RS Module 3 Notes

Content-Based Filtering Recommendations

Systems implementing a content-based recommendation approach analyze a set of documents and/or descriptions of items previously rated by a user, and build a model or profile of user interests based on the features of the objects rated by that user. The profile is a structured representation of user interests, adopted to recommend new interesting items. The recommendation process basically consists in matching up the attributes of the user profile against the attributes of a content object. The result is a relevance judgment that represents the user's level of interest in that object. If a profile accurately reflects user preferences, it is of tremendous advantage for the effectiveness of an information access process.

High Level Architecture of Content-based Systems

Content-based Information Filtering (IF) systems need proper techniques for representing the items and producing the user profile, and some strategies for comparing the user profile with the item representation. The high level architecture of a content-based recommender system is depicted in below.

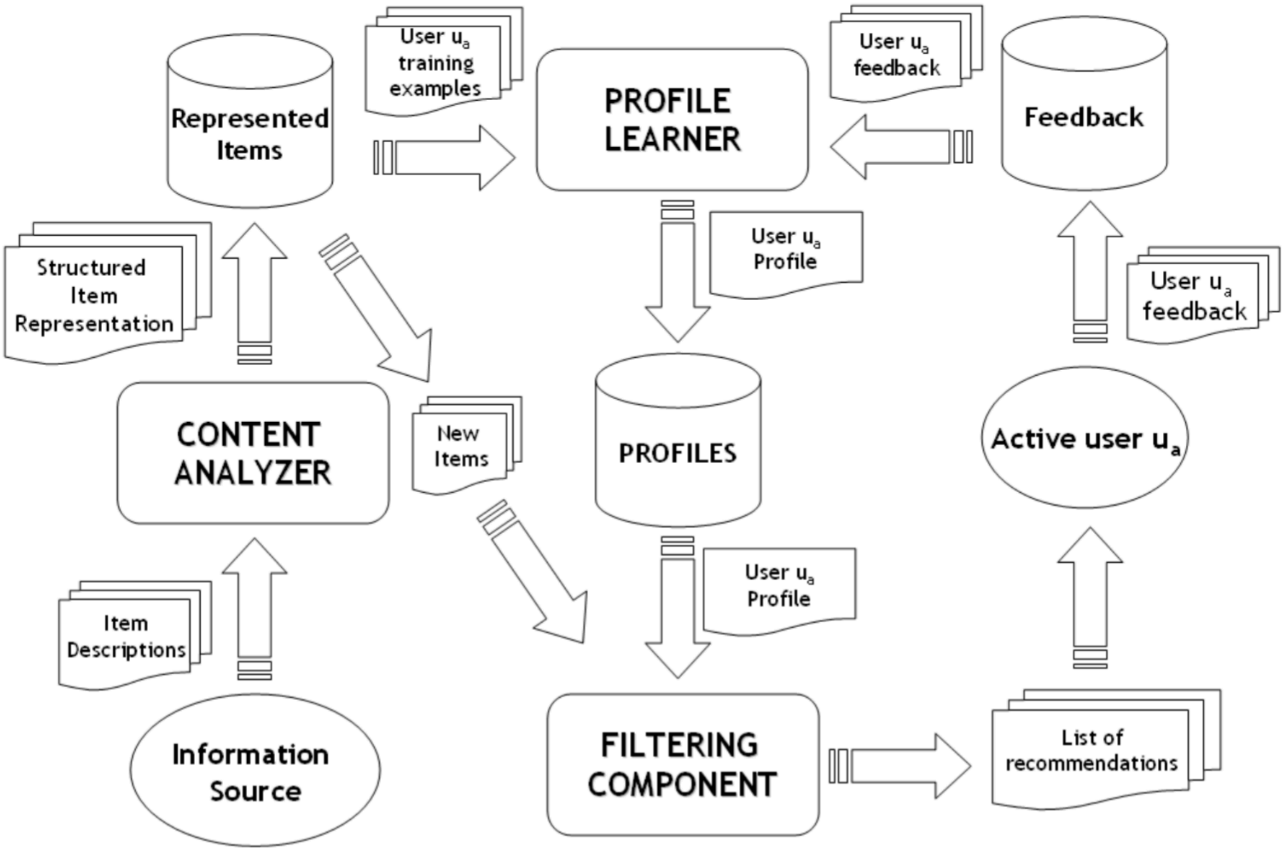


Fig. 3.1: High level architecture of a Content-based Recommender

The recommendation process is performed in three steps, each of which is handled by a separate component:

1. **CONTENT ANALYZER**: When information has no structure (e.g. text), some kind of preprocessing step is needed to extract structured relevant information. The main responsibility of the component is to represent the content of items (e.g. documents, Web pages, news, product descriptions, etc.) coming from information sources in a form suitable for the next processing steps.
2. **PROFILE LEARNER**: This module collects data representative of the user preferences and tries to generalize this data, in order to construct the user profile. Usually, the generalization strategy is realized through machine learning techniques, which are able to infer a model of user interests starting from items liked or disliked in the past. For instance, the PROFILE LEARNER of a Web page recommender can implement a relevance feedback method in which the learning technique combines vectors of positive and negative examples into a prototype vector representing the user profile.
3. **FILTERING COMPONENT**: This module exploits the user profile to suggest relevant items by matching the profile representation against that of items to be recommended. The result is a binary or continuous relevance judgment (computed using some similarity metrics), the latter case resulting in a ranked list of potentially interesting items. In the above mentioned example, the matching is realized by computing the cosine similarity between the prototype vector and the item vectors.

Advantages and Drawbacks of Content-based Filtering

The adoption of the content-based recommendation paradigm has several advantages when compared to the collaborative one:

1. **User Independence** - Content-based recommenders exploit solely ratings provided by the active user to build her own profile. Instead, collaborative filtering methods need ratings from other users in order to find the “nearest neighbors” of the active user, i.e., users that have similar tastes since they rated the same items similarly. Then, only the items that are most liked by the neighbors of the active user will be recommended.
2. **Transparency** - Explanations on how the recommender system works can be provided by explicitly listing content features or descriptions that caused an item to occur in the list of recommendations. Those features are indicators to consult in order to decide whether to trust a recommendation. Conversely, collaborative systems are black boxes since the only explanation for an item recommendation is that unknown users with similar tastes liked that item.
3. **New Item** - Content-based recommenders are capable of recommending items not yet rated by any user. As a consequence, they do not suffer from the first-rater problem, which affects collaborative recommenders which rely solely on users’ preferences to make recommendations. Therefore, until the new item is rated by a substantial number of users, the system would not be able to recommend it.

Nonetheless, content-based systems have several shortcomings:

1. **Limited Content Analysis** - Content-based techniques have a natural limit in the number and type of features that are associated, whether automatically or manually, with the objects they recommend. Domain knowledge is often needed, e.g., for movie recommendations the system needs to know the actors and directors, and sometimes, domain ontologies are also needed. No content-based recommendation system can provide suitable suggestions if the analyzed content does not contain enough information to discriminate items the user likes from items the user does not like.
2. **Over-Specialization** - Content-based recommenders have no inherent method for finding something unexpected. The system suggests items whose scores are high when matched against the user profile, hence the user is going to be recommended items similar to those already rated. This drawback is also called *serendipity* problem to highlight the tendency of the content-based systems to produce recommendations with a limited degree of *novelty*.
   To give an example, when a user has only rated movies directed by Stanley Kubrick, she will be recommended just that kind of movies. A “perfect” content-based technique would rarely find anything *novel*, limiting the range of applications for which it would be useful.
3. **New User** - Enough ratings have to be collected before a content-based recommender system can really understand user preferences and provide accurate recommendations. Therefore, when few ratings are available, as for a new user, the system will not be able to provide reliable recommendations.

Item Representation

Items that can be recommended to the user are represented by a set of features, also called attributes or properties. An **item profile** is a structured representation of an item’s features. It includes various attributes that describe the item, which are then used to match against user preferences.

- **For a movie**: The item profile may include the movie’s title, genres (action, drama), cast, director, release year, and keywords like "heroic" or "thrilling".
- **For a product**: The item profile may include the product's name, brand, price, category, specifications, etc.

These features are crucial in determining the similarity between items and matching them to users’ preferences.

Keyword-Based Vector Space Model

Most content-based recommender systems use relatively simple retrieval models, such as keyword matching or the **Vector Space Model (VSM) with basic TF-IDF weighting**. VSM is a spatial representation of text documents. In that model, each document is represented by a vector in a n-dimensional space, where each dimension corresponds to a term from the overall vocabulary of a given document collection.

Document representation in the VSM raises 2 issues: weighting the terms & measuring the feature vector similarity. The most commonly used term weighting scheme, **TF-IDF** (Term Frequency-Inverse Document Frequency) weighting, is based on empirical observations regarding text:

1. Rare terms are not less relevant than frequent terms (IDF assumption)
2. Multiple occurrences of a term in a document are not less relevant than single occurrences (TF assumption)
3. Long documents are not preferred to short documents (normalization assumption).

In other words, terms that occur frequently in one document (TF=term-frequency), but rarely in the rest of the corpus (IDF=inverse-document-frequency), are more likely to be relevant to the topic of the document. In addition, normalizing the resulting weight vectors prevent longer documents from having a better chance of retrieval. These assumptions are well exemplified by the TF-IDF function:

$$TF - IDF(t_k, d_j) = TF(t_k, d_j) \cdot log\frac{N}{n_k}$$

where N denotes the number of documents in the corpus, and n_k denotes the number of documents in the collection in which the term t_k occurs at least once. In order for the weights to fall in the [0, 1] interval and for the documents to be represented by vectors of equal length, weights obtained are usually normalized by cosine normalization:

$$w_{k,j} = \frac{TF - IDF(t_k, d_j)}{\sqrt{\sum_{s=1}^{|T|} TF - IDF(t_s, d_j)^2}}$$

Discovering Features of Documents

This involves extracting features (keywords, entities, etc.) from textual content (e.g., articles, blogs, or descriptions). Common methods include:

- **Keyword Extraction**: Identifying the most relevant terms in the document.
- **Named Entity Recognition (NER)**: Identifying named entities such as people, locations, or organizations from the text.
- **Topic Modeling**: Extracting hidden topics in a set of documents using techniques like Latent Dirichlet Allocation (LDA).

These features can then be used to create item profiles.

Obtaining Item Features from Tags

Tags are user-generated labels or keywords that describe the content of an item. Tags provide additional descriptive information that might not be present in the formal item features.

- **Example**: A movie may have the genre “Action,” but users might tag it with terms like “explosive,” “thrilling,” or “adventure.” These tags enrich the item profile with user-generated content.

Tags are often gathered from user interaction with content (e.g., social tagging) and provide valuable insight into how users perceive items.

Representing Item Profiles

Item profiles are represented as **feature vectors** in a vector space, where each dimension corresponds to a particular feature (e.g., genre or keyword).

- **Example**: A movie’s item profile could be represented by a binary vector where each element corresponds to the presence (1) or absence (0) of a genre or tag.

This vector representation allows for easy comparison between different items based on their features.

Methods for Learning User Profiles

Machine learning techniques, generally used in the task of inducing content-based profiles, are well-suited for text categorization. In a ML approach to text categorization, an inductive process automatically builds a text classifier by learning from a set of training documents (documents labeled with the categories they belong to) the features of the categories. The problem of learning user profiles can be cast as a binary text categorization task: each document has to be classified as interesting or not with respect to the user preferences. Therefore, the set of categories is $C = \{c_+, c_-\}$, where c_+ is the +ve class (user-likes) & c_- the negative one (user-dislikes).

1. **Probabilistic Methods and Naive Bayes:**
   Naive Bayes is a probabilistic approach to inductive learning, and belongs to the general class of Bayesian classifiers. These approaches generate a probabilistic model based on previously observed data. The model estimates the a posteriori probability, $P(c|d)$, of document d belonging to class c. This estimation is based on the a priori probability, $P(c)$, the probability of observing a document in class c, $P(d|c)$, the probability of observing the document d given c, and $P(d)$, the probability of observing the instance d. Using these probabilities, the Bayes theorem is applied to calculate $P(c|d)$:

   $$P(c|d) = \frac{P(c) \cdot P(d|c)}{P(d)}$$

   To classify the document d, the class with the highest probability is chosen:

$$c = argmax_{c_j} \frac{P(c_j) \cdot P(d|c_j)}{P(d)}$$

However, estimating $P(d|c)$ in this way is problematic, as it is very unlikely to see the same document more than once: the observed data is generally not enough to be able to generate good probabilities. The na¨ıve Bayes classifier overcomes this problem by simplify- ing the model through the independence assumption: all the words or tokens in the observed document d are conditionally independent of each other given the class. Individual probabilities for the words in a document are estimated one by one rather than the complete document as a whole. The conditional independence assumption is clearly violated in real-world data, however, despite these violations, empirically the naive Bayes classifier does a good job in classifying text documents. There are two commonly used working models of the naive Bayes classifier, the **multivariate Bernoulli** event model and the **multinomial** event model. Both models treat a document as a vector of values over the corpus vocabulary, V , where each entry in the vector represents whether a word occurred in the document, hence both models lose information about word order.

2. **Relevance Feedback and Rocchio's Algorithm:**

Relevance feedback is a technique adopted in Information Retrieval that helps users to incrementally refine queries based on previous search results. It consists of the users feeding back into the system decisions on the relevance of retrieved documents with respect to their information needs. Relevance feedback and its adaptation to text categorization, the well-known *Rocchio's* formula, are commonly adopted by content-based recommender systems. The general principle is to allow users to rate documents suggested by the recommender system with respect to their information need. This form of feedback can subsequently be used to incrementally refine the user profile or to train the learning algorithm that infers the user profile as a classifier. Some linear classifiers consist of an explicit profile (or prototypical document) of the category. The Rocchio's method is used for inducing linear, profile-style classifiers. This algorithm represents documents as vectors, so that documents with similar content have similar vectors. More formally, **Rocchio's** method computes a classifier $\vec{c_i} = \langle \omega_{1i}, \ldots, \omega_{|T|i} \rangle$ for the category ci (T is the vocabulary, that is the set of distinct terms in the training set) by means of the formula:

$$\omega_{ki} = \beta \cdot \sum_{\{d_j \in POS_i\}} \frac{\omega_{kj}}{|POS_i|} - \gamma \cdot \sum_{\{d_j \in NEG_i\}} \frac{\omega_{kj}}{|NEG_i|}$$

where ω_{kj} is the $TF-IDF$ weight of the term t_k in document $d_j, POS_i \text{ and } NEG_i$ are the set of positive and negative examples in the training set for the specific class $c_j, \beta \text{ and } \gamma$ are control parameters that allow to set the relative importance of all positive and negative eg. To assign a class $\bar{c}$ to a document d_j, the similarity between each prototype vector $\vec{c_i}$ and the document vector $\vec{d_j}$ is computed and $\bar{c}$ will be the c_i with the highest value of similarity.

User profiles are constructed by aggregating the features of items that the user has interacted with in the past. There are different ways to build these profiles:

1. **Simple Averaging**: Aggregate the item profiles (average the feature vectors) of all items that the user has rated positively.
2. **Weighted Averaging**: Give more weight to items that the user interacted with more frequently or rated more highly.
3. **Feedback Loops**: Continuously update the user profile as they interact with more items, providing more refined recommendations over time.

User profiles evolve as the system learns more about user preferences, improving the accuracy of future recommendations.

Similarity-Based Retrieval

This involves retrieving items similar to the ones that the user has liked before, based on comparing item and user profiles.
Example: For a user who likes "action" and "thrilling" movies, the system will recommend other movies with similar features.

Common Similarity Measures:

- **Cosine Similarity**: Measures the cosine of the angle between two feature vectors.
- **Jaccard Similarity**: Measures similarity based on the overlap of common features.

Classification Algorithms

In content-based systems, classification algorithms can be used to categorize items into predefined categories based on their features. These algorithms are also used to predict whether a user would like or dislike a particular item. regard the given data as a training set, and for each user, build a classifier that predicts the rating of all items.

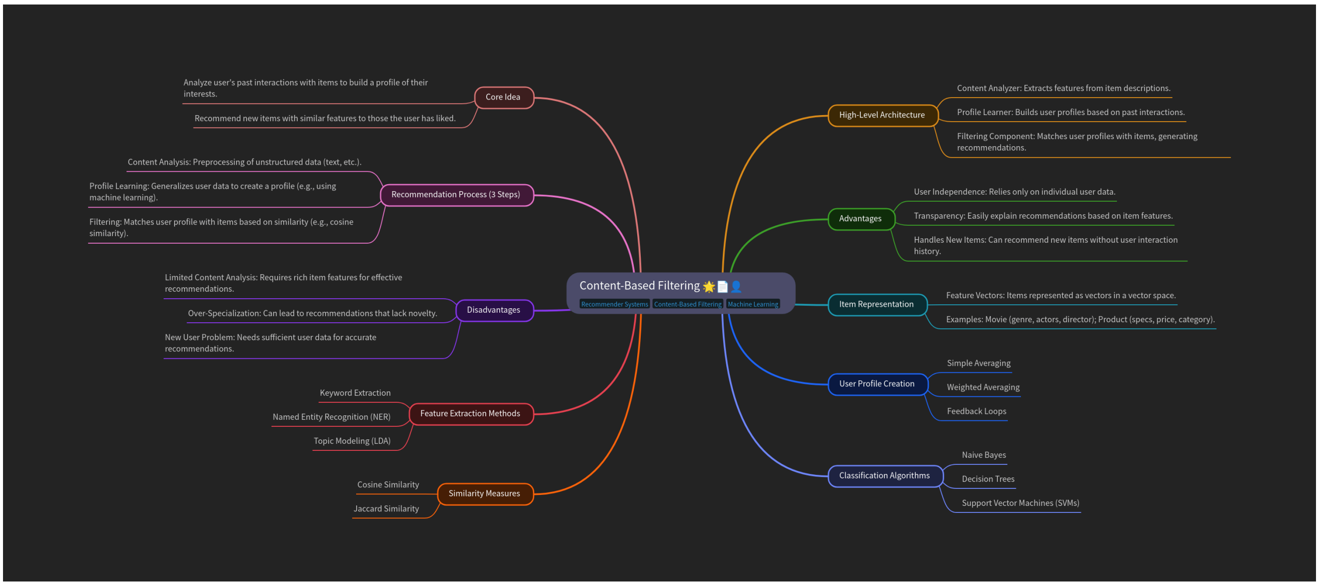
Common Algorithms:

1. **Naive Bayes**: A probabilistic classifier that calculates the likelihood of a user liking an item based on its features.
2. **Decision Trees**: A tree-based classifier that makes predictions based on decision rules derived from the item's attributes. A decision tree is a collection of nodes, arranged as a binary tree. The leaves render decisions; in our case, the decision would be "likes" or "doesn't like." Each interior node is a condition on the objects being classified; in our case the condition would be a predicate involving one or more features of an item. To classify an item, we start at the root, and apply the predicate at the root to the item. If the predicate is true, go to the left child, and if it is false, go to the right child. Then repeat the same process at the node visited, until a leaf is reached. That leaf classifies the item as liked or not.

3. **Support Vector Machines (SVMs)**: A powerful classifier that separates items into different categories by finding the optimal hyperplane between them.

Summary

Content-based filtering systems build **user profiles** based on the features of items a user has liked. These systems excel at recommending items similar to those a user has already enjoyed, though they face challenges such as **overspecialization** and **cold-start problems** for new users. By using techniques like **feature extraction**, **tagging**, and **similarity-based retrieval**, content-based recommenders provide personalized recommendations but require a rich set of item features to work effectively.


Content-Based Filtering 🌟📄👤
Recommender Systems
Content-Based Filtering
Machine Learning
Core Idea
Analyze user's past interactions with items to build a profile of their interests.
Recommend new items with similar features to those the user has liked.
Recommendation Process (3 Steps)
Content Analysis: Preprocessing of unstructured data (text, etc.).
Profile Learning: Generalizes user data to create a profile (e.g., using machine learning).
Filtering: Matches user profile with items based on similarity (e.g., cosine similarity).
Disadvantages
Limited Content Analysis: Requires rich item features for effective recommendations.
Over-Specialization: Can lead to recommendations that lack novelty.
New User Problem: Needs sufficient user data for accurate recommendations.
Feature Extraction Methods
Keyword Extraction
Named Entity Recognition (NER)
Topic Modeling (LDA)
Similarity Measures
Cosine Similarity
Jaccard Similarity
High-Level Architecture
Content Analyzer: Extracts features from item descriptions.
Profile Learner: Builds user profiles based on past interactions.
Filtering Component: Matches user profiles with items, generating recommendations.
Advantages
User Independence: Relies only on individual user data.
Transparency: Easily explain recommendations based on item features.
Handles New Items: Can recommend new items without user interaction history.
Item Representation
Feature Vectors: Items represented as vectors in a vector space.
Examples: Movie (genre, actors, director); Product (specs, price, category).
User Profile Creation
Simple Averaging
Weighted Averaging
Feedback Loops
Classification Algorithms
Naive Bayes
Decision Trees
Support Vector Machines (SVMs)