CSCE 670 Final Review Sheet

Modeling Text

✸ **TF**: term frequency. $tf_{t,d}$ = Number of occurrences of term t in document d. Determines the importance of a word by its occurrence in a document; neglects ordering, prefers frequent words.
The **log frequency weight** of term t in d is defined as $w_{t,d} = 1 + log_{10} tf_{t,d}$ if $tf_{t,d} > 0$, 0 otherwise. Better scaling of the raw TF.
✸ **IDF**: inverse document frequency. $idf_t = log \frac{N}{df_t}$, where df_t = the number of documents in the collection that contain a term t. It rewards rare terms. It's a measure of the informativeness of the term.
✸ **TF-IDF**: a combined weighting scheme.
$tf - idf_{t,d} = tf_{t,d} \times idf_t$. It ↑ w/ the # of occurrences within a doc (TF), or with the rarity of the term in the collection (IDF).
❂ **Vector Space Model**: $\vec{V}(d)$ is the vector derived from document d, with one component in the vector for each dictionary term. Very high dim but sparse! (IR fundamental)
❂ **Cosine similarity**: the standard way of quantifying the similarity between two documents d_1 and d_2. Assume that vector representations are $\vec{V}(d_1)$ and $\vec{V}(d_2)$.
$\therefore sim(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}$
Note: same idea for the cosine similarity could be applied to normalizing weights as well. Normalized weight = $\frac{ori_{weight}}{|Doc-vector|}$ (ori_{weight} could be tf, log tf, ...)
❂ **Zipf's law**: The i^{th} most frequent term has frequency cf_i proportional to $1/i$. That is, $cf_i \propto \frac{1}{i}$

Evaluation

✎ **Test collections**: consist of 1. A document collection, 2. A test suite of information needs, expressible as queries, 3. A set of relevance judgments, standardly a binary assessment of either relevant or nonrelevant for each query-document pair.
✎ **Ground truth**: decision over a doc (rel or not).
✎ Relevance is assessed relative to an **information need** rather than a query. Info needs got translated into queries.
✎ **McNamara's fallacy**: don't measure easy stuffs but neglect the true objective
⋆ **CG**: cumulative gain. The CG at a particular rank position p is defined as: $CG_p = \sum_{i=1}^{p} rel_i$, where rel_i is the graded relevance of the result at position i. (unaffected by changes in the ordering of search results)
⋆ **DCG**: discounted cumulative gain. The DCG at a particular rank position p is defined as:
$DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i}-1}{log_2(i+1)}$. It penalizes rel doc showing up lower in the search result list. Also assuming highly rel docs are more useful.
⋆ **NDCG**: Normalized DCG. For a query, its NDCG at a particular rank position p is defined as:
$NDCG_p = \frac{DCG_p}{IDCG_p}$, where IDCG = ideal discounted cumulative gain = $\sum_{i=1}^{|REL|} \frac{2^{rel_i}-1}{log_2(i+1)}$. This metric is used to compare a search engine's performance between queries (as queries might have different lengths).
❤ **Precision**: the fraction of relevant instances among the retrieved instances. = $\frac{tp}{tp+fp}$ = P(relevant|retrieved)
❤ **Precision@k**: measures precision but up to position k. Note that this is a ranked evaluation.
❤ **Recall**: the fraction of relevant instances that have been retrieved over the total amount of relevant instances. = $\frac{tp}{tp+fn}$ = P(retrieved|relevant)
There is an inverse relationship between precision and recall, which means that it's possible to optimize one at the expense of the other. (One could ↑ precision by returning more docs; recall is a non-decreasing function of the number of docs retrieved.)
❤ **F-measure**: a measure of a test's accuracy. It is defined as the weighted harmonic mean of the precision and recall of the test. F = $2 * \frac{precision \cdot recall}{precision+recall}$
Note: NDCG is better than precision and recall because it naturally incorporates **graded relevance** and order. Also precision/recall cannot distinguish duplicates; neither do them consider rank positions as well. Precision/recall rely on binary assessment, and they're both heavily skewed by corpus/authorship.

Basic ML

❑ **Loss function**: measures how "bad" a system's prediction is in comparison to the truth. In particular, if y is the truth and $\hat{y}$ is the system's prediction, then $l(y, \hat{y})$ is a measure of error. (Regression: squared loss or absolute loss, binary/multiclass classification: zero/one loss. SVM: hinge loss)
❑ **Regularization**: controls the complexity of the function (to avoid overfitting per se)

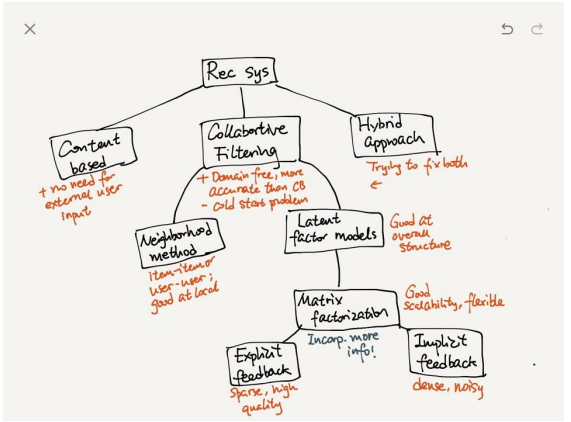
Rocchio

Classical, simple; uses centroids to define the boundaries. The centroid of a class c is computed as the vector average or center of mass of its members:
$\vec{\mu}(c) = \frac{1}{|D_c|} \sum_{d \in D_c} \vec{v}(d)$. Do this on all the training examples.
Draw boundaries (hyperplanes). The boundary between two classes in Rocchio classification is the set of points with equal distance from the two centroids. Then for unseen x, classify it in accordance with the region it falls into.
! Does not guarantee that classifications are consistent with the training data!

KNN

A supervised, non parametric (no assumption of the underlying data dist.) and instance-based learning algorithm. Good for getting a pretty accurate classifier up and running in a short time w/o constraints on efficiency.
Given a positive integer K, an unseen observation x and a similarity metric d:
1.computes d between x and each training obs. through the whole dataset. Set A = K points in the training data that are closest to x.
2.estimates the conditional probability for each class: $P(y = j|X = x) = \frac{1}{K} \sum_{i \in A} I(y^{(i)} = j)$
3.assigns x to the class with the largest prob.
K is indeed a hyperparameter. (Basically the category of x is determined by its k^{th} nearest neighbors)
Increasing k will decrease variance and increase bias. Decreasing it will go the other way. If we increase k too much, then we no longer follow the true boundary line and we observe high bias. (tradeoff)
✓ No training necessary; accurate w/ large training data; linear training time
✗ Not accurate if not enough training data

Recommenders overview



Content-based recommendation

✯ **Basic idea:** recommend items to customer x similar to previous items rated highly by x.

✯ **Item representation:** For each item, create an item profile consisting of a set of feature vectors. (Pick important words using heuristics or TFIDF. "Handtuned" similar to BM25)

✯ **User representation:** Model users as collections of rated item profiles. Could be weighted average, weight by difference from average rating for item, etc.

✯ **Prediction**: Given user profile x and item profile i, estimate $u(x,i) = cos(x,i)$

✯ **Pros of CB**: Do not need other users' data - no cold start/sparsity. Can recommend to users with unique tastes, and new/unpopular items. Can justify recommendations.

✯ **Cons of CB**: Find appropriate features, new user problem, overspecialization (never recommend stuffs outside the content profile, unable to exploit users' multiple interests & quality judgments of other users), indiscernible problem when two different items share the same set of features

Basic collaborative filtering

✰ **Basic idea:** The user will be recommended items that...

user-user: For user x, find set N of other users whose ratings are "similar" to xs ratings. Estimate xs ratings based on ratings of users in N.

item-item: For item s, find other similar items. Estimate rating for item based on ratings for similar items. Can use same similarity metrics and prediction functions as in user-user model; normally works better

(aka neighborhood approach)

✰ ** **Ways to measure similarity**:

Jaccard index: $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$. Count the # of items purchased. Easy to interpret but ignores the value of the rating.

Cosine: same as the cosine similarity above. It works for arbitrary vectors but treats missing ratings as "negative".

Pearson correlation: Used when there are numerical ratings. Subtract average from ratings first then compared.

$sim(u,v) = \frac{\sum_{i \in I_u \cap I_v}(R_{u,i} - \bar{R}_u)(R_{v,i} - \bar{R}_v)}{\sqrt{\sum_{i \in I_u \cap I_v}(R_{u,i} - \bar{R}_u)^2 \sum_{i \in I_u \cap I_v}(R_{v,i} - \bar{R}_v)^2}}$

(only consider things in common)

✰ **Aggregate ratings**: multiple methods possible.

Simple average: $r_{c,s} = \frac{1}{N} \sum_{c' \in \hat{C}} r_{c',s}$

Weighted sum: $r_{c,s} = k \sum_{c' \in \hat{C}} sim(c,c') \times r_{c',s}$ (most common, but fails to realize the different rating scale among users)

Adjusted weighted sum:

$r_{c,s} = \bar{r_c} + k \sum_{c' \in \hat{C}} sim(c,c') \times r_{c',s} - \bar{r}_{c'}$ (overcome issues above)

✰ **Pros of CF**: Works for any kind of item w/o feature selection. Domain free, more accurate than CB.

✰ ✈ **Pros of neighborhood models**: able to handle new users/ratings without re-training, offer direct explanations to the recommendations.

✰ ✉ **Pros of latent factor models**: good for overall structures in general, makes items and users directly comparable in the same latent factor space

✰ **Cons of CF**: Cold start (need enough users), sparsity of the matrix, first rater (cannot recommend an item that has not been previously rated), and popularity bias (popular items more favorable, not able to accommodate weirdos)

✰ ✈ **Cons of neighborhood models**: too concentrate on localized relationships, ignoring weak signals

✰ ✉ **Cons of latent factor models**: poor at localized connections

✰ **Possible extensions (CB/CF)**: 1) improve the understanding of users and items; 2) incorporate the contextual information into the recommendation process; 3) support multcriteria ratings; 4) provide more flexible and less intrusive types of recommendations.

Evaluation (Recommenders)

✱ ** **RMSE**: $RMSE = \sqrt{\frac{1}{n} \sum_{j=1}^{n} (y_j - \hat{y}_j)^2}$. Compare predictions with known ratings; the lower the better. Usually averages around a large user group.

✱ **Coverage**: measures the % of items for which a rec sys is capable of making predictions.

✱ **Accuracy**: could be statistical or decision-support. Stat: estimated vs ground truth, e.g. RMSE, MAE, corr. Decision-support: determines how well a rec sys can make pred of high rel items, e.g. precision, recall, F-measure, ROC.

✱ **Diversity** measures how many distinct items a rec sys recommends to users, whereas **Novelty** refers to how different it is with respect to "what has been previously seen", by a specific user, or by a community as a whole.

✱ **Limitations of the metrics**: 1) biased, do not test how well a sys evaluates a random item not explicitly rated by users. 2) measures do not capture "usefulness"/"quality" of the recommendations → study other measures that may bring business values, e.g. ROI!

BellKor approach to Netflix Prize

✲ ** **global**: baseline rating = mean item rating μ + user-bias b_u + item-bias b_i

✲ **regional**: Latent factor to capture user-item interaction. $q_i^T(p_u + |N(u)|^{-1/2} \sum_{j \in N(u)} y_j)$

✲ **local**: item-item CF. Base form: $\hat{r}_{xi} = \frac{\sum_{j \in N} s_{ij} * r_{ij}}{\sum_{j \in N} s_{ij}}$ where N are the items similar to i that have been rated by x, s_{ij} is the similarity between item i and item j. Eventually: $\sum_{j \in N}(r_{uj} - b_{uj})w_{ij}$ (weight to be learned) + some normalized terms

✲ Combining global, regional and local: a 3 tier integrated model, able to further decrease RMSE (to 0.8868) but comes with longer training time.

✲ **Temporal component**: affects item biases, $b_i(t)$; user biases, $b_u(t)$; and user preferences, $p_u(t)$. **These effects were strong!!** Since user preferences may drift over time, user-item rating interaction changes accordingly. Nonetheless, item characteristics q_i stays the same.

Matrix Factorization / Latent Factor Models

✳ **SVD**: original version - $A = U\Sigma V^T$. One can always decompose a real matrix A. Think of it as U is the user-to-concept similarity matrix, V is the movie-to-concept similarity matrix, and Σ measures the 'strength' of each concept.

Dim reduction of SVD: Set smallest singular values to zero, cross out the corresponding rows and cols in U, Σ and V. Also possible with PCA.

Version used in recommending sys: $R = QP^T$. Need to extract latent factors for users and items. Unknown ratings can be estimated as inner-products of factors.

✳ **Pros of MF**: useful and applicable in a wide area of applications; memory efficient!

✳ **Issues with SVD**: treat missing values as 0, but the assumption is invalid. So just train the model using the presented values in the matrix.

✳ To estimate a rating given latent factors:

$\hat{r}_{ui} = b_{ui} + p_u^T q_i$ (combine with the global term) OR...

✳ To estimate unknown ratings for user u on item i as inner-products of factors: use the ith row in the item-factor matrix and the uth row in the factor-user matrix. (1*n vector * n*1 vector = 1*1 rating)

✳ **optimization setting**: goal is to minimize SSE and avoid overfitting. Add regularization terms to allow rich model where there are sufficient data & shrink aggressively where data are scarce.

stochastic gradient descent: ✓ faster, works well on sparse matrices; ✗ costly on dense matrices

alternating least squares (ALS): ✓ can potentially exploit parallelization, works fine for implicit data.

✳ **Query example**: Map query into a "concept space". Say that a user has rated only 1 movie (Matrix). In order to get the strength of each concept: multiply the user rating vector by the movie-to-concept similarity matrix V. That is, $q_{concept} = qV$.

Implicit feedback

✳ **SVD++**: improved model - $\hat{r}_{ui} = b_{ui} + q_i^T(p_u + |N(u)|^{-1/2}\sum_{j\in N(u)} y_j)$, where p_u are the latent factors from explicit ratings, $N(u)$ is set of items that user u has implicit feedback on.

✓ more accurate than the original SVD/asymmetric SVD

✗ more parameters, not handling new users well, results not easily inteprable

✳ **** Differences between explicit and implicit feedback**: *implicit* - no negative feedback, inherently noisy, indication of confidence, tricky evaluation. Usually dense. *explicit* - indication of preferences. High quality, most convenient. Usually sparse.

✳ Note: for the framework - it's similar to MF on explicit feedback but: 1) an implicit feedback r_{ui} will always be decomposed into two parts: p_{ui} and c_{ui}. 2) optimization should be based upon all possible u, i pairs.

✳ **Confidence/preference framework**: now, r_{ui} captures user actions. User preference $p_{ui} = 1$ if $r_{ui} > 0$, 0 otherwise. If a user consumes an item, then there is some preference for it. c_{ui} = Confidence in observing $p = 1 + \alpha r_{ui}$. Confidence goes up as a user consumes an item more.

Why confidence level? 1) not all item deserved the same weight, 2) to avoid attacks from adversarial users, 3) system could be implicit data-based.

✳ **Final model**: optimization problem = $\min_{x*,y*}\sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(\sum_u ||x_u||^2 + \sum_i ||y_i||^2)$. It should be based upon all possible u, i pairs. Dense cost function → use alternating least square to estimate (linear time possible).

✳ **Experiment**: on digital TV service. Quality measure: the expected percentile ranking (EPR) of a watching unit in the test period (the lower the better). (Better than neighborhood/popularity)

✳ Standard training/testing split not applicable because a rec sys should identify items that users have not been exposed to before.

BPR

✳ **Motivation**: Given a "one-class" prediction task (like purchase prediction) we might want to optimize a ranking function rather than trying to factorize a matrix directly

✳ **Goal**: to count how many times we identified item i as being more preferable than item j for a user u.

For each user, estimate a personalized ranking function $x(u, i, j)$. Positive if i is preferred, negative otherwise. (Similar to ranking SVM, not the actual rating but relative position)

✳ **Evaluation metric = Area under the curve AUC**. It counts how many times the model correctly identifies that u prefers the item they bought (positive feedback) over the item they did not. AUC = 1.0 ↔ we always guess correctly. (Optimize AUC approx. by maximizing $\sigma(\hat{x}_{uij})$)

✳ **** How this works:**

Want to know: $P(\theta| >_u)$. Per Bayesian: $P(\theta| >_u) \propto p(i >_u j|\theta) * p(\theta)$.

$\sigma(\hat{x}_{uij})$ is a smooth function of the original counting function $\delta(\hat{x}_{uij} > 0)$. Similar to hinge loss in SVM, the loss is no longer 0/1 but smoothed by the logistic sigmoid transformation. It is equivalent to $p(i >_u j|\theta)$, which stands for the individual probability that a user really prefers item i to item j.

? *Why smooth*? One cannot simply state that the indiv. prob. is 0 even though no obs. is available. This is a more conservative bound!

$\hat{x}_{uij}(\theta)$: an arbitrary model capturing the relationship between user u, item i and item j. Could be matrix factorization, adaptive KNN, etc. $\hat{x}_{uij} := \hat{x}_{ui} - \hat{x}_{uj}$

Prior density function $p(\theta)$ can be approx. as $N(0, \sum_\theta)$ (a normal distribution with zero mean and variance-covariance matrix $\sum_\theta$)

✳ New optimization criterion: BPR-OPT = $\ln P(\theta| >_u)$ = ... = $\sum_{(u,i,j)\in D_S} \ln\sigma(\hat{x}_{uij}) - \lambda_\theta||\theta||^2$

New learning algorithm: LearnBPR (based on stochastic gradient descent - taking first derivative of BPR-OPT, but gets the flavor of bootstrap sampling)

✳ **Experiments**: using the Rossmann online shopping dataset and the Netfilx video rental dataset. For MF, BPR is better than SVD and WR (another criterion). But for kNN, BPR hardly beats the cosine-kNN model. BPR always better than the most popular model. → good for personalized ranking!

Attacks

Shilling Recommender Systems for Fun and Profit, Lam and Riedl 2004: (first paper on rec sys attacks, data from MovieLens)

✶ **Motivation**: Push or nuke certain items, or joker. Can manipulate reviews!

✶ **Possible approaches**: *naive* - create many fake accounts and issue high or low ratings to the "target item". May be good for 1-nn, but can easily be detected by the rec and nullify the effort!!

Random - Take random values for filler items , assuming that typical distribution of ratings is known. ✗ Not that effective but better than the naive one...

Average - Use the individual item's rating average for the filler items. ✗ Costly as it needs to find out the average rating of an item; ✓ More effective than Random Attack in user-based CF.

✶ **4 hypotheses investigated**:

Hypo 1: Different ACF algorithms respond differently to shilling attacks. (Yes - item-item and user-user respond differently)

Hypo 2: Shilling attacks affect recommender algorithms differently from prediction algorithms. (yeah, somehow, choose the metric wisely)

Hypo 3: Shilling attacks are not detectable using traditional measures of algorithm performance. (Neither confirm nor deny. Think about other metrics please!)

Hypo 4: Ratings distribution of the target item influences attack effectiveness. (Rejected - the authors look into correlation between [popularity, entropy, likeability] and [PredShift, ExpTop40] for the 2 algo. Some of them are linearly correlated, some are not.)

✶ **Takeaway**: use item-item algorithm - this conclusion could be biased as the authors only considered creating fake users! Push stronger than nuke on user-user, but less effective on item-item (whereas nuke stays roughly the same). Use recommendation metrics (e.g. Expected Top-N Occupancy). Watch (trad. Like MAE) metrics, but worry anyway. Protect new items (easier to be attacked). More work possible under the intent, target, knowledge dimensions etc.

✶ **Other strategies**: *Bandwagon Strategy* - Add profiles that contain high ratings for "blockbusters" (in the selected items, e.g. HP); use random values for the filler items. ✓ Low-cost attack; Does not require additional knowledge about mean item ratings

Segment Strategy: Find items that are similar to target item and give them high ratings (to target a specific group), random or lower ratings for other genres. The

target item will be pushed to the relevant community. ✗ Additional knowledge (e.g. genre of a book) is required

✶ **Issues:** cost, algorithm dependability, detectability

✶ **TwoFace**: identify review manipulators from crowdsourced websites. Hard to detect coz they write mixed reviews: half authentic half fake. Data from Amazon & crowdsourced websites. Starting with seed users who reviewed 2+ target products. Propagate suspiciousness to nearby users through the graph edges (using TrustRank - random walk w/ restart). Uncover (hidden) distant users who serve structurally similar roles using review graph embeddings (node2Vec).

High recall in presence of very few seeds, but low precision (too many misclassification). Can uncover many fraudulent reviewers. Better than trad feature approaches like D-cube. Suspicious propagation is needed!

Dark Side / Fake News

✷ Revise peer review process to also focus on negative impacts of research

✷ Algorithm discrimination: e.g. online behavioral advertising (more lucrative jobs for men), bias towards races, etc.

✷ **Sources of concern**: data and algorithms.

✷ Source of biases: could come from the imbalanced dataset, human biases, or majority perception in data distribution.

✷ misinformation vs disinformation: Both info is false, but the person who is disseminating misinformation believes that it is true, while the person who is disseminating disinformation knows it is false - a deliberate, intentional lie.

✷ Ways to gather **ground truth**:

❝ Encode our intuitions - mostly ruled based, biased

❝ Ask the Experts - ✓ reliable; ✗ costly, may not have access to all resources

❝ Exploit Crowd Signals - ✓ fast, cheaper; ✗ can amplify false info

❝ Rely on platform - ✓ more robust, directly embedded, could notify the source for falsehood; ✗ incentive not aligned between the platform and users, potential to be hacked

✷ Efficiency Improvement of Neutrality-Enhanced Recommendation, Kamashima et al. 2013: this paper proposes an information-neutral rec sys. Obj func: $\sum_D \text{loss}(r, \hat{r}(x, y, v)) + \eta\text{neutral}(R, V) + \lambda\text{reg}(\theta)$ (whereas $\hat{r}(x, y, v)$ can be MF)

Adjust the current rec scheme to incorporate the state of viewpoints. Add a neutrality term objective function. Accuracy (MAE) may suffer as the neutrality increases. The neutrality term quantifies the stat indep. btwn a rec result R and a viewpoint feature V. It could be calculated as:

Mutual information: negative mutual information between R and V. A really complicated model... not easy to calculate.

Calders-Verwer's Discrimination Score: measures the difference between distributions of target variable given $V = 0$ and $V = 1$: $PR(R|V = 0) - PR(R|V = 1)$. Could be in the form of r-match and m-match. ✓ easy to compute than MI above, hard to trap at local minima; ✗ not applicable for extension, m-match may not work well homogeneously on users but r-match takes longer time to compute, etc.

Paper Takeaways

✷ **Most of them should be noted above but below are just some additional details.**

✷ The BellKor 2008 Solution to the Netflix Prize, Bell et al. 2008:

Feature selection: Collaborative filtering is good at handling sparse matrices, but other features should be looked at as well (e.g. users deliberate rating on selected movies). Well need to be careful here, some of the other features turn out to be not that useful in terms of improving accuracy (e.g. movie descriptions).

Choose the right model: working with well designed models is also important!

Power of a single model: one can fit over 10 billion parameters but still achieve a higher accuracy! (not easily converge, able to handle really high, complex dimensions) It indicates a complex multifaceted nature of user-movie interaction.

Majority votes help: besides the complex collaborative filtering based model, a combination of simpler models also behaves well. Nonetheless, blending too many simple models does little good for an excellent system.

✷ Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model, Koren:

The paper presents two **improvements**: 1) a new neighborhood based model (+ global biases and implicit feedback); 2) an improved latent factor model (+ implicit feedback). The improved neighborhood model can integrate with the new latent factor model to make the 3-tier blended model (see BellKor section above).

Evaluation: measuring the success of a top-K recommender. For each movie i rated 5-star by user u, randomly draw 1000 movies + i. For the 1000 movies, predicted a rating for each of them, then rank the 1001 movies in a descend order based on their ratings. The hope is that i should somehow appears in the top K results, as u may not like all the 1000 movies. For each of the 384,573 ratings in the validation set, the experiment above was performed. A ranking between 0-100% is derived (the lower the better); the final distribution for all these movies was analyzed.

A major **insight** beyond this work is that improved recommendation quality depends on successfully addressing different aspects of the data, i.e. implicit feedback