Recommender system based on latent classes

Gaëlle Guillou, Marie Laval

Introduction

Recommender systems have been a major hype in data science recently with the growth of ad-targeting (Criteo, millemercis), media websites and services (Netflix, Google News, Spotify) for whom recommendation is a major revenue-driver. Recommender systems, who have been around since the 1990's, use in general content-based or collaborative approaches. Using movies as examples, content-based filtering is like saying: if you enjoyed Star Trek, you should like Star Wars, because they are both Science-Fiction movies. Collaborative approaches have proved themselves to be quite efficient in general: you empirically consider that if you have watched the Silence of the Lambs and loved it you should like the Shawshank Redemption even though it's not the same kind of movie, because most users who like one movie liked the other.

In this paper, we present the Probabilistic Semantic Latent Indexing approach to recommender systems, a collaborative system. Introduced by Hoffman in 1999, the PLSI approach introduces latent classes to recommend items to users. Users, like items, can belong to several latent classes. Those can be better apprehended if we consider them as groups gathering fans of similar movies, for example. To test our algorithm, we will be using the Movielens dataset from the Grouplens website. It is an aggregation of about twenty million ratings applied to 27,000 movies by 138,000 users.

Description of the PLSI algorithm

Background: assumptions, introducing latent classes (Hoffman and al., 1999)

For the sake of simplicity and realism - considering privacy regulations -, we assume that we have no external knowledge about the objects or the users.

Now, let's dive into the statistical model presented by Hoffman in 1999. We introduce three random variables: users u, objects s and a latent class variable z.The main statistical assumption we make is that u and s are independent conditioned on z. Therefore, we have:

$$P(u, s) = \sum_{z \in \mathcal{Z}} P(z) P(u \mid z) P(s \mid z)$$

This yields a most relevant result for our recommender system:

$$P(s \mid u) = \sum_{z \in \mathcal{Z}} P(z \mid u) P(s \mid z)$$

As Hoffman underlines it himself, there is no assumption that persons form groups or that objects form clusters, which allows for great flexibility regarding our recommendations.

Expectation-Maximization algorithm (Dempster and al., 1977)

For maximizing likelihood estimation in latent variable models, it is usual to use the Expectation Maximization algorithm.

By maximizing the log likelihood, we are minimizing the empirical logarithmic loss, defined here:

$$L(\theta) = -\frac{1}{T}\sum_{t=1}^{T} log\big(p(s \mid u;\theta)\big)$$

The EM algorithm consists of two steps:

- Expectation (E) step: computing posterior probabilities for latent variables, based on current estimates,
- Maximization (M) step: updating parameters for posterior probabilities.

As indicated, we will be using it in our own implementation of PLSI.

Our implementation of the PLSI algorithm, inspired by The Google News implementation (Das and al., 2007)

Our implementation is largely based on the Google News MapReduce implementation, as presented in Das and al., 2007.

The PLSI algorithm that we will implement here is based on Das description of the Google News recommendation system. The algorithm is based on the following :

- The relationship between users and movies is learned by modeling the joint distribution of users and items as a mixture distribution,
- To capture this relationship, we introduce the hidden variable z (latent variable), that can be seen as a representation of user communities (same preferences) and movie communities (sames genres).

All in all, we try to compute the following probability for each (user, movie) couple : $p(s \mid u) = \sum p(s \mid z)p(z \mid u)$, which is the probability for a given user to see a given movie. This is obtained by summing for each community the probability for a movie s to be seen given a community z times the probability to be in the community z given a user u.

Initialisation

E-STEP - Compute $q(z \mid (u,s))$: the probability that the (user, movie) couple belongs to the class z

This step is first initialized at random :

- To each couple (u,s), assign each possible community through a cartesian product Example: with number of classes = 2, the lines (Marie, Star Wars) and (Gaëlle, Matrix) will give (Marie, Star Wars, 1), (Marie, Star Wars, 2), (Gaëlle, Matrix, 1), (Gaëlle, Matrix, 2)
- To each line, assign a random probability. This random probability corresponds to $q^*(z \mid (u,s))$. For example if I have (Marie, Star Wars, 1, 0.3), then the probability that the couple (Marie, Star Wars) is in class 1 is 0.3.
- Initialise the log-likelihood at 0:

LogLik = 0

Iteration

M-STEP - Compute $p(s \mid z)$ and $p(z \mid u)$ based on $q(z \mid (u,s))$ - Compute $p(s \mid z)$: sum the probas associated to every couple (s,z) and divide it by the sum of probas associated to this z - Compute $p(z \mid u)$: sum the probas associated to every couple (u,z) and divide by the sum of probas associated to this u

E-STEP - Compute the new $q(z \mid (u,s)) = \frac{p(s|z)p(z|u)}{\sum p(s|z)p(z|u)}$ - For each (u,s,z), compute $p(s \mid z) * p(z \mid u)$ - For each (u,s), compute $\sum_z p(s \mid z) * p(z \mid u)$ which is $p(s \mid u)$ - For each (u,s,z), compute $\frac{p(s|z)p(z|u)}{\sum p(s|z)p(z|u)}$ which is the new $q(z \mid (u,s))$

Update LogLik

$$\sum_{(s,u)} \Big(log\big(\sum_{z} p(s \mid z) * p(z \mid u) \big) \Big) = \sum_{(s,u)} log\big(p(s \mid u) \big)$$

Iterate again until LogLik converges : this means that it has reached its maximum and we have found the best estimation of $p(z \mid u)$ and $p(s \mid z)$.

We can now predict the probability that Gaëlle will watch Star Wars :

$$p(StarWars \mid Gaëlle) = p(1 \mid Gaëlle) * p(StarWars \mid 1) + p(2 \mid Gaëlle) * p(StarWars \mid 2)$$

Integrating preference values

To do

Comparison with LDA recommender systems

To do

If you are very motivated, you can try to integrate preference values to PLSI, as described in Hoffman (1999). You can also compare your results to Spark LDA implementation and discuss the use of LDA versus PLSI.

Appendix: code