

Ensembled- Based and Hybrid Recommendation System

revision.....

Collaborative methods use the ratings of a community of users in order to make recommendations, whereas **content-based methods** use the ratings of a single user in conjunction with attribute-centric item descriptions to make recommendations.

Knowledge-based methods require the explicit specification of user requirements to make recommendations, and they do not require any historical ratings at all. Therefore, these methods use different sources of data, and they have different strengths and weaknesses. [Persistent personalization remembers your preferences over time to deliver constantly relevant recommendations.](#)

For example, **knowledge-based systems can address cold-start issues** much better than either content-based or collaborative systems because they do not require ratings. On the other hand, they are weaker than content-based and collaborative systems in terms of using persistent personalization from historical data. If a different user enters the same requirements and data in a knowledge based interactive interface, she might obtain exactly the same result.

Ensembled- Based and Hybrid Recommendation System

All these models seem rather restrictive in isolation, especially when multiple sources of data are available. In general, one would like to make use of all the knowledge available in different data sources and also use the algorithmic power of various recommender systems to make robust inferences. Hybrid recommender systems have been designed to explore these possibilities. There are three primary ways of creating hybrid recommender systems:

1. Ensemble design

Individual recommendation system models can be limiting, especially when you have a rich dataset with various user and item information. Hybrid recommender systems address this by combining the strengths of different approaches to provide more robust and accurate recommendations.

2. Monolithic design

Choosing Between Ensemble and Monolithic Designs:

3. Mixed systems

Data Availability: If you have a rich dataset with diverse data sources, a monolithic design might be a good choice to exploit the potential for deeper learning.

Development Resources: Ensemble methods are generally easier to implement and maintain, especially if you're starting with a new recommender system.

Need for Interpretability: If it's crucial to understand why specific recommendations are made, ensemble methods with simpler underlying models might be preferable.

Ensemble design

This approach combines the predictions from multiple recommendation system models. Imagine you have a content-based system recommending books based on genre and a collaborative filtering system recommending books based on similar user preferences. In ensemble design, you'd take the recommendations from both models and use a technique like weighted averaging to create a final list. This leverages the strengths of each model for a more comprehensive recommendation.

1. Ensemble design: In this design, results from off-the-shelf algorithms are combined into a single and more robust output. For example, one might combine the rating outputs from a content-based and a collaborative recommender into a single output. A significant variation exists in terms of the specific methodologies used for the combination process. The basic principle at work is not very different from the design of ensemble methods in many data mining applications such as clustering, classification, and outlier analysis

Ensemble design can be formalized as follows.

Let \hat{R}^k be an $m \times n$ matrix containing the predictions of the m users for the n items by the k th algorithm, where $k \in \{1 \dots q\}$.

Therefore, a total of q different algorithms are used to arrive at these predictions. The (u, j) th entry of \hat{R}^k contains the predicted rating of user u for item j by the k th algorithm.

Note that the observed entries of the original ratings matrix R are replicated in each \hat{R}^k , and only the unobserved entries of R vary in different \hat{R}^k because of the different predictions of different algorithms. The final result of the algorithm is obtained by combining the predictions $\hat{R}^1 \dots \hat{R}^q$ into a single output. This combination can be performed in various ways, such as the computation of the weighted average of the various predictions. Furthermore, in some sequential ensemble algorithms, the prediction \hat{R}^k may depend on the results of the previous component \hat{R}^{k-1} . In yet other cases, the outputs may not be directly combined. Rather, the output of one system is used as an input to the next as a set of content features. The common characteristics of all these systems are that (a) they use existing recommenders in off-the-shelf fashion, and (b) they produce a unified score or ranking.

Monolithic Design: This approach builds a single, unified model that incorporates different data sources within its architecture. For instance, you could create a model that considers user-item interactions (collaborative filtering) alongside item features (content-based) during the training process. This allows the model to learn the relationships between these factors simultaneously.

Monolithic design

A clear distinction may sometimes not exist between the various parts (e.g., content and collaborative) of the algorithm. In other cases, existing collaborative or content-based recommendation algorithms may need to be modified to be used within the overall approach, even when there are clear distinctions between the content-based and collaborative stages. Therefore, this approach tends to integrate the various data sources more tightly, and one cannot easily view individual components as off-the-shelf black-boxes.

monolithic design in recommender systems. You're right, it creates a more tightly integrated approach compared to ensemble methods. Here are some additional points to consider:

Advantages of Monolithic Design:

Deeper Learning: By feeding various data sources (user interactions, item features, context) into a single model, it can potentially learn more complex relationships between the factors. This can lead to more accurate and personalized recommendations.

Flexibility in Data Sources: Monolithic designs are adaptable to incorporating new data sources in the future without needing major architectural changes. As long as the data is relevant, the model can be retrained to leverage it.

Disadvantages of Monolithic Design:

Complexity: Building and maintaining a monolithic model can be more challenging compared to using off-the-shelf algorithms in ensemble methods. The model needs to be carefully designed to handle different data types effectively.

Interpretability: Understanding how a monolithic model arrives at its recommendations can be difficult. The complex interactions within the model make it less transparent compared to simpler, individual models in ensemble designs.

Mixed recommender systems share some similarities with ensembles but with a key difference in presentation. Here's a breakdown of mixed systems:

Modular Approach: Similar to ensembles, mixed systems utilize multiple recommendation algorithms as independent components.

Mixed systems

Black-Box Treatment: Each algorithm acts as a "black box," meaning you focus on the output (recommendations) rather than the internal workings of the model.

Like ensembles, these systems use multiple recommendation algorithms as black-boxes, but the items recommended by the various systems are presented together side by side

Side-by-Side Presentation: Unlike ensembles where the outputs are combined, mixed systems present recommendations from each algorithm separately. This can be in the form of separate lists, sections within the interface, or different weights (e.g., prominence) given to recommendations from various algorithms.

Benefits of Mixed Systems:

Flexibility and Customization: You can easily combine different types of algorithms to cater to specific user needs or item characteristics.

Transparency: Users can see the recommendations from different perspectives (content-based, collaborative, etc.) and potentially make more informed choices.

Simplicity: Mixed systems are often easier to implement and maintain compared to monolithic designs.

Challenges of Mixed Systems:

Presentation Overload: Presenting too many separate recommendations can overwhelm users and make it difficult to choose. Careful design and prioritization are crucial.

Redundancy: There might be overlap in recommendations from different algorithms, reducing the perceived value of the overall list. Strategies to filter or de-duplicate recommendations can be helpful.

Examples of Mixed Systems:

A music streaming service might use a content-based system to recommend similar songs based on genre and a collaborative filtering system to recommend songs popular among users with similar tastes. Both sets of recommendations would be displayed for the user to explore.

A news recommendation system could use a content-based system to suggest articles based on a user's reading history and a collaborative filtering system to recommend articles popular among users with similar interests. The user would see both content and socially relevant articles.

mixed systems offer a versatile approach to recommender systems, leveraging the strengths of different algorithms while maintaining a level of transparency for users.

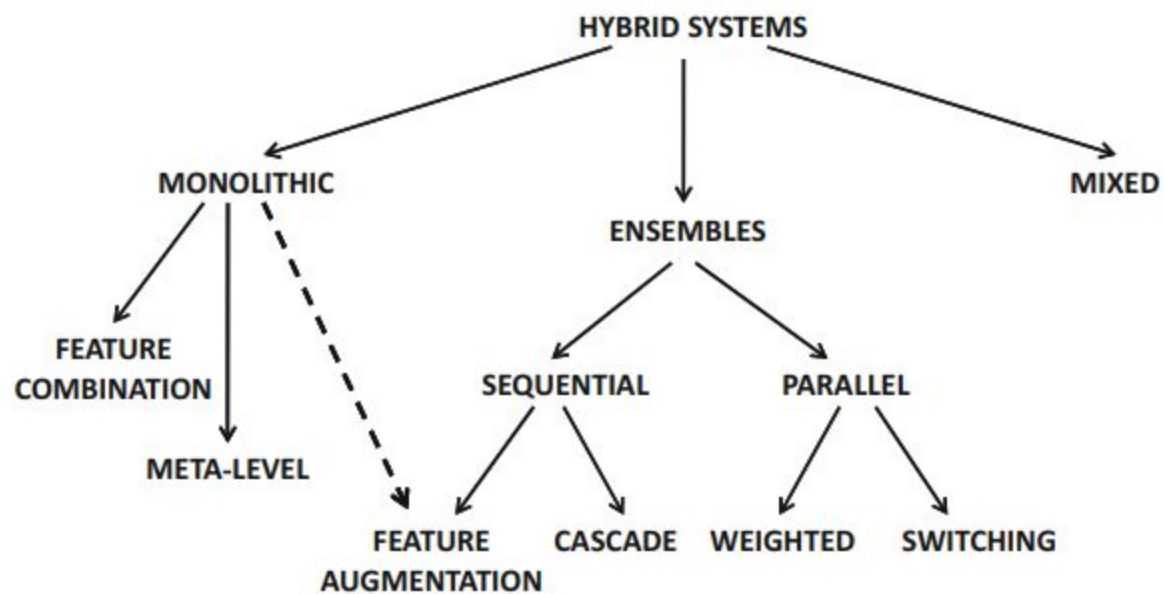


Figure 6.1: The taxonomy of hybrid systems

1. **Weighted:** In this case, the scores of several recommender systems are combined into a single unified score by computing the weighted aggregates of the scores from individual ensemble components. The methodology for weighting the components may be heuristic, or it might use formal statistical models.
2. **Switching:** The algorithm switches between various recommender systems depending on current needs. For example, in earlier phases, one might use a knowledge-based recommender system to avoid cold-start issues. In later phases, when more ratings are available, one might use a content-based or collaborative recommender. Alternatively, the system might adaptively select the specific recommender that provides the most accurate recommendation at a given point in time

Imagine a recommendation relay race. The first system picks a bunch of items, the second system refines that list based on the user, giving you the most relevant recommendations. In boosting (a complex type of cascade), each runner trains based on how the previous runner did, getting better as a team.

3. Cascade: In this case, one recommender system refines the recommendations given by another. In generalized forms of cascades, such as boosting, the training process of one recommender system is biased by the output of the previous one, and the overall results are combined into a single output.

4. Feature augmentation: The output of one recommender system is used to create input features for the next. While the cascade hybrid successively refines the recommendations of the previous system, the feature augmentation approach treats them as features as input for the next system

Feature Augmentation: Imagine adding features (extra information) based on one system's output for the next system to consider. It's like giving the next runner in the race additional clues to make better decisions. The focus is on enriching the data used for recommendations.

Feature Augmentation: A data scientist and a chef. The data scientist analyzes past customer behavior and creates recommendations for dishes (features). The chef uses these recommendations alongside their expertise to create the perfect meal.

Cascade: Refines recommendations through stages.

Feature Augmentation: Enriches data for recommendations using one system's output as features for another.

Feature Combination: This approach combines the features (data points) from different sources into a single dataset. This enriched dataset is then used to train a single recommender system. The model learns the relationships between all the features (user interactions, item characteristics, etc.) to generate recommendations.

Feature Combination: All ingredients are thrown into a single pot, and one chef creates a dish based on everything available.

5. Feature combination: In this case, the features from different data sources are combined and used in the context of a single recommender system. This approach can be viewed as a monolithic system, and therefore it is not an ensemble method.

Combines content data with collaborative filtering for better predictions.

Modifies collaborative filtering, making it a monolithic system.

Collaboration via Content: Leverages both content and ratings for improved recommendations.

6. Meta level: The collaborative system is modified to use the content features to determine peer groups. Then, the ratings matrix is used in conjunction with this peer group to make predictions. Note that this approach needs to modify the collaborative system to use a content matrix for finding peer groups, although the final predictions are still performed with the ratings matrix. Therefore, the collaborative system needs to be modified, and one cannot use it in an off-the-shelf fashion. This makes the meta-level approach a monolithic system rather than an ensemble system. Some of these methods are also referred to as “collaboration via content” because of the way in which they combine collaborative and content information

Focus on Composite Entities: This approach is often used when recommending sets of related items, like outfit recommendations or movie bundles.

Black-Box Treatment: Similar to ensembles, mixed systems treat individual recommender systems as "black boxes," focusing on their outputs rather than internal workings.

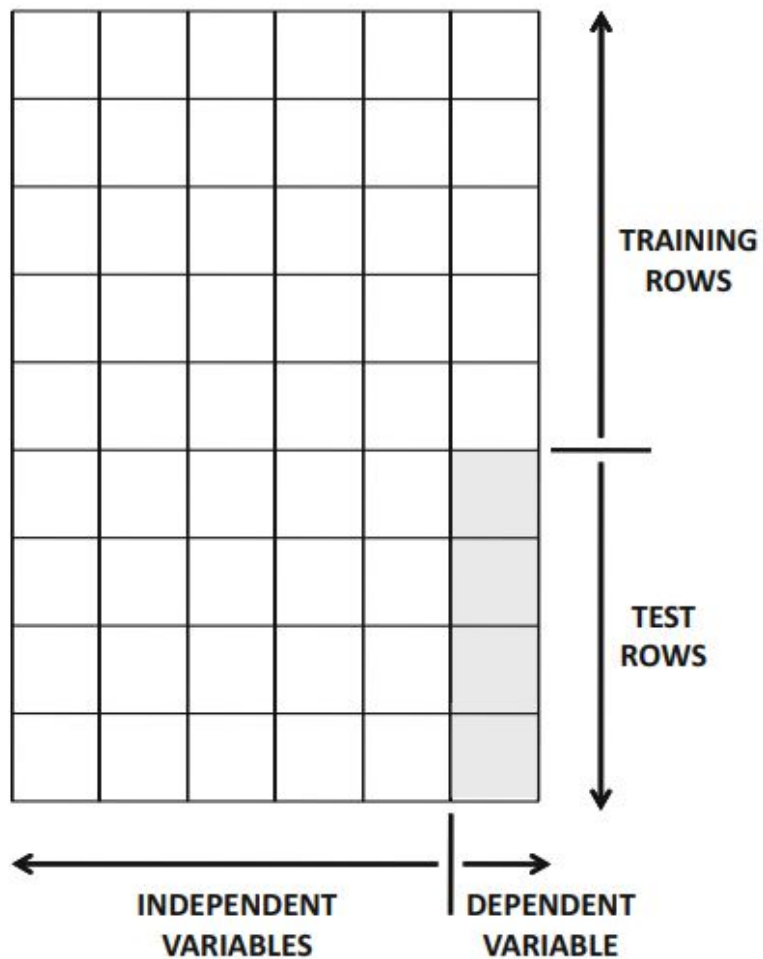
Neither Ensemble nor Monolithic: Mixed systems differ from ensembles (combining predictions) and monolithic systems (combining features within a single model). They present recommendations independently.

Mixed: It does not explicitly combine the scores . Furthermore, this approach is often used when the recommendation is a composite entity in which multiple items can be recommended as a related set. On the one hand, it does use other recommenders as black-boxes (like ensembles), but it does not combine the predicted ratings of the same item from different recommenders. Therefore, mixed recommenders cannot be viewed either as monolithic or ensemble based methods and are classified into a distinct category of their own. The approach is most relevant in complex item domains, and it is often used in conjunction with knowledge-based recommender systems

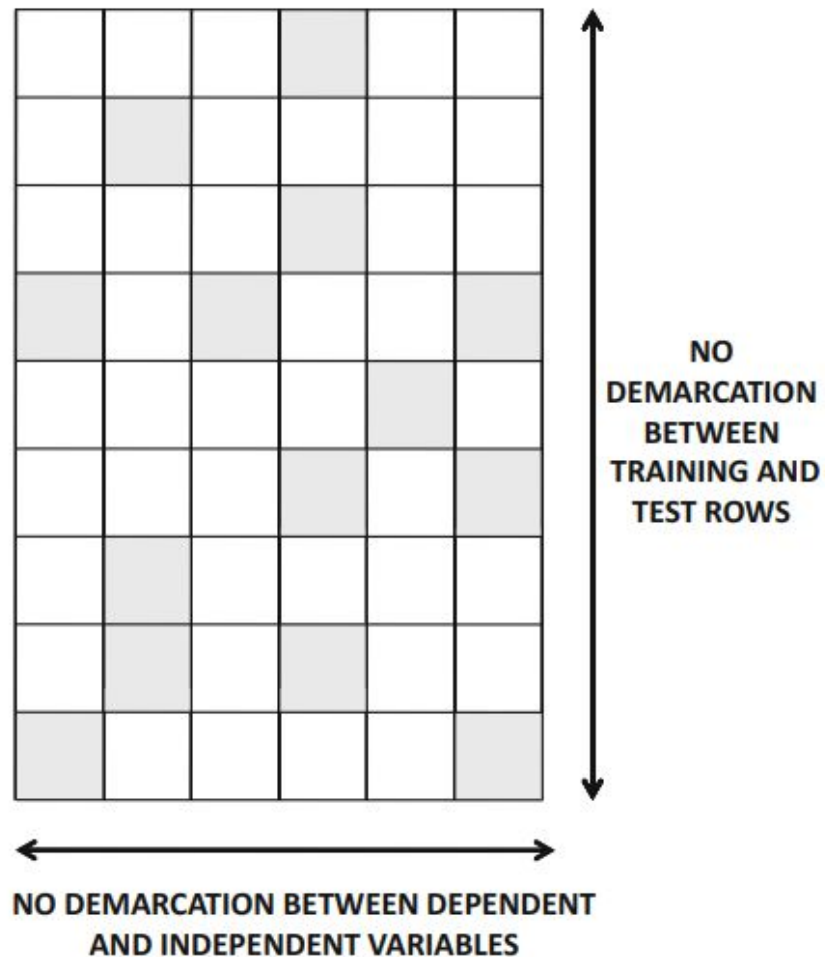
Complex Domains: Mixed systems are particularly relevant for complex item domains where showcasing diverse recommendation perspectives can be beneficial.

Knowledge-Based Integration: Mixed systems can be used alongside knowledge-based recommender systems, leveraging additional domain knowledge for richer recommendations.

In essence, mixed systems offer a versatile approach for scenarios where you want to present recommendations from different angles, catering to diverse user preferences or complex item relationships.



(a) Classification



(b) Collaborative filtering

Ensemble Methods from the Classification Perspective

Ensemble methods are commonly used in the field of data classification to improve the robustness of learning algorithms. As we will discuss below, much of this theory also applies to various forms of recommender systems.

For example, content-based recommender systems are often straightforward applications of text classification algorithms. Therefore, a direct application of existing ensemble methods in classification is usually sufficient to obtain high-quality results.

Repeated experiments have shown that combining multiple collaborative recommender systems often leads to more accurate results

This is because the bias-variance theory, which is designed for classification, also applies to the collaborative filtering scenario.

Ensemble methods are powerful tools in both classification and recommender systems, and there's significant overlap in their application.

Here's a breakdown of the key points:

Ensemble Methods for Classification:

Boosting Robustness: Ensemble methods combine the predictions from multiple classifiers to achieve better overall accuracy and reduce the impact of errors in any single model. This leads to more robust learning algorithms.

Content-Based Recommenders as Classifiers: Content-based recommender systems can often be viewed as a form of classification. They categorize items based on features like genre, theme, or description, similar to how a classifier might categorize emails as spam or not spam. So, directly applying ensemble methods from classification to content-based recommenders can be quite effective.

Ensemble Methods in Collaborative Filtering:

The good news is that the benefits of ensemble methods extend to collaborative filtering as well:

Improved Accuracy: Experiments have shown that combining predictions from multiple collaborative filtering systems can lead to more accurate recommendations. This aligns with the bias-variance trade-off principle from classification, where ensemble methods often help reduce variance (overfitting) and improve generalization.

Bias-Variance Theory Applies: The bias-variance theory, which explains the relationship between model complexity, variance, and bias, applies to both classification and collaborative filtering. Ensemble methods can help achieve a better balance between these factors in collaborative filtering, leading to more accurate recommendations.

In essence, ensemble methods are a valuable tool for both classification and recommender systems. By combining the strengths of multiple models, you can achieve more robust and accurate predictions, regardless of whether you're dealing with classifying data or recommending items.

Variance refers to the sensitivity of a machine learning model to the specific training data it is trained on.

Imagine you are training a model to predict if someone will buy a certain product based on their age and income. If you train the model on data from a wealthy neighborhood, the model might learn that people with a high income are more likely to buy the product. However, this might not be true for the general population. This is because the model has high variance - it's too sensitive to the specific training data it was shown.

1. **Bias:** Every classifier makes its own modeling assumptions about the nature of the decision boundary between classes. For example, a linear SVM classifier assumes that the two classes may be separated by a linear decision boundary. This is, of course, not true in practice. In other words, any given linear support vector machine will have an inherent *bias*. When a classifier has high bias, it will make *consistently incorrect* predictions over particular choices of test instances near the incorrectly modeled decision-boundary, even when different samples of the training data are used for the learning process.
2. **Variance:** Random variations in the choices of the training data will lead to different models. As a result, the dependent variable for a test instance might be inconsistently predicted by different choices of training data sets. Model variance is closely related to overfitting. When a classifier has an overfitting tendency, it will make *inconsistent* predictions for the same test instance over different training data sets.
3. **Noise:** The noise refers to the intrinsic errors in the target class labeling. Because this is an intrinsic aspect of data quality, there is little that one can do to correct it. Therefore, the focus of ensemble analysis is generally on reducing bias and variance.

The expected mean-squared error of a classifier over a set of test instances can be shown to be sum of the bias, variance, and noise. This relationship can be stated as follows:

$$\text{Error} = \text{Bias}^2 + \text{Variance} + \text{Noise} \quad (6.1)$$

Reducing Bias or Variance Improves Overall Error: Just like in regular machine learning, reducing either bias or variance in a CF model leads to a lower overall prediction error.

Ensemble Methods to the Rescue: The passage mentions two types of ensemble methods:

Bagging: This technique reduces variance by training multiple models on different subsets of the data and then averaging their predictions.

Boosting: This technique focuses on reducing bias by sequentially training models, each focusing on the errors made by the previous ones.

CF and Missing Entries: Unlike classification problems where missing entries are usually limited to the target variable (class label), CF can have missing entries in any user-item interaction data.

Bias-Variance Trade-Off Still Applies: Despite the difference in missing entries, the core idea of bias-variance trade-off remains relevant in CF. You can still optimize models to minimize overall error by addressing these components.

Ensemble Methods Apply to CF Too: The good news is that the principles behind ensemble methods used for classification can be adapted for CF tasks.

It is noteworthy that by reducing either the bias or variance components, one can reduce the overall error of a classifier. For example, classification ensemble methods such as bagging reduce the variance, whereas methods such as boosting can reduce the bias.

It is noteworthy that the only difference between classification and collaborative filtering is that missing entries can occur in any column rather than only in the class variable.

Nevertheless, the bias-variance result still holds when applied to the problem of predicting a specific column, whether the other columns are incompletely specified or not.

This means that the basic principles of ensemble analysis in classification are also valid for collaborative filtering. Indeed, as we will see later in this chapter, many classical ensemble methods in classification, such as bagging and boosting, have also been adapted to collaborative filtering.

Examples to Come: The passage hints that later sections will discuss how established ensemble methods like bagging and boosting are specifically used in collaborative filtering. In essence, the passage highlights that the concept of bias-variance trade-off is crucial for understanding and improving the performance of collaborative filtering models. By leveraging techniques like bagging and boosting, we can build more accurate CF systems that handle missing data effectively.

Weighted Hybrids

Let $R = [r_{uj}]$ be an $m \times n$ ratings matrix. In weighted hybrids, the outputs of various recommender systems are combined using a set of weights. Let $\hat{R}_1 \dots \hat{R}_q$ be the $m \times n$ *completely specified* ratings matrices, in which the unobserved entries of R are predicted by q different algorithms. Note that the entries r_{uj} that are already observed in the original $m \times n$ ratings matrix R are already fixed to their observed values in each prediction matrix \hat{R}_k . Then, for a set of weights $\alpha_1 \dots \alpha_q$, the weighted hybrid creates a combined prediction matrix $\hat{R} = [\hat{r}_{uj}]$ as follows:

$$\hat{R} = \sum_{i=1}^q \alpha_i \hat{R}_i \quad (6.2)$$

In the simplest case, it is possible to choose $\alpha_1 = \alpha_2 = \dots = \alpha_q = 1/q$. However, it is ideally desired to weight the various systems in a differential way, so as to give greater importance to the more accurate systems. A number of methods exist for such differential weighting. One can also write the aforementioned equation in terms of individual entries of the matrix:

$$\hat{r}_{uj} = \sum_{i=1}^q \alpha_i \hat{r}_{uj}^i \quad (6.3)$$

In order to determine the optimal weights, it is necessary to be able to evaluate the effectiveness of a particular combination of weights $\alpha_1 \dots \alpha_q$.

A simple approach is to hold out a small fraction (e.g., 25%) of the known entries in the $m \times n$ ratings matrix $R = [r_{uj}]$ and create the prediction matrices $\hat{R}_1 \dots \hat{R}_q$ by applying the q different base algorithms on the remaining 75% of the entries in R . The resulting predictions $\hat{R}_1 \dots \hat{R}_q$ are then combined to create the ensemble-based prediction \hat{R} according to Equation 6.2. Let the user-item indices (u, j) of these held-out entries be denoted by H . Then, for a given vector $\bar{\alpha} = (\alpha_1 \dots \alpha_q)$ of weights, the effectiveness of a particular scheme can be evaluated using either the mean-squared error (MSE) or the mean absolute error (MAE) of the predicted matrix $\hat{R} = [\hat{r}_{uj}]_{m \times n}$ over the held-out ratings in H :

$$MSE(\bar{\alpha}) = \frac{\sum_{(u,j) \in H} (\hat{r}_{uj} - r_{uj})^2}{|H|}$$

$$MAE(\bar{\alpha}) = \frac{\sum_{(u,j) \in H} |\hat{r}_{uj} - r_{uj}|}{|H|}$$

The linear regression approach is, however, sensitive to presence of noise and outliers.

This is because the squared error function is overly influenced by the largest errors in the data. A variety of robust regression methods are available, which are more resistant to the presence of noise and outliers. One such method uses the mean absolute error (MAE) as the objective function as opposed to the mean-squared error. The MAE is well known to be more robust to noise and outliers because it does not overemphasize large errors.



$$\frac{\partial MAE(\overline{\alpha})}{\partial \alpha_i} = \frac{\sum_{(u,j) \in H} \frac{\partial |(\hat{r}_{uj} - r_{uj})|}{\partial \alpha_i}}{|H|} \quad (6.4)$$

The value of \hat{r}_{uj} can be expanded using Equation 6.3, and the partial derivative may be simplified in terms of the ratings of individual ensemble components as follows:


$$\frac{\partial MAE(\overline{\alpha})}{\partial \alpha_i} = \frac{\sum_{(u,j) \in H} \text{sign}(\hat{r}_{uj} - r_{uj}) \hat{r}_{uj}^i}{|H|} \quad (6.5)$$

The gradient can be written in terms of the individual partial derivatives:

$$\overline{\nabla MAE} = \left(\frac{\partial MAE(\overline{\alpha})}{\partial \alpha_1} \dots \frac{\partial MAE(\overline{\alpha})}{\partial \alpha_q} \right)$$



This gradient is then used to descend through the parameter space α with an iterative gradient descent approach as follows:

1. Initialize $\alpha(0) = (1/q \dots 1/q)$ and $t = 0$.
 2. Iterative Step 1: Update $\alpha(t+1) \leftarrow \alpha(t) - \gamma \cdot \nabla \text{MAE}$. The value of $\gamma > 0$ can be determined using a line search so that the maximum improvement in MAE is achieved.
 3. Iterative Step 2: Update the iteration index as $t \leftarrow t + 1$.
 4. Iterative Step 3 (convergence check): If MAE has improved by at least a minimum amount since the last iteration, then go to iterative step 1.
 5. Report $\alpha(t)$
- 

Regularization can be added to prevent overfitting. It is also possible to add other constraints on the various values of α_i such as non-negativity or ensuring that they sum to 1.

Such natural constraints improve generalizability to unseen entries. The gradient descent equations can be modified relatively easily to respect these constraints. After the optimal

weights have been determined, all ensemble models are retrained on the entire ratings matrix without any held-out entries. The predictions of these models are combined with the use of the weight vector discovered by the iterative approach.

Various Types of Model Combinations

1. Homogeneous data type and model classes: In this case, different models are applied on the same data. For example, one might apply various collaborative filtering engines such as neighborhood-based methods, SVD, and Bayes techniques on a ratings matrix. The results are then aggregated into a single predicted value.
2. Heterogeneous data type and model classes: In this cases, different classes of models are applied to different data sources. For example, one component of the model might be a collaborative recommender that uses a ratings matrix, whereas another component of the model might be a content-based recommender. This approach essentially fuses the power of multiple data sources into the combination process.

Adapting Bagging from Classification

bias-variance trade-off - common weighted combination techniques used in the classification problem is that of bagging

the bagging approach needs to be slightly modified in order to adjust for the fact that the collaborative filtering problem is formulated somewhat differently from that of classification

The basic idea in bagging is to reduce the variance component of the error in classification. In bagging, q training data sets are created with bootstrapped sampling. In bootstrapped sampling, rows of the data matrix are sampled with replacement in order to create a new training data set of the same size as the original training data set

A total of q training models are created with each of the sampled training data sets. For a given test instance, the average prediction from these q models is reported. Bagging generally improves the classification accuracy because it reduces the variance component of the error. A particular variant of bagging, known as subbagging, subsamples the rows, rather than sampling with replacement. For example, one can simply use all the distinct rows in a bootstrapped sample for training the models. The bagging and subbagging methods can be generalized to collaborative filtering as follows:

subbagging: Creates multiple training datasets by subsampling the original data (without replacement). Models are trained on these smaller datasets

1. Row-wise bootstrapping: In this case, the rows of the ratings matrix R are sampled with replacement to create a new ratings matrix of the same dimensions. A total of q such ratings matrices $R_1 \dots R_q$ are thus created. An existing collaborative filtering algorithm (e.g., latent factor model) is then applied to each of the q training data sets. For each training data set, an item rating can be predicted for a user only if that user is represented at least once in the matrix.

The predicted rating is then averaged over all the ensemble components in which that user is present. Note that for reasonably large values of q , each user will typically be present in at least one ensemble component with a high probability value of $1 - (1/e)^q$. Therefore, all users will be represented with high probability.

2. Row-wise subsampling: This approach is similar to row-wise bootstrapping, except that the rows are sampled without replacement. The fraction f of rows sampled is chosen randomly from $(0.1, 0.5)$. The number of ensemble components q should be significantly greater than 10 to ensure that all rows are represented. The main problem with this approach is that it is difficult to predict all the entries in this setting, and therefore one has to average over a smaller number of components. Therefore, the benefits of variance reduction are not fully achieved.

Entry-wise bagging samples individual ratings (with replacement) from the rating matrix, creating weighted datasets for training. It requires compatible collaborative filtering algorithms (e.g., weighted matrix factorization) that can handle these weights. This approach helps reduce variance and potentially improve recommendation accuracy.

3. Entry-wise bagging: In this case, the entries of the original ratings matrix are sampled with replacement to create the q different ratings matrices $R_1 \dots R_q$. Because many entries may be sampled repeatedly, the entries are now associated with weights. Therefore, a base collaborative filtering algorithm is required that can handle entries with weights

Each data point (rating) can have a different weight assigned to it.

Weights are determined by the sampling process in entry-wise bagging.

Ratings that are sampled more frequently (with replacement) will have higher weights, indicating their increased influence during model training.

4. Entry-wise subsampling: In entry-wise subsampling, a fraction of the entries are retained at random from the ratings matrix R to create a sampled training data set. Typically, a value of f is sampled from $(0.1, 0.5)$, and then a fraction f of the entries in the original ratings matrix are randomly chosen and retained. This approach is repeated to create q training data sets $R_1 \dots R_q$. Thus, each user and each item is represented in each subsampled matrix, but the number of specified entries in the subsampled matrix is smaller than that in the original training data. A collaborative filtering algorithm (e.g., latent factor model) is applied to each ratings matrix to create a predicted matrix. The final prediction is the simple average of these q different predictions.

In general, it is desirable to use uncorrelated base models with low bias and high variance in order to extract the maximum benefit from bagging. In cases where bagging does not work because of high correlations across base predictors, it may be helpful to explicitly use **randomness injection**.

Bagging thrives on diversity:

For bagging to work effectively, the models it combines (base models) should be different from each other. This is achieved by using models with low bias (good at capturing underlying pattern) and high variance (sensitive to changes in the training data).

Highly correlated models (all learn very similar things) won't offer much improvement when averaged in bagging.

Randomness injection to the rescue:

If you find bagging isn't working well, it might be due to high correlation among models.

To address this, you can introduce randomness during training to make the models learn more diverse patterns.

Think of it like this:

Imagine a group discussion. If everyone has similar ideas (low variance), averaging their opinions won't give you much new perspective.

But if you have people with different backgrounds and viewpoints (high variance), averaging their insights can lead to a richer understanding. Randomness injection helps create this diversity in bagging.

Randomness Injection

Randomness injection is an approach that shares many principles of random forests in classification [22]. The basic idea is to take a base classifier and explicitly inject randomness into the classifier. Various methods can be used for injecting the randomness.





Randomness injection in matrix factorization and why bagging doesn't work well for factorized neighborhood models:

Randomness in Matrix Factorization:

Matrix factorization is indeed inherently random due to two key aspects:

Random Initialization: The factor matrices (representing user and item latent factors) are initialized with random values. This starting point guides the model's learning process but isn't the only solution.

Gradient Descent: The model uses gradient descent to find an optimal solution by iteratively adjusting the factors. This process is not deterministic (always reaching the same point) and can lead to slightly different solutions depending on the initial random values.

These elements introduce randomness, making different runs of the model potentially find slightly different, but hopefully good, solutions.

2. Injecting randomness into a matrix factorization model: Matrix factorization methods are inherently randomized methods because they perform gradient descent over the solution space after randomly initializing the factor matrices. Therefore, by choosing different initializations, different solutions are often obtained. The combinations of these different solutions often provide more accurate results.

It is noteworthy that the bagging approach does not work very well in the case of the factorized neighborhood model.

bagging isn't very effective due to the similarity of solutions found with different initializations.

Switching Hybrids

Switching hybrids are used most commonly in recommender systems in the context of the problem of model selection, but they are often not formally recognized as hybrid systems. The original motivation for switching systems [117] was to handle the cold-start problem, where a particular model works better in earlier stages when there is a paucity of available data. However, in later stages, a different model is more effective, and therefore one switches to the more effective model.

1. Switching Mechanisms for Cold-Start Issues

One might use a knowledge-based recommender, when few ratings are available because knowledge-based recommender systems can function without any ratings, and they are dependent on user specifications of their needs. However, as more ratings become available, one might switch to a collaborative recommender. One can also combine content based and collaborative recommenders in this way, because content-based recommenders can work well for new items, whereas collaborative recommenders cannot effectively give recommendations for new items

Switching Mechanisms for Cold-Start Issues examples

Eg : First, a nearest neighbor content classifier is used, followed by a collaborative system, and finally a naive Bayes content classifier is used to match with the long-term profile. This approach does not fully address the cold-start problem because all the underlying learners need some amount of data.

Eg : Another work [659] combines hybrid versions of collaborative and knowledge-based systems. The knowledge-based system provides more accurate results during the cold-start phase, whereas the collaborative system provides more accurate results in later stages. Incorporating knowledge-based systems is generally more desirable for handling the coldstart problem.

2. Bucket-of-Models

The held-out entries are then used to evaluate the effectiveness of the model in terms of a standard measure, such as the MSE or the MAE. The model that yields the lowest MSE or MAE is used as the relevant one. This approach is also commonly used for parameter tuning. For example, each model may correspond to a different value of the parameter of the algorithm, and the value providing the best result is selected as the relevant one. Once the relevant model has been selected, it is retrained on the entire ratings matrix, and the results are reported. Instead of using a hold-out approach, a different technique known as cross-validation is also used

Bucket-of-Models Approach

Train and evaluate multiple collaborative filtering models on the same dataset.

Choose the model with the lowest error (e.g., MSE) on a held-out set.

Useful for model selection and parameter tuning.

Alternative: Cross-Validation

Splits data into folds for more robust evaluation compared to a single held-out set.

Key Points:

Explore different models/parameter settings.

Select best performer based on error metrics.

Consider cross-validation for a more robust assessment.

Cascade Hybrids

Here, we take a broader view of cascade hybrids in which a recommender is allowed to use recommendations of the previous recommender in any way (beyond just direct refinement), and then combine the results to make the final recommendation. This broader definition encompasses larger classes of important hybrids, such as boosting, which would not otherwise be included in any of the categories of hybrids. Correspondingly, we define two different categories of cascade recommenders.

This broader view allows cascade hybrids to utilize the recommendations from previous stages in any way for the final output.
This enables richer interactions and opens doors to powerful techniques like boosting.

1 Successive Refinement of Recommendations

In this approach, a recommender system successively refines the output of recommendations from the previous iteration. For example, the first recommender can provide a rough ranking and also eliminate many of the potential items. The second level of recommendation then uses this rough ranking to further refine it and break ties. The resulting ranking is presented to the user

The first knowledge-based recommender is clearly of higher priority because the second-level recommender cannot change the recommendations made at the first level. The other observation is that the second level recommender is much more efficient because it needs to focus only on the ties within each bucket. Therefore, the item-space of each application of the second recommender is much smaller.

2. Boosting

Boosting has been used popularly in the context of classification and regression . One of the earliest methods for boosting was the AdaBoost algorithm. The regression variant of this algorithm is referred to as AdaBoost.RT . The regression variant is more relevant to collaborative filtering because it is easier to treat ratings as numeric attributes. In traditional boosting, a sequence of training rounds is used with weighted training examples.

6.5.2.1 Weighted Base Models

The boosting and bagging methods require the use of weighted base models, in which entries are associated with weights. In this section, we show how existing collaborative filtering models can be modified so that they can work with weights.

Let us assume that the weight w_{uk} be associated with a particular entry in the ratings matrix for user u and item k . It is relatively straightforward to modify existing models to work with weights on the entries:

1. *Neighborhood-based algorithms:* The average rating of a user is computed in a weighted way for mean-centering the ratings. Both the Pearson and the cosine measures can be modified to take weights into account. Therefore, Equation 2.2 of Chapter 2 can be modified as follows to compute the Pearson coefficient between users u and v :

$$\text{Pearson}(u, v) = \frac{\sum_{k \in I_u \cap I_v} \max\{w_{uk}, w_{vk}\} \cdot (r_{uk} - \mu_u) \cdot (r_{vk} - \mu_v)}{\sqrt{\sum_{k \in I_u \cap I_v} w_{uk} (r_{uk} - \mu_u)^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} w_{vk} (r_{vk} - \mu_v)^2}} \quad (6.6)$$

The reader should refer to section 2.3 of Chapter 2 for the details of the notations. A different way⁴ of modifying the measure is as follows:

$$\text{Pearson}(u, v) = \frac{\sum_{k \in I_u \cap I_v} w_{uk} \cdot w_{vk} \cdot (r_{uk} - \mu_u) \cdot (r_{vk} - \mu_v)}{\sqrt{\sum_{k \in I_u \cap I_v} w_{uk}^2 (r_{uk} - \mu_u)^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} w_{vk}^2 (r_{vk} - \mu_v)^2}} \quad (6.7)$$

⁴The work in [67] proposes only the first technique for computing the similarity.

2. *Latent factor models*: Latent factor models are defined as optimization problems in which the sum of the squares of the errors of the specified entries are minimized. In this case, the *weighted* sum of the squares of the optimization problem must be minimized. Therefore, the objective function in section 3.6.4.2 of Chapter 3 can be modified as follows:

$$\text{Minimize } J = \frac{1}{2} \sum_{(i,j) \in S} w_{ij} e_{ij}^2 + \frac{\lambda}{2} \sum_{i=1}^m \sum_{s=1}^k u_{is}^2 + \frac{\lambda}{2} \sum_{j=1}^n \sum_{s=1}^k v_{js}^2 \quad (6.8)$$

Here, $U = [u_{ij}]$ and $V = [v_{ij}]$ are the $m \times k$ and $n \times k$ user-factor and item-factor matrices, respectively. Note the weights associated with the errors on the entries. The corresponding change in the gradient descent method is to weight the relevant updates:

$$\begin{aligned} u_{iq} &\Leftarrow u_{iq} + \alpha(w_{ij} \cdot e_{ij} \cdot v_{jq} - \lambda \cdot u_{iq}) \\ v_{jq} &\Leftarrow v_{jq} + \alpha(w_{ij} \cdot e_{ij} \cdot u_{iq} - \lambda \cdot v_{jq}) \end{aligned}$$

Many other base collaborative filtering algorithms can be modified to work with weighted entries. These types of weighted base algorithms are useful for many collaborative filtering ensembles, such as boosting and bagging.

Feature Augmentation Hybrids

The Libra system combines Amazon.com's recommender system with its own Bayes classifier. The approach uses the “related authors” and “related titles” that Amazon generates as features describing the items. Note that Amazon generates these recommendations with the use of a collaborative recommender system. These data are then used in conjunction with a content-based recommender to make the final predictions. Note that any off-the-shelf content-based system can be used in principle, and therefore the approach can be viewed as an ensemble system. The approach in opts for a naive Bayes text classifier. It was found through experiments that the features generated by Amazon's collaborative system were of high quality, and they contributed significantly to better quality recommendations.

Instead of using a collaborative system first, it is also possible to use the content-based system first. The basic idea is to use the content-based system to fill in the missing entries of the ratings matrix so that it is no longer sparse. Thus, the missing entries are estimated by the content-based system to create a denser ratings matrix. These newly added ratings are referred to as pseudo-ratings. Then, a collaborative recommender is used on the dense ratings matrix to make rating predictions. Finally, the collaborative prediction is combined.

Determining Weights for Pseudo-Ratings (Content-Based Filling):

Content-based filling: This approach uses a CBF system to estimate missing ratings (pseudo-ratings) in the CF matrix, creating a denser data set.

Weighting pseudo-ratings: The final recommendation might combine CF predictions with these pseudo-ratings. The weight assigned to a pseudo-rating reflects the system's confidence in the CBF prediction.

A heuristic function could consider factors like the number of user ratings ($|I_i|$), with higher numbers suggesting more confidence in the pseudo-rating.

Modifications to CF algorithms: This approach often requires modifications to existing CF algorithms to handle pseudo-ratings effectively.

How can such weights be determined? The weight of a pseudo-rating intuitively represents the algorithm's certainty in the prediction of the first phase, and it is an increasing function of the number of ratings $|I_i|$ of that user. A number of heuristic functions are used to weight various ratings, and the reader is referred to for details. Note that this approach requires modifications to the second phase of collaborative filtering, and off-the-shelf algorithms cannot be used. Such methods can be viewed as monolithic systems.

Advantages:

Improved User Profiling: CBF analysis adds valuable information to user profiles, leading to a more comprehensive understanding of user preferences.

Enhanced Similarity Computation: By considering both interactions and content aspects, user similarities become more accurate, reflecting deeper connections.

Meta-Level Hybrids

Potentially Better Recommendations: More accurate user profiles and similarities can translate into better-targeted recommendations.

Challenges:

Modifications to CF Algorithms: Integrating CBF often requires modifications to existing CF algorithms to handle the enriched user profiles.

before, one can robustly compute the similarities between users with this new representation. The main idea here is that the content-based peer group is used to determine the most similar users of the target user. Once the peer group has been determined, then the weighted average of the ratings of the peer group are used to determine the predicted ratings. Note that this approach does require a certain amount of change to the original collaborative recommender, at least in terms of how the similarity is computed.

Not True Ensemble Systems: Since CBF acts as a preprocessing step, these systems might not qualify as pure ensembles (combining multiple off-the-shelf models).

LaboUr System (Another Example):

An instance-based model analyzes user behavior to learn a content-based profile.

This profile is then compared with other users using collaborative approaches.

This combined analysis allows for user comparisons and prediction generation.

Furthermore, the first phase of the approach cannot use off-the-shelf content-based models in their entirety because it is mostly a feature selection (preprocessing) phase. Therefore, in many cases, these systems cannot be considered **true ensembles**, because they do not use existing methods as off-the-shelf recommenders.

Meta-Level vs. Feature Augmentation:

While both leverage CBF, they differ in approach:

Feature Augmentation: CBF recommendations serve as additional features for a final model.

Meta-Level Hybrids: CBF enriches user profiles before similarity computation in CF.

In essence, meta-level hybrids provide a valuable technique for enhancing collaborative filtering by harnessing the power of content-based analysis to improve user profiles and

ultimately deliver more accurate and relevant recommendations.

Meta-level hybrids utilize content-based filtering (CBF) as a preprocessing step to enhance user profiles for collaborative filtering (CF). Here's how it works:

CBF for User Profile Enrichment: A CBF system analyzes user behavior or item properties to create a more nuanced representation of user profiles.

Modified Similarity Computation: This enriched user profile is then used in a modified CF algorithm to compute user similarities more accurately. Similarity measures now consider both user interaction patterns and content-based insights.

Another example of a meta-level system was **LaboUr** in which an instance-based model is used to learn the content-based user's profile. The profiles are then compared using a collaborative approach. These models are compared across users to make predictions. Many of these methods fall within the category of “collaboration via content,” though that is not the only way in which such hybrids can be constructed.

Recommendation Generation: Based on the enhanced user similarities, CF predicts ratings for the target user.

Example:

Imagine a user who frequently reads science fiction books by female authors.

A CBF system might identify keywords like "space exploration" and "strong female characters" as relevant to this user's profile.

These content-based insights are then incorporated into the user profile used for CF.

CF can now find other users who share similar interests based on both explicit ratings and the enriched profile, leading to potentially more relevant recommendations.

feature combination hybrids work, as illustrated in the image:

Ratings Matrix: The system starts with a ratings matrix, where rows represent users and columns represent items. Each entry indicates a user's rating for an item (e.g., on a scale of 1 to 5 stars).

Feature Combination Hybrids

Feature Vectors: For both users and items, additional feature vectors are created. These vectors contain characteristics that go beyond just ratings. For users, this might include demographics or browsing history. For items, it could include genre, director, or product description keywords.

Another approach is to augment a ratings matrix and add columns for keywords in addition to items. Therefore, the ratings matrix becomes an $m \times (n + d)$ matrix, where n is the number of items and d is the number of keywords. The weights of “keyword items” are based on the weighted aggregation of the descriptions of the items accessed, bought, or rated by the user. A traditional neighborhood or matrix factorization approach can be used with this augmented matrix. The relative weights of the two types of columns can be learned through cross-validation (see Chapter 7). This type of combination of two optimization models is common in hybrid settings, where the objective function is set up as follows in terms of a parameter vector $\bar{\theta}$:

$$J = \text{CollaborativeObjective}(\bar{\theta}) + \beta \text{ContentObjective}(\bar{\theta}) + \text{Regularization} \quad (6.9)$$

Combining Features: The key idea is to augment the ratings matrix with these feature vectors. This can be done by adding new columns for user features and item features, resulting in a larger matrix.

Recommendation Algorithm: A machine learning algorithm is then applied to this augmented matrix. This algorithm can be a traditional matrix factorization approach used in collaborative filtering, but it can now also leverage the additional feature information to make predictions.

Regression and Matrix Factorization

Final Recommendations: Based on the analysis of the augmented matrix, the system recommends items to the target user. These recommendations consider both the user's past behavior (through ratings) and the characteristics of the items themselves (through features).

Meta-level Features

Id.	Description
1	Constant value of 1 (using only this feature amounts to using the global linear regression model of section 6.3)
2	A binary variable indicating whether the user rated more than 3 movies on this particular date
3	The log of the number of times a movie has been rated
4	The log of the number of distinct dates on which a user has rated movies
5	A Bayesian estimate of the mean rating of the movie after having subtracted out the user's Bayesian estimated mean
6	The log of the number of user ratings
16	The standard deviation of the user ratings
17	The standard deviation of the movie ratings
18	The log of (Rating Date – First User Rating Date +1)
19	The log of the number of user ratings on the date +1

Table 6.1: A subset of the meta-features used in [554] for ensemble combination on the Netflix Prize data set

Mixed Hybrids

The main characteristic of mixed recommender systems is that they combine the scores from different components in terms of presentation, rather than in terms of combining the predicted scores. In many cases, the recommended items are presented next to one another. Therefore, the main distinguishing characteristic of such systems is the combination of presentation rather than the combination of predicted scores.

Mixed hybrids utilize multiple recommendation techniques but don't directly combine their predicted scores.

Instead, they present recommendations side-by-side, showcasing the strengths of each approach.

Example:

Imagine a system that recommends movies. It might use:

A collaborative filtering (CF) model to suggest movies similar to those you've rated highly.

A content-based filtering (CBF) model to recommend movies based on your watch history (e.g., genre preferences).

In a mixed hybrid, the system would display two separate lists:

A list of CF recommendations (movies similar to your past favorites).

A list of CBF recommendations (movies based on genre preferences).

example of a mixed hybrid has been proposed in the tourism domain. In this case, bundles of recommendations are created, where each bundle contains multiple categories of items.

For example, in a tourism recommender system, the different categories may correspond to accommodations, leisure activities, air-tickets, and so on.

Tourists will typically buy bundles of these items from various categories in order to create their trips.

For each category, a different recommender system is employed.

The basic idea here is that the recommender system that is most appropriate for obtaining the best accommodations, may not be the one that is most appropriate for recommending tourism activities.

Therefore, each of these different aspects is treated as a different category for which a different recommender system is employed.

Furthermore, it is important to recommend bundles in which the items from multiple categories are not mutually inconsistent. [items from different categories can be combined meaningfully \(e.g., close activity to chosen hotel\).](#)

For example, if a tourist is recommended a leisure activity that is very far away from her place of accommodation, then the overall recommendation bundle will not be very convenient for the tourist.

Therefore, a knowledge base containing a set of domain constraints is incorporated for the bundling process. The constraints are designed to resolve inconsistencies in the product domain. A constraint satisfaction problem is employed to determine a mutually consistent bundle.

It is noteworthy that many of the mixed hybrids are often used in conjunction with knowledge-based recommender systems as one of the components . This is not a coincidence. Mixed hybrids are generally designed for complex product domains with multiple components like knowledge-based recommender systems.