# CBF Programming Assignment

## Overview

In this assignment, you will implement a content-based recommender as a LensKit recommender algorithm. LensKit provides tools to produce recommendations from a user; your task is to implement the logic of the recommender itself.

There are 2 parts to this assignment, implementing two variants of a TF-IDF recommender.

## Downloads and Resources

- Project template (on course website)
- LensKit for Teaching website (links to relevant documentation and the LensKit tutorial video)
- JavaDoc for included code

Additionally, you will need:

- Java — download the Java 8 JDK. On Linux, install the OpenJDK 'devel' package (you will need the devel package to have the compiler).
- An IDE; I recommend IntelliJ IDEA Community Edition.

## Notation

Here's the mathematical notation we are using:

$\vec{u}$ The user's vector (in this assignment, the user profile vector).
$\vec{i}$ The item vector.
$I(u)$ The set of items rated by user u.
$u_t$, $i_t$ User u's or item i's score for tag t
$r_{ui}$ User u's rating for item i.
$\mu_u$ The average of user u's ratings.

## Part 1: TF-IDF Recommender with Unweighted Profiles (85 points)

Start by downloading the project template. This is a Gradle project; you can import it into your IDE directly (IntelliJ users can open the build.gradle file as a project). The code should compile as-is; you can test this by running the `build`

Gradle target from your IDE, or running `./gradlew build` at the command line.

There are 3 things you need to implement to complete the first part of the assignment:

**Compute item-tag vectors (the model)** For this task, you need to modify the model builder (`TFIDFModelBuilder`, your modifications go in the get() method) to compute the unit-normalized TF-IDF vector for each movie in the data set. We provide the skeleton of this; TODO comments indicate where you need to implement missing pieces. When this piece is done, the model should contain a mapping of item IDs to TF-IDF vectors, normalized to unit vectors, for each item.

**Build user profile for each query user** The `UserProfileBuilder` interface defines classes that take a user's history – a list of ratings — and produce a vector representing that user's profile. For part 1, the profile should be the sum of the item-tag vectors of all items the user has rated positively (>= 3.5 stars); this implementation goes in `ThresholdUserProfileBuilder`.

**Generate item scores for each user** The heart of the recommendation process in many LensKit recommenders is the score method of the item scorer, in this case `TFIDFItemScorer`. Modify this method to score each item by using cosine similarity: the score for an item is the cosine between that item's tag vector and the user's profile vector. Cosine similarity is defined as follows:

$$cos(u, i) = \frac{\vec{u} \cdot \vec{i}}{\|\vec{u}\|_2 \|\vec{i}\|_2} = \frac{\sum_t u_t i_t}{\sqrt{\sum_t u_t^2}\sqrt{\sum_t i_t^2}}$$

You can run your program from the command line using Gradle:

```
./gradlew recommendBasic -PuserId=42
```

Try different user IDs.

### Example Output for Unweighted User Profile

The following example gives actual outputs for users 42 and 91 in the data set. It was executed using `./gradlew recommendBasic -PuserId=42,91` in a Unix-like console.

```
recommendations for user 42:
  862 (Toy Story (1995)): 0.287
  557 (Spider-Man (2002)): 0.251
  11 (Star Wars: Episode IV - A New Hope (1977)): 0.196
  1892 (Star Wars: Episode VI - Return of the Jedi (1983)): 0.191
  9741 (Unbreakable (2000)): 0.182
  807 (Seven (a.k.a. Se7en) (1995)): 0.180
  812 (Aladdin (1992)): 0.180
  752 (V for Vendetta (2006)): 0.177
  141 (Donnie Darko (2001)): 0.175
  1891 (Star Wars: Episode V - The Empire Strikes Back (1980)): 0.170
recommendations for user 91:
  9806 (The Incredibles (2004)): 0.259
  2164 (Stargate (1994)): 0.238
  9741 (Unbreakable (2000)): 0.228
  807 (Seven (a.k.a. Se7en) (1995)): 0.209
  812 (Aladdin (1992)): 0.191
  36658 (X2: X-Men United (2003)): 0.184
  141 (Donnie Darko (2001)): 0.171
  550 (Fight Club (1999)): 0.171
  36657 (X-Men (2000)): 0.168
  393 (Kill Bill: Vol. 2 (2004)): 0.157
```

## Part 2: Weighted User Profile (15 points)

For this part, adapt your solution from Part 1 to compute weighted user profiles. Put your weighted user profile code in `WeightedUserProfileBuilder`.

In this variant, rather than just summing the vectors for all positively-rated items, compute a weighted sum of the item vectors for all rated items, with weights being based on the user's rating. Your solution should implement the following formula:

$$\vec{u} = \sum_{i \in I(u)} (r_{ui} - \mu_u)\vec{i}$$

### Example Output for Weighted User Profile

The following example gives actual outputs for users 42 and 91 in the data set. It was executed using `./gradlew recommendWeighted -PuserId=42, 91` in a Unix-like console.

```
recommendations for user 42:
  13 (Forrest Gump (1994)): 0.095
  862 (Toy Story (1995)): 0.090
  1422 (The Departed (2006)): 0.070
  1892 (Star Wars: Episode VI - Return of the Jedi (1983)): 0.060
  1891 (Star Wars: Episode V - The Empire Strikes Back (1980)): 0.057
  11 (Star Wars: Episode IV - A New Hope (1977)): 0.056
  568 (Apollo 13 (1995)): 0.053
  581 (Dances with Wolves (1990)): 0.051
  7443 (Chicken Run (2000)): 0.049
  812 (Aladdin (1992)): 0.046
recommendations for user 91:
  77 (Memento (2000)): 0.172
  393 (Kill Bill: Vol. 2 (2004)): 0.172
  807 (Seven (a.k.a. Se7en) (1995)): 0.170
  9741 (Unbreakable (2000)): 0.152
  24 (Kill Bill: Vol. 1 (2003)): 0.150
  550 (Fight Club (1999)): 0.101
  107 (Snatch (2000)): 0.089
  278 (The Shawshank Redemption (1994)): 0.078
  1900 (Traffic (2000)): 0.063
  238 (The Godfather (1972)): 0.055
```

## Submitting

As with all programming assignments in this class, you are strongly encouraged to work with a partner. Not only does working with a partner make the assignment easier and more educational, it also makes it faster to grade! When submitting the assignment, please include a name and ID number for both you and your partner. Only one submission is necessary. Submit your code as a zip file to the TA (taijala@cs.umn.edu).

To create this zip file, please use the pre-created archive functionality in the Gradle build:

```
./gradlew prepareSubmission
```

This will ensure that your submission contains all required files. It will produce a submission file in `build/distributions`.