging to class *A* and that in fact do not belong to class *A*; **False Positives** (*FP*): number of instances classified as class *A* but that do not belong to class *A*; **False Negatives** (*FN*): instances not classified as belonging to class v but that in fact do belong to class *A*.

The most commonly used measure for model performance is its *Accuracy* defined as the ratio between the instances that have been correctly classified (as belonging or not to the given class) and the total number of instances: $Accuracy = (TP + TN)/(TP + TN + FP + FN)$. However, accuracy might be misleading in many cases. Imagine a 2-class problem in which there are 99,900 samples of class A and 100 of class B. If a classifier simply predicts everything to be of class A, the computed accuracy would be of 99.9 % but the model performance is questionable because it will never detect any class B examples. One way to improve this evaluation is to define the cost matrix where we declare the "cost" of misclassifying class B examples as being of class A. In real world applications different types of errors may indeed have very different costs. For example, if the 100 samples above correspond to defective airplane parts in an assembly line, incorrectly rejecting a non-defective part (one of the 99,900 samples) has a negligible cost compared to the cost of mistakenly classifying a defective part as a good part.

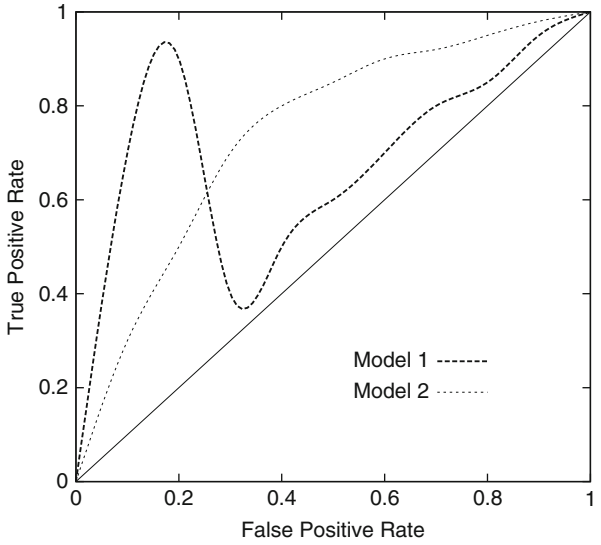

Fig. 7.7 Example of ROC curve. Model 1 performs better for low False Positive Rates while Model 2 is fairly consistent throughout and outperforms Model 1 for False Positive Rates higher than 0.25

Other common measures of model performance, particularly in Information Retrieval, are Precision and Recall. Precision, defined as $P = TP/(TP + FP)$, is a measure of how many errors we make in classifying samples as being of class A. On the other hand, recall, $R = TP/(TP + FN)$, measures how good we are in not leaving out samples that should have been classified as belonging to the class. Note that these two measures are misleading when used in isolation in most cases. We could build a classifier of perfect precision by not classifying any sample as being of class A (therefore obtaining 0 TP but also 0 FP). Conversely, we could build a classifier of perfect recall by classifying all samples as belonging to class A. As a matter of fact, there is a measure, called the F_1-measure that combines both Precision and Recall into a single measure as: $F_1 = \frac{2RP}{R+P} = \frac{2TP}{2TP+FN+FP}$

Sometimes we would like to compare several competing models rather than estimate their performance independently. In order to do so we use a technique developed in the 1950s for analysis of noisy signals: the Receiver Operating Characteristic (ROC) Curve. An ROC curve characterizes the relation between positive hits and false alarms. The performance of each classifier is represented as a point on the curve (see Fig. 7.7).

Ziegler et al. show [100] that evaluating recommender algorithms through top-N lists measures still does not map directly to the user's utility function. However, it does address some of the limitations of the more commonly accepted accuracy measures, such as MAE. Basu et al. [16], for instance, use this approach by analyzing which of the items predicted in the top quartile of the rating scale were actually evaluated in the top quartile by the user. McLaughlin and Herlocker [66]

propose a *modified precision* measure in which non-rated items are counted as *not recommendable*. This precision measure in fact represents a lower-bound of the "real" precision. Although the F-measure can be directly derived from the precision-recall values, it is not common to find it in RS evaluations. Huang et al. [56] and Bozzon et al. [22], and Miyahara and Pazzani [68] are some of the few examples of the use of this measure.

ROC curves have also been used in evaluating RS. Zhang et al. [82] use the value of the area under the ROC curve as their evaluation measure when comparing the performance of different algorithms under attack. Banerjee and Ramanathan [13] also use the ROC curves to compare the performance of different models.

It must be noted, though, that the choice of a good evaluation measure, even in the case of a top-N RS, is still a matter of discussion. Many authors have proposed measures that are only indirectly related to these traditional evaluation schemes. Deshpande and Karypis [35], for instance, propose the use of the *hit rate* and the *average reciprocal hit-rank*. On the other hand, Breese et al. [24] define a measure of the utility of the recommendation in a ranked list as a function of the neutral vote. It is also becoming increasingly common to treat a top-N RS as a learning-to-rank problem. In that context, it is common to use ranking metrics such as Mean Average Precision (MAP), Normalized Discounted Cumulative Gain (NDCG), Fraction of Concordant Pairs (FCP), or Mean Reciprocal Rank (MRR).

Chapter 8 focuses on the use of some of these evaluation measures in the context of RS and is therefore a good place to continue if you are interested on this topic.

7.4 Unsupervised Learning

7.4.1 Clustering

The main problem for scaling a CF classifier is the amount of operations involved in computing distances—for finding the best k-nearest neighbors, for instance. A possible solution is, as we saw in Sect. 7.2.3, to reduce dimensionality. But, even if we reduce dimensionality of features, we might still have many objects to compute the distance to. This is where clustering algorithms can come into play. The same is true for content-based RS, where distances among objects are needed to retrieve similar ones. Clustering is sure to improve efficiency because the number of operations is reduced. However, the improve in accuracy is not guaranteed.

Clustering [54] consists of assigning items to groups so that the items in the same groups are more similar than items in different groups: the goal is to discover natural (or meaningful) groups that exist in the data. Similarity is determined using a distance measure, such as the ones reviewed in Sect. 7.2.1. The goal of a clustering algorithm is to minimize intra-cluster distances while maximizing inter-cluster distances.

There are two main categories of clustering algorithms: hierarchical and partitional. Partitional clustering algorithms divide data items into non-overlapping clusters such that each data item is in exactly one cluster. Hierarchical clustering algorithms successively cluster items within found clusters, producing a set of nested cluster organized as a hierarchical tree.

Many clustering algorithms try to minimize a function that measures the quality of the clustering. Such a quality function is often referred to as the objective function, so clustering can be viewed as an optimization problem: the ideal clustering algorithm would consider all possible partitions of the data and output the partitioning that minimizes the quality function. But the corresponding optimization problem is NP hard, so many algorithms resort to heuristics. The main point is that clustering is a difficult problem for which finding optimal solutions is often not possible. For that same reason, selection of the particular clustering algorithm and its parameters (e.g., similarity measure) depend on many factors, including the characteristics of the data. In the following paragraphs we describe the *k*-means clustering algorithm and some of its alternatives.

7.4.1.1 *k*-Means

k-Means clustering is a partitioning method. The function partitions the data set of N items into k disjoint subsets S_j that contain N_j items so that they are as close to each other as possible according a given distance measure. Each cluster in the partition is defined by its N_j members and by its centroid λ_j. The centroid for each cluster is the point to which the sum of distances from all items in that cluster is minimized. Thus, we can define the *k*-means algorithm as an iterative process to minimize $E = \sum_1^k \sum_{n \in S_j} d(x_n, \lambda_j)$, where x_n is a vector representing the n-th item, λ_j is the centroid of the item in S_j and d is the distance measure. The *k*-means algorithm moves items between clusters until E cannot be decreased further.

The algorithm works by randomly selecting k centroids. Then all items are assigned to the cluster whose centroid is the closest to them. The new cluster centroid needs to be updated to account for the items who have been added or removed from the cluster and the membership of the items to the cluster updated. This operation continues until there are no further items that change their cluster membership. Most of the convergence to the final partition takes place during the first iterations of the algorithm, and therefore, the stopping condition is often changed to "until relatively few points change clusters" in order to improve efficiency.

The basic *k*-means is an extremely simple and efficient algorithm. However, it does have several shortcomings: (1) it assumes prior knowledge of the data in order to choose the appropriate k ; (2) the final clusters are very sensitive to the selection of the initial centroids; and (3), it can produce empty cluster. *k*-means also has several limitations with regard to the data: it has problems when clusters are of differing sizes, densities, and non-globular shapes; and it also has problems when the data contains outliers.

Xue et al. [97] present a typical use of clustering in the context of a RS by employing the k-*means* algorithm as a pre-processing step to help in neighborhood formation. They do not restrict the neighborhood to the cluster the user belongs to but rather use the distance from the user to different cluster centroids as a pre-selection step for the neighbors. They also implement a cluster-based smoothing technique in which missing values for users in a cluster are replaced by cluster representatives. Their method is reported to perform slightly better than standard *k*NN-based CF. In a similar way, Sarwar et al. [37] describe an approach to implement a scalable *k*NN classifier. They partition the user space by applying the *bisecting* k-*means* algorithm and then use those clusters as the base for neighborhood formation. They report a decrease in accuracy of around 5 % as compared to standard *k*NN CF. However, their approach allows for a significant improvement in efficiency.

Connor and Herlocker [31] present a different approach in which, instead of users, they cluster items. Using the Pearson Correlation similarity measure they try out four different algorithms: average link hierarchical agglomerative [52], robust clustering algorithm for categorical attributes (ROCK) [53], kMetis, and hMetis.[4] Although clustering did improve efficiency, all of their clustering techniques yielded worse accuracy and coverage than the non-partitioned baseline. Finally, Li et al. [79] and Ungar and Foster [92] present a very similar approach for using k-*means* clustering for solving a probabilistic model interpretation of the recommender problem.

7.4.1.2 Alternatives to *k*-Means

Density-based clustering algorithms such as DBSCAN work by building up on the definition of density as the number of points within a specified radius. DBSCAN, for instance, defines three kinds of points: *core points* are those that have more than a specified number of neighbors within a given distance; *border points* have fewer than the specified number but belong to a *core point* neighborhood; and *noise points* are those that are neither core or border. The algorithm iteratively removes *noise points* and performs clustering on the remaining points.

Message-passing clustering algorithms are a very recent family of graph-based clustering methods. Instead of considering an initial subset of the points as centers and then iteratively adapt those, message-passing algorithms initially consider all points as centers—usually known as *exemplars* in this context. During the algorithm execution points, which are now considered nodes in a network, exchange messages until clusters gradually emerge. *Affinity Propagation* is an important representative of this family of algorithms [44] that works by defining two kinds of messages between nodes: "responsibility", which reflects how well-suited receiving point is to serve as exemplar of the point sending the message, taking into account other

[4] http://www.cs.umn.edu/~karypis/metis.

potential exemplars; and "availability", which is sent from candidate exemplar to the point and reflects how appropriate it would be for the point to choose the candidate as its exemplar, taking into account support from other points that are choosing that same exemplar. Affinity propagation has been applied, with very good results, to problems as different as DNA sequence clustering, face clustering in images, or text summarization.

Hierarchical Clustering, produces a set of nested clusters organized as a hierarchical tree (*dendogram*). Hierarchical Clustering does not have to assume a particular number of clusters in advanced. Also, any desired number of clusters can be obtained by selecting the tree at the proper level. Hierarchical clusters can also sometimes correspond to meaningful taxonomies. Traditional hierarchical algorithms use a similarity or distance matrix and merge or split one cluster at a time. There are two main approaches to hierarchical clustering. In *agglomerative* hierarchical clustering we start with the points as individual clusters and at each step, merge the closest pair of clusters until only one cluster (or k clusters) are left. In *divisive* hierarchical clustering we start with one, all-inclusive cluster, and at each step, split a cluster until each cluster contains a point (or there are k clusters).

To the best of our knowledge, the previous alternatives to *k*-means have not been applied to RS. On the other hand, other approaches such as Locality-Sensitive Hashing or Bayesian non-parametric models have already proved useful in practical applications.

Locality-sensitive hashing (LSH) [10] is a technique for solving a nearest-neighbor search in high dimensionality spaces. The algorithm relies on the use of hashing functions that preserve "locality" or, in other words, bucket together items that are similar. LSH is an unsupervised method that can be considered as an approach to clustering. However, since it is an approximate solution to the nearest-neighbor problem, it can also be used for supervised classification as explained in Sect. 7.3.1.1. Due to its performance and scalability, LSH is used as a preprocessing step to group similar users in some industrial RS approaches. LinkedIn, for example, has publicly described its application for people recommendation [18].

Latent Dirichlet Allocation (LDA) [20] is a generative unsupervised model that can also be considered a form of clustering. As opposed to the previous methods though, LDA is a mixed membership model in which we consider that each data point may belong to more than a single clusters. A typical application of LDA is to identify topics in collections of documents. In that sense, LDA is also very related to Latent Semantic Analysis, and therefore techniques such as SVD (see Sect. 7.2.3). LDA has been used in different ways for content-based recommendations. For example, Jin et. al use LDA to identify topics in webpages in order to implement a hybrid content/CF recommender system [58]. LDA is also a common approach to tag recommendation (see [62], for example).

Finally, **Bayesian non-parametric models** is a family of methods that combines the power of mixed-membership models such as LDA, and the flexibility of dynamic methods that adapt the number of clusters to the underlying data distribution. **Hierarchical Dirichlet Processes (HDP)** [89] and **Recurrent Chinese Restaurant**

Processes (RCRP) have been used to cluster documents and users to later perform recommendations [3]. These initial results are promising, and highlight the applicability of these flexible approaches for RS.

7.4.2 Association Rule Mining

Association Rule Mining focuses on finding rules that will predict the occurrence of an item based on the occurrences of other items in a transaction. The fact that two items are found to be related means co-occurrence but not causality. Note that this technique should not be confused with rule-based classifiers presented in Sect. 7.3.1.3.

We define an *itemset* as a collection of one or more items (e.g. (Milk, Beer, Diaper)). A *k-itemset* is an itemset that contains k items. The frequency of a given itemset is known as *support count* (e.g. (Milk, Beer, Diaper) = 131). And the *support* of the itemset is the fraction of transactions that contain it (e.g. (Milk, Beer, Diaper) = 0.12). A *frequent itemset* is an itemset with a support that is greater or equal to a *minsup* threshold. An association rule is an expression of the form $X \Rightarrow Y$, where X and Y are itemsets. (e.g. $Milk, Diaper \Rightarrow Beer$). In this case the *support* of the association rule is the fraction of transactions that have both X and Y. On the other hand, the *confidence* of the rule is how often items in Y appear in transactions that contain X.

Given a set of transactions T, the goal of association rule mining is to find all rules having $support \geq minsupthreshold$ and $confidence \geq minconfthreshold$. The brute-force approach would be to list all possible association rules, compute the support and confidence for each rule and then prune rules that do not satisfy both conditions. This is, however, computationally very expensive. For this reason, we take a two-step approach: (1) Generate all itemsets whose support $\geq$ minsup (**Frequent Itemset Generation**); (2) Generate high confidence rules from each frequent itemset (**Rule Generation**).

Several techniques exist to optimize the generation of frequent itemsets. On a broad sense they can be classified into those that try to minimize the number of candidates (M), those that reduce the number of transactions (N), and those that reduce the number of comparisons (NM). The most common approach though, is to reduce the number of candidates using the *Apriori principle*. This principle states that if an itemset is frequent, then all of its subsets must also be frequent. This is verified using the support measure because the support of an itemset never exceeds that of its subsets. The Apriori Algorithm is a practical implementation of the principle.

Given a frequent itemset L, the goal when generating rules is to find all non-empty subsets that satisfy the minimum confidence requirement. If $|L| = k$, then there are $2k-2$ candidate association rules. So, as in the frequent itemset generation, we need to find ways to generate rules efficiently. For the Apriori Algorithm we can generate candidate rules by merging two rules that share the same prefix in the rule consequent.

The effectiveness of association rule mining for uncovering patterns and driving personalized marketing decisions has been known for a some time [2]. However, and although there is a clear relation between this method and the goal of a RS, they have not become mainstream. The main reason is that this approach is similar to item-based CF but is less flexible since it requires of an explicit notion of *transaction*—e.g. co-occurrence of events in a given session. In the next paragraphs we present some promising examples, some of which indicate that association rules still have not had their last word.

Mobasher et al. [69] present a system for web personalization based on association rules mining. Their system identifies association rules from pageviews co-occurrences based on users navigational patterns. Their approach outperforms a *k*NN-based recommendation system both in terms of precision and coverage. Smyth et al. [87] present two different case studies of using association rules for RS. In the first case they use the a priori algorithm to extract item association rules from user profiles in order to derive a better item-item similarity measure. In the second case, they apply association rule mining to a *conversational* recommender. The goal here is to find co-occurrent *critiques*—i.e. user indicating a preference over a particular feature of the recommended item. Lin et al. [65] present a new association mining algorithm that adjusts the minimum support of the rules during mining in order to obtain an appropriate number of significant rule therefore addressing some of the shortcomings of previous algorithms such as the a priori. They mine both association rules between users and items. The measured accuracy outperforms previously reported values for correlation-based recommendation and is similar to the more elaborate approaches such as the combination of SVD and ANN.

Finally, as already mentioned in Sect. 7.3.1.2, Cho et al. [28] combine Decision Trees and Association Rule Mining in a web shop RS. In their system, association rules are derived in order to link related items. The recommendation is then computed by intersecting association rules with user preferences. They look for association rules in different transaction sets such as purchases, basket placement, and click-through. They also use a heuristic for weighting rules coming from each of the transaction sets. Purchase association rules, for instance, are weighted higher than click-through association rules.

7.5 Conclusions

This chapter has introduced the main data mining methods and techniques that can be applied in the design of a RS. We have also surveyed their use in the literature and provided some rough guidelines on how and where they can be applied.

We started by reviewing techniques that can be applied in the pre-processing step. First, there is the choice of an appropriate distance measure, which is reviewed in Sect. 7.2.1. This is required by most of the methods in the following steps. The cosine similarity and Pearson correlation are commonly accepted as the best choice. Then, in Sect. 7.2.2, we reviewed the basic sampling techniques that need to be

applied in order to select a subset of an originally large data set, or to separating a training and a testing set. Finally, we discussed the use of dimensionality reduction techniques such as Principal Component Analysis and Singular Value Decomposition in Sect. 7.2.3 as a way to address the *curse of dimensionality* problem.

In Sect. 7.3, we reviewed the main classification methods: namely, nearest-neighbors, decision trees, rule-based classifiers, Bayesian networks, logistic regression, support vector machines, and artificial neural networks. We saw that, although *k*NN (see Sect. 7.3.1.1) CF is the preferred approach, all those classifiers can be applied in different settings. Decision trees (see Sect. 7.3.1.2) can be used to derive a model based on the content of the items or to model a particular part of the system. Decision rules (see Sect. 7.3.1.3) can be derived from a pre-existing decision trees, or can also be used to introduce business or domain knowledge. Bayesian networks (see Sect. 7.3.1.4) are a popular approach to content-based recommendation, but can also be used to derive a model-based CF system. In a similar way, Artificial Neural Networks can be used to derive a model-based recommender but also to combine/hybridize several algorithms. Finally, support vector machines (see Sect. 7.3.1.6) are gaining popularity also as a way to infer content-based classifications or derive a CF model.

Choosing the right classifier for a RS is not easy and is in many senses task and data-dependent. In the case of CF, some results seem to indicate that model-based approaches using classifiers such as the SVM or Bayesian Networks can slightly improve performance of the standard *k*NN classifier. However, those results are non-conclusive and hard to generalize. In the case of a content-based RS there is some evidence that in some cases Bayesian Networks will perform better than simpler methods such as decision trees. However, it is not clear that more complex non-linear classifiers such as the ANN or SVMs can perform better.

The choice of the right classifier for a specific recommending task still has nowadays much of exploratory. A practical rule-of-thumb is to start with the simplest approach and only introduce complexity if the performance gain obtained justifies it. The performance gain should of course balance different dimensions. In Sect. 7.3.3 we reviewed different ways to evaluate the performance of a classifier. Another option is to combine different classifiers in an ensemble. We described different techniques to build ensembles in Sect. 7.3.2.

We reviewed clustering algorithms in Sect. 7.4.1. Clustering is usually used in RS to improve performance. A previous clustering step, either in the user of item space, reduces the number of distance computations we need to perform. The simplicity and relative efficiency of the *k*-means algorithm (see Sect. 7.4.1.1) make it hard to find a practical alternative. We reviewed some of them such as Hierarchical Clustering or Message-passing algorithms in Sect. 7.4.1.2.

Finally, in Sect. 7.4.2, we described association rules and surveyed their use in RS. Association rules offer an intuitive framework for recommending items whenever there is an explicit or implicit notion of *transaction*. Although there exist efficient algorithms for computing association rules, and they have proved more accurate than standard *k*NN CF, they are still not a favored approach.

The choice of the right DM technique in designing a RS is a complex task that is bound by many problem-specific constraints. However, we hope that the short review of techniques and experiences included in this chapter can help the reader make a much more informed decision. Besides, we have also touched upon areas that are open to many further improvements, and where there is still much exciting and relevant research to be done in the coming years.

References

1. G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.
2. R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, 1994.
3. A. Ahmed and E. Xing. Scalable dynamic nonparametric bayesian models of content and users. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI'13, pages 3111–3115. AAAI Press, 2013.
4. X. Amatriain. Big & personal: data and models behind netflix recommendations. In *Proceedings of the 2nd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, pages 1–6. ACM, 2013.
5. X. Amatriain. Mining large streams of user data for personalized recommendations. *ACM SIGKDD Explorations Newsletter*, 14(2):37–48, 2013.
6. X. Amatriain, N. Lathia, J. M. Pujol, H. Kwak, and N. Oliver. The wisdom of the few: A collaborative filtering approach based on expert opinions from the web. In *Proc. of SIGIR '09*, 2009.
7. X. Amatriain, J. M. Pujol, and N. Oliver. I like it... i like it not: Evaluating user ratings noise in recommender systems. In *UMAP '09*, 2009.
8. X. Amatriain, J. M. Pujol, N. Tintarev, and N. Oliver. Rate it again: Increasing recommendation accuracy by user re-rating. In *Recys '09*, 2009.
9. M. Anderson, M. Ball, H. Boley, S. Greene, N. Howse, D. Lemire, and S. McGrath. Racofi: A rule-applying collaborative filtering system. In *Proc. IEEE/WIC COLA'03*, 2003.
10. A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, Jan. 2008.
11. B. D. Baets. Growing decision trees in an ordinal setting. *International Journal of Intelligent Systems*, 2003.
12. S. Balakrishnan and S. Chopra. Collaborative ranking. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 143–152. ACM, 2012.
13. S. Banerjee and K. Ramanathan. Collaborative filtering on skewed datasets. In *Proc. of WWW '08*, 2008.
14. D. Barber. Bayesian Reasoning and Machine Learning. Cambridge University Press. 2012.
15. C. Basu, H. Hirsh, and W. Cohen. Recommendation as classification: Using social and content-based information in recommendation. In *In Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 714–720. AAAI Press, 1998.
16. C. Basu, H. Hirsh, and W. Cohen. Recommendation as classification: Using social and content-based information in recommendation. In *AAAI Workshop on Recommender Systems*, 1998.
17. R. M. Bell, Y. Koren, and C. Volinsky. The bellkor solution to the netflix prize. Technical report, AT&T Labs – Research, 2007.

18. A. Bhasin. Beyond ratings and followers. In *Proceedings of the 6th ACM Conference on Recommender Systems*, RecSys '12, 2012.
19. C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
20. D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, Mar. 2003.
21. A. Bouza, G. Reif, A. Bernstein, and H. Gall. Semtree: ontology-based decision tree algorithm for recommender systems. In *International Semantic Web Conference*, 2008.
22. A. Bozzon, G. Prandi, G. Valenzise, and M. Tagliasacchi. A music recommendation system based on semantic audio segments similarity. In *Proceeding of Internet and Multimedia Systems and Applications - 2008*, 2008.
23. M. Brand. Fast online svd revisions for lightweight recommender systems. In *SIAM International Conference on Data Mining (SDM)*, 2003.
24. J. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence*, page 43–52, 1998.
25. L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
26. R. Burke. Hybrid web recommender systems. pages 377–408. 2007.
27. W. Cheng, J. Hühn, and E. Hüllermeier. Decision tree and instance-based learning for label ranking. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 161–168, New York, NY, USA, 2009. ACM.
28. Y. Cho, J. Kim, and S. Kim. A personalized recommender system based on web usage mining and decision tree induction. *Expert Systems with Applications*, 2002.
29. C. Christakou and A. Stafylopatis. A hybrid movie recommender system based on neural networks. In *ISDA '05: Proceedings of the 5th International Conference on Intelligent Systems Design and Applications*, pages 500–505, 2005.
30. W. Cohen. Fast effective rule induction. In *Machine Learning: Proceedings of the 12th International Conference*, 1995.
31. M. Connor and J. Herlocker. Clustering items for collaborative filtering. In *SIGIR Workshop on Recommender Systems*, 2001.
32. T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.
33. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, March 2000.
34. S. Deerwester, S. T. Dumais, G. W. Furnas, L. T. K., and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41, 1990.
35. M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, 2004.
36. I. D. E. Montanés, J.-R. Quevedo and J. Ranilla. Collaborative tag recommendation system based on logistic regression. In *ECML PKDD Discovery Challenge 09*, 2009.
37. B. S. et al. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Proceedings of the Fifth International Conference on Computer and Information Technology*, 2002.
38. K. O. et al. Context-aware svm for context-dependent information recommendation. In *International Conference On Mobile Data Management*, 2006.
39. P. T. et al. *Introduction to Data Mining*. Addison Wesley, 2005.
40. S. G. et al. Tv content recommender system. In *AAAI/IAAI 2000*, 2000.
41. S. H. et al. Aimed- a personalized tv recommendation system. In *Interactive TV: a Shared Experience*, 2007.
42. T. B. et al. A trail based internet-domain recommender system using artificial neural networks. In *Proceedings of the Int. Conf. on Adaptive Hypermedia and Adaptive Web Based Systems*, 2002.
43. Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, 2003.

44. B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 307, 2007.
45. J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.
46. N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Mach. Learn.*, 29(2–3):131–163, 1997.
47. S. Funk. Netflix update: Try this at home, 2006.
48. R. Ghani and A. Fano. Building recommender systems using a knowledge base of product semantics. In *In 2nd International Conference on Adaptive Hypermedia and Adaptive Web Based Systems*, 2002.
49. N. Golbandi, Y. Koren, and R. Lempel. Adaptive bootstrapping of recommender systems using decision trees. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 595–604. ACM, 2011.
50. K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Journal Information Retrieval*, 4(2):133–151, July 2001.
51. G. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 14(5):403–420, April 1970.
52. E. Gose, R. Johnsonbaugh, and S. Jost. *Pattern Recognition and Image Analysis*. Prentice Hall, 1996.
53. S. Guha, R. Rastogi, and K. Shim. Rock: a robust clustering algorithm for categorical attributes. In *Proc. of the 15th Int'l Conf. On Data Eng.*, 1999.
54. J. A. Hartigan. *Clustering Algorithms (Probability & Mathematical Statistics)*. John Wiley & Sons Inc, 1975.
55. J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, 2004.
56. Z. Huang, D. Zeng, and H. Chen. A link analysis approach to recommendation under sparse data. In *Proceedings of AMCIS 2004*, 2004.
57. A. Isaksson, M. Wallman, H. Göransson, and M. G. Gustafsson. Cross-validation and bootstrapping are unreliable in small sample classification. *Pattern Recognition Letters*, 29:1960–1965, 2008.
58. X. Jin, Y. Zhou, and B. Mobasher. A maximum entropy web recommendation system: Combining collaborative and content features. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 612–617, New York, NY, USA, 2005. ACM.
59. I. T. Jolliffe. *Principal Component Analysis*. Springer, 2002.
60. H. Kang and S. Yoo. Svm and collaborative filtering-based prediction of user preference for digital fashion recommendation systems. *IEICE Transactions on Inf & Syst*, 2007.
61. Y. Koren. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 2009.
62. R. Krestel, P. Fankhauser, and W. Nejdl. Latent dirichlet allocation for tag recommendation. In *Proceedings of the third ACM conference on Recommender systems*, pages 61–68. ACM, 2009.
63. M. Kurucz, A. A. Benczur, and K. Csalogany. Methods for large scale svd with missing values. In *Proceedings of KDD Cup and Workshop 2007*, 2007.
64. N. Lathia, S. Hailes, and L. Capra. The effect of correlation coefficients on communities of recommenders. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 2000–2005, New York, NY, USA, 2008. ACM.
65. W. Lin and S. Alvarez. Efficient adaptive-support association rule mining for recommender systems. *Data Mining and Knowledge Discovery Journal*, 6(1), 2004.
66. M. R. McLaughlin and J. L. Herlocker. A collaborative filtering algorithm and evaluation metric that accurately model the user experience. In *Proc. of SIGIR '04*, 2004.
67. S. M. McNee, J. Riedl, and J. A. Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 1097–1101, New York, NY, USA, 2006. ACM Press.

68. K. Miyahara and M. J. Pazzani. Collaborative filtering with the simple bayesian classifier. In *Pacific Rim International Conference on Artificial Intelligence*, 2000.
69. B. Mobasher, H. Dai, T. Luo, and M. Nakagawa. Effective personalization based on association rule discovery from web usage data. In *Workshop On Web Information And Data Management, WIDM '01*, 2001.
70. K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.
71. D. Nikovski and V. Kulev. Induction of compact decision trees for personalized recommendation. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 575–581, New York, NY, USA, 2006. ACM.
72. M. P. O'mahony. Detecting noise in recommender system databases. In *In Proceedings of the International Conference on Intelligent User Interfaces (IUI'06), 29th–1st*, pages 109–115. ACM Press, 2006.
73. D. Parra, A. Karatzoglou, X. Amatriain, and I. Yavuz. Implicit feedback recommendation via implicit-to-explicit ordinal logistic regression mapping. 2011.
74. A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD Cup and Workshop 2007*, 2007.
75. M. J. Pazzani. A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review*, 13:393–408, 1999.
76. M. J. Pazzani and D. Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27(3):313–331, 1997.
77. V. Pronk, W. Verhaegh, A. Proidl, and M. Tiemann. Incorporating user control into recommender systems based on naive bayesian classification. In *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems*, pages 73–80, 2007.
78. D. Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann, second edition, 1999.
79. B. K. Q. Li. Clustering approach for hybrid recommender system. In *Web Intelligence 03*, 2003.
80. J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, March 1986.
81. T. T. R. Zhang and Y. Mao. Recommender systems from words of few mouths. In *Proceedings of IJCAJ 11*, 2011.
82. J. F. S. Zhang, Y. Ouyang and F. Makedon. Analysis of a low-dimensional linear model under recommendation attacks. In *Proc. of SIGIR '06*, 2006.
83. R. Salakhutdinov, A. Mnih, and G. E. Hinton. Restricted Boltzmann machines for collaborative filtering. In *Proc of ICML '07*, New York, NY, USA, 2007. ACM.
84. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Incremental svd-based algorithms for highly scalable recommender systems. In *5th International Conference on Computer and Information Technology (ICCIT)*, 2002.
85. B. M. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl. Application of dimensionality reduction in recommender systems—a case study. In *ACM WebKDD Workshop*, 2000.
86. A. Schclar, A. Tsikinovsky, L. Rokach, A. Meisels, and L. Antwarg. Ensemble methods for improving the performance of neighborhood-based collaborative filtering. In *RecSys '09: Proceedings of the third ACM conference on Recommender systems*, pages 261–264, New York, NY, USA, 2009. ACM.
87. B. Smyth, K. McCarthy, J. Reilly, D. O'Sullivan, L. McGinty, and D. Wilson. Case studies in association rule mining for recommender systems. In *Proc. of International Conference on Artificial Intelligence (ICAI '05)*, 2005.
88. E. Spertus, M. Sahami, and O. Buyukkokten. Evaluating similarity measures: A large-scale study in the orkut social network. In *Proceedings of the 2005 International Conference on Knowledge Discovery and Data Mining (KDD-05)*, 2005.
89. Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei. Hierarchical dirichlet processes. *Journal of the American Statistical Association*, 101, 2004.
90. M. Tiemann and S. Pauws. Towards ensemble learning for hybrid music recommendation. In *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems*, pages 177–178, New York, NY, USA, 2007. ACM.

91. A. Toescher, M. Jahrer, and R. Legenstein. Improved neighborhood-based algorithms for large-scale recommender systems. In *In KDD-Cup and Workshop 08*, 2008.
92. L. H. Ungar and D. P. Foster. Clustering methods for collaborative filtering. In *Proceedings of the Workshop on Recommendation Systems*, 2000.
93. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, second edition, 2005.
94. M. Wu. Collaborative filtering via ensembles of matrix factorizations. In *Proceedings of KDD Cup and Workshop 2007*, 2007.
95. Z. Xia, Y. Dong, and G. Xing. Support vector machines for collaborative filtering. In *ACM-SE 44: Proceedings of the 44th annual Southeast regional conference*, pages 169–174, New York, NY, USA, 2006. ACM.
96. J. Xu and K. Araki. A svm-based personal recommendation system for tv programs. In *Multi-Media Modelling Conference Proceedings*, 2006.
97. G.-R. Xue, C. Lin, Q. Yang, W. Xi, H.-J. Zeng, Y. Yu, and Z. Chen. Scalable collaborative filtering using cluster-based smoothing. In *Proceedings of the 2005 SIGIR*, 2005.
98. K. Yu, V. Tresp, and S. Yu. A nonparametric hierarchical bayesian framework for information filtering. In *SIGIR '04*, 2004.
99. Y. Zhang and J. Koren. Efficient bayesian hierarchical user modeling for recommendation system. In *SIGIR 07*, 2007.
100. C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *Proc. of WWW '05*, 2005.
101. J. Zurada. *Introduction to artificial neural systems*. West Publishing Co., St. Paul, MN, USA, 1992.