**Chapter 1: Getting Started with Recommender Systems**

* **Collaborative Filtering Systems** – looks at similarities between users and recommends based on what others have bought
  + If 2 users share the same interests in the last, they also have similar tastes in the future
    - Ex: User A and B have similar purchase history. User A buys book 1. User B is recommended book 1.
  + Does not take features/contents of items being recommended into account – only looks at user preferences
  + Requires a large set of user preferences
* **Content-based Systems** – considers only the user’s past preferences and the properties/features of the items
  + Recommends items by taking the similarity of items and user profile into consideration -- items like those that the user has liked in the past
  + Similarity of items is calculated based on the features associated with the other compared items and is matched with user’s historical preferences
  + Does not take additional user preferences into consideration so does not need a large user group’s preferences for better recommendation accuracy
* **Knowledge-based Systems –** takes in knowledge about the items (ex: features) and recommendation criteria and asks user preferences explicitly
  + Constraint-based systems because the user provides details about requirements
  + Ex: when recommending an air conditioner, the user is asked what size they would like it to be

**Chapter 2: Data Mining Techniques Used in Recommender Systems**

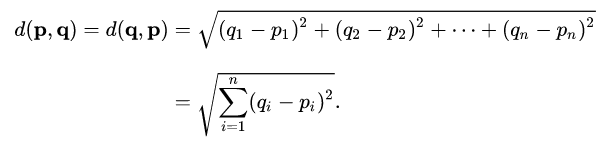
**Similarity Measures**

* **Euclidean Distance** – similarity measure where (q,p) are two consecutive data points and n is the number of attributes in the dataset

**x1 <- rnorm(30)**

**x2 <- rnorm(30)**

**dist(rbind(x1,x2), method = “Euclidean**”)



* **Cosine Distance** – similarity measure between 2 vectors of an inner product space that measures the cosine of the angle between them
  + ***Outperforms other similarity measures in item-based collaborative filtering***

**vec1 <- c(1,1,1,0,0,0,0,0,0,0,0,0**

**vec2 <- c(0,0,1,1,1,1,1,0,1,0,0,0)**

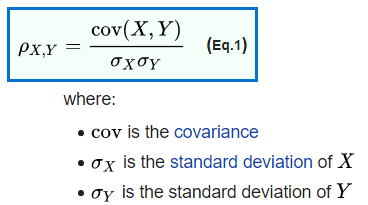
**library(lsa)**

**cosine(vec1,vec2**)



* **Pearson correlation** – similarity measure; correlation coefficient calculated between 2 variables as the covariance of the 2 variables divided by the product of their standard deviation
  + ***Outperforms other similarity measures for user-based collaborative filtering recommender systems***

**Coef <- cor(mtcars, method = “pearson”)**



**Dimensionality Reduction**

* **Principal Component Analysis** – transforms the data with high-dimensional space to a space with fewer dimensions; allows us to discard features that have less variance
  + We create n new features, each of which is a combination of 2 of the original features
  + The first principle component has the largest possible variance and accounts for as much of the variability in the data as possible by considering highly correlated features.
  + Each succeeding component has the highest variance using the features that are less correlated with the first principal component

**Data Mining Techniques**

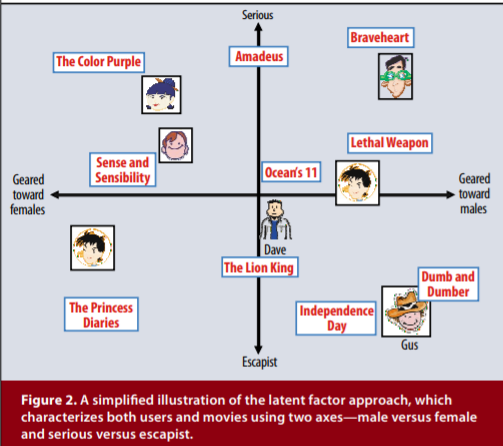
* **K-means clustering** – unsupervised; cluster similar points together based on distance
* **Support Vector Machine** – supervised; classification problems; forms a boundary between classes based on the maximum margin to separate classes
* **Ensemble Methods**
  + **Bagging** **(bootstrap aggregating)** – multiple copies of bootstrap samples (sampling with replacement) are created, a new model is fitted for each subset, and all individual models are combined (aggregated or averaged) to create a single, predictive value
    - Helps to avoid overfitting and reduces variance
    - Focus: less variance (overfit)
    - Used mostly with decision trees
  + **Random Forest** – select only a portion of the variables for the bootstrapped samples; predictions made by averaging the results of each model
    - Removes dependency of strong predictors in the dataset because we intentionally select fewer variables
    - De-correlates variables, resulting in less variability in the model and hence, more reliability
  + **Boosting** – each new model is built using information from previously built models; new model is built from the residuals from the previous model – each new model learns from previous mistakes
    - Focus: less biased (underfit)

**Data Mining Evaluation**

* **Bias**: Underfitting
* **Variance**: Overfitting
* **Cross validation**: split into training & testing; calculate error on testing; do this many times and average the test error
* **Regularization**: variables are penalized to reduce the complexity of the model with the objective to minimize the cost function; try to reduce the coefficients to 0, so a smaller number of variables will fit the data optimally
* **Confusion Matrix**: precision (sensitivity) and recall (specificity)
  + **Precision (Positive Predictive Value – PPV)**: pct of positively classified records that are true; true classified positives/all classified positives
    - PPV = TP/ (TP + FP)
  + **Recall** **(Sensitivity/ True Positive Rate):** pct of positive records classified properly; true classified positive/all true positives
    - TPR = TP/P = TP/ (TP+FN)
  + **Specificity (True Negative Rate):** pct of negative records classified properly; true classified negatives/all true negatives
    - SPC = TN/N = TN / (TN+FP)

**Matrix Factorization Techniques for Recommender Systems – External Source**

* **Latent Filtering** – a type of collaborative filtering methodology (depends on past user history) that characterizes BOTH users and items and attempts to find recommendations that way



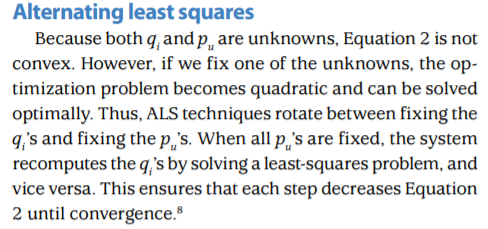
* **Matrix factorization** is the basis of latent filtering – it characterizes both items and users by vectors of factors inferred from item rating patterns
  + When explicit feedback (ex: ratings) is not available, recommender systems can infer user preferences using implicit feedback, which indirectly reflects opinion by observing user behavior including purchase history, browsing history, search patterns, or even mouse movements
  + User-item interactions are modeled as dot products
* **Implicit feedback** usually denotes the presence or absence of an event, so it is a densely filled matrix
* Each user, *i,* has an associated vector in the vector-item space (*q\_i*). Similarly, each item *u* is associated with a vector (*p\_u*)
  + In the user matrix, the values represent the attitude towards the features (positive or negative)
  + In the item matrix, the values represent the extent to which the item has the features (positive or negative)
* The user’s rating of the product is denoted by the dot product of the two:



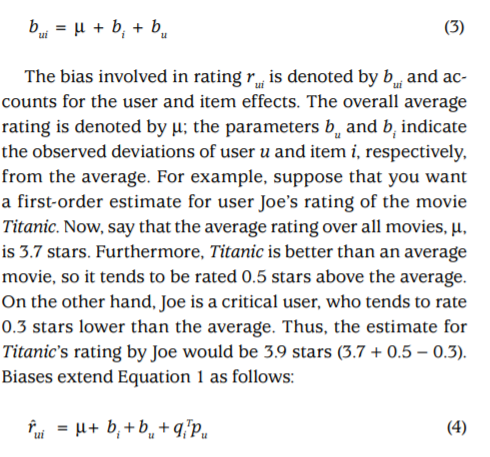
* This technique is like **Singular Value Decomposition (SVD)** but with the additional step of factoring the user-item rating matrix
  + SVD requires no missing values
* To use Singular Value Decomposition, we must only take into account the non-zero ratings, which can result in overfitting of the model. To account for this, we will add in a regularization factor/ To learn the factor vectors (pu and qi), the system minimizes the regularized squared error on the set of known ratings:



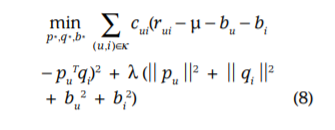
* **Alternating Least Squares (ALS):** Rotates between fixing the q and p vectors and solves by least squares.



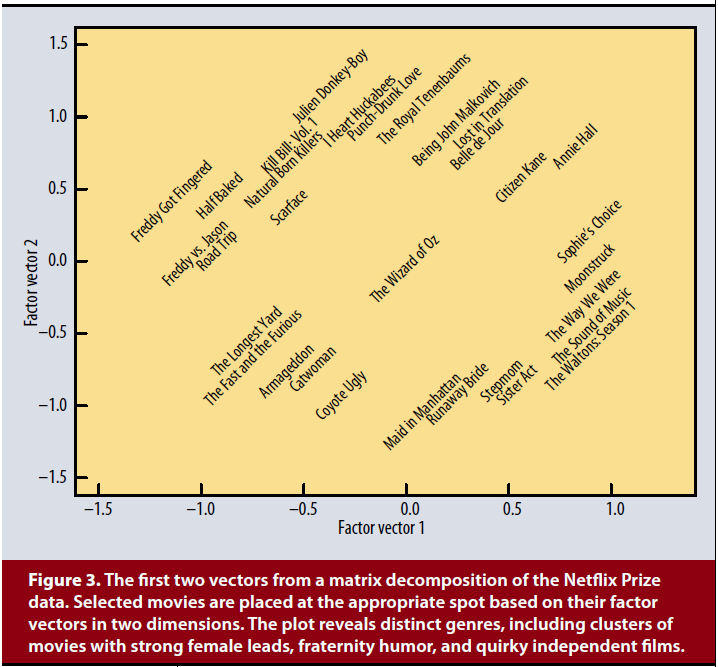
* + Can be done in parallel because each computation is independent of the others
* **Accounting for bias:** The first equation (rating) is affected by user bias (users that tend to rate high or low) and item bias (items that are typically higher rater or lower rated)



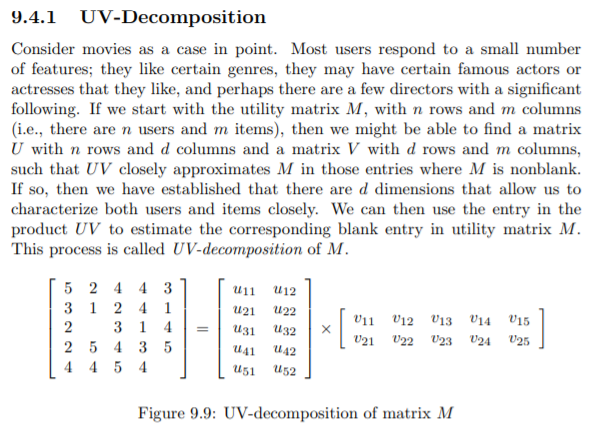
* We can also incorporate confidence in our predictions by applying a weight to each item that represents a numeric strength:



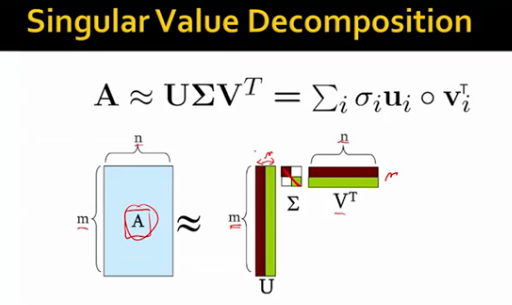
* In matrix factorization, we are given the final matrix (user-item ratings) and we try to back into the user vectors and item vectors. Depending on the method that we use, we will get a different number of components of the vectors. Remember, each vector represents features associated with a particular item and the user preferences for those features.
  + Each factor vector represents the vector across all items/users associated with one particular feature. Example below:



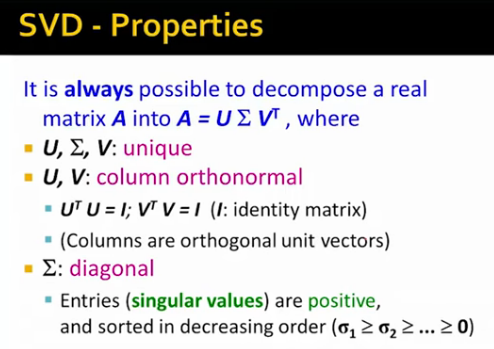
* We can use **Singular Value Decomposition** to take the user-item ratings matrix and split it into the separate user and item matrices.

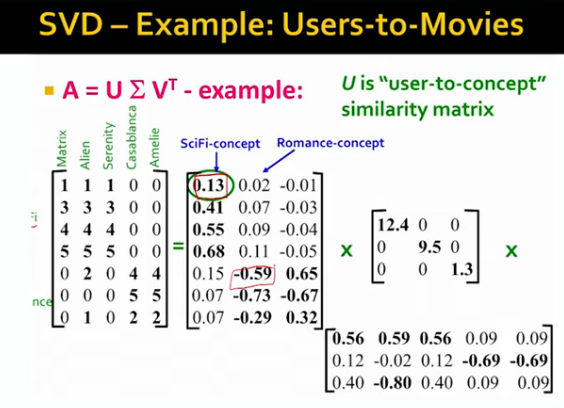


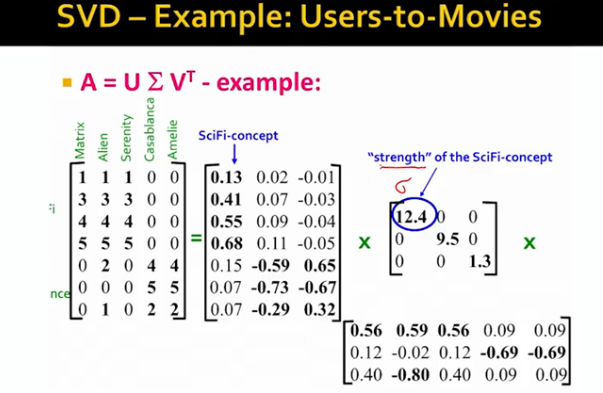
* + In this instance, we can consider the first matrix our user-item matrix, our second matrix as the user matrix and the third as the item matrix

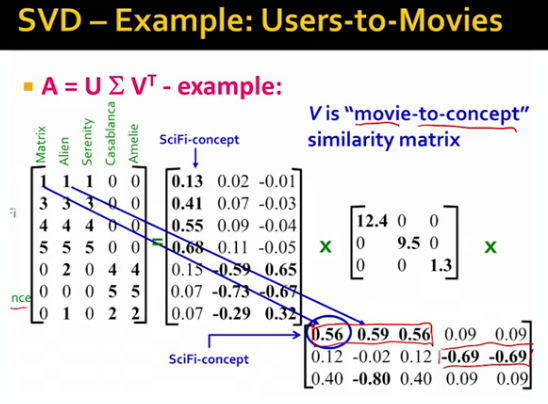


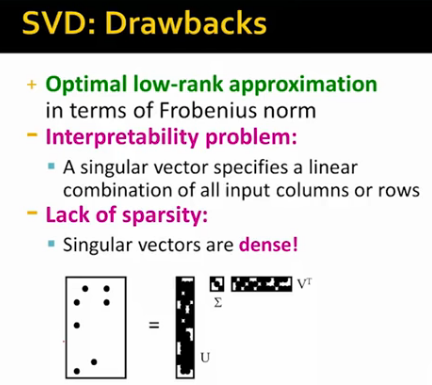
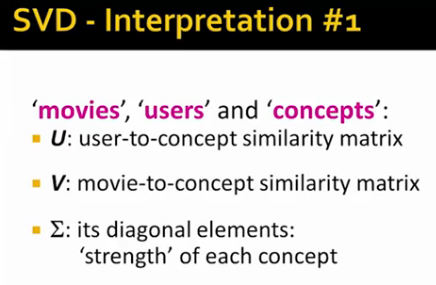
* m: number of users, n: number of movies, r: number of features
* **U** represents the “user-to concept” matrix: it is the affinity of each user for that particular concept
* **Sigma** represents the concept matrix: it is the strength of a particular feature
* **V** represents the “movie-to-concept” matrix: it is the strength of the movie to that particular concept











* If we allow ourselves to take my data and represent it using a small number of dimensions, then SVD will be able to identify the best possible number of dimensions and the sum of the squares of the error are minimized