PageRank: $v' = Mv$, Deadend: page ranks = 0, SpiderTrap: a pagerank = 1, Avoid spidertrap and deadend: $v' = Mv + (1-\beta)\frac{e}{n}$, e is a vector of all 1s. n is the number of nodes in the web. Topic Sensitive Page Rank: $v' = \beta Mv + (1-\beta)\frac{e_s}{|S|}$, S is the teleport set. Sparm Farm: tek oklu: $y = x + \beta(1-\beta)\frac{M}{N} + \frac{1-\beta}{N}$ $y = \frac{x}{1-\beta^2} + \frac{\beta}{1+\beta}\frac{m}{n}$ Spam Mass: $p = \frac{r-t}{r}$, spam mass close to 1, page probably a spam.

SIMILAR ITEMS: Jaccard Similarity: $\frac{|S \cap T|}{|S \cup T|}$ Minhash Signatures: 1. Compute $h_1(r), h_2(r), \ldots, h_n(r)$ 2. For each column c do: (a) If c has 0 in row r, do nothing. (b) If c has 1 in row r, then for each i = 1 to n set SIG(i, c) to the smaller of the current value of SIG(i, c) and $h_i(r)$. LSH for Minhash Signature: "hash" items several times, in such a way that similar items are more likely to be hashed to the same bucket. We consider any pair that hashed to the same bucket for any of the hashings to be a candidate pair. We check only the candidate pairs for similarity. The hope is that most of the dissimilar pairs will never hash to the same bucket. We also hope that most of the truly similar pairs will hash to the same bucket under at least one of the hash functions. If we have minhash signatures for the items, an effective way to choose the hashings is to divide the signature matrix into b bands consisting of r rows each. For each band, there is a hash function that takes vectors of r integers and hashes them to some large number of buckets. We can use the same hash function for all the bands, but we use a separate bucket array for each band, so columns with the same vector in different bands will not hash to the same bucket. Make candidate pairs with band comparison.

Same bucket and not similar -> false positive. different bucket and similar -> false negative. Euclidean Distance: $(\sum_{i=1}^{n} |x_i - y_i|^r)^{1/r}$. Jaccard Distance: $d(x,y) = 1 - SIM(x,y)$. Cosine Distance: $\frac{Dotproduct}{{L_2}^2}$. Edit distance: The smallest number of insertions/deletions of single characters that will convert x to y.

MapReduce: When the Master is executing fails. In this case, the entire MapReduce job must be restarted. But only this one node can bring the entire process down; other failures will be managed by the Master, and the MapReduce job will complete eventually. a Map worker resides fails, All the Map tasks that were assigned to this Worker will have to be redone, even if they had completed. The Master sets the status of each of these Map tasks to idle and will schedule them on a Worker when one becomes available. Dealing with a failure at the node of a Reduce worker is simpler. The Master simply sets the status of its currently executing Reduce tasks to idle.

Recommendation Systems: The Utility Matrix: each user-item pair, a value that represents what is known about the degree of preference of that user for that item. not even be necessary to find all items with the highest expected ratings, but only to find a large subset of those with the highest ratings. Populating The Utility Matrix: We can ask users to rate items. We can make inferences from users' behavior. Content-Based Recommendations: Similarity of items is determined by measuring the similarity in their properties. In a content-based system, we must construct for each item a profile, which is a record or collection of records representing important characteristics of that item. In simple cases, the profile consists of some characteristics of the item that are easily discovered. Discovering Features of Documents: First, eliminate stop words – the several hundred most common words, which tend to say little about the topic of a document. For the remaining words, compute the TF.IDF score for each word in the document. The ones with the highest scores are the words that characterize the document. To compute the cosine distance in option (2), think of the sets of high- TF.IDF words as a vector. To be precise, the dot product is the size of the intersection of the two sets of words, and the lengths of the vectors are the square roots of the numbers of words in each set. That calculation lets us compute the cosine of the angle between the vectors as the dot product divided by the product of the vector lengths. User Profiles: If the utility matrix is not boolean, e.g., ratings 1–5, then we can weight the vectors representing the profiles of items by the utility value. It makes sense to normalize the utilities by subtracting the average value for a user. That way, we get negative weights for items with a below-average rating, and positive weights for items with above-average ratings. Recommending Items to Users Based on Content: With profile vectors for both users and items, we can estimate the degree to which a user would prefer an item by computing the cosine distance between the user's and item's vectors. The random-hyperplane and locality-sensitive-hashing techniques can be used to place (just) item profiles in buckets. In that way, given a user to whom we want to recommend some items, we can apply the same two techniques – random hyperplanes and LSH – to determine in which buckets we must look for items that might have a small cosine distance from the user. Collaborative Filtering: Users are similar if their vectors are close according to some distance measure such as Jaccard or cosine distance. Recommendation for a user U is then made by looking at the users that are most similar to U in this sense, and recommending items that these users like. The process of identifying similar users and recommending what similar users like is called collaborative filtering. Measuring Similarity: If we normalize ratings, by subtracting from each rating the average rating of that user, we turn low ratings into negative numbers and high ratings into positive numbers. If we then take the cosine distance, we find that users with opposite views of the movies they viewed in common will have vectors in almost opposite directions, and can be considered as far apart as possible. However, users with similar opinions about the movies rated in common will have a relatively small angle between them. User-user: predicting the value of the utility-matrix entry for user U and item I is to find the n users most similar to U and average their ratings for item I, counting only those among the n similar users who have rated I. It is generally better to normalize the matrix first. That is, for each of the n users subtract their average rating for items from their rating for i. Average the difference for those users who have rated I, and then add this average to the average rating that U gives for all items. This normalization adjusts the estimate in the case that U tends to give very high or very low ratings, or a large fraction of the similar users who rated I are users who tend to rate very high or very low. Item-item: Find the m items most similar to I, for some m, and take the average rating, among the m items, of the ratings that U has given. As for user-user similarity, we consider only those items among the m that U has rated, and it is probably wise to normalize item ratings first. $b_{xi} = u + b_x + b_i$ -> u = global movie rating, b_x = Rating deviation of user x, b_i = Rating deviation of movie i. $r_{xi} = b_{xi} + \frac{(\sum s_{ij}(r_{xj} - b_{xj}))}{(\sum s_{ij})}$

Root-Mean-Square Error: Sum, over all nonblank entries in M the square of the difference between that entry and the corresponding entry in the product UV. Generally, we stop at this point, but if we want to compute the true RMSE, we divide by the number of nonblank entries in M and take the square root.