

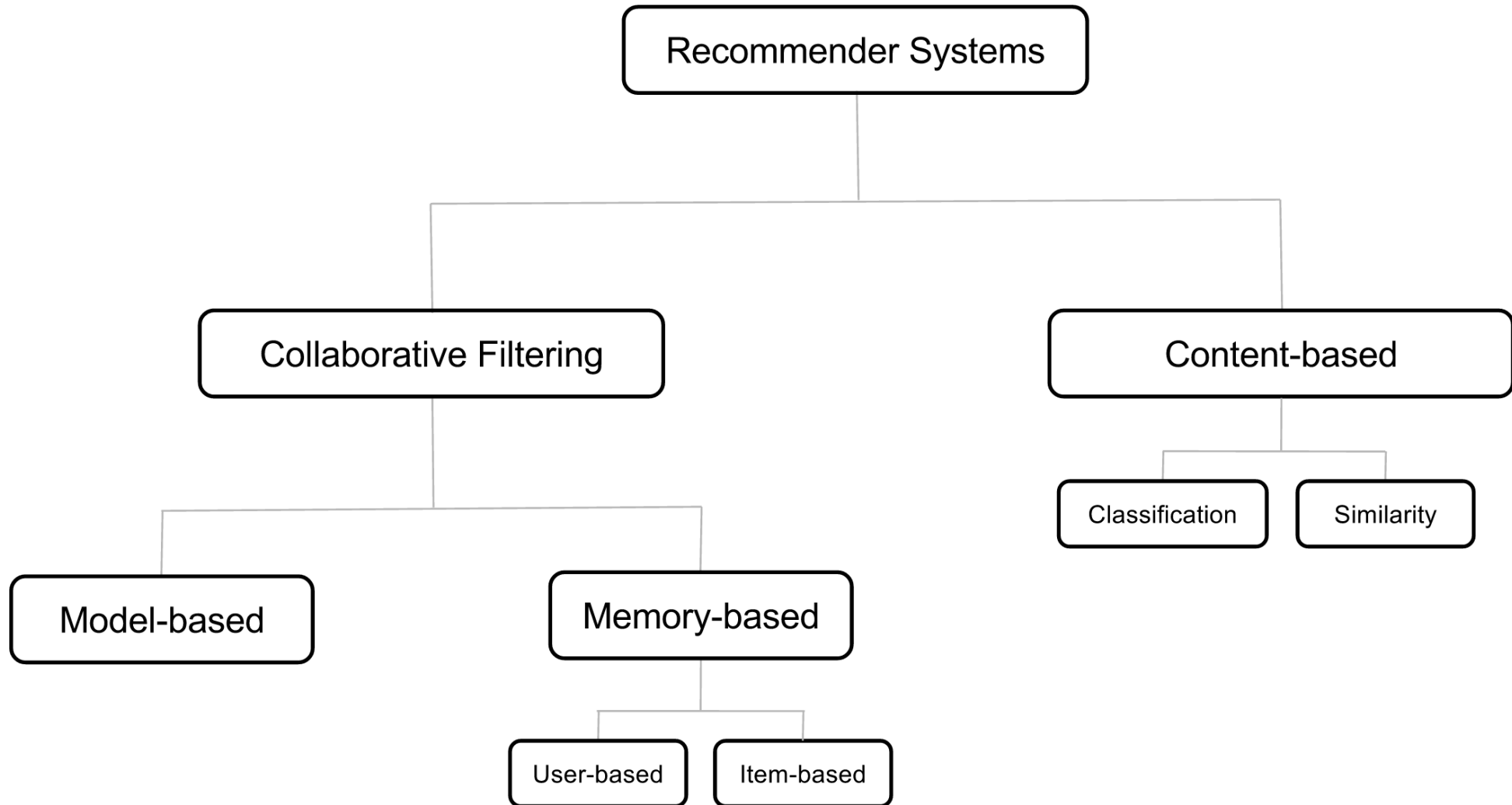
# Professional Matchmaking in R

Duncan Stoddard

[duncan@dsanalytics.co.uk](mailto:duncan@dsanalytics.co.uk)

[www.dsanalytics.co.uk](http://www.dsanalytics.co.uk)





# Fill in the gaps...

**advisors**

		ITEMS				
		a	b	c	d	e
USERS projects / professionals	1	1	0			
	2		1	1		1
	3	0				0
	4				0	
	5	1	1	1		1

# User-personalised recommendations

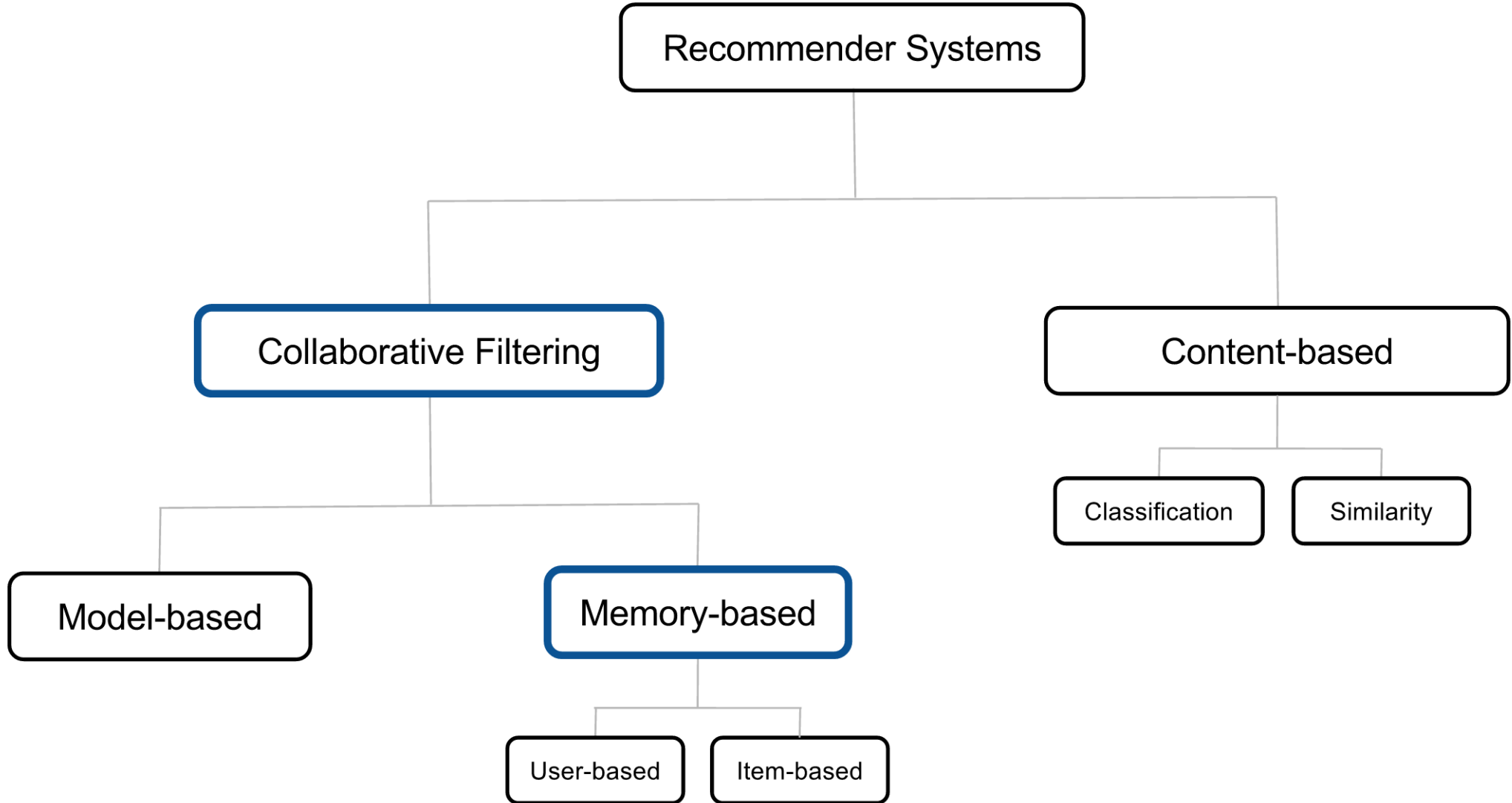


# Recommender evaluation

actual / predicted	negative	positive
negative	$a$	$b$
positive	$c$	$d$

$$\textit{Precision} = \frac{\textit{correctly recommended items}}{\textit{total recommended items}} = \frac{d}{b + d}$$

$$\textit{Recall} = \frac{\textit{correctly recommended items}}{\textit{total useful recommendations}} = \frac{d}{c + d}$$



# User-based

## 1. Get similarity scores

	ITEMS				
	a	b	c	d	e
1	1	0			
2		1	1		1
3	0				0
4				0	
5	1	1	1		1



# User-based

## 1. Get similarity scores

$$\text{sim}_{\text{Pearson}}(\vec{x}, \vec{y}) = \frac{\sum_{i \in I} (\vec{x}_i - \bar{\vec{x}})(\vec{y}_i - \bar{\vec{y}})}{(|I| - 1) \text{sd}(\vec{x}) \text{sd}(\vec{y})}$$

$$\text{sim}_{\text{Cosine}}(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \|\vec{y}\|},$$

# User-based

1. Get similarity scores
2. Predict blanks as weighted average of neighbours'

$$\hat{r}_{aj} = \frac{1}{\sum_{i \in \mathcal{N}(a)} s_{ai}} \sum_{i \in \mathcal{N}(a)} s_{ai} r_{ij}$$

s = sim

r = rating

$\mathcal{N}(a)$  = users in neighbourhood of active user,  $a$

# User-based

1. Get similarity scores
2. Predict blanks as weighted average of neighbours'
3. Recommend top n items not already used by user

## Problems:

- 'Cold-start': how do you predict prefs of new users?
- Computationally intensive

# Item-based

1. Get similarity/distance score between each item vector of users' preferences
2. For a given *active user*, create a weighted list of items similar to their top rated items

# Item-based

1. Get similarity/distance score between each item vector of users' preferences
2. For a given *active user*, create a weighted list of items similar to their top rated items

*E.g. Wedding Crashers = 5, Skyfall = 1*

$$\text{pred\_rating}(\text{marley \& me}) = (5 * \text{sim}(\text{wedding}, \text{marley}) + 1 * \text{sim}(\text{sky}, \text{marley})) / \text{sum}(\text{sims})$$

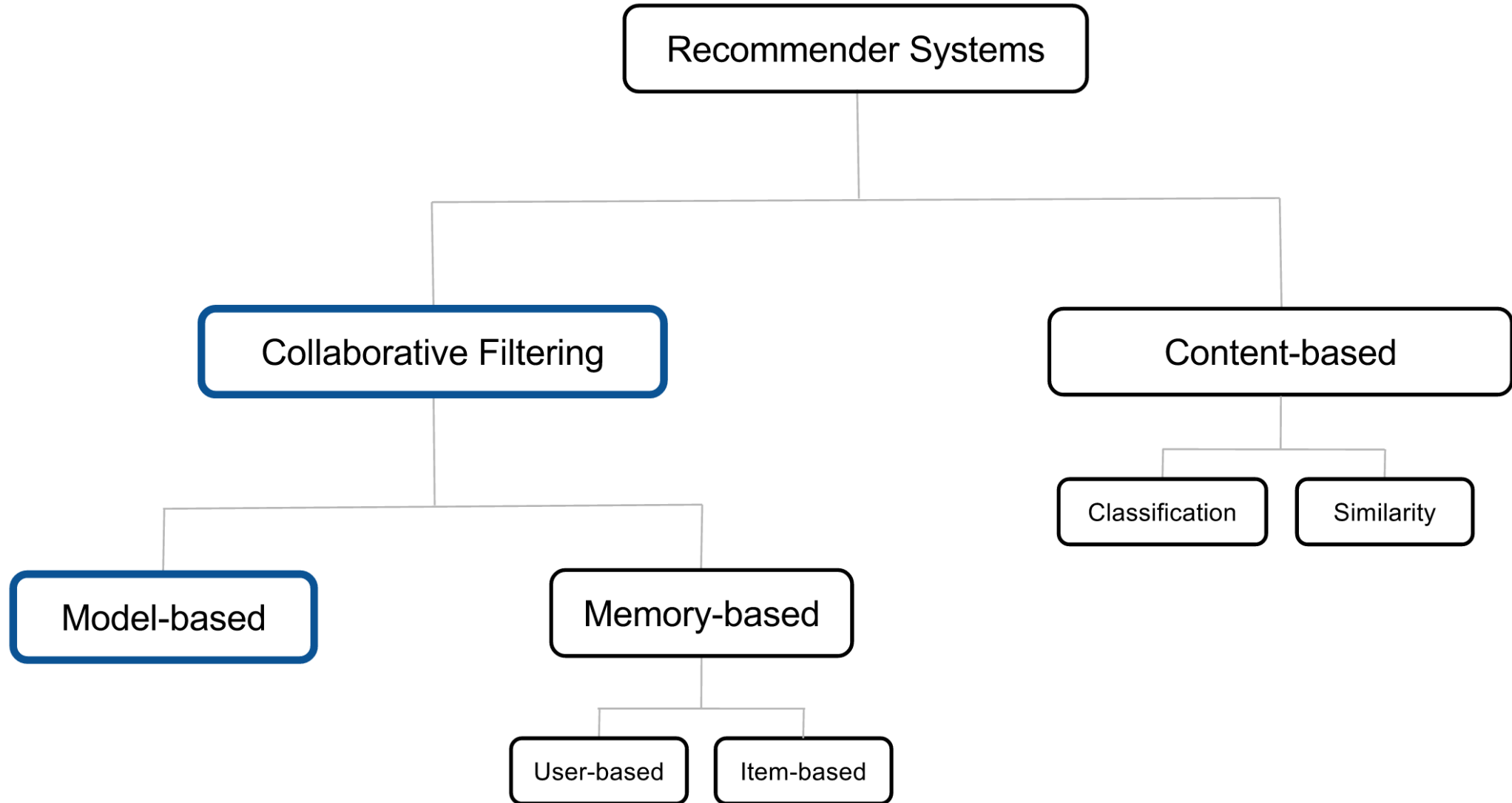
# Item-based

## Pros:

1. Better when items available are less varying than users
2. Can pre-calculate similarities matrix
3. Don't need attributes

## Problems:

- 'Cold-start': how do you predict prefs of new users?



# recommenderlab

```
r <- as(m, "realRatingMatrix")
```



# recommenderlab

```
r <- as(m, "realRatingMatrix")
```

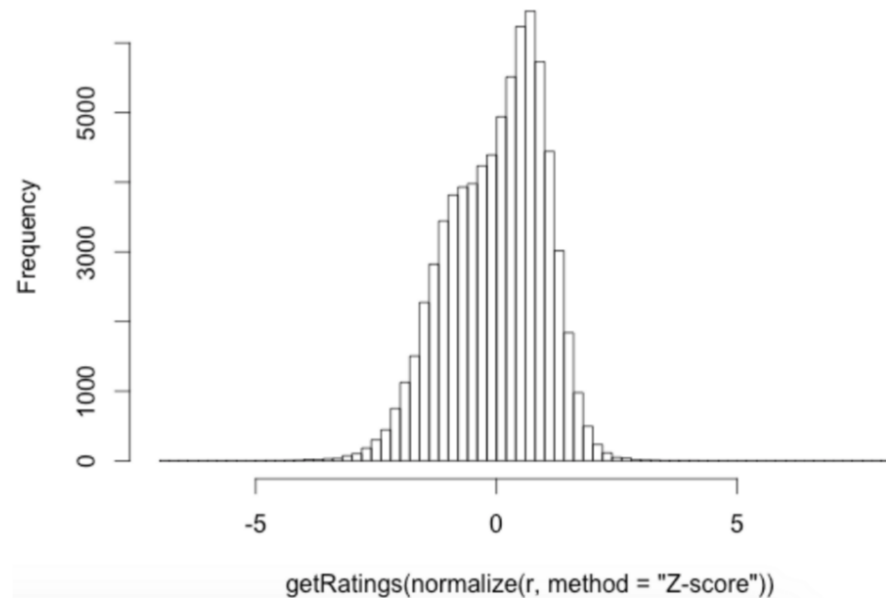
```
r_m <- normalize(r)
```

# recommenderlab

```
r <- as(m, "realRatingMatrix")
```

```
r_m <- normalize(r)
```

Histogram of `getRatings(normalize(r, method = "Z-score"))`

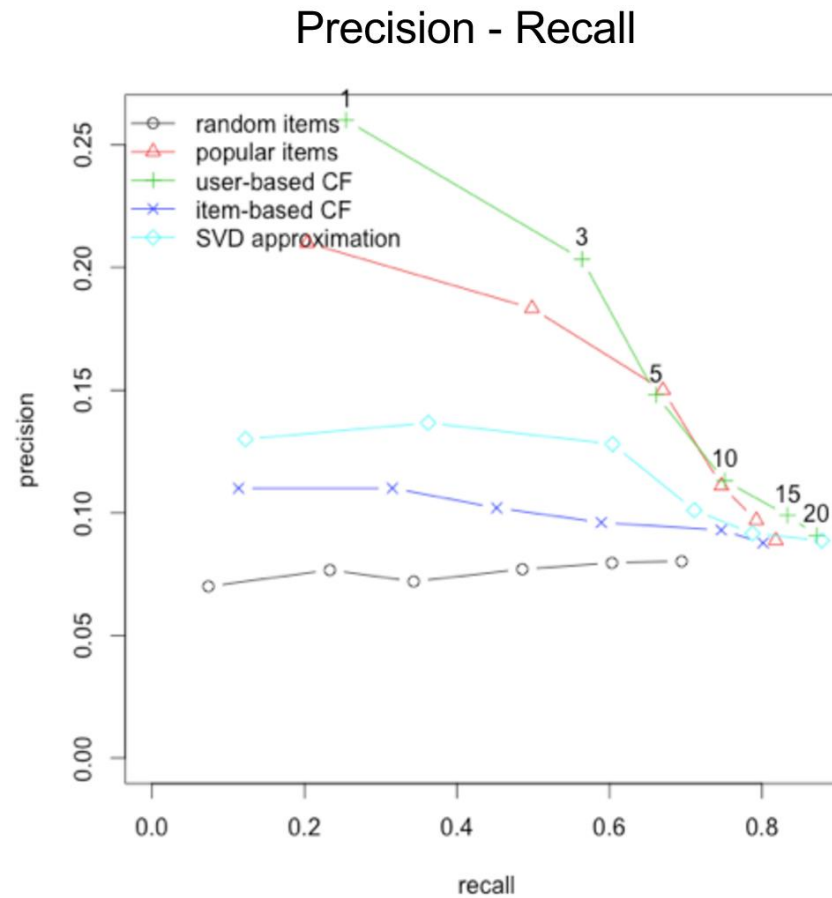
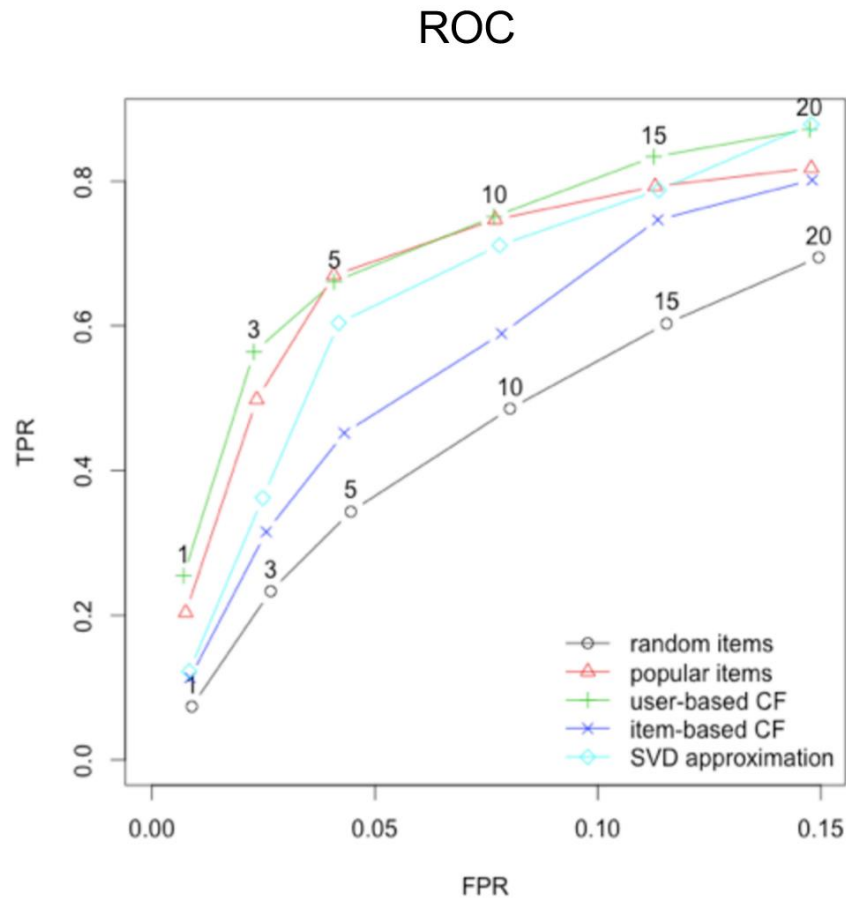


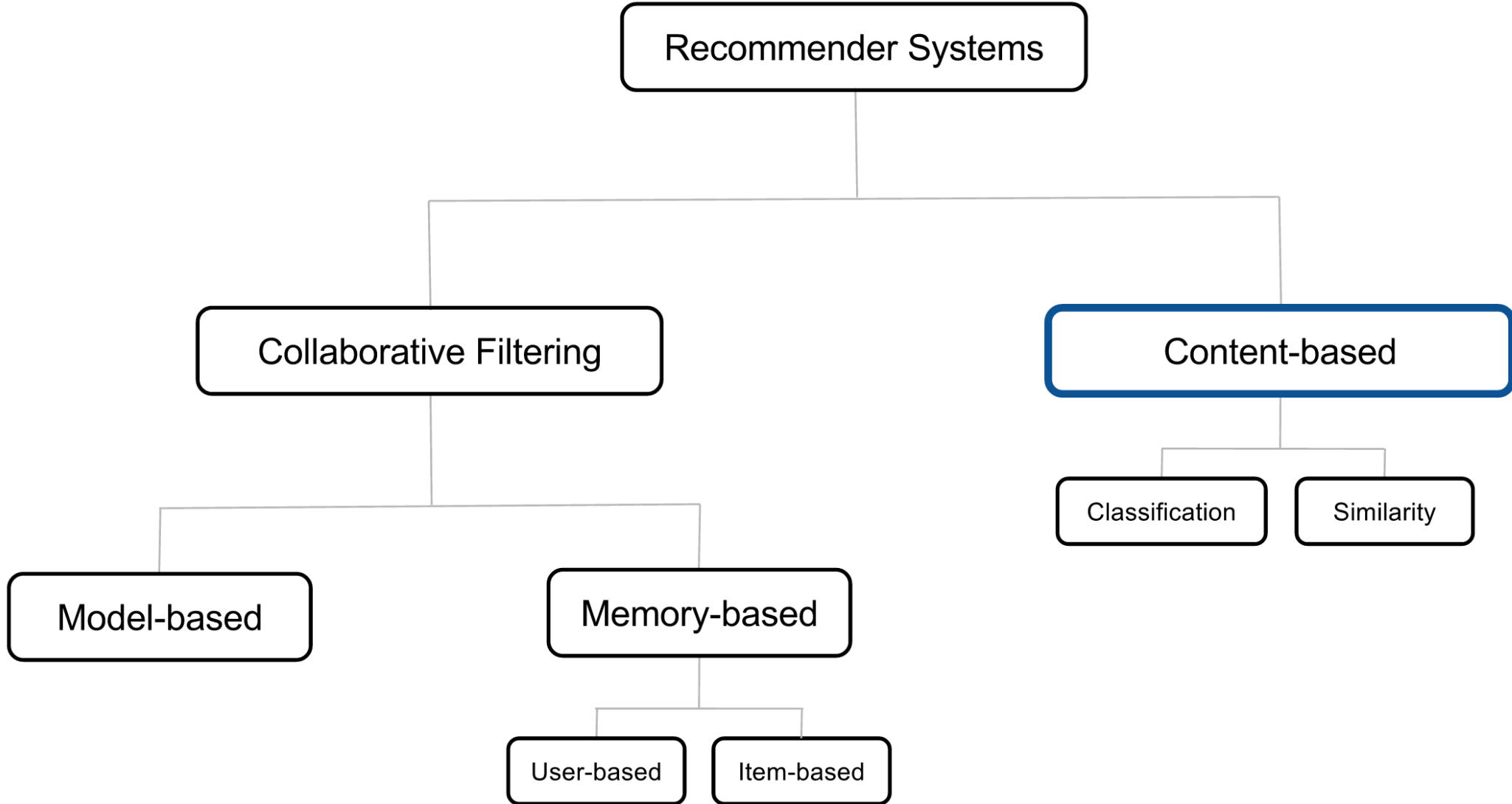
# recommenderlab

```
e <- evaluationScheme(data, method="split", train=0.8,  
                      given=10, goodRating=5)
```

# recommenderlab

evaluate(scheme, algorithms, ...)





# Content-based

## Classification method

- Train a model with ratings / behaviour and item features e.g. decision tree
- Problem: need one per user. Only useful when problem is small

# Content-based

## Similarity method

- Item *profile* vector, with elements representing presence of feature (e.g. tf-idf for top words)

# Content-based

## Similarity method

- Item *profile* vector, with elements representing presence of feature (e.g. tf-idf for top words)

wedding\_crashers = { 'owen\_wilson' : 1, 'christopher\_walken' : 1, ... }



# Content-based

## Similarity method

- Item *profile* vector, with elements representing presence of feature (e.g. tf-idf for top words)
- User *profile* vector obtained from weighted item scores, where weights = values in utility matrix

# Content-based

## Similarity method

- Item *profile* vector, with elements representing presence of feature (e.g. tf-idf for top words)
- User *profile* vector obtained from weighted item scores, where weights = values in utility matrix

$$\text{christopher\_walken} = ( 5 * \text{crashers\_profile}(\text{'walken'}) + 1 * \text{skyfall\_profile}(\text{'walken'}) ) / ( 5 + 1 )$$

# Possible approaches for AlphaSights

- **Collaborative filtering** after clustering projects based on project text
  - Problem: will it just propose popular advisors for cluster?
- **Content-based** using project text to create user profile vectors, then similarities between users and items
- **Content-based** using project text to create user profile vectors, then similarities between active project and others, then recommend items as weighted average of similar projects' items?
- **Content-based** classification model. Cluster projects by text, use as levels in predictive model
- Hierarchical cluster - most popular advisors

# Thanks

[duncan@dsanalytics.co.uk](mailto:duncan@dsanalytics.co.uk)