Item Based Recommenders

Fall 2010

Item Based Recommendations

Produced by grouping similar items together, rather than users

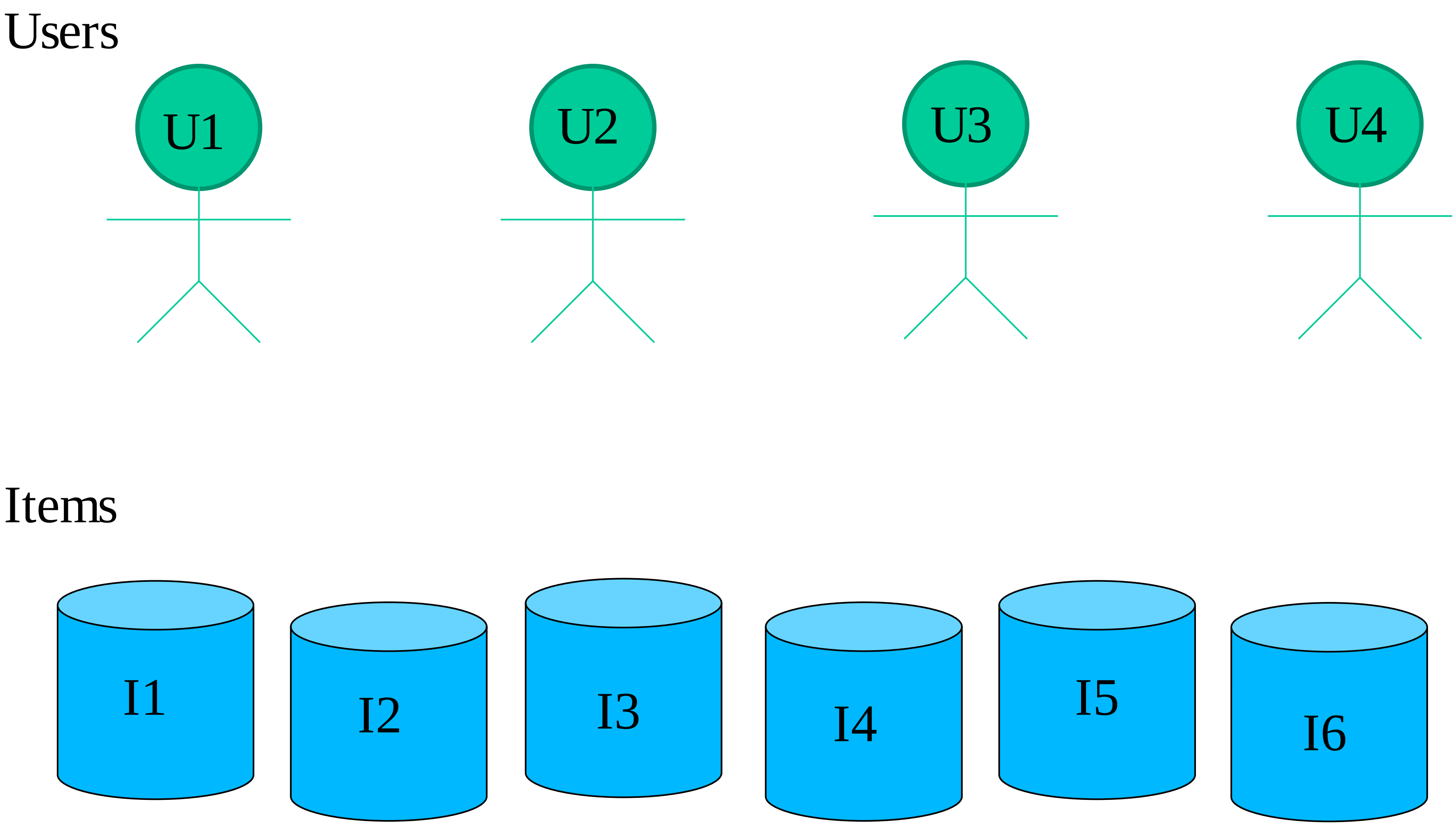
Item Based Algorithm

for every item i that u has no preference for yet
    for every item j that u has a preference for
        compute a similarity s between i and j
        add u's preference for j, weighted by s, to a running average
return the top items, ranked by weighted average

User Based Algorithm

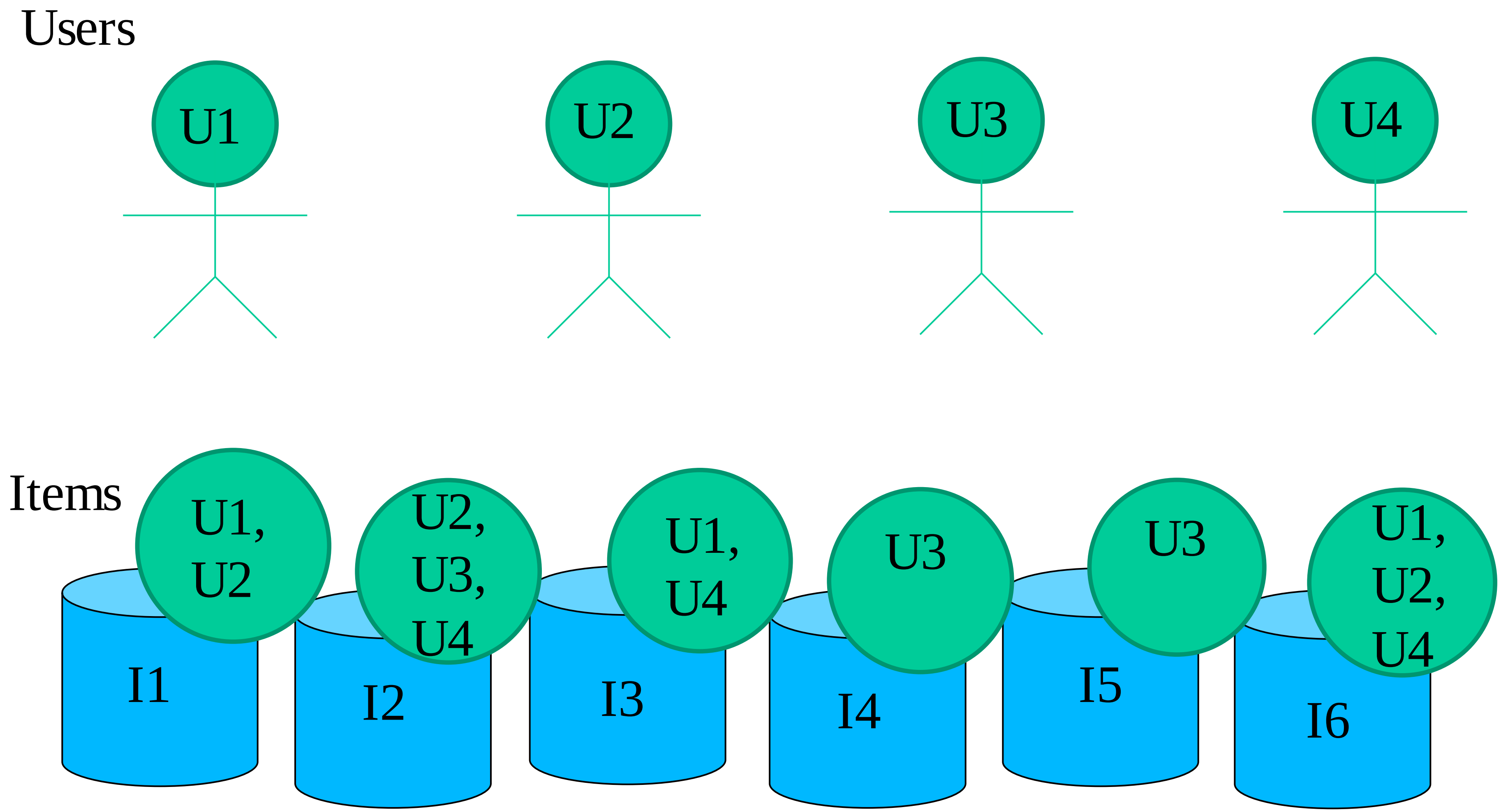
for every item i that u has no preference for yet
    for every other user v that has a preference for i
        compute a similarity s between u and v
        add v's preference for i, weighted by s, to a running average
return the top items, ranked by weighted average

Users and items



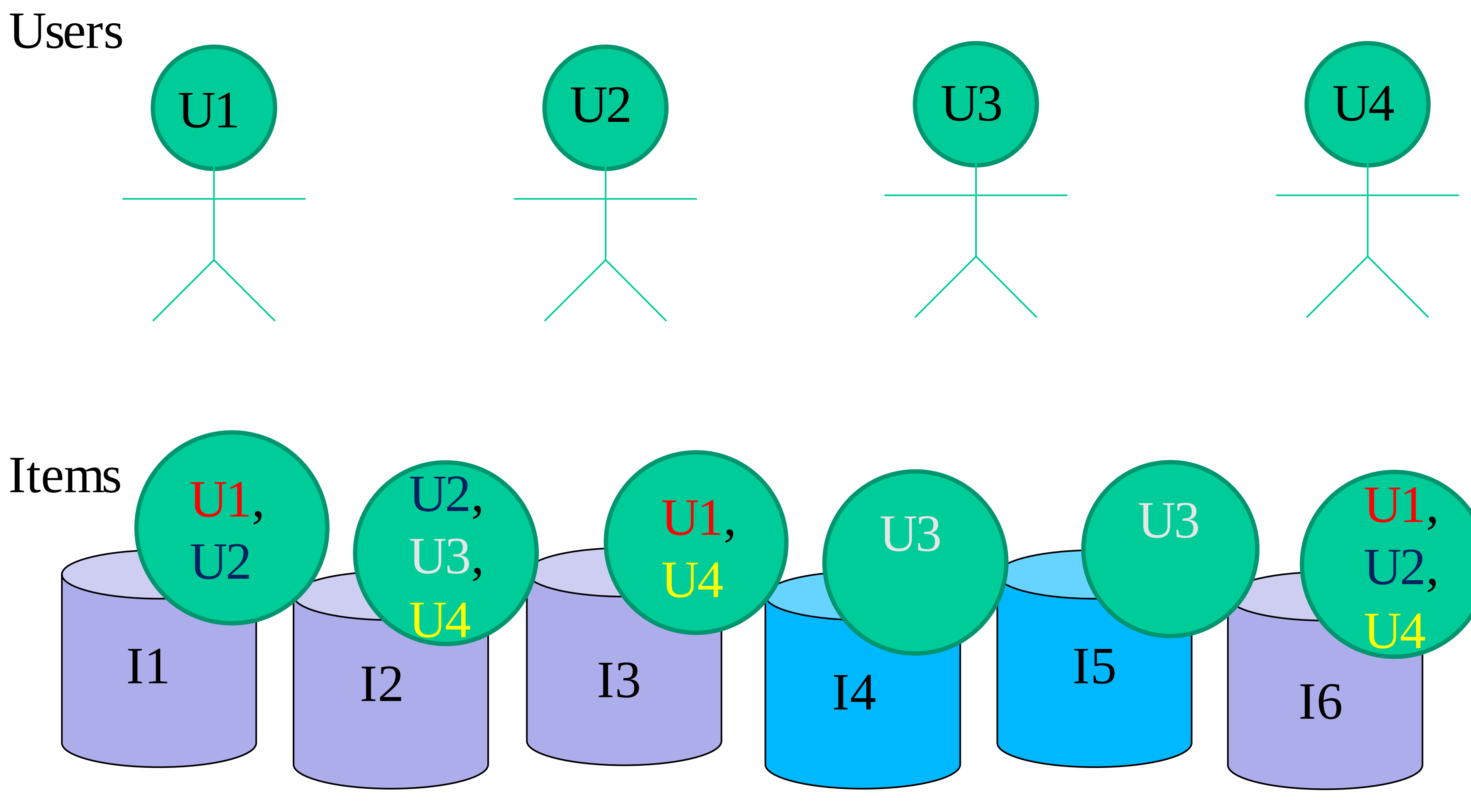
Users relate with items

- When an item is bought, clicked on, or recommended by a user that item can be associated with the user



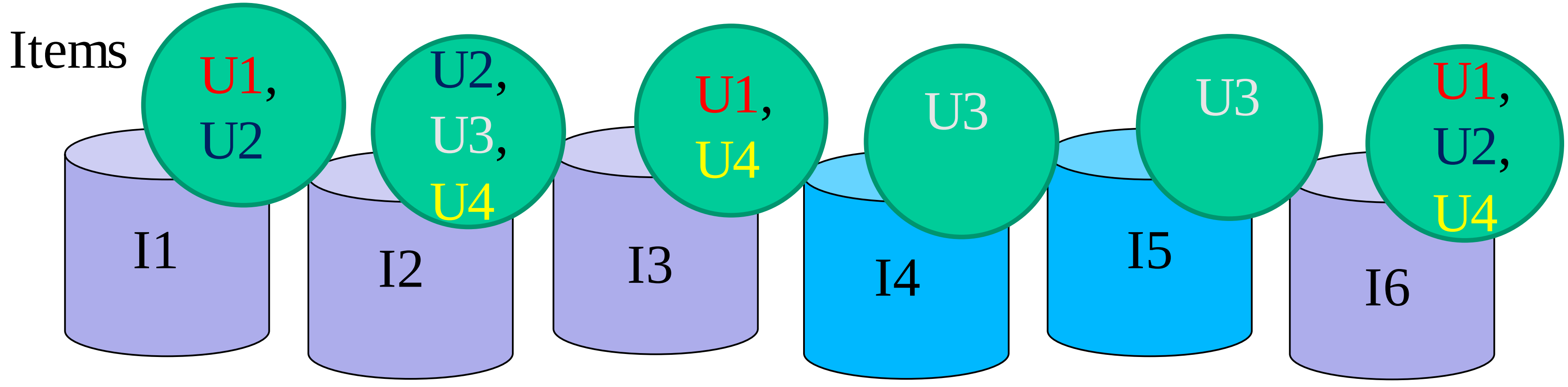
Items grouped

- Items I1 and I6 share two users, as do items I2 and I6 and items I3 and I6
- Items I1, I2, I3, and I6 are similar items



Items grouped

- Groups can be used to recommend items
- Items I1 and I6 are similar so we can recommend I1 to U4
- Items I2 and I6 are similar so we can recommend I2 to U1 and I6 to U3
- Items could be similar for other reasons than sharing users



Item Based vs. User Based

- Recommenders scale with the number of items or users they must deal with, so there are scenarios in which each type can perform better than the other
- Similarity estimates between items are more likely to converge over time than similarities between users
- We can compute and cache similarities that converge, which can give item based recommenders a performance advantage
- Item based recommenders begin with a list of a user's preferred items and therefore do not need a nearest item neighborhood as user based recommenders do

Mahout Example

```
// open the data set
DataModel model = new FileDataModel(new File("data"));

// create an ItemSimilarity object for testing similarity
ItemSimilarity sim = new LogLikelihoodSimilarity(model);

// Create an Item Based recommender using the model and log
likelihood similarity measure
Recommender recommender =  new
    GenericItemBasedRecommender(model, sim);
```

Mahout Example

```
// no neighborhood is necessary as with user-based similarity

// produce numRecommendations for userId
for (RecommendedItem recommendation :
        recommender.recommend(userId,numRecommendations))
{
    System.out.println(recommendation);

}
```

Comparing Similarity Measures

- Training/testing split is 70/30
- Data set is a subset of the grouplens 10m set

	Euclidean Distance	*Pearson Correlation*	*Tanimoto Coefficient*	*Log Likelihood*
Run time	1s	1s	4s	3s
Item-Based Average Absolute Distance	0.85	0.92	0.80	0.80
User-Based Average Absolute Distance	0.86	0.82	0.85	0.80

Slope One Recommender

- Slope one [1] uses the difference in user ratings between items to predict a user's rating
- Does not use a similarity measure
- Preprocessing step calculates the average difference in rating between every pair of items
- Recommendation step uses these differences to predict a user’s rating for a given item

Slope One Recommender

Preprocessing Algorithm

for every item i
    for every other item j
        for every user u expressing a preference for both i and j
            add the difference in u’s preference for i and j to an average

Recommendation Algorithm

for every item i the user u expresses no preference for
    for every item j that user u expresses a preference for
        find the average preference difference between j and i
            add this diff to u’s preference value for j
            add this to a running average
return the top items, ranked by these averages

Slope One Recommender - Example

- Begin by computing the average preference value difference between all item pairs
- Items 102 and 101: $\frac{(3.5-5)+(5-2)+(3.5-4.5)}{3} = \frac{0.5}{3}$
- Items 103 and 101: $\frac{(4-2)+(1-4.5)}{2} = \frac{-1.5}{2}$
- Items 104 and 101: $\frac{(2-2)+(4-4.5)}{2} = \frac{-0.5}{2}$
- Items 103 and 102: $\frac{(4-5)+(1-3.5)}{2} = \frac{-3.5}{2}$
- And so on…

	101	102	103	104
User X	5	3.5		
User Y	2.0	5.0	4.0	2.0
User Z	4.5	3.5	1	4.0

Slope One Recommender - Example

- When done we have a table we can use to look up the average difference between any two items
- This completes the preprocessing step
- Empty cells contain inverses that are omitted here

Item	101	102	103	104
101	-			
102	0.17	-		
103	-0.75	-1.75	-	
104	-0.25	-1.25	0.5	-

Slope One Recommender - Example

Illinois Institute of Technology

- Let's recommend an item for user X
- There are two potential candidates: item 103 and item 104
- We want to predict X's preferences for both items and recommend the one user X would prefer
- We need to do this using all of X's existing items: 101,102

	101	102	103	104
User X	5	3.5		
User Y	2.0	5.0	4.0	2.0
User Z	4.5	3.5	1	4.0

Slope One Recommender - Example

Illinois Institute of Technology

- Predict X's preference for item 103 using item 101
  - Look up the pre-computed average preference difference between items 103 and 101: -0.75
  - Use this to predict X's rating for item 103 based on item 101
    - X's rating for item 101: 5
    - X's predicted preference for item 103 using 101: -0.75 + 5 = 4.25

Item	101	102	103	104
101	-			
102	0.17	-		
103	-0.75	-1.75	-	
104	-0.25	-1.25	0.5	-

	101	102	103	104
User X	5	3.5		

Slope One Recommender - Example

- Predict X's preference for item 103 using item 102
  - Look up the pre-computed average preference difference between items 103 and 102: -1.75
  - Use this to predict X’s rating for item 103 based on item 102
    - X’s rating for item 102: 3.5
    - X’s predicted preference for item 103 using 102: -1.75 + 3.5 = 1.75

Item	101	102	103	104
101	-			
102	0.17	-		
103	-0.75	-1.75	-	
104	-0.25	-1.25	0.5	-

	101	102	103	104
User X	5	3.5		

Slope One Recommender - Example

- Predict X's preference for item 103
  - Preference for item 103 based on item 101: 4.25
  - Preference for item 103 based on item 102: 1.75
  - Averaging these predictions together gives us a final prediction of X's preference value for item 103
  - $\frac{4.25+1.75}{2} = 3$

Slope One Recommender - Example

- Next predict X's preference for item 104 using 101
  - Look up the pre-computed average preference difference between items 104 and 101: -0.25
  - Use this to predict X’s rating for item 104 based on item 101
    - X’s rating for item 101: 5
    - X’s predicted preference for item 104: -0.25 + 5 = 4.75

Item	101	102	103	104
101	-			
102	0.17	-		
103	-0.75	-1.75	-	
104	-0.25	-1.25	0.5	-

	101	102	103	104
User X	5	3.5		

Slope One Recommender - Example

- Predict X's preference for item 104 using 102
  - Look up the pre-computed average preference difference between items 104 and 102: -1.25
  - Use this to predict X's rating for item 104 based on item 102
    - X's rating for item 102: 3.5
    - X's predicted preference for item 104: -1.25 + 3.5 = 2.25

Item	101	102	103	104
101	-			
102	0.17	-		
103	-0.75	-1.75	-	
104	-0.25	-1.25	0.5	-

	101	102	103	104
User X	5	3.5		

Slope One Recommender - Example

- Predict X's preference for item 104
  - Preference for item 104 based on item 101: 4.75
  - Preference for item 104 based on item 102: 2.25
  - Averaging these predictions together gives us a final prediction of X’s preference value for item 104
  - $\frac{4.75+2.25}{2} = 3.5$
- Now make a recommendation based on the predicted values
  - Item 103: 3
  - Item 104: 3.5
  - Item 104 has the highest predicted preference value
  - Recommend item 104 to user X

Slope One - Mahout

```
// This estimates preference from user A to new item itemID
private float doEstimatePreference(long userID, long itemID) {
  double count = 0.0, totalPreference = 0.0;
  PreferenceArray prefs = getDataModel().getPreferencesFromUser(userID);
  RunningAverage[] averages = diffStorage.getDiffs(userID, itemID, prefs);
  int size = prefs.length();

  // loop through all existing preferences and compute running avg.
  for (int i = 0; i < size; i++) {
    RunningAverage averageDiff = averages[i];
    if (averageDiff != null) {
      double averageDiffValue = averageDiff.getAverage();
      totalPreference += prefs.getValue(i) + averageDiffValue;
      count += 1.0;
    }
  }
    return (float) (totalPreference / count);
}
```

Slope One - Weighting

- We haven't considered how much a preference value varies
- Items with less variance in their preference values should be weighted higher than items with high variance
- Use the standard deviation of an item's average preference
- $w_i = \frac{c_i}{1+\sigma_i}$, where
  - c_i = the number of users expressing a preference for item i
  - σ_i = the standard deviation of the average preference for I
- Users' estimated preference values are multiplied by this weight when a recommendation is calculated

SVD Recommender

Singular value decomposition [2] factors a m x n matrix M of rank r into three matrices of sizes m x m, m x n, and n x n

$$U = \begin{matrix} u_{1,1} & u_{1,2} \\ u_{2,1} & u_{2,2} \end{matrix}, \quad \Sigma = \begin{matrix} \Sigma_1 & 0 & 0 \\ 0 & \Sigma_2 & 0 \end{matrix}, \quad V^T = \begin{matrix} v_{1,1} & v_{1,2} & v_{1,3} \\ v_{2,1} & v_{2,2} & v_{2,3} \\ v_{3,1} & v_{3,2} & v_{3,3} \end{matrix}$$

- The matrices are then used to produce a lower dimensional representation of the underlying data, which can be thought of as features of the original data
- Neighborhoods and recommendations are then computed using the lower dimensional data
- Considering features rather than individual items can help with sparse data
- SVDRecommender takes the number of features to target and the number of steps to run as arguments

SVD Recommender

Singular value decomposition [2] factors a m x n matrix M of rank r into three matrices of sizes m x m, m x n, and n x n

- $M = U\Sigma V^T$
- A lower dimensional representation of the data is created by removing all but the r largest singular values from Σ
- This representation can be seen as an approximation of M
- With Mahout the rank r is passed to the SVD Recommender as the number of features to target
- The lower the rank the fewer nonzero values in Σ

SVD Recommender - Mahout

Num Features is number of singular values to keep. If we think there are 20 categories of users, we might set this to 20. A value of 10 for num steps is often reasonable.

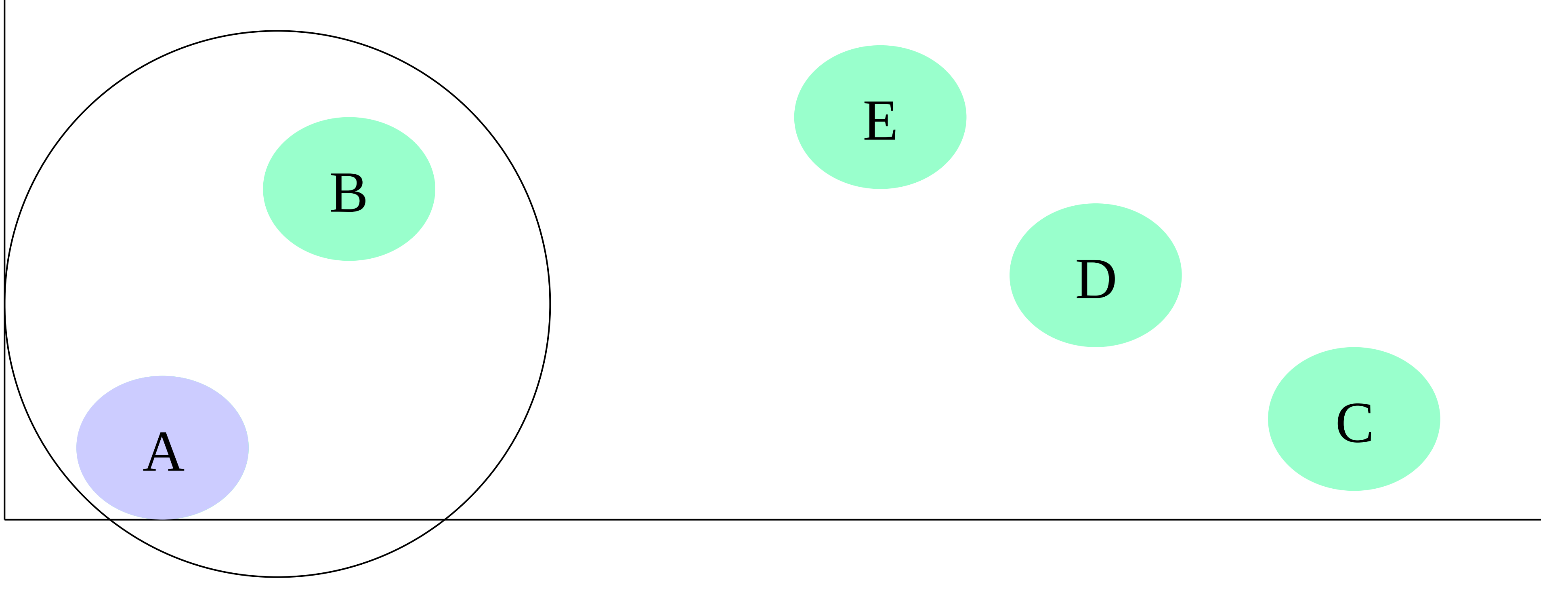
```
DataModel model = new FileDataModel(new File("data"));
Recommender recommender =
   new SVDRecommender(model, numFeatures, numSteps);

for (RecommendedItem recommendation :
           recommender.recommend(userId,numRecommend))
{
       System.out.println(recommendation);

}
```

One pass Clustering

- Choose a user and declare it to be in a cluster of size one.
- Now compute distance from this cluster to all remaining users.
- Add “closest” node to the cluster. If no node is really close (within some threshold), start a new cluster between the two closest nodes.

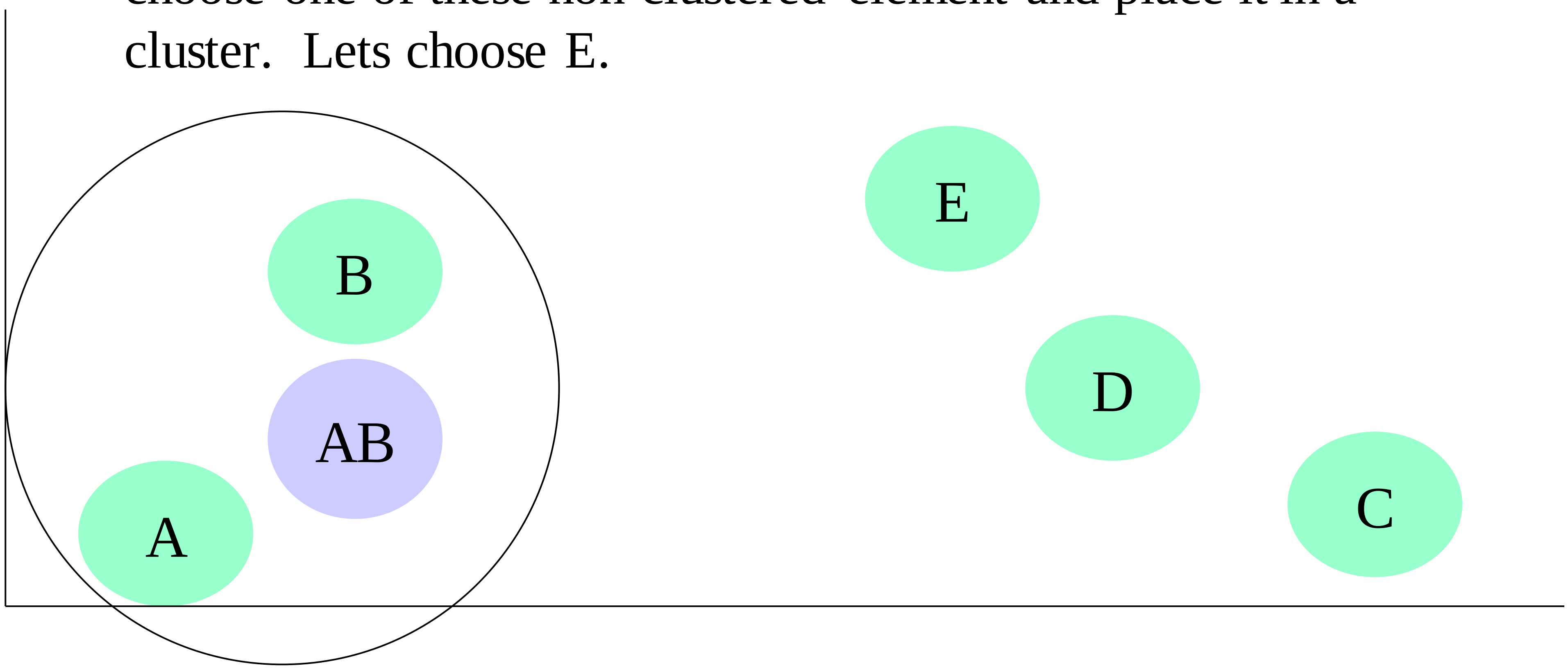
Example (One pass Clustering)

- Choose user A as the first cluster
- Now compute similarity coefficient (SC) as SC(A,B), SC(A,C) and SC(A,D). B is the closest so we now cluster B.



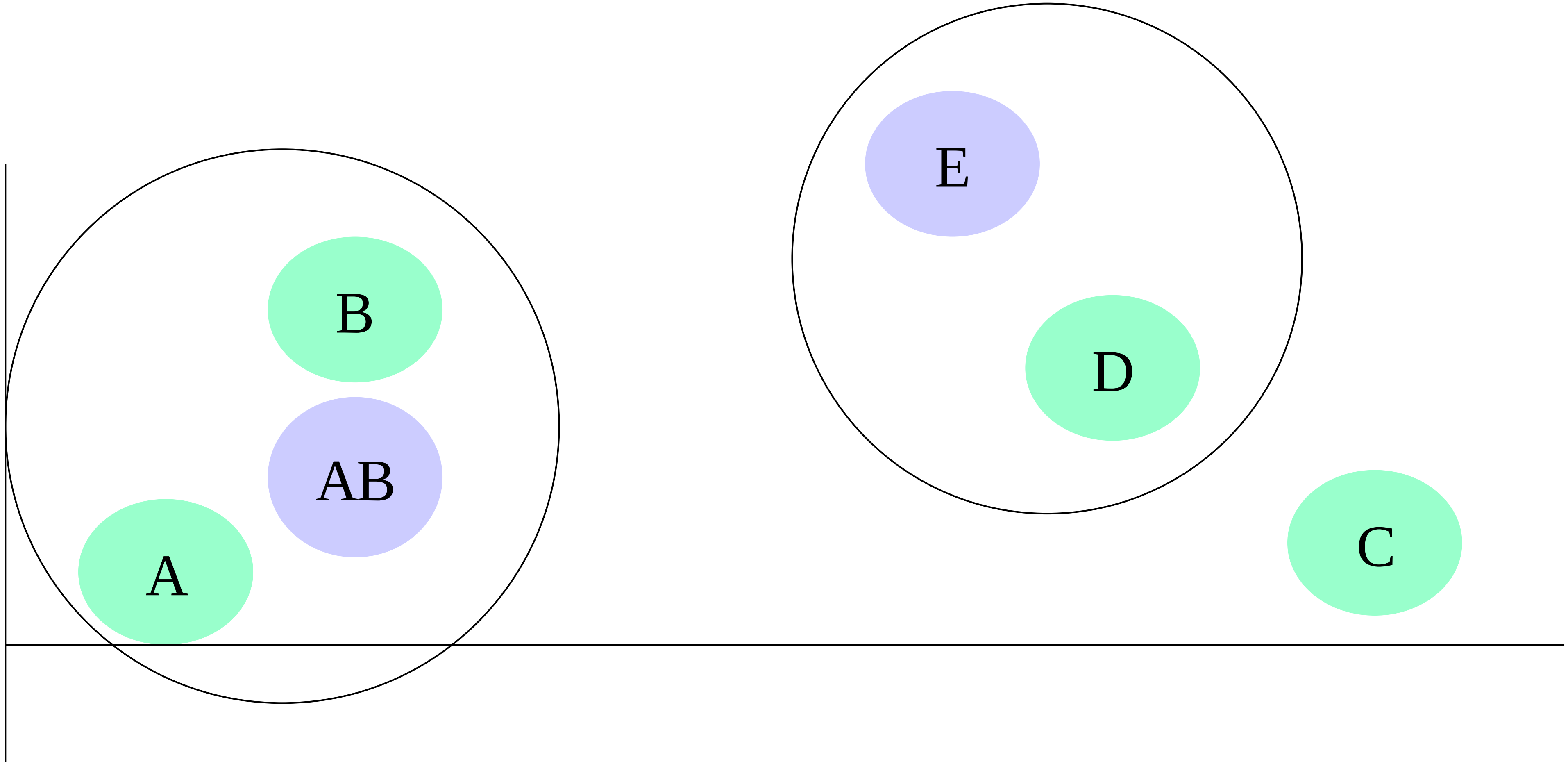
Example

- Now we treat AB as a single cluster but to measure its similarity compute the distance min(SC(A,E), SC(B,E)) to compute SC(AB, E).
- Lets assume its too far from AB to E, D, and C. So now we choose one of these non-clustered element and place it in a cluster. Lets choose E.



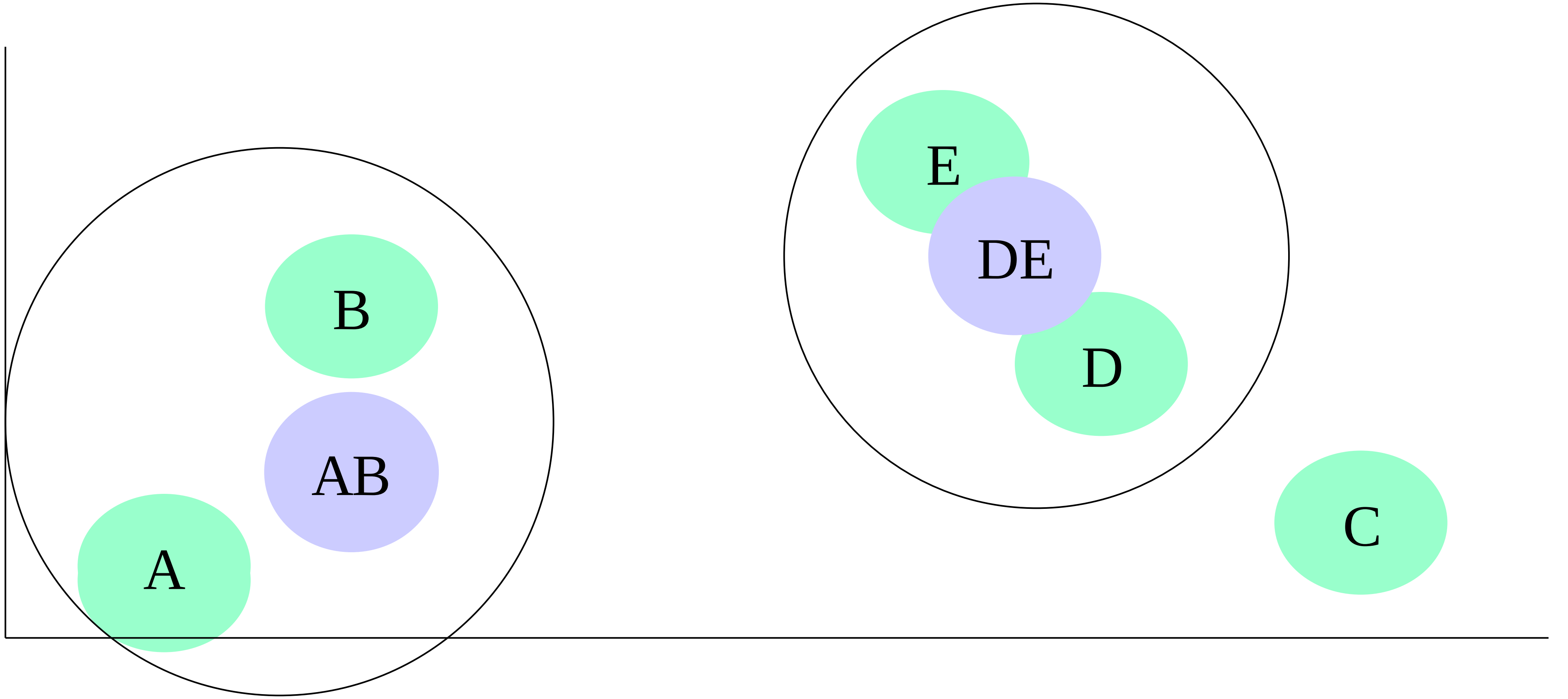
Example

- Now we compute the distance from E to D and E to C. E to D is closer so we form a cluster of E and D.



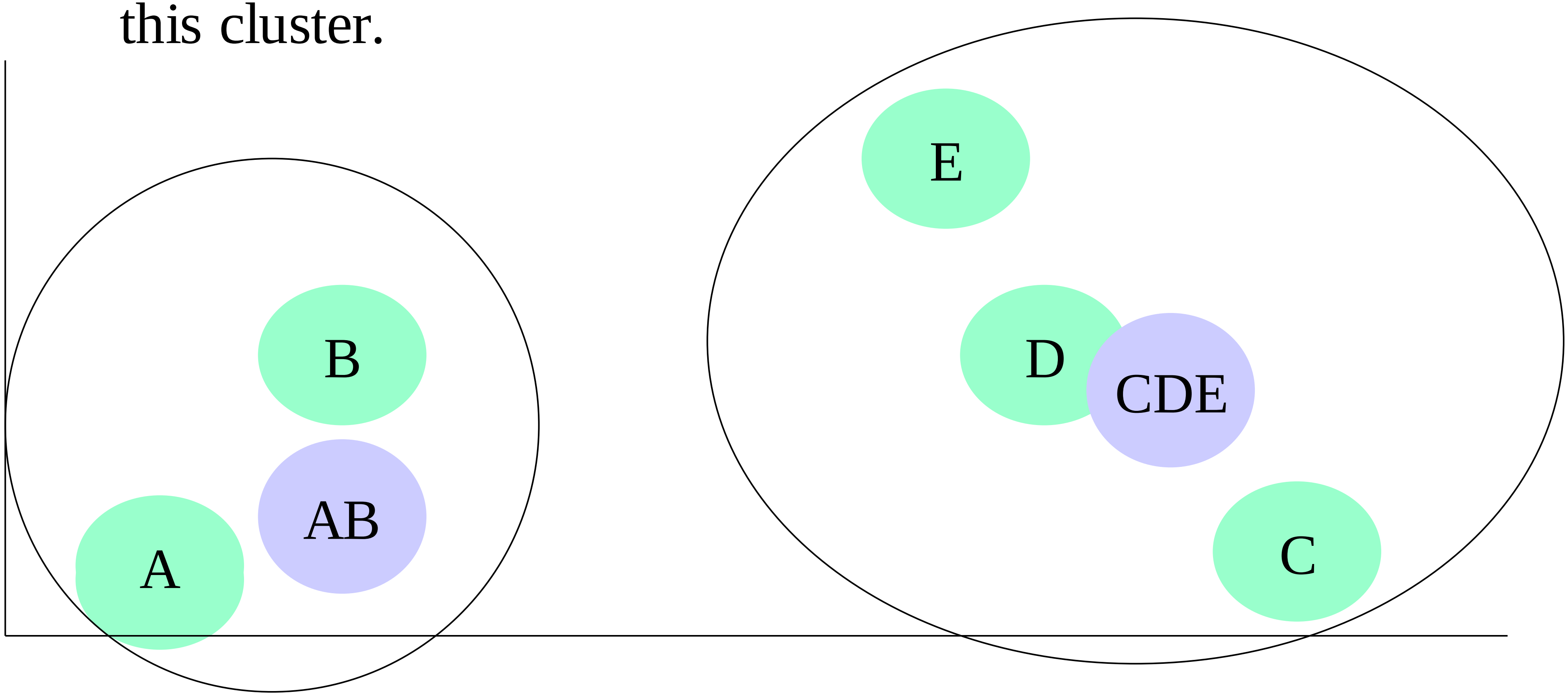
Example (Cont'd)

- Now we compute the centroid of D and E, which is DE.



Example (Cont'd)

- Now we compute the distance from DE to C, SC(DE, C). It is within the threshold so we now include C in this cluster.



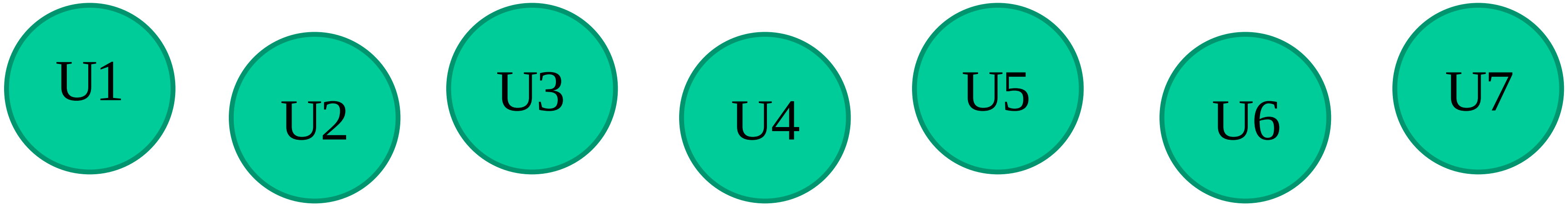
Cluster Recommender

Cluster based recommenders [3] operate by grouping users

- Recommendations are made for the entire cluster, not for individual users as with previous methods
- Can work well for new users with few preferences
- Similarity between clusters is defined as either
  - the similarity between the most similar users in a cluster
  - the similarity between the least similar users in a cluster
- Mahout requires as an argument either the number of clusters to create or a cluster similarity threshold

Cluster Recommender - Example

- Euclidean Distance similarity, and a target of 2 clusters
- Begin by creating one cluster for each user



	101	102	103	104	105
U1	5.0	3.5	2.5		4.0
U2	2.0	5.0	5.0	2.0	
U3	4.5	3.5		4.0	5.0
U4	1.0	4.0		4.5	1.5
U5	2.5	4.0			2.0
U6	3.5			4.0	4.0
U7				5.0	4.0

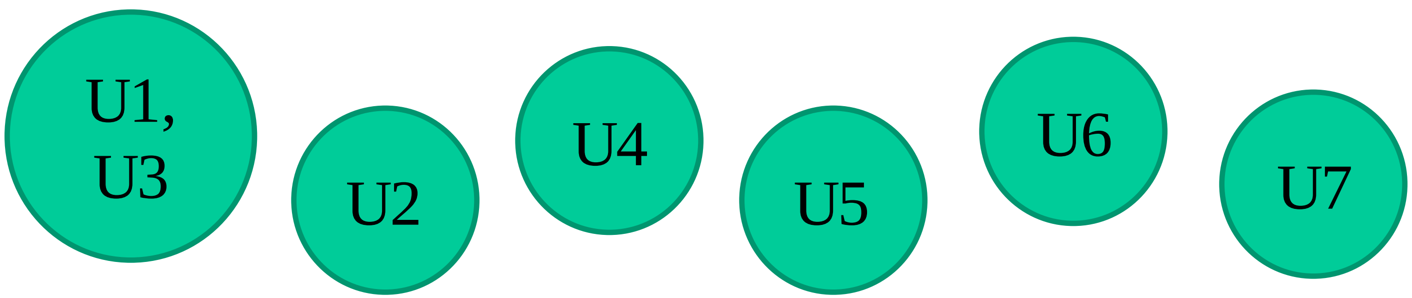
Cluster Recommender - Example

- We have more than two clusters, so merge two clusters
- Find the most similar pair of clusters by finding the Euclidean Distance between each pair of users in the clusters
- Recall that the distance is computed as $D(X,Y) = \frac{n}{1+\sqrt{\sum_i (x_i - y_i)^2}}$

Distance	U1	U2	U3	U4	U5	U6	U7
U1	-	0.58	1.0	0.55	0.71	0.8	1.0
U2		-	0.66	1.0	0.94	0.57	0.25
U3			-	0.56	0.65	1.0	0.83
U4				-	0.8	0.44	0.22
U5					-	0.62	0.33
U6						-	1.0
U7							-

Cluster Recommender - Example

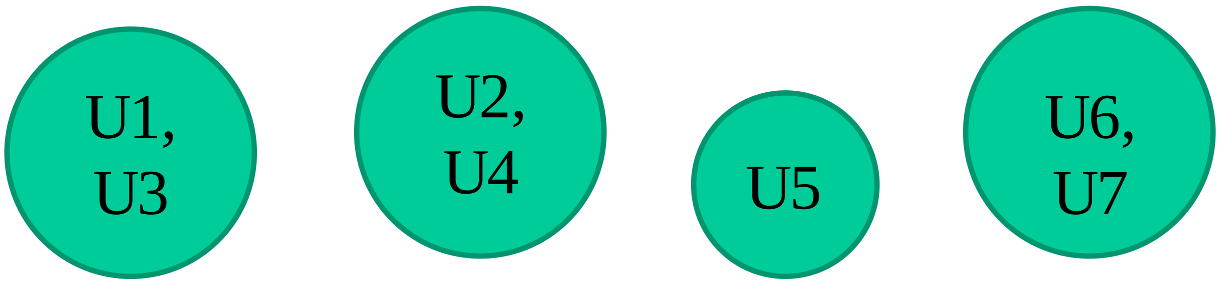
- Similarity between two clusters is defined as the furthest distance between a pair of users in the two clusters
- Three one-member clusters have a similarity of 1.0
- Start by merging the clusters containing U1 and U3



Cluster Recommender - Example

Illinois Institute of Technology

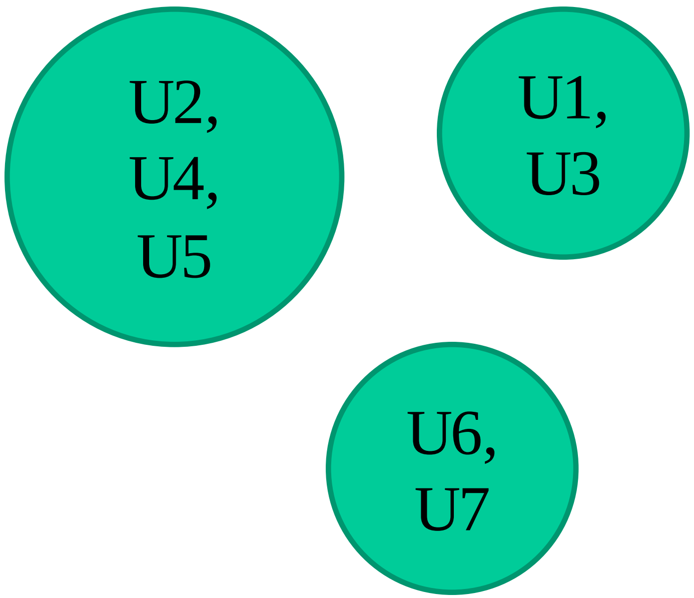
- We know that similarity can't be greater than 1.0, so we can also now merge the other two clusters with a similarity of 1.0



Distance	U1	U2	U3	U4	U5	U6	U7
U1	-	0.58	1.0	0.55	0.71	0.8	1.0
U2		-	0.66	1.0	0.94	0.57	0.25
U3			-	0.56	0.65	1.0	0.83
U4				-	0.8	0.44	0.22
U5					-	0.62	0.33
U6						-	1.0
U7							-

Cluster Recommender - Example

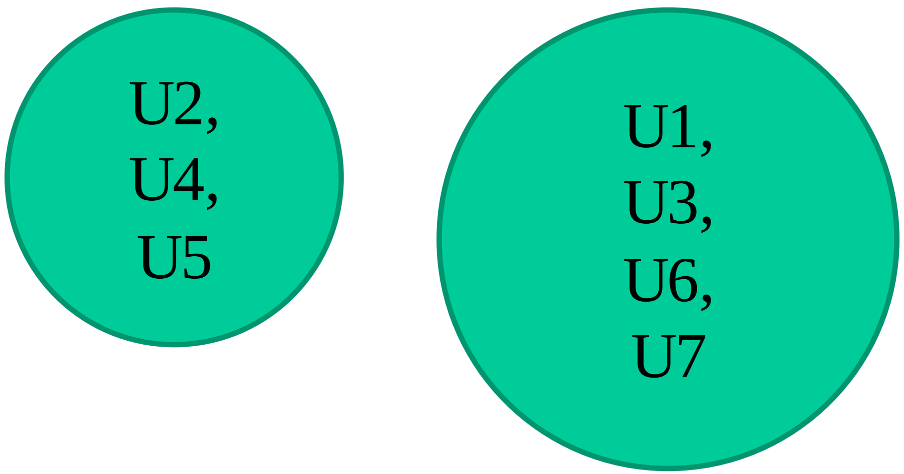
- Compute similarity again using the new clusters and merge the two most similar clusters: [U2,U4] and U5



	U1, U3	U2, U4	U5	U6, U7
U1, U3	-	0.55	0.65	0.8
U2, U4		-	0.8	0.22
U5			-	0.33
U6, U7				-

Cluster Recommender - Example

- Compute similarity again using the new clusters and merge the two most similar clusters: [U1,U3] and [U6,U7]
- We have reached the target of 2 clusters and can stop merging



	U1,U3	U2,U4,U5	U6,U7
U1,U3	-	0.55	0.8
U2,U4,U5		-	0.22
U6,U7			-

Cluster Recommender - Example

- Let's recommend an item for U7
- Find the items with the highest average preference within each cluster; for U7's missing items the average preferences are
  - Item 101: $\frac{5+4.5+3.5}{3} = 4.3$
  - Item 102: $\frac{3.5+3.5}{2} = 3.5$
  - Item 103: $\frac{2.5}{1} = 2.5$
- Recommend item 101 to U7 with an estimated preference value of 4.3

	101	102	103	104	105
U1	5.0	3.5	2.5		4.0
U3	4.5	3.5		4.0	5.0
U6	3.5			4.0	4.0
U7				5.0	4.0

Tree Clustering Recommender - Mahout

```
DataModel model = new FileDataModel(new File("data"));

// Use Euclidean Distance for user pair similarity
UserSimilarity usim = new
    EuclideanDistanceSimilarity(model);

// Farthest neighbor cluster similarity measure with similarity uSim
ClusterSimilarity clusterSim =
    new FarthestNeighborClusterSimilarity(uSim);

// Create a recommender
Recommender recommender = new
    TreeClusteringRecommender(model, clusterSim, numCluster);
```

Comparing Recomenders

- Training/testing split is 70/30
- Data set is 2m records taken from the grouplens 10m set
- SVD: 10 features and 10 steps
- Tree Clustering: 100 clusters

	Average Absolute Distance	*Precision*	*Recall*	*Build run time*	*Evaluation run time*
Slope One	0.75	0.07	0.10	1s	189s
SVD	0.78	0.14	0.19	1s	149s
Tree Clustering	0.96	0.07	0.24	34s	102353s (1d 4h+)

Recommender Summary

Type	*Parameters*	*Features*
User based	user similarity metric, user neighborhood	Fast when there are fewer users than items
Item based	item similarity metric	Fast when there are fewer items than users, useful with a specialized item similarity definition
Slope one		Fast at runtime, slow to precompute, good with low item numbers
SVD	number of target features	Slow to precompute
Tree clustering	user similarity metric, cluster similarity metric, number of clusters	Fast at runtime, slow to precompute, good with low user numbers

References

D. Lemire and A. Maclachlan, Slope One Predictors for Online Rating-Based Collaborative Filtering. In Proceedings SIAM Data Mining (SDM'05), 2005.

B. Sarwar, G. Karypis, J. Konstan and J. Riedl. Application of Dimensionality Reduction in Recommender System -- A Case Study. In Proceedings WebKDD Workshop (WebKDD’04), 2004.

P. Franti, O. Virmajoki, and V. Hautamaki. Fast Agglomerative Clustering using K-Nearest Neighbor Graph. Transactions on Pattern Analysis and Machine Intelligence, Volume 28, Number 11, November 2006.