Comparative performance of open-source recommender systems

Lenskit vs Mahout

Laurie James
This presentation

`Whistle-stop' tour of recommendation systems.

– Information overload & the need for recommenders:
  • Early solutions;
  • Impact of increasingly powerful computation on recommenders;
  • Industry interest.

– Collaborative Filtering & similarity matrices:
  • Early approaches - user-Item models & their drawbacks;
  • Dimensionality reduction;
  • Amazon - Item-item CF.

– This project:
  • Mahout and Lenskit (among others!);
  • What we’re testing;
  • The `show so far'
Information Overload  (for the theoretically inclined).

Issues (premises):

– We like to consume media, but have limited time;
– Some material is more enjoyable than others;
– There already exists enough media to fill a lifetime;
– And new material is being produced faster than it’s possible to consume!

• Maximisation problem:
  – Given your lifespan $T$, find the set of set of items that has the highest total enjoyment:
  – Maximise $\sum \beta_c$ such that $\sum a_t \leq T$
Information Overload: the problem

...But no two person’s tastes are identical, so the previous is (by definition) impossible to solve.

So, find some systematic way of selecting ‘good’ media (or filtering out ‘bad’ media).

Of huge industry relevance – 30—40% of Amazon’s sales come from automated recommendations.
   – And almost all of Netflix’s rentals.
An early solution.

- Trusted third parties – Radio presenters! Magazine reviewers! Friends! Family!
  - A `trusted’ source. Someone else samples more media than you can, and relays their opinion.
  - Assuming your tastes are similar, this should be effective.
  - But it’s not tractable – no-one can sample all the media, even if they’re working full-time.
  - Also, people have radically different tastes...

- Diversification – hire more people, split them up into subgroups!
  - Turtles all the way down...
Collaborative Filtering – a new challenger appears!

- Harness the power of the masses – have everyone rate media, find trends.

- Simplest model: everyone gives a yes/no rating.
  - Items with higher yes percentages are ‘better’;
  - Takes no account of individuals’ preferences;
  - Large enough samples should represent the whole population;
  - Items which score well with everyone generally do well – very specialised items obviously do not;
  - Might seem naive, but is actually in use – Rotten tomatoes, anyone?
  - Appropriate for certain domains.
Surely we can do better than that?

• Some types clusters of users have broadly similar tastes
  – (metal/pop/classical - action/romance/documentary...).
  – So give higher importance to ratings from ‘similar’ users.

• ‘User-Item’ recommendation.
  – For each user, build a user-item matrix [0,1,1...].
  – Then compute similarity between user pairs.
    • Cosine distance, or similar.
  – Find users with high similarity metrics.
    • Then recommend from their disjoint sets.
Close, but no cigar.

- With lots of tweaking for the dataset, user-item CF scores good precision and recall.

- But it’s computationally expensive
  - $O(M+N)$ average cost to recommend one item.
  - With databases in the tens of millions, this is prohibitive.

- Clustering & dimensionality reduction alleviate issues, but at the cost of performance.
  - SVD/LDA/K-means
Amazon and Item-Item CF

• The Amazon algorithm compares *items* to *items*, dropping the users.

• Items are likely to be `similar’ when bought or viewed together. Metadata probably helps.

• Essentially, build a giant Item-Item similarity matrix.
  – Very expensive to compute – worst-case $O(N^2M)$.
  – ...But we can do it offline!
  – With a pre-computed similarity matrix, recommendation is fast.
  – Periodically update similarity matrix to maintain best performance.
Open-source recommenders

- **Apache Mahout (Taste)**
  - Scalable machine learning library by Apache.
  - Runs on-top of Hadoop;
  - Covers many traditional ML problems, including clustering and Collaborative Filtering.
- **Lenskit**
  - Made by the GroupLens project, a leading research group in recsys;
  - Java, modular, very extensible.
- **EasyRec**
  - Quick deployment of simple recommender;
  - Web API.
- **MyMediaLite**
  - C#, otherwise similar to Lenskit.
This project

• Take Mahout and Lenskit, evaluate performance:
  – Accuracy & recall;
  – Time taken to bootstrap the dataset.

• Against different datasets:
  – Implicit/Explicit;
  – 100K, 1M, 10M ratings;

• Provide:
  – Tools for cleaning standard datasets (Python);
  – Implementation of DAOs to efficiently load certain standard datasets.
The show so far:

• Setting up environments – configuring Lenskit & Mahout.
  – (AKA the open-source documentation nightmare...)

• Cleaning implicit rating dataset.
  – 16M user-item pairs ripped straight from Last.fm API; lots of bad metadata.

• Simple user-item recommenders built & ready.
  – Both are Java, so we’re running in Tomcat to simulate deployment.
  – Preliminary test – Lenskit with 300k ratings took ~30 minutes to bootstrap.
    • 1M ratings was still running 8 hours later...

Next up:
• Accurately measure times, get precision & recall.
Thank you!

Questions...