Making Recommendations

Summary

Recommender systems have increasingly become a common part of everyday life. Common examples include the recommendations Amazon makes for you about products you might like to buy, and the movies Netflix recommends you might enjoy watching.

This lecture summarises the different types of recommender systems and looks in detail at a form of recommendation called Collaborative Filtering in which the past behaviour of users is captured in order to make predictions about what other items the users may like (or dislike). In particular we're going to look at *neighbourhood-based* Collaborative Filtering approaches.

Key points

Key Terminology

- Typically based around the idea of people (Users) *buying/using/wanting* some items.
    - items could be anything (even other people in the case of dating websites!)

Recommender systems fall into a number of different categories:

- Content-based approaches use characteristics of items in order to recommend other items with similar properties.
    - Doesn't rely on the users; only information about items is used to make predictions
- Systems based on collaborative filtering make use of users' past behaviour in order to recommend items.
    - Doesn't explicitly rely on the attributes of the items, only on user behaviour
- *Hybrid* recommendation systems fall in between and combine user information with content attributes of the items.

Types of Collaborative Filtering

- Neighbourhood-based (or Memory-based) work by comparing users or items based on the similarity user user ratings
- Model-based attempt to build "models" that better explain rating behaviour and make better predictions
- Hybrid approaches combine neighbourhood-based and model-based techniques

Collaborative filtering (CF) recommender systems

- Key idea: "Similar users like similar items"
    - Personal preferences are correlated
        - If Jack loves A and B, and Jill loves A, B, and C, then Jack is more likely to love C
    - If we can find out which users are similar to each other, we might be able to predict whether a user will like (or dislike) an item they have not *rated* before on the basis of the tastes of users that are similar.
- CF systems:
    - Discover patterns in observed preference behavior (e.g. purchase history, item ratings, click counts) across community of users

        - Predict new preferences based on those patterns

Collecting user preferences

- User preferences or ratings are the key to collaborative filtering
    - Can be collected implicitly:
        - e.g. by looking at what a user browsed and/or brought
    - or explicitly:
        - e.g. by asking users to provide ratings for things they interacted with
- To perform neighbourhood-based CF ratings need to be converted to numeric scores
- User preferences are usually sparse; not all users will have preferences recorded for all items
    - Requires special data structures to deal with efficiently
    - sparse vector of ratings for a user or item is a *featurevector*

Feature spaces

- A feature space is an abstract mathematical space defined by a feature extraction procedure that transforms raw data into featurevectors of some fixed number of elements.
    - A featurevector is just a list of numbers that represents a point in the corresponding feature space.
    - The vector can also be considered to represent a direction in the space.
- The number of elements of the vector is known as the dimensionality of the space (and the vector)

Distance and similarity

- Feature extractors are often defined so that they produce vectors that are close together for similar inputs (for some given notion of similarity between the input objects)
- Closeness of two vectors can be computed in the feature space by measuring a distance between the vectors.
- perhaps the most common distance measure is the Euclidean Distance or L2 distance, which represents the straight-line distance between two points p = $(p_1, p_2, ..., p_n)$ and q = $(q_1, q_2, ..., q_n)$:

$$\mathrm{D}_2(\mathbf{p},\mathbf{q}) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

    - and equivalently in vector form:

$$\mathrm{D}_2(\mathbf{p},\mathbf{q}) = ||\mathbf{p}-\mathbf{q}|| = \sqrt{(\mathbf{p}-\mathbf{q})^T(\mathbf{p}-\mathbf{q})}$$

- The L1 distance (aka taxicab or Manhattan distance) is also often used:

$$\mathrm{D}_1(\mathbf{p},\mathbf{q}) = ||\mathbf{p}-\mathbf{q}||_1 = \sum_{i=1}^{n}|p_i - q_i|$$

- The Lp distances are a generalisation to other orders:

$$\mathrm{D}_p(\mathbf{x},\mathbf{y}) = ||\mathbf{x}-\mathbf{y}||_p = (\sum_{i=1}^{n}|x_i - y_i|^p)^{\frac{1}{p}}$$

- The Cosine similarity is another commonly used measure:
$$\cos(\theta) = \frac{p \cdot q}{||p||||q||} = \frac{\sum\limits_{i=1}^{n} p_i q_i}{\sqrt{\sum\limits_{i=1}^{n} p_i^2}\sqrt{\sum\limits_{i=1}^{n} q_i^2}}$$
    - This is not a distance measure!
    - similarity=1 if vectors are in the same direction; decreases as angle increases
    - Useful if you only care about relative direction, but not magnitude

Measuring user similarity

- By considering the featurevector of ratings for each user we can compare users.
- But not all users have ratings for all items (i.e. the vectors are sparse)
    - Need to take this into account, typically by only computing similarity over items users have in common (denoted below by the set I_{xy} for users x and y)
    - for example:
        - "Euclidean Similarity":
$$\text{sim}_{L2}(x, y) = \frac{1}{1 + \sqrt{\sum\limits_{i \in I_{xy}} (r_{x,i} - r_{y,i})^2}}$$
        - "Pearson correlation":
$$\text{sim}_{pearson}(x, y) = \frac{\sum\limits_{i \in I_{xy}} (r_{x,i} - \bar{r_x})(r_{y,i} - \bar{r_y})}{\sqrt{\sum\limits_{i \in I_{xy}} (r_{x,i} - \bar{r_x})^2 \sum\limits_{i \in I_{xy}} (r_{y,i} - \bar{r_y})^2}}$$
            - Pearson correlation automatically adjusts for grade inflation, where users have differing ideas of absolute ratings but are relatively consistent across items
                - Grade inflation can be removed by data standardisation (mean-centring and normalising)

User-based Collaborative Filtering

- Ranking users
    - Typically want the *top-N* users most similar to a target user.
        - Compute the similarity between the target user and other users and choose the N users with the highest similarity.
        - Rather than considering all users, might want to only consider those that rated a specific item...
- Recommending items for a user
    - Predict the rating, $r_{u,i}$, of an item i by user u as an aggregation of the ratings of item i by users similar to u: $r_{u,i} = \text{aggr}_{\hat{u} \in U}(r_{\hat{u},i})$, where U is the set N of top users most similar to u that rated item i.
    - Possible aggregation functions:

- $r_{u,i} = \frac{1}{N} \sum_{\hat{u} \in U} r_{\hat{u},i}$
- $r_{u,i} = \frac{\sum_{\hat{u} \in U} \operatorname{sim}(u,\hat{u}) r_{\hat{u},i}}{\sum_{\hat{u} \in U} |\operatorname{sim}(u,\hat{u})|}$
- $r_{u,i} = \bar{r}_u + \frac{\sum_{\hat{u} \in U} \operatorname{sim}(u,\hat{u})(r_{\hat{u},i} - \bar{r}_{\hat{u}})}{\sum_{\hat{u} \in U} |\operatorname{sim}(u,\hat{u})|}$

Problems with user based CF

- Computationally expensive if there are lots of users
- Potential problems if there is not enough overlap between users (e.g. as a result of having large number of products)
    - i.e. doesn't work well with very sparse data

Item-based Collaborative Filtering

- Designed as a workaround to user-based CF problems
- Top-N items are computed and cached for every item
    - Only updated occasionally on the assumption that product similarity doesn't change as frequently as user similarity
- Recommendations are created by taking each item a user u has rated and scoring all products similar to those (excluding any the user has rated)
    - Rating for an unrated item $\hat{i}$ computed as follows: $r_{u,\hat{i}} = \frac{\sum_{i \in I} \operatorname{sim}(\hat{i}, i) r_{u,i}}{\sum_{i \in I} \operatorname{sim}(\hat{i}, i)}$, where I is the subset of N items similar to $\hat{i}$

Item-based versus user-based

- With small and dense data, performance of the two approaches is about equivalent
- Item-based CF has additional overheads:
    - Computation and storage of item similarities
    - More complex to implement
- But generally has better performance with large and sparse datasets

Problems with CF

- What happens when a new user or item is added?
    - don't have any ratings to compare them...
        - the cold start problem
- For new items, can use content-based techniques to assess similarity and bootstrap ratings
- For new users, need to get information from elsewhere
    - Questioning
    - Monitoring/tracking behaviour

Further Reading

- Chapter 2 of “Programming Collective Intelligence” gives a good overview of the basic techniques.
- Wikipedia has a good overview of Recommender Systems and Collaborative Filtering:
    - https://en.wikipedia.org/wiki/Recommender_system (https://en.wikipedia.org/wiki/Recommender_system)
    - https://en.wikipedia.org/wiki/Collaborative_filtering (https://en.wikipedia.org/wiki/Collaborative_filtering)
- Recommender systems (http://comp6237.ecs.soton.ac.uk/reading/summary_recommender_systems.pdf) , Melville and Sindwhani, Encyclopaedia of Machine Learning, 2010
- Amazon.com recommendations: item-to-item collaborative filtering (http://comp6237.ecs.soton.ac.uk/reading/amazon_recommender_system_2003.pdf) , Linden, Smith, and York, 2003 (overview of the basic components of Amazon’s recommender system)
- Methods and Metrics for Cold-Start Recommendations (http://citeseer.ist.psu.edu/schein02methods.html) , Schein, Popescul, Ungar and Pennock, SIGIR 2002.
- Recommender systems: from algorithms to user experience (http://comp6237.ecs.soton.ac.uk/reading/recommendations_from_algorithms_to_user_experience_2012.pdf) , Konstain and Riedl, 2012 (emphasizes that the user experience is important, not just predictive accuracy)
- Item-based collaborative filtering recommendation algorithms (http://dl.acm.org/citation.cfm?id=372071) , Sarwar et al., In proc. WWW 2001.
- Audioscrobbler: Real-time Data Harvesting and Musical Collaborative Filtering (http://comp6237.ecs.soton.ac.uk/reading/audioscrobbler.pdf) , Jones, ECS Project Report, 2003 (The original description of the Audioscrobbler CF system developed as part of an ECS third year project)

Practical exercises

- Have a play with the demos in the slides and make sure the results you get match those you compute by hand on the same data.
- Implement cosine similarity and see what effect it has compared to Pearson correlation
- Try playing with a bigger dataset - you can use `MovieData.loadMovieLens100K()` instead of `MovieData.SMALLDATA` in the interactive slides to load the MovieLens100K dataset, which has 100,000 ratings from 1000 users over 1700 movies!