# 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:
  2081 (Little Mermaid, The (1989)): 0.500
  2671 (Notting Hill (1999)): 0.292
  2724 (Runaway Bride (1999)): 0.286
  1265 (Groundhog Day (1993)): 0.241
  58299 (Horton Hears a Who! (2008)): 0.224
  96861 (Taken 2 (2012)): 0.215
  1210 (Star Wars: Episode VI - Return of the Jedi (1983)): 0.187
  1202 (Withnail & I (1987)): 0.127
  26048 (Human Condition II, The (Ningen no joken II) (1959)): 0.110
  260 (Star Wars: Episode IV - A New Hope (1977)): 0.109
```

```
recommendations for user 91:
  4816 (Zoolander (2001)): 0.343
  8376 (Napoleon Dynamite (2004)): 0.252
  1247 (Graduate, The (1967)): 0.232
  4361 (Tootsie (1982)): 0.210
  6350 (Laputa: Castle in the Sky (Tenkû no shiro Rapyuta) (1986)): 0.169
  2167 (Blade (1998)): 0.161
  2858 (American Beauty (1999)): 0.159
  6947 (Master and Commander: The Far Side of the World (2003)): 0.158
  5459 (Men in Black II (a.k.a. MIIB) (a.k.a. MIB 2) (2002)): 0.158
  2724 (Runaway Bride (1999)): 0.151
```

## 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:
  2081 (Little Mermaid, The (1989)): 0.191
  2724 (Runaway Bride (1999)): 0.109
  1265 (Groundhog Day (1993)): 0.107
  2671 (Notting Hill (1999)): 0.093
  96861 (Taken 2 (2012)): 0.082
  58299 (Horton Hears a Who! (2008)): 0.077
  1202 (Withnail & I (1987)): 0.056
  1295 (Unbearable Lightness of Being, The (1988)): 0.047
  7361 (Eternal Sunshine of the Spotless Mind (2004)): 0.039
  3147 (Green Mile, The (1999)): 0.036
recommendations for user 91:
  4816 (Zoolander (2001)): 0.438
  8376 (Napoleon Dynamite (2004)): 0.323
  1247 (Graduate, The (1967)): 0.296
  4361 (Tootsie (1982)): 0.268
  6350 (Laputa: Castle in the Sky (Tenkû no shiro Rapyuta) (1986)): 0.216
  2167 (Blade (1998)): 0.206
  6947 (Master and Commander: The Far Side of the World (2003)): 0.202
  5459 (Men in Black II (a.k.a. MIIB) (a.k.a. MIB 2) (2002)): 0.202
  2724 (Runaway Bride (1999)): 0.193
  2161 (NeverEnding Story, The (1984)): 0.190
```

## Submitting

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`.