

User Modeling and Personalization

7: Content-Based and Hybrid Recommender Systems

Eelco Herder

L3S Research Center / Leibniz University of Hanover
Hannover, Germany

23 May 2016

Outline I

Content-Based and Hybrid Recommender Systems

Content-based Recommenders

- Application Areas

- Utility-Based Recommender

- Prediction and Critiquing

Information Retrieval Approach

- HTML and Text Preprocessing

- Term weighting

- Weighted-Term User Model

- Prediction of document relevance

- Example: Beijing Duck Recipe

- Discussion

Hybrid Recommender Systems

Outline II

Weighted Hybrid

Switching Hybrid

Feature-Augmented Hybrid

Cascade Hybrid

Content-Based and Hybrid Recommender Systems

Recommender Systems

Recommender systems work from a specific type of information filtering system technique that attempts to recommend items (movies, TV program/show/episode, video on demand, music, books, news, images, web pages, scientific literature such as research papers etc.) that are likely to be of interest to the user.

In the previous lecture, we focused on recommender systems that used collaborative filtering as the underlying information filtering technique.

Collaborative Filtering

Collaborative Filtering is the process of filtering or evaluating items using the opinions of other people. Collaborative filtering systems produce predictions or recommendations for a given user and one or more items.

In the first part of this lecture, we focus on content-based recommenders:

Content-Based Recommender Systems

Content-based recommender systems are systems that recommend an item to a user based upon the description of (the features of) an item and a profile of the user.

Content-based Recommenders

Content-based recommenders treat recommendation as a user-specific classification or rating problem.

The classification or rating is based on the similarity between *item features* and the user's *preferences* regarding these features.

Other users' preferences or characteristics are *not* taken into account.

Recommendation process

Content-based recommendation consists of the following steps:

1. Creating a vector representation of an item
 - ▶ Based on explicit features (or metadata): for example, features of a camera include: price, type, manufacturer, resolution, type of lens, zoom level, battery lifetime, memory, ... (*Utility-Based Approach*)
 - ▶ Based on implicit features, such as the keywords in a news article or on a web page (*Information Retrieval Approach*)
2. Applying a weighting scheme to the components of the vector (to distinguish important from less important features)
3. Creating a user model, which is also vector-based and contains the same features as the item representation
4. Generating recommendations, based on vector similarity (or using machine learning techniques)

Application Areas

Content-based recommendations are a useful alternative for collaborative filtering techniques, in various situations, such as:

- ▶ There are objective criteria for goodness; which criteria are more important may differ from user to user (e.g. price versus resolution of a camera)
- ▶ Items do not persist long enough to receive sufficient ratings (e.g. news stories) - the *new item cold start problem*
- ▶ There are insufficient ratings in the system for collaborative filtering; this may be due to the *new community cold start problem* or because user ratings are not desirable or feasible (e.g. web site recommenders)

Assumptions

Further, the following assumptions should be met:

- ▶ Items are sufficiently homogeneous: the system should be able to (fairly) compare two items based on an item vector (e.g. would that be possible for a hammer and a refrigerator?)
- ▶ It is possible to create a user model that covers the item features

Utility-Based Recommender

A utility-based recommender works on *structured data*, with items that have explicit features. Typically, these are items that one can buy or visit (books, consumer electronics, hotels, restaurants, ...).

As these items are normally retrieved from a relational database, it is quite easy to obtain the feature vector.

ID	Name	Cuisine	Service	Cost	Rating
10001	Mike's Pizza	Italian	Counter	Low	70%
1002	Chris' Cafe	French	Table	Medium	85%
10003	Jacques Bistro	French	Table	High	65%

Unstructured fields, such as user comments are not suitable for utility-based recommenders:

- ▶ A charming cafe with attentive staff overlooking the river.

Weighting scheme

This is very application-dependent:

- ▶ features may have different formats (discrete, categorical, ordinal, numerical, non-numerical)
- ▶ features may be orthogonal or related
- ▶ feature importance may be based on expert opinions, user surveys, test panels, query statistics, ...
- ▶ some features may even contain sub-features (e.g. image quality may be composed of resolution, sharpness, contrast, colors)

User model

The user model is a description of the (features of the) types of items that may be of interest to the user.

The user model may be:

- ▶ explicitly provided: for example, by issuing a query and selecting feature options in a product comparison website
- ▶ derived from a user profile: for example, stereotyping based on demographics, or refinements of coarse-grained indicated user interests
- ▶ based on user's interactions: items that a user queried for, inspected, rated, commented upon, bought, ...

User model representation

The representation of the user model may be:

- ▶ a list of items that the user searched for, inspected or bought
- ▶ an overlay model of the product features in the system
- ▶ a log of the item features that the user is interested in
- ▶ a detailed list on how important certain features are for the user
- ▶ ...

Prediction

Predictions for item interest can be done with various methods (utility functions):

- ▶ neighborhood: similarity of the item with the items already in the user profile
- ▶ naive Bayes classifiers or Bayesian networks
- ▶ rule-based (users who bought CDs from the Wise Guys might want to buy their next album)
- ▶ other machine learning techniques (i.e. association rules) or custom utility functions

In its most general form, a utility function is a sum of criteria matchings and the weights associated with these criteria.

$$\sum_{i=1}^n w_i c_i$$


Critiquing

In order to improve results, the weights for each criterion, or the matching criteria, need to be adjusted to the user's (perceived) importance of this criterion.

This is usually achieved by asking the user's feedback, for example:

- ▶ Criteria that can be ignored (weight is decreased)
- ▶ Criteria that need to be kept (weight remains the same)
- ▶ Criteria that need to be adjusted (e.g. higher threshold for criterion matching)

To find similar products with better values than this one



Canon PowerShot S2 IS Digital Camera [Add to related list](#)
 Canon, 5.3 M pixels, 12x optical zoom, 16 MB memory, 1.8 in screen size,
 2.37 in thickness, 434.7 g weight [\[data\]](#)

Would you like to improve some values?

	Keep	Improve	Take any suggestion
Manufacturer	<input checked="" type="radio"/> Canon	<input type="radio"/> Sony	<input type="radio"/>
Price	<input type="radio"/> \$424.15	<input checked="" type="radio"/> less expensive	<input type="radio"/>
Resolution	<input checked="" type="radio"/> 5.3 M pixels	<input type="radio"/> 10 M pixels	<input type="radio"/>
Optical Zoom	<input checked="" type="radio"/> 12x	<input type="radio"/> 180x zoom	<input type="radio"/>
Removable Flash Memory	<input checked="" type="radio"/> 16 MB	<input checked="" type="radio"/> more memory	<input type="radio"/>
LCD Screen Size	<input checked="" type="radio"/> 1.8 in	<input type="radio"/> larger	<input type="radio"/>
Thickness	<input checked="" type="radio"/> 2.37 in	<input checked="" type="radio"/> thinner	<input type="radio"/>
Weight	<input checked="" type="radio"/> 434.7 g	<input type="radio"/> lighter	<input type="radio"/>

[Show Results](#) [Reset](#)

Discussion

Utility-based recommenders can be used if:

- ▶ Items are described in a structured format (e.g. product features)
- ▶ All items have a number of features in common
- ▶ There is a way of constructing a user model (explicitly or implicitly)

In stores like Amazon, utility-based recommenders can only work on the level of *product categories*: the DVD department should use a different recommender than the Book department - and, hence, recommendations for books based on DVDs you have seen (e.g. Harry Potter) are not feasible.

The utility function is only based on the selected features. It may be that the function will be flawed due to other features that are unknown or than cannot be taken into account: availability, customer reviews, ...

Information Retrieval Approach

Many Web pages contain unrestricted text, which is a form of *unstructured data*.

Unlike structured data, there are no attribute names with well-defined values. Furthermore, the full complexity of natural language may be present in the text field including polysemous words (the same word may have several meanings) and synonyms (different words may have the same meaning).

For automatic systems that operate on the Web (e.g. adaptive Web-Based systems) such documents often need to be *pre-processed*: recasting them in representations different from primitive ones and more apt to undergo a particular elaboration.

A common approach to dealing with free text fields is to convert the free text to a structured representation.

For example, each word may be viewed as an attribute

- ▶ with a Boolean value indicating whether the word is in the article
- ▶ or with an integer value indicating the number of times the word appears in the article.

HTML and Text Preprocessing

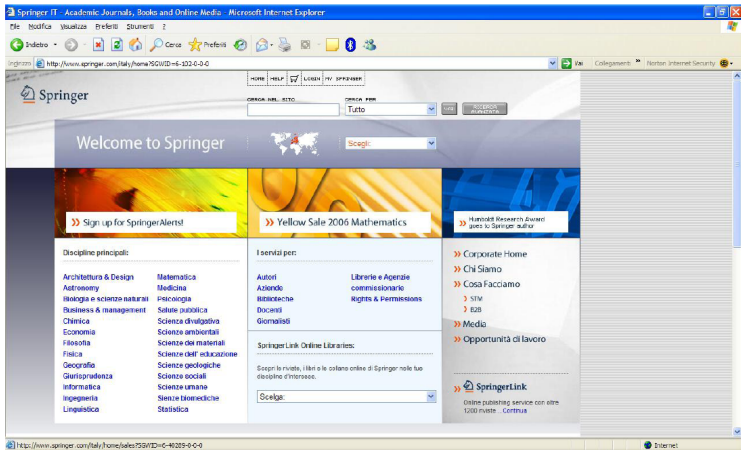
HTML stands for HyperText Markup Language and is the predominant markup language for web pages.

HTML is written in the form of HTML elements consisting of tags, enclosed in angle brackets (like `<html>`), within the web page content. HTML tags normally come in pairs like `<h1>` and `</h1>`. In between these tags web designers can add text, tables, images, etc.

The purpose of a web browser is to read HTML documents and compose them into visual or audible web pages. The browser does not display the HTML tags, but uses the tags to interpret the content of the page.

HTML has included semantic markup from its inception, but has also included presentational markup such as *font*, *i* and *center* tags. It can embed scripts in languages such as JavaScript which affect the behavior of HTML web pages.

This is what the browser sees



But this is the ‘text’ a recommender system need to work with

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!-- saved from url=(0049)http://www.springer.com/uk/home?SGWID=3-102-0-0-0 -->
<?xml version="1.0" encoding="UTF-8" ?><HTML lang=en-UK xml:lang="en"
xmlns="http://www.w3.org/1999/xhtml">
<HEAD>
<TITLE>
Springer UK - Academic Journals, Books and Online Media
</TITLE>
<!-- FA1.1 --><!-- Revision: 12435 -->
<META http-equiv=content-type content="text/html; charset=utf-8">
<META
content="Buy academic journals, books and online media at Springer. Choose from thousands
name=description>
<META
content="Springer, Publishing, Springer Online, Springer-Verlag, Books, Journals, Publisher
name=keywords>
<META http-equiv=imagetoolbar content=no>
<META content=noindex,nofollow name=robots>
<META content=all name=audience>
<META content=global name=distribution><LINK title="ICRA labels"
href="/sgw/labels.rdf" type=application/rdf+xml rel=Meta><LINK
href="springer_new_file/springer.css" type=text/css rel=stylesheet><LINK
href="/favicon.ico" type=image/ico rel=icon><LINK href="/favicon.ico"
type=image/x-icon rel="shortcut icon">
<SCRIPT language=JavaScript>
```

Step 1: Tag Removal

Consider the following excerpt:

```
<link rel="stylesheet" href="include/style\_0.css"
type="text/css"> <script language="JavaScript"
src="include/item.js"></script>
<script language="JavaScript"
src="include/fw\_menu.js"></script>
<span class="bodytext">
Benefit from attractive savings on Springer books
by signing up for Springer's free new book e-mail
notification service. New title info, news and
special announcements: with Springer Alerts it
pays to be informed.
</span>
```

This phase consists of removing all HTML instructions (i.e., tags) from an HTML page. The simplest approach excludes all terms belonging to HTML tags. In this example, only this text would remain:

Benefit from attractive savings on Springer books by signing up for Springer's free new book e-mail notification service. New title info, news and special announcements: with Springer Alerts it pays to be informed.

Step 2: Stopword Removal

Not all terms of a document are necessarily relevant. Some frequently used terms, within the document itself, tend to be removed: these terms are known as **Stopwords** (i.e., 'a', 'the', 'in', 'to'; or pronouns: 'I', 'he', 'she', 'it').

Stopwords obviously vary according to the language. It is possible to download sets of standard stopwords, of about 500 stopwords. By exploiting such a list for our text fragment, we obtain:

Benefit attractive savings Springer books signing
Springer book e-mail notification service title info
special announcements Springer Alerts pays informed

Step 3: Stemming or lemmatization

Stemming

Stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form generally a written word form. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root.

The goal of this phase is to reduce a term to its morphologic root, in order to recognize morphologic variations of the word itself. For example, the root *comput* is the reduced version of 'comput-er', 'comput-ational', 'comput-ation' and 'compute'.

One of the most widely used packages for stemming is the *Porter's Stemmer*. It is a simple (yet very elaborated) procedure, which cyclically recognizes and removes known suffixes and prefixes without having to use a dictionary.

After having applied the Porter Stemmer, we obtain the following terms:

Original Terms	Stemmed Terms	Original Terms	Stemmed Terms
benefit	benefit	notification	notif
attractive	attract	service	servic
savings	save	title	titl
springer	springer	info	info
books	book	special	special
signing	sign	announcements	announc
springer	springer	alerts	alert
book	book	pays	pai
email	email	informed	inform

Lemmatization

Lemmatization is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. Computational lemmatization is the algorithmic process of determining the lemma for a given word.

Lemmatization is closely related to stemming. The difference is that a stemmer operates on a single word without knowledge of the context, and therefore cannot discriminate between words which have different meanings depending on part of speech.

However, stemmers are typically easier to implement and run faster, and the reduced accuracy may not matter for some applications.

For instance:

- ▶ The word "better" has "good" as its lemma. This link is missed by stemming, as it requires a dictionary look-up.
- ▶ The word "walk" is the base form for word "walking", and hence this is matched in both stemming and lemmatization.
- ▶ The word "meeting" can be either the base form of a noun or a form of a verb ("to meet") depending on the context, e.g., "in our last meeting" or "We are meeting again tomorrow". Unlike stemming, lemmatization does select the right lemma depending on the context.

Term weighting

We now have our document $d = \{t1, t2, \dots, t16\}$, i.e., $d = \{\text{benefit, attract, save, springer, book, sign, email, notif, servic, titl, info, special, announc, alert, pai, inform}\}$.

Each term t_k of the document d belonging to a document collection D is named *feature* or index term and its relevance within the document d is expressed by means of an associated numeric weight w_k .

A first simple weight, also used as a starting point to build more sophisticated weights, is Term Frequency:

- ▶ $w = freq$: the number of times term t appears in the document d .

The more a term t appears in a document d , the more this term can characterize the topic dealt with by the document itself.

However, not all the terms in a document d have the same relevance in discerning the document d itself for a correct representation and retrieval.

- ▶ A term t that appears in almost all documents of the collection D does not entail relevant information content for the characterization of the topic of a particular document.

Hence, Term Weighting involves at least two components:

- ▶ the frequency of a term t within a document d
- ▶ the frequency of term t within the whole collection D .

Term frequency and inverse document frequency: $tf \times idf$

Let D be the collection of documents in the system and d_i be the set of documents in which the index term t_i appears.

Let $freq_{i,j}$ be the raw frequency of term t_i in the document d_j (i.e., the number of times the term t_i is mentioned in the text of the document d_j).

Then, the normalized frequency $f_{i,j}$ of term t_i in document d_j is given by

$$f_{i,j} = \frac{freq_{i,j}}{\sum_k freq_{k,j}}$$

where the sum is computed over all terms which are mentioned in the text of the document d_j . If the term t_i does not appear in the document d_j then $f_{i,j} = 0$.

Further, let idf_i , inverse document frequency for t_i , be given by

$$idf_i = \log \frac{|D|}{|d_i|}$$

The best known term-weighting schemes use weights which are given by

$$w_{i,j} = f_{i,j} \times idf_i$$

or by a variation of this formula. Such term-weighting strategies are called tf-idf schemes.

Weighted-Term User Model

The $tf \times idf$ approach can also be used to construct a model of the user's interests:

$$w_{i,u} = \left(0.5 + \frac{0.5 * freq_{i,u}}{\sum_k freq_{l,k}} \right) * \log \frac{|D|}{|d_i|}$$

where $w_{i,u} = 0$ when $freq_{i,u} = 0$.

The use of a minimum level of 0.5 is a common approach from the field of Information Retrieval to assign weights to query terms. It is also a useful approach to ensure that terms are given sufficient weight when the user model does not contain many terms (which is often the case).

After having applied the $\text{tf} \times \text{idf}$ weighting to the terms in the document d_j and the user model u , we can construct the term vectors

$$\vec{d}_j = (w_{1,j}, w_{2,j}, \dots, w_{k,j}), \quad \vec{u} = (w_{1,u}, w_{2,u}, \dots, w_{k,u})$$

with k the total number of terms taken into consideration.

Prediction of document relevance

After having created the weighted term vectors for the documents and the user model, we can calculate the similarity between the document d_j and the user model u , for example using cosine similarity:

$$\begin{aligned} \text{sim}(d_j, u) &= \frac{\vec{d}_j * \vec{u}}{\|\vec{d}_j\| * \|\vec{u}\|} \\ &= \frac{\sum_{i=1}^t w_{i,j} * w_{i,u}}{\sqrt{\sum_{i=1}^t w_{i,j}^2} * \sqrt{\sum_{i=1}^t w_{i,u}^2}} \end{aligned}$$

Note that in the previous lecture, for simplicity, we normalized the cosine similarity with the terms that were available in both d_j and u .

It would have been more correct to normalize with respect to the length of the *whole vectors* (including all terms $i = 1..t$), to compensate for differences in length between documents, queries and user profiles.

This IR-based recommender would recommend the top- k documents with the highest similarity to the user model.

Example: Beijing Duck Recipe

There are 5 different documents in the collection:

D1 = "If it walks like a duck and quacks like a duck, it must be a duck."

D2 = "Beijing Duck is mostly prized for the thin, crispy duck skin with authentic versions of the dish serving mostly the skin."

D3 = "Bugs' ascension to stardom also prompted the Warner animators to recast Daffy Duck as the rabbit's rival, intensely jealous and determined to steal back the spotlight while Bugs remained indifferent to the duck's jealousy, or used it to his advantage. This turned out to be the recipe for the success of the duo."

D4 = "6:25 PM 1/7/2007 blog entry: I found this great recipe for Rabbit Braised in Wine on cookingforengineers.com."

D5 = "Last week Li, from Beijing, has shown you how to make the Sechuan duck. Today we'll be making Chinese dumplings (Jiaozi), a popular dish that I had a chance to try last summer in Beijing. There are many recipees for Jiaozi, but the Beijing variety is the best."

First we count the frequency of terms in all the documents. Counting is case insensitive (Duck = duck) and we need to perform stemming (recipes = recipe).

For simplicity, we only consider the following six terms and assume that all other words have been removed.

term / count	D1	D2	D3	D4	D5
beijing		1			3
dish		1			1
duck	3	2	2		1
rabbit			1	1	
recipe			1	1	1
roast					

Next we normalize the frequencies by dividing each value in the TF table by the length of each document (for simplicity, we take the sum of term occurrences for each column).

We also compute the IDF values for each term using the formula

$$idf_i = \log \frac{|D|}{|d_i|}$$

term / freq	D1	D2	D3	D4	D5	IDF
beijing		0.25			0.5	0.3979
dish		0.25			0.167	0.3979
duck	1	0.5	0.5		0.167	0.0969
rabbit			0.25	0.5		0.3979
recipe			0.25	0.5	0.167	0.2218
roast						0.0000

We then compute the TFIDF weights by multiplying each cell in the previous table by the corresponding IDF value.

term / TFIDF	D1	D2	D3	D4	D5	IDF
beijing		0.0995			0.199	0.3979
dish		0.0995			0.066	0.3979
duck	0.0969	0.0485	0.0485		0.016	0.0969
rabbit			0.0995	0.199		0.3979
recipe			0.0555	0.1109	0.037	0.2218
roast						0.0000

Query: Beijing Duck Recipe

We will use the weighted query term vector, Q is the unweighted vector and Q_{idf} is obtained from Q by multiplying each cell by the corresponding IDF value, same as for documents.

term	Q	Q_{idf}
beijing	1	0.3979
dish	0	0
duck	1	0.0969
rabbit	0	0
recipe	1	0.2218
roast	0	0

Finally we use the cosine measure to compute the similarity. We take the cosine between each document vector and the Qidf vector.

For example, the similarity for D2 is the cosine of the angle between the query vector: (0.389, 0, 0.097, 0, 0.222, 0) and the TFIDF vector for D2: (0.0995, 0.0995, 0.0485, 0, 0, 0)

D1	D2	D3	D4	D5
0.2081	0.639	0.295	0.232	0.8941

Discussion

Although there are different approaches to learning a model of the user's interest with content-based recommendation, no content-based recommendation system can give good recommendations if the content does not contain enough information to distinguish items the user likes from items the user doesn't like.

In recommending some items, e.g., jokes or poems, there often isn't enough information in the word frequency to model the user's interests. While it would be possible to tell a lawyer joke from a chicken joke based upon word frequencies, it would be difficult to distinguish a *funny* lawyer joke from other lawyer jokes.

Hybrid Recommender Systems

Collaborative filtering recommenders and content-based recommenders have their own strengths and weaknesses:

- ▶ Content-based recommenders do not need any user ratings, but they do need a suitable structure or metadata for comparison with a user model
- ▶ Content-based recommenders are useful for answering queries or searches for specific products or news items (based on immediate need)

- ▶ Collaborative recommenders work better in situations where relations between items are not particularly content-driven (e.g. cd albums of two different artists)
- ▶ Collaborative recommenders are able to relate items from different (product) categories
- ▶ Collaborative recommenders improve when the amount of ratings grows

Hybrid recommender systems are those that combine two or more recommendation techniques to improve recommendation performance.

Weighted Hybrid

Perhaps the simplest design for a hybrid system is a weighted one. Each component of the hybrid scores a given item and the scores are combined using a linear formula.

First, both recommenders are trained independently.

In order to generate recommendations:

- ▶ Both recommenders generate candidates
- ▶ The two candidate sets are merged - either the intersection or the union of the sets is used
- ▶ If a union is used, the system must decide how to handle cases if one recommender did not provide a rating for a candidate (neutral rating perhaps?)
- ▶ Each candidate is rated by a linear combination of the two scores of the recommenders.

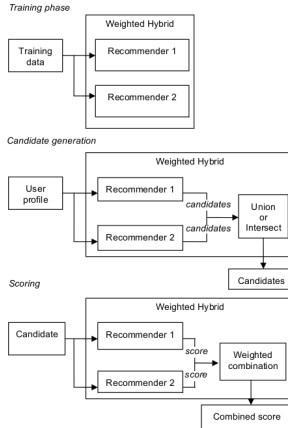


Figure: Weighted hybrid

Example weighted hybrid

A movie recommender system has two components:

- ▶ one, using collaborative techniques, identifies similarities between rating profiles and makes predictions based on this information.
- ▶ the second component uses simple semantic knowledge about the features of movies, compressed dimensionally via latent semantic analysis, and recommends movies that are semantically similar to those the user likes.

The output of the two components is combined using a linear weighting scheme.

Advantages:

- ▶ Both recommendation techniques are taken into account in the results
- ▶ It is easy to check which of both techniques performed best (to adjust the weighting scheme)

Disadvantage:

- ▶ Both recommendation techniques are assumed to perform equally well for all items. This might not be the case - for example, collaborative filtering does not work well for items with few ratings.

Switching Hybrid

A switching hybrid is one that selects a single recommender from among its constituents based on the recommendation situation.

For a different profile, a different recommender might be chosen.

This approach takes into account the problem that components may not have consistent performance for all types of users.

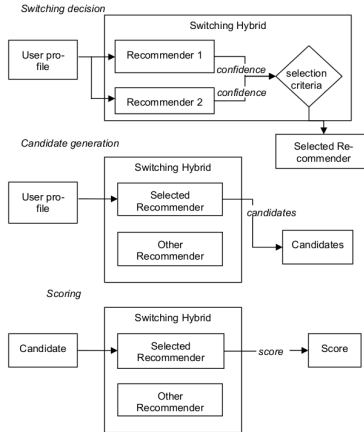


Figure: Switching hybrid

Example switching hybrid

NewsDude recommends news stories. It has three recommendation components:

- ▶ a content-based nearest-neighbor recommender
- ▶ a collaborative recommender
- ▶ and a second content-based algorithm using a naive Bayes classifier.

The recommenders are ordered. The nearest neighbor technique is used first. If it cannot produce a recommendation with high confidence, then the collaborative recommender is tried, and so on, with the naive Bayes recommender at the end of line.

Disadvantage:

- ▶ A switching recommender requires reliable switching criteria

Feature-Augmented Hybrid

A feature augmentation hybrid generates a new feature for each item by using the recommendation logic of the contributing domain.

- ▶ At each step, the contributing recommender intercepts the data header for the actual recommender and augments it with its own contribution.
- ▶ The contribution is added as one or more additional features to be taken into account by the actual recommender.

For example, a content-based (utility-based) recommender may use an item-based collaborative filtering recommender in order to find products that do not exactly fit the user's search criteria (e.g. a different brand with about the same specifications).

Advantage:

- ▶ The augmentation can usually be done offline

Disadvantage:

- ▶ It is not always immediately obvious how to create a feature augmentation recommender for any two recommendation components.

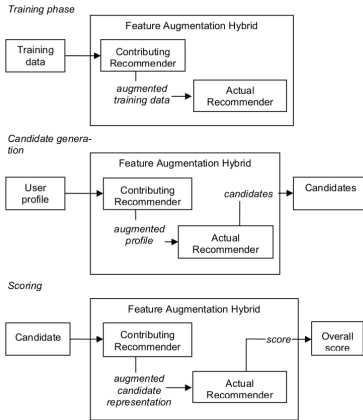


Figure: Feature-Augmented Hybrid

Cascade Hybrid

The idea of a cascade hybrid is to create a strictly hierarchical hybrid, one in which a weak recommender cannot overturn decisions made by a stronger one, but can merely refine them.

In its order-dependence, it is similar to the feature augmentation hybrid, but it is an approach that retains the function of the recommendation component as providing predicted ratings.

A cascade recommender uses a secondary recommender only to break ties in the scoring of the primary one.

Example Cascade Hybrid

The knowledge-based Entree restaurant recommender was found to return too many equally-scored items, which could not be ranked relative to each other.

Rather than additional labor-intensive knowledge engineering (to produce finer discriminations), the hybrid EntreeC was created by adding a collaborative re-ranking of only those items with equal scores.

Advantage:

- ▶ A clearly defined, easy to understand, way of combining recommenders

Disadvantage:

- ▶ Many recommendation techniques have real-valued outputs and so the probability of actual numeric ties is small. This would give the secondary recommender in a cascade little to do.

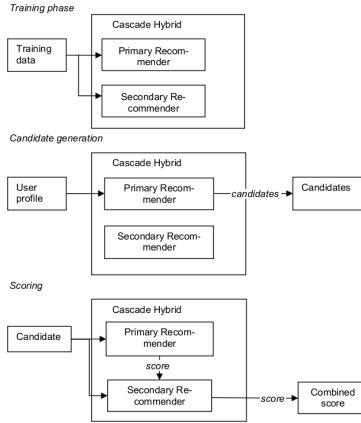


Figure: Cascade Hybrid