COMP4121 Advanced Algorithms

Aleks Ignjatović

School of Computer Science and Engineering
University of New South Wales Sydney

Recommender Systems

Recommender Systems

- Main purpose: the noble goal of selling you as much stuff as possible, regardless of whether you need it or not.

- Examples of recommender systems:
  - Netflix's, to recommend to you which movie to see next.
  - Amazon's, to recommend to you which book to buy next.
  - Kogan's to recommend which gizmo to buy next.
  - IEEE's Xplore: to recommend which articles might be of interest to you, given what article you have just look at.
- Two major kinds of recommender systems:
  - **content based**: items are recommended by their intrinsic similarity (i.e., similar properties, qualities, kind etc.)
    For example a book might be recommended because you bought a book on a similar topic)
  - **collaborative filtering**: items are recommended based on some similarity measure between users and between items based on ratings of items by the community of users.

Recommender Systems

- Main purpose: the noble goal of selling you as much stuff as possible, regardless of whether you need it or not.

- Examples of recommender systems:
  - Netflix's, to recommend to you which movie to see next.
  - Amazon's, to recommend to you which book to buy next.
  - Kogan's to recommend which gizmo to buy next.
  - IEEE's Xplore: to recommend which articles might be of interest to you, given what article you have just look at.

- Two major kinds of recommender systems:
  - **content based**: items are recommended by their intrinsic similarity (i.e., similar properties, qualities, kind etc.) For example a book might be recommended because you bought a book on a similar topic)
  - **collaborative filtering**: items are recommended based on some similarity measure between users and between items based on ratings of items by the community of users.

Recommender Systems

- Main purpose: the noble goal of selling you as much stuff as possible, regardless of whether you need it or not.

- Examples of recommender systems:
  - Netflix's, to recommend to you which movie to see next.
  - Amazon's, to recommend to you which book to buy next.
  - Kogan's to recommend which gizmo to buy next.
  - IEEE's Xplore: to recommend which articles might be of interest to you, given what article you have just look at.

- Two major kinds of recommender systems:
  - **content based**: items are recommended by their intrinsic similarity (i.e., similar properties, qualities, kind etc.) For example a book might be recommended because you bought a book on a similar topic)
  - **collaborative filtering**: items are recommended based on some similarity measure between users and between items based on ratings of items by the community of users.

Recommender Systems

- Main purpose: the noble goal of selling you as much stuff as possible, regardless of whether you need it or not.

- Examples of recommender systems:
  - Netflix's, to recommend to you which movie to see next.
  - Amazon's, to recommend to you which book to buy next.
  - Kogan's to recommend which gizmo to buy next.
  - IEEE's Xplore: to recommend which articles might be of interest to you, given what article you have just look at.

- Two major kinds of recommender systems:
  - **content based**: items are recommended by their intrinsic similarity (i.e., similar properties, qualities, kind etc.) For example a book might be recommended because you bought a book on a similar topic)
  - **collaborative filtering**: items are recommended based on some similarity measure between users and between items based on ratings of items by the community of users.

Recommender Systems

- Main purpose: the noble goal of selling you as much stuff as possible, regardless of whether you need it or not.

- Examples of recommender systems:
  - Netflix's, to recommend to you which movie to see next.
  - Amazon's, to recommend to you which book to buy next.
  - Kogan's to recommend which gizmo to buy next.
  - IEEE's Xplore: to recommend which articles might be of interest to you, given what article you have just look at.

- Two major kinds of recommender systems:
  - **content based**: items are recommended by their intrinsic similarity (i.e., similar properties, qualities, kind etc.) For example a book might be recommended because you bought a book on a similar topic)
  - **collaborative filtering**: items are recommended based on some similarity measure between users and between items based on ratings of items by the community of users.

Recommender Systems

- Main purpose: the noble goal of selling you as much stuff as possible, regardless of whether you need it or not.

- Examples of recommender systems:
  - Netflix's, to recommend to you which movie to see next.
  - Amazon's, to recommend to you which book to buy next.
  - Kogan's to recommend which gizmo to buy next.
  - IEEE's Xplore: to recommend which articles might be of interest to you, given what article you have just look at.

- Two major kinds of recommender systems:
  - **content based**: items are recommended by their intrinsic similarity (i.e., similar properties, qualities, kind etc.) For example a book might be recommended because you bought a book on a similar topic)
  - **collaborative filtering**: items are recommended based on some similarity measure between users and between items based on ratings of items by the community of users.

Recommender Systems

- Main purpose: the noble goal of selling you as much stuff as possible, regardless of whether you need it or not.
- Examples of recommender systems:
  - Netflix's, to recommend to you which movie to see next.
  - Amazon's, to recommend to you which book to buy next.
  - Kogan's to recommend which gizmo to buy next.
  - IEEE's Xplore: to recommend which articles might be of interest to you, given what article you have just look at.
- Two major kinds of recommender systems:
  - **content based**: items are recommended by their intrinsic similarity (i.e., similar properties, qualities, kind etc.) For example a book might be recommended because you bought a book on a similar topic)
  - **collaborative filtering**: items are recommended based on some similarity measure between users and between items based on ratings of items by the community of users.

- Main purpose: the noble goal of selling you as much stuff as possible, regardless of whether you need it or not.
- Examples of recommender systems:
  - Netflix's, to recommend to you which movie to see next.
  - Amazon's, to recommend to you which book to buy next.
  - Kogan's to recommend which gizmo to buy next.
  - IEEE's Xplore: to recommend which articles might be of interest to you, given what article you have just look at.
- Two major kinds of recommender systems:
  - **content based**: items are recommended by their intrinsic similarity (i.e., similar properties, qualities, kind etc.) For example a book might be recommended because you bought a book on a similar topic)
  - **collaborative filtering**: items are recommended based on some similarity measure between users and between items based on ratings of items by the community of users.

- Main purpose: the noble goal of selling you as much stuff as possible, regardless of whether you need it or not.

- Examples of recommender systems:
  - Netflix's, to recommend to you which movie to see next.
  - Amazon's, to recommend to you which book to buy next.
  - Kogan's to recommend which gizmo to buy next.
  - IEEE's Xplore: to recommend which articles might be of interest to you, given what article you have just look at.

- Two major kinds of recommender systems:
  - **content based**: items are recommended by their intrinsic similarity (i.e., similar properties, qualities, kind etc.) For example a book might be recommended because you bought a book on a similar topic)
  - **collaborative filtering**: items are recommended based on some similarity measure between users and between items based on ratings of items by the community of users.

- Content based recommender systems suffer a serious problem: classification according to content usually has to be done by humans because content is a semantic notion and machines are still not good at dealing with semantics.
- Collaborative filtering techniques are superior in performance and does not rely on human advice.
- **A Representative Example:** Assume users are rating movies that they have seen. On the basis of such information we would like to recommend to a user a movie he has not already seen.

- Two main approaches: the **Neighbourhood Method** and the **Latent Factor Method**

- The Neighbourhood Method comes in two flavours:

  (I) based on similarity of users:
  - assume it happens that two users A and B gave “similar” evaluations to movies that they have both seen;
  - if there is a movie which user A liked a lot but user B has not seen,
  - then it is reasonable to recommend such a movie to user B.

- Content based recommender systems suffer a serious problem: classification according to content usually has to be done by humans because content is a semantic notion and machines are still not good at dealing with semantics.
- Collaborative filtering methods are superior in performance and does not rely on human advice.
- **A Representative Example:** Assume users are rating movies that they have seen. On the basis of such information we would like to recommend to a user a movie he has not already seen.
- Two main approaches: the **Neighbourhood Method** and the **Latent Factor Method**
- The Neighbourhood Method comes in two flavours:
  - (I) based on similarity of users:
    - assume it happens that two users A and B gave "similar" evaluations to movies that they have both seen;
    - if there is a movie which user A liked a lot but user B has not seen,
    - then it is reasonable to recommend such a movie to user B.

- Content based recommender systems suffer a serious problem: classification according to content u[illegible]o be done by humans because content is a semantic notion and m[illegible] still not good at dealing with semantics.
- Collaborative filtering [illegible] superior in performance and does not rely on human advice.
- **A Representative Example:** Assume users are rating movies that they have seen. On the basis of such information we would like to recommend to a user a movie he has not already seen.

- Two main approaches: the **Neighbourhood Method** and the **Latent Factor Method**

- The Neighbourhood Method comes in two flavours:

  (I) based on similarity of users:

  - assume it happens that two users A and B gave "similar" evaluations to movies that they have both seen;
  - if there is a movie which user A liked a lot but user B has not seen,
  - then it is reasonable to recommend such a movie to user B.

- Content based recommender systems suffer a serious problem: classification according to content u[illegible]o be done by humans because content is a semantic notion and m[illegible] still not good at dealing with semantics.
- Collaborative filtering [illegible] superior in performance and does not rely on human advice.
- **A Representative Example:** Assume users are rating movies that they have seen. On the basis of such information we would like to recommend to a user a movie he has not already seen.
- Two main approaches: the **Neighbourhood Method** and the **Latent Factor Method**
- The Neighbourhood Method comes in two flavours:
  (I) based on similarity of users:
  - assume it happens that two users A and B gave "similar" evaluations to movies that they have both seen;
  - if there is a movie which user A liked a lot but user B has not seen,
  - then it is reasonable to recommend such a movie to user B.

Recommender Systems

- Content based recommender systems suffer a serious problem: classification according to content usually has to be done by humans because content is a semantic notion and machines are still not good at dealing with semantics.
- Collaborative filtering methods are superior in performance and does not rely on human advice.
- **A Representative Example:** Assume users are rating movies that they have seen. On the basis of such information we would like to recommend to a user a movie he has not already seen.

- Two main approaches: the **Neighbourhood Method** and the **Latent Factor Method**

- The Neighbourhood Method comes in two flavours:

  (I) based on similarity of users:
  - assume it happens that two users A and B gave "similar" evaluations to movies that they have both seen;
  - if there is a movie which user A liked a lot but user B has not seen,
  - then it is reasonable to recommend such a movie to user B.

- Content based recommender systems suffer a serious problem: classification according to content u[illegible] to be done by humans because content is a semantic notion and m[illegible] still not good at dealing with semantics.
- Collaborative filtering [illegible] superior in performance and does not rely on human advice.
- **A Representative Example:** Assume users are rating movies that they have seen. On the basis of such information we would like to recommend to a user a movie he has not already seen.
- Two main approaches: the **Neighbourhood Method** and the **Latent Factor Method**
- The Neighbourhood Method comes in two flavours:
  - (I) based on similarity of users:
    - assume it happens that two users A and B gave "similar" evaluations to movies that they have both seen;
    - if there is a movie which user A liked a lot but user B has not seen,
    - then it is reasonable to recommend such a movie to user B.

- Content based recommender systems suffer a serious problem: classification according to content usually has to be done by humans because content is a semantic notion and machines are still not good at dealing with semantics.
- Collaborative filtering methods are superior in performance and does not rely on human advice.
- **A Representative Example:** Assume users are rating movies that they have seen. On the basis of such information we would like to recommend to a user a movie he has not already seen.

- Two main approaches: the **Neighbourhood Method** and the **Latent Factor Method**

- The Neighbourhood Method comes in two flavours:

(I) based on similarity of users:
  - assume it happens that two users A and B gave "similar" evaluations to movies that they have both seen;
  - if there is a movie which user A liked a lot but user B has not seen,
  - then it is reasonable to recommend such a movie to user B.

- Content based recommender systems suffer a serious problem: classification according to content u[illegible]to be done by humans because content is a semantic notion and m[illegible]e still not good at dealing with semantics.
- Collaborative filtering [illegible]e superior in performance and does not rely on human advice.
- **A Representative Example:** Assume users are rating movies that they have seen. On the basis of such information we would like to recommend to a user a movie he has not already seen.

- Two main approaches: the **Neighbourhood Method** and the **Latent Factor Method**

- The Neighbourhood Method comes in two flavours:

(I) based on similarity of users:
   - assume it happens that two users A and B gave “similar” evaluations to movies that they have both seen;
   - if there is a movie which user A liked a lot but user B has not seen,
   - then it is reasonable to recommend such a movie to user B.

- Content based recommender systems suffer a serious problem: classification according to content usually has to be done by humans because content is a semantic notion and machines are still not good at dealing with semantics.
- Collaborative filtering methods are superior in performance and does not rely on human advice.
- **A Representative Example:** Assume users are rating movies that they have seen. On the basis of such information we would like to recommend to a user a movie he has not already seen.

- Two main approaches: the **Neighbourhood Method** and the **Latent Factor Method**

- The Neighbourhood Method comes in two flavours:

(I) based on similarity of users:
  - assume it happens that two users A and B gave "similar" evaluations to movies that they have both seen;
  - if there is a movie which user A liked a lot but user B has not seen,
  - then it is reasonable to recommend such a movie to user B.

- (II) based on similarity of items:
  - assume it happens that two movies M_1 and M_2 received similar ratings by most [illegible]
  - a user has seen [illegible] and liked it;
  - then it is reasonable to recommend movie M_2 to such a user.

- Note that in both approaches movies are not categorised and compared by their "intrinsic" features but we rely only on the "wisdom of the crowd".
- This is an example of *collaborative filtering*.

- We now want to explore how such similarities of users and of items are measured in a most informative way.

- We can construct a sparsely populated table of ratings R; the rows will correspond to movies, the columns to users. The entry $r(j, i)$ of the table, if non empty, represents the rating user U_i gave to movie M_j (in general, *item* M_j).

- Usually, such a rating is the "number of stars", in range $0 - 5$ (or a similar, relatively small rating range, usually with at most 10 or so levels).

- (II) based on similarity of items:
  - assume it happens that two movies M_1 and M_2 received similar ratings by most users;
  - a user has seen movie M_1 and liked it;
  - then it is reasonable to recommend movie M_2 to such a user.

- Note that in both approaches movies are not categorised and compared by their "intrinsic" features but we rely only on the "wisdom of the crowd".
- This is an example of *collaborative filtering*.
- We now want to explore how such similarities of users and of items are measured in a most informative way.
- We can construct a sparsely populated table of ratings R; the rows will correspond to movies, the columns to users. The entry $r(j, i)$ of the table, if non empty, represents the rating user U_i gave to movie M_j (in general, *item* M_j).
- Usually, such a rating is the "number of stars", in range $0 - 5$ (or a similar, relatively small rating range, usually with at most 10 or so levels).

- (II) based on similarity of items:
  - assume it happens that two movies M_1 and M_2 received similar ratings by most [illegible]
  - a user has seen [illegible] and liked it;
  - then it is reasonable to recommend movie M_2 to such a user.

- Note that in both approaches movies are not categorised and compared by their "intrinsic" features but we rely only on the "wisdom of the crowd".
- This is an example of *collaborative filtering*.

- We now want to explore how such similarities of users and of items are measured in a most informative way.

- We can construct a sparsely populated table of ratings R; the rows will correspond to movies, the columns to users. The entry $r(j,i)$ of the table, if non empty, represents the rating user U_i gave to movie M_j (in general, *item* M_j).

- Usually, such a rating is the "number of stars", in range $0-5$ (or a similar, relatively small rating range, usually with at most 10 or so levels).

- (II) based on similarity of items:
  - assume it happens that two movies M_1 and M_2 received similar ratings by most users;
  - a user has seen movie M_1 and liked it;
  - then it is reasonable to recommend movie M_2 to such a user.

- Note that in both approaches movies are not categorised and compared by their "intrinsic" features but we rely only on the "wisdom of the crowd".
- This is an example of *collaborative filtering*.

- We now want to explore how such similarities of users and of items are measured in a most informative way.

- We can construct a sparsely populated table of ratings R; the rows will correspond to movies, the columns to users. The entry $r(j, i)$ of the table, if non empty, represents the rating user U_i gave to movie M_j (in general, *item* M_j).

- Usually, such a rating is the "number of stars", in range $0 - 5$ (or a similar, relatively small rating range, usually with at most 10 or so levels).

- (II) based on similarity of items:
  - assume it happens that two movies M_1 and M_2 received similar ratings by most [illegible]
  - a user has seen [illegible] and liked it;
  - then it is reason[illegible]commend movie M_2 to such a user.
- Note that in both approaches movies are not categorised and compared by their "intrinsic" features but we rely only on the "wisdom of the crowd".

- This is an example of *collaborative filtering*.

- We now want to explore how such similarities of users and of items are measured in a most in[illegible] way.

- We can construct a sparsely populated table of ratings R; the rows will correspond to movies, [illegible]rs. The entry $r(j, i)$ of the table, if non empty, represents the rating user U_i gave to movie M_j (in general, *item* M_j).

- Usually, such a rating is the "number of stars", in range $0 - 5$ (or a similar, relatively small rating range, usually with at most 10 or so levels).

- (II) based on similarity of items:
  - assume it happens that two movies M_1 and M_2 received similar ratings by most [illegible]
  - a user has seen [illegible] and liked it;
  - then it is reason[illegible]commend movie M_2 to such a user.

- Note that in both approaches movies are not categorised and compared by their "intrinsic" features but we rely only on the "wisdom of the crowd".
- This is an example of *collaborative filtering*.

- We now want to explore how such similarities of users and of items are measured in a most informative way.

- We can construct a sparsely populated table of ratings R; the rows will correspond to movies, the columns to users. The entry $r(j, i)$ of the table, if non empty, represents the rating user U_i gave to movie M_j (in general, *item* M_j).

- Usually, such a rating is the "number of stars", in range $0 - 5$ (or a similar, relatively small rating range, usually with at most 10 or so levels).

- (II) based on similarity of items:
  - assume it happe[illegible]o movies M_1 and M_2 received similar ratings by most [illegible]
  - a user has seen [illegible] and liked it;
  - then it is reason[illegible]commend movie M_2 to such a user.

- Note that in both approaches movies are not categorised and compared by their "intrinsic" features but we rely only on the "wisdom of the crowd".
- This is an example of *collaborative filtering*.
- We now want to explore how such similarities of users and of items are measured in a most informative way.
- We can construct a sparsely populated table of ratings R; the rows will correspond to movies, the columns to users. The entry $r(j,i)$ of the table, if non empty, represents the rating user U_i gave to movie M_j (in general, *item* M_j).
- Usually, such a rating is the "number of stars", in range $0-5$ (or a similar, relatively small rating range, usually with at most 10 or so levels).

Recommender Systems

- (II) based on similarity of items:
  - assume it happens that two movies M_1 and M_2 received similar ratings by most [illegible]
  - a user has seen [illegible] and liked it;
  - then it is reason[illegible]commend movie M_2 to such a user.

- Note that in both approaches movies are not categorised and compared by their "intrinsic" features but we rely only on the "wisdom of the crowd".
- This is an example of *collaborative filtering*.

- We now want to explore how such similarities of users and of items are measured in a most informative way.

- We can construct a sparsely populated table of ratings R; the rows will correspond to movies, the columns to users. The entry $r(j, i)$ of the table, if non empty, represents the rating user U_i gave to movie M_j (in general, *item* M_j).

- Usually, such a rating is the "number of stars", in range $0 - 5$ (or a similar, relatively small rating range, usually with at most 10 or so levels).

- (II) based on similarity of items:
  - assume it happe[illegible]o movies M_1 and M_2 received similar ratings by most [illegible]
  - a user has seen [illegible] and liked it;
  - then it is reason[illegible]ommend movie M_2 to such a user.

- Note that in both approaches movies are not categorised and compared by their "intrinsic" features but we rely only on the "wisdom of the crowd".
- This is an example of *collaborative filtering*.
- We now want to explore how such similarities of users and of items are measured in a most informative way.
- We can construct a sparsely populated table of ratings R; the rows will correspond to movies, the columns to users. The entry $r(j, i)$ of the table, if non empty, represents the rating user U_i gave to movie M_j (in general, *item* M_j).
- Usually, such a rating is the "number of stars", in range $0 - 5$ (or a similar, relatively small rating range, usually with at most 10 or so levels).

- We replace these integers with more informative numbers.
- A more informative number can be obtained by computing the mean $\overline{r}$ of all ratings of all users for [illegible] (thus, the mean of all numbers in our partial table of ratings R)
- We now obtain from t[illegible]w table $\overline{R}$ by replacing all ratings $r(j,i)$ in R by the values $r^*(j,i) =$ [illegible]
- Now numbers $r^*(j,i)$ are already more informative: if $r^*(j,i) > 0$ this means, in a sense, that user U_j has liked movie M_i above the "global average".
- The fact that numbers $r^*(j,i)$ can be both positive and negative with about equal likelihood is important for the subsequent steps to be taken.
- Some users are more generous and tend to give higher scores than the average user; some are more critical and tend to give lower scores.
- We are not interested in evaluating generosity of users, we want to assess only the "taste" of users: w[illegible] and what they like less.
- Similarly, some movies get higher scores because they are popular at the moment for whatever [illegible]s have less "hype" about them because they might be older and less trendy.
- Again, we are not interested in "absolute popularity" or "trendiness" of a movie, rather, we would like to assess how "intrinsically likeable" a movie is.

Neighbourhood Method

- We replace these integers with more informative numbers.
- A more informative number can be obtained by computing the mean $\overline{r}$ of all ratings of all users for all movies (thus, the mean of all numbers in our partial table of ratings R)
- We now obtain from t[illegible] table $\overline{R}$ by replacing all ratings $r(j,i)$ in R by the values $r^*(j,i) =$ [illegible]
- Now numbers $r^*(j,i)$ are already more informative: if $r^*(j,i) > 0$ this means, in a sense, that user U_j has liked movie M_i above the “global average”.
- The fact that numbers $r^*(j,i)$ can be both positive and negative with about equal likelihood is important for the subsequent steps to be taken.
- Some users are more generous and tend to give higher scores than the average user; some are more critical and tend to give lower scores.
- We are not interested in evaluating generosity of users, we want to assess only the “taste” of users: what they like more and what they like less.
- Similarly, some movies get higher scores because they are popular at the moment for whatever reason, while other movies have less “hype” about them because they might be older and less trendy.
- Again, we are not interested in “absolute popularity” or “trendiness” of a movie, rather, we would like to assess how “intrinsically likeable” a movie is.

Neighbourhood Method

- We replace these integers with more informative numbers.
- A more informative number can be obtained by computing the mean $\overline{r}$ of all ratings of all users for [illegible] (thus, the mean of all numbers in our partial table of ratings R)
- We now obtain from t[illegible]w table $\overline{R}$ by replacing all ratings $r(j,i)$ in R by the values $r^*(j,i) =$ [illegible]
- Now numbers $r^*(j,i)$ are already more informative: if $r^*(j,i) > 0$ this means, in a sense, that user U_j has liked movie M_i above the "global average".
- The fact that numbers $r^*(j,i)$ can be both positive and negative with about equal likelihood is important for the subsequent steps to be taken.
- Some users are more generous and tend to give higher scores than the average user; some are more critical and tend to give lower scores.
- We are not interested in evaluating generosity of users, we want to assess only the "taste" of users: what they like more and what they like less.
- Similarly, some movies get higher scores because they are popular at the moment for whatever reason, and some movies have less "hype" about them because they might be older and less trendy.
- Again, we are not interested in "absolute popularity" or "trendiness" of a movie, rather, we would like to assess how "intrinsically likeable" a movie is.

Neighbourhood Method

- We replace these integers with more informative numbers.
- A more informative number can be obtained by computing the mean $\bar{r}$ of all ratings of all users for [illegible] (thus, the mean of all numbers in our partial table of ratings R)
- We now obtain from t[illegible]w table $\overline{R}$ by replacing all ratings $r(j, i)$ in R by the values $r^*(j, i) =$ [illegible]
- Now numbers $r^*(j, i)$ are already more informative: if $r^*(j, i) > 0$ this means, in a sense, that user U_j has liked movie M_i above the "global average".
- The fact that numbers $r^*(j, i)$ can be both positive and negative with about equal likelihood is important for the subsequent steps to be taken.
- Some users are more generous and tend to give higher scores than the average user; some are more critical and tend to give lower scores.
- We are not interested in evaluating generosity of users, we want to assess only the "taste" of users: what they like more and what they like less.
- Similarly, some movies get higher scores because they are popular at the moment for whatever reason, and some movies have less "hype" about them because they might be older and less trendy.
- Again, we are not interested in "absolute popularity" or "trendiness" of a movie, rather, we would like to assess how "intrinsically likeable" a movie is.

- We replace these integers with more informative numbers.
- A more informative number can be obtained by computing the mean $\bar{r}$ of all ratings of all users for [illegible] (thus, the mean of all numbers in our partial table of ratings R)
- We now obtain from t[illegible]w table $\overline{R}$ by replacing all ratings $r(j,i)$ in R by the values $r^*(j,i) =$ [illegible]
- Now numbers $r^*(j,i)$ are already more informative: if $r^*(j,i) > 0$ this means, in a sense, that user U_j has liked movie M_i above the "global average".
- The fact that numbers $r^*(j,i)$ can be both positive and negative with about equal likelihood is important for the subsequent steps to be taken.
- Some users are more generous and tend to give higher scores than the average user; some are more critical and tend to give lower scores.
- We are not interested in evaluating generosity of users, we want to assess only the "taste" of users: what they like more and what they like less.
- Similarly, some movies get higher scores because they are popular at the moment for whatever reason; other movies have less "hype" about them because they might be older and less trendy.
- Again, we are not interested in "absolute popularity" or "trendiness" of a movie, rather, we would like to assess how "intrinsically likeable" a movie is.

- We replace these integers with more informative numbers.
- A more informative number can be obtained by computing the mean $\overline{r}$ of all ratings of all users for all movies (thus, the mean of all numbers in our partial table of ratings R)
- We now obtain from t[illegible]w table $\overline{R}$ by replacing all ratings $r(j,i)$ in R by the values $r^*(j,i) =$
- Now numbers $r^*(j,i)$ are already more informative: if $r^*(j,i) > 0$ this means, in a sense, that user U_j has liked movie M_i above the "global average".
- The fact that numbers $r^*(j,i)$ can be both positive and negative with about equal likelihood is important for the subsequent steps to be taken.
- Some users are more generous and tend to give higher scores than the average user; some are more critical and tend to give lower scores.
- We are not interested in evaluating generosity of users, we want to assess only the "taste" of users: what they like more and what they like less.
- Similarly, some movies get higher scores because they are popular at the moment for whatever reason and other movies have less "hype" about them because they might be older and less trendy.
- Again, we are not interested in "absolute popularity" or "trendiness" of a movie, rather, we would like to assess how "intrinsically likeable" a movie is.

Neighbourhood Method

- We replace these integers with more informative numbers.
- A more informative number can be obtained by computing the mean $\overline{r}$ of all ratings of all users for [illegible] (thus, the mean of all numbers in our partial table of ratings R)
- We now obtain from t[illegible]w table $\overline{R}$ by replacing all ratings $r(j,i)$ in R by the values $r^*(j,i) =$ [illegible]
- Now numbers $r^*(j,i)$ are already more informative: if $r^*(j,i) > 0$ this means, in a sense, that user U_j has liked movie M_i above the "global average".
- The fact that numbers $r^*(j,i)$ can be both positive and negative with about equal likelihood is important for the subsequent steps to be taken.
- Some users are more generous and tend to give higher scores than the average user; some are more critical and tend to give lower scores.
- We are not interested in evaluating generosity of users, we want to assess only the "taste" of users: what they like more and what they like less.
- Similarly, some movies get higher scores because they are popular at the moment for whatever reason and other movies have less "hype" about them because they might be older and less trendy.
- Again, we are not interested in "absolute popularity" or "trendiness" of a movie, rather, we would like to assess how "intrinsically likeable" a movie is.

Neighbourhood Method

- We replace these integers with more informative numbers.
- A more informative number can be obtained by computing the mean $\bar{r}$ of all ratings of all users for [illegible] (thus, the mean of all numbers in our partial table of ratings R)
- We now obtain from t[illegible]w table $\overline{R}$ by replacing all ratings $r(j,i)$ in R by the values $r^*(j,i) =$ [illegible]
- Now numbers $r^*(j,i)$ are already more informative: if $r^*(j,i) > 0$ this means, in a sense, that user U_j has liked movie M_i above the "global average".
- The fact that numbers $r^*(j,i)$ can be both positive and negative with about equal likelihood is important for the subsequent steps to be taken.
- Some users are more generous and tend to give higher scores than the average user; some are more critical and tend to give lower scores.
- We are not interested in evaluating generosity of users, we want to assess only the "taste" of users: what they like more and what they like less.
- Similarly, some movies get higher scores because they are popular at the moment for whatever reason and some movies have less "hype" about them because they might be older and less trendy.
- Again, we are not interested in "absolute popularity" or "trendiness" of a movie, rather, we would like to assess how "intrinsically likeable" a movie is.

Neighbourhood Method

- We replace these integers with more informative numbers.
- A more informative number can be obtained by computing the mean $\bar{r}$ of all ratings of all users for [illegible] (thus, the mean of all numbers in our partial table of ratings R)
- We now obtain from t[illegible]w table $\overline{R}$ by replacing all ratings $r(j,i)$ in R by the values $r^*(j,i) =$ [illegible]
- Now numbers $r^*(j,i)$ are already more informative: if $r^*(j,i) > 0$ this means, in a sense, that user U_j has liked movie M_i above the "global average".
- The fact that numbers $r^*(j,i)$ can be both positive and negative with about equal likelihood is important for the subsequent steps to be taken.
- Some users are more generous and tend to give higher scores than the average user; some are more critical and tend to give lower scores.
- We are not interested in evaluating generosity of users, we want to assess only the "taste" of users: what they like more and what they like less.
- Similarly, some movies get higher scores because they are popular at the moment for whatever reason, and some movies have less "hype" about them because they might be older and less trendy.
- Again, we are not interested in "absolute popularity" or "trendiness" of a movie, rather, we would like to assess how "intrinsically likeable" a movie is.

Neighbourhood Method

- For that reason we want to remove the "systematic biases" of both the users and the movies, thus taking out the individual "generosity" of each user and the "hype" of each movie.
- For that purpose we introduce for every user U_i a variable υ_i standing for the "individual bias" of user U_i, reflecting his tendency to give overall higher or lower scores.
- We also introduce for each movie M_j a variable μ_j standing for the "hype bias" of movie M_j which is due to how "fashionable" the movie is (which anyhow usually quickly fades with time).
- We now remove both such systematic biases by seeking the values of variables υ_i and variables μ_j which minimises the expression

$$S(\vec{\upsilon},\vec{\mu}) = \sum_{(j,i)\in R} \left(r^*(j,i) - \upsilon_i - \mu_j\right)^2$$

- Note that μ 's are constant shifts of rows (each row corresponding to a movie) and υ's are constant shifts of columns (each corresponding to a user)
- We chose such constant shifts of each row and of each column which minimise the residuals.
- Each such residual $\widetilde{r}(j,i) = r^*(j,i) - \upsilon_i - \mu_j$ better represents the "intrinsic" sentiment of a user U_i for a movie M_j than $\widetilde{r}(j,i)$.

Neighbourhood Method

- For that reason we want to remove the "systematic biases" of both the users and the movies, thus taking out the individual "generosity" of each user and the "hype" of each movie.
- For that purpose we introduce for every user U_i a variable v_i standing for the "individual bias" of user U_i reflecting his tendency to give overall higher or lower scores.
- We also introduce for each movie M_j a variable μ_j standing for the "hype bias" of movie M_j which is due to how "fashionable" the movie is (which anyhow usually quickly fades with time).
- We now remove both such systematic biases by seeking the values of variables v_i and variables μ_j which minimises the expression

$$S(\vec{v},\vec{\mu}) = \sum_{(j,i)\in R} \left(r^*(j,i) - v_i - \mu_j\right)^2$$

- Note that μ 's are constant shifts of rows (each row corresponding to a movie) and v's are constant shifts of columns (each corresponding to a user)
- We chose such constant shifts of each row and of each column which minimise the residuals.
- Each such residual $\widetilde{r}(j,i) = r^*(j,i) - v_i - \mu_j$ better represents the "intrinsic" sentiment of a user U_i for a movie M_j than $\widetilde{r}(j,i)$.

Neighbourhood Method

- For that reason we want to remove the "systematic biases" of both the users and the movies, thus taking out the individual "generosity" of each user and the "hype" of each movie.
- For that purpose we in[illegible] every user U_i a variable v_i standing for the "individual bias" of us[illegible]ting his tendency to give overall higher or lower scores.
- We also introduce for [illegible] M_j a variable μ_j standing for the "hype bias" of movie M_j which is due to how "fashionable" the movie is (which anyhow usually quickly fades with time).
- We now remove both such systematic biases by seeking the values of variables v_i and variables μ_j which minimises the expression

$$S(\vec{v}, \vec{\mu}) = \sum_{(j,i)\in R} \left(r^*(j,i) - v_i - \mu_j\right)^2$$

- Note that μ 's are constant shifts of rows (each row corresponding to a movie) and v's are constant shifts of columns (each corresponding to a user)
- We chose such constant shifts of each row and of each column which minimise the residuals.
- Each such residual $\tilde{r}(j,i) = r^*(j,i) - v_i - \mu_j$ better represents the "intrinsic" sentiment of a user U_i for a movie M_j than $\tilde{r}(j,i)$.

Neighbourhood Method

- For that reason we want to remove the "systematic biases" of both the users and the movies, thus taking out the individual "generosity" of each user and the "hype" of each movie.
- For that purpose we in[illegible] every user U_i a variable υ_i standing for the "individual bias" of us[illegible]ting his tendency to give overall higher or lower scores.
- We also introduce for [illegible] M_j a variable μ_j standing for the "hype bias" of movie M_j which is due to how "fashionable" the movie is (which anyhow usually quickly fades with time).
- We now remove both such systematic biases by seeking the values of variables υ_i and variables μ_j which minimises the expression

$$S(\vec{\upsilon}, \vec{\mu}) = \sum_{(j,i)\in R} (r^*(j,i) - \upsilon_i - \mu_j)^2$$

- Note that μ 's are constant shifts of rows (each row corresponding to a movie) and υ's are constant shifts of columns (each corresponding to a user)
- We chose such constant shifts of each row and of each column which minimise the residuals.
- Each such residual $\widetilde{r}(j,i) = r^*(j,i) - \upsilon_i - \mu_j$ better represents the "intrinsic" sentiment of a user U_i for a movie M_j than $\widetilde{r}(j,i)$.

Neighbourhood Method

- For that reason we want to remove the "systematic biases" of both the users and the movies, thus taking out the individual "generosity" of each user and the "hype" of each movie.
- For that purpose we in[illegible] every user U_i a variable υ_i standing for the "individual bias" of us[illegible]ting his tendency to give overall higher or lower scores.
- We also introduce for [illegible]e M_j a variable μ_j standing for the "hype bias" of movie M_j which is due to how "fashionable" the movie is (which anyhow usually quickly fades with time).

- We now remove both such systematic biases by seeking the values of variables υ_i and variables μ_j which minimises the expression

$$S(\vec{\upsilon}, \vec{\mu}) = \sum_{(j,i)\in R} \left(r^*(j,i) - \upsilon_i - \mu_j\right)^2$$

- Note that μ 's are constant shifts of rows (each row corresponding to a movie) and υ's are constant shifts of columns (each corresponding to a user)

- We chose such constant shifts of each row and of each column which minimise the residuals.
- Each such residual $\widetilde{r}(j,i) = r^*(j,i) - \upsilon_i - \mu_j$ better represents the "intrinsic" sentiment of a user U_i for a movie M_j than $\widetilde{r}(j,i)$.

Neighbourhood Method

- For that reason we want to remove the "systematic biases" of both the users and the movies, thus taking out the individual "generosity" of each user and the "hype" of each movie.
- For that purpose we ir every user U_i a variable υ_i standing for the "individual bias" of us ting his tendency to give overall higher or lower scores.
- We also introduce for e M_j a variable μ_j standing for the "hype bias" of movie M_j which is due to how "fashionable" the movie is (which anyhow usually quickly fades with time).
- We now remove both such systematic biases by seeking the values of variables υ_i and variables μ_j which minimises the expression

$$S(\vec{\upsilon}, \vec{\mu}) = \sum_{(j,i)\in R} (r^*(j,i) - \upsilon_i - \mu_j)^2$$

- Note that μ 's are constant shifts of rows (each row corresponding to a movie) and υ's are constant shifts of columns (each corresponding to a user)
- We chose such constant shifts of each row and of each column which minimise the residuals.
- Each such residual $\widetilde{r}(j,i) = r^*(j,i) - \upsilon_i - \mu_j$ better represents the "intrinsic" sentiment of a user U_i for a movie M_j than $\widetilde{r}(j,i)$.

Neighbourhood Method

- For that reason we want to remove the "systematic biases" of both the users and the movies, thus taking out the individual "generosity" of each user and the "hype" of each movie.
- For that purpose we in[illegible] every user U_i a variable υ_i standing for the "individual bias" of us[illegible]ting his tendency to give overall higher or lower scores.
- We also introduce for [illegible] M_j a variable μ_j standing for the "hype bias" of movie M_j which is due to how "fashionable" the movie is (which anyhow usually quickly fades with time).
- We now remove both such systematic biases by seeking the values of variables υ_i and variables μ_j which minimises the expression

$$S(\vec{\upsilon}, \vec{\mu}) = \sum_{(j,i)\in R} \left(r^*(j,i) - \upsilon_i - \mu_j\right)^2$$

- Note that μ 's are constant shifts of rows (each row corresponding to a movie) and υ's are constant shifts of columns (each corresponding to a user)
- We chose such constant shifts of each row and of each column which minimise the residuals.
- Each such residual $\widetilde{r}(j,i) = r^*(j,i) - \upsilon_i - \mu_j$ better represents the "intrinsic" sentiment of a user U_i for a movie M_j than $\widetilde{r}(j,i)$.

- This is a *Least Squares* problem and is easily reducible to a system of linear equations:
- $S(\vec{v}, \vec{\mu})$ achieves a min [illegible] $\vec{\mu}$ for which all the partial derivatives $\frac{\partial}{\partial \mu_j} S(\vec{v}, \vec{\mu})$ for all j an [illegible]) for all i are equal to 0:

$$\frac{\partial}{\partial \mu_j} S(\vec{v}, \vec{\mu}) = \frac{\partial}{\partial \mu_j} \sum_{(j,i)\in R} (r^*(j,i) - v_i - \mu_j)^2$$
$$= -2 \sum_{i:(j,i)\in R} (r^*(j,i) - v_i - \mu_j) = 0$$

and

$$\frac{\partial}{\partial v_i} S(\vec{v}, \vec{\mu}) = \frac{\partial}{\partial v_i} \sum_{(j,i)\in R} (r^*(j,i) - v_i - \mu_j)^2$$
$$= -2 \sum_{j:(j,i)\in R} (r^*(j,i) - v_i - \mu_j) = 0$$

Neighbourhood Method

- This is a *Least Squares* problem and is easily reducible to a system of linear equations:
- $S(\vec{v}, \vec{\mu})$ achieves a minimum at $\vec{v}, \vec{\mu}$ for which all the partial derivatives $\frac{\partial}{\partial \mu_j} S(\vec{v}, \vec{\mu})$ for all j and $\frac{\partial}{\partial v_i} S(\vec{v}, \vec{\mu})$ for all i are equal to 0:

$$\frac{\partial}{\partial \mu_j} S(\vec{v}, \vec{\mu}) = \frac{\partial}{\partial \mu_j} \sum_{(j,i) \in R} (r^*(j,i) - v_i - \mu_j)^2$$
$$= -2 \sum_{i:(j,i) \in R} (r^*(j,i) - v_i - \mu_j) = 0$$

and

$$\frac{\partial}{\partial v_i} S(\vec{v}, \vec{\mu}) = \frac{\partial}{\partial v_i} \sum_{(j,i) \in R} (r^*(j,i) - v_i - \mu_j)^2$$
$$= -2 \sum_{j:(j,i) \in R} (r^*(j,i) - v_i - \mu_j) = 0$$

- This is a *Least Squares* problem and is easily reducible to a system of linear equations:
- $S(\vec{v},\vec{\mu})$ achieves a minimum at $\vec{v},\vec{\mu}$ for which all the partial derivatives $\frac{\partial}{\partial \mu_j}S(\vec{v},\vec{\mu})$ for all j and $\frac{\partial}{\partial v_i}S(\vec{v},\vec{\mu})$ for all i are equal to 0:

$$\frac{\partial}{\partial \mu_j}S(\vec{v},\vec{\mu}) = \frac{\partial}{\partial \mu_j}\sum_{(j,i)\in R}\left(r^*(j,i) - v_i - \mu_j\right)^2$$

$$= -2\sum_{i:(j,i)\in R}\left(r^*(j,i) - v_i - \mu_j\right) = 0$$

and

$$\frac{\partial}{\partial v_i}S(\vec{v},\vec{\mu}) = \frac{\partial}{\partial v_i}\sum_{(j,i)\in R}\left(r^*(j,i) - v_i - \mu_j\right)^2$$

$$= -2\sum_{j:(j,i)\in R}\left(r^*(j,i) - v_i - \mu_j\right) = 0$$

Neighbourhood Method

- This is a *Least Squares* problem and is easily reducible to a system of linear equations:
- $S(\vec{v}, \vec{\mu})$ achieves a minimum for $\vec{v}$ and $\vec{\mu}$ for which all the partial derivatives $\frac{\partial}{\partial \mu_j} S(\vec{v}, \vec{\mu})$ for all j and $\frac{\partial}{\partial v_i} S(\vec{v}, \vec{\mu})$ for all i are equal to 0:

$$\frac{\partial}{\partial \mu_j} S(\vec{v}, \vec{\mu}) = \frac{\partial}{\partial \mu_j} \sum_{(j,i)\in R} (r^*(j,i) - v_i - \mu_j)^2$$

$$= -2 \sum_{i:(j,i)\in R} (r^*(j,i) - v_i - \mu_j) = 0$$

and

$$\frac{\partial}{\partial v_i} S(\vec{v}, \vec{\mu}) = \frac{\partial}{\partial v_i} \sum_{(j,i)\in R} (r^*(j,i) - v_i - \mu_j)^2$$

$$= -2 \sum_{j:(j,i)\in R} (r^*(j,i) - v_i - \mu_j) = 0$$

Neighbourhood Method

- Unfortunately, Least Squares fits usually suffer from overfitting: they often minimise the objective function by choosing excessively large values for the variables, relying on m[illegible]elations of positive and negative terms.
- The solution to this pr[illegible]lled *regularisation*: we introduce a term which penalises for lar[illegible] the variables.
- Thus, instead, we mini[illegible]m

$$S(\vec{v},\vec{\mu},\lambda) = \sum_{(j,i)\in R} (r^*(j,i) - v_i - \mu_j)^2 + \lambda\left(\sum_i v_i^2 + \sum_j \mu_j^2\right)$$

where λ is a suitably chosen small positive constant, usually $10^{-10} \leq \lambda \leq 10^{-2}$.

- Optimal value of λ can be "learned" in a way to be described later.
- We now obtain from table $\overline{R}$ a new table $\widetilde{R}$ by replacing all $r^*(j,i)$ in $\overline{R}$ with values $\widetilde{r}(j,i) = r^*(j,i) - v_i - \mu_j$ where $v's$ and $\mu's$ were obtained by our regularised least squares fit.
- Having removed the systematic biases of users and trendiness of movies, we are now ready to estimate similarities of users and similarities of movies.

Neighbourhood Method

- Unfortunately, Least Squares fits usually suffer from overfitting: they often minimise the objective function by choosing excessively large values for the variables, relying on m[illegible]elations of positive and negative terms.
- The solution to this pr[illegible]lled *regularisation*: we introduce a term which penalises for lar[illegible] the variables.
- Thus, instead, we mini[illegible]m

$$S(\vec{v}, \vec{\mu}, \lambda) = \sum_{(j,i) \in R} (r^*(j,i) - v_i - \mu_j)^2 + \lambda \left(\sum_i v_i^2 + \sum_j \mu_j^2 \right)$$

where λ is a suitably chosen small positive constant, usually $10^{-10} \leq \lambda \leq 10^{-2}$.

- Optimal value of λ can be "learned" in a way to be described later.

- We now obtain from table $\overline{R}$ a new table $\widetilde{R}$ by replacing all $r^*(j,i)$ in $\overline{R}$ with values $\widetilde{r}(j,i) = r^*(j,i) - v_i - \mu_j$ where $v's$ and $\mu's$ were obtained by our regularised least squar[illegible]

- Having removed the systematic biases of users and trendiness of movies, we are now ready to estimate similarities of users and similarities of movies.

Neighbourhood Method

- Unfortunately, Least Squares fits usually suffer from overfitting: they often minimise the objective function by choosing excessively large values for the variables, relying on m[illegible]elations of positive and negative terms.
- The solution to this pr[illegible]lled *regularisation*: we introduce a term which penalises for lar[illegible] the variables.
- Thus, instead, we mini[illegible]um

$$S(\vec{\upsilon}, \vec{\mu}, \lambda) = \sum_{(j,i)\in R} (r^*(j,i) - \upsilon_i - \mu_j)^2 + \lambda \left(\sum_i \upsilon_i^2 + \sum_j \mu_j^2 \right)$$

where λ is a suitably chosen small positive constant, usually $10^{-10} \leq \lambda \leq 10^{-2}$.

- Optimal value of λ can be "learned" in a way to be described later.
- We now obtain from table $\overline{R}$ a new table $\widetilde{R}$ by replacing all $r^*(j,i)$ in $\overline{R}$ with values $\widetilde{r}(j,i) = r^*(j,i) - \upsilon_i - \mu_j$ where $\upsilon's$ and $\mu's$ were obtained by our regularised least squares fit.
- Having removed the systematic biases of users and trendiness of movies, we are now ready to estimate similarities of users and similarities of movies.

Neighbourhood Method

- Unfortunately, Least Squares fits usually suffer from overfitting: they often minimise the objective function by choosing excessively large values for the variables, relying on mutual cancellations of positive and negative terms.
- The solution to this problem is so called *regularisation*: we introduce a term which penalises for large values of the variables.
- Thus, instead, we minimise the following sum

$$S(\vec{v}, \vec{\mu}, \lambda) = \sum_{(j,i)\in R} (r^*(j,i) - v_i - \mu_j)^2 + \lambda \left(\sum_i v_i^2 + \sum_j \mu_j^2 \right)$$

where λ is a suitably chosen small positive constant, usually $10^{-10} \leq \lambda \leq 10^{-2}$.

- Optimal value of λ can be "learned" in a way to be described later.

- We now obtain from table $\overline{R}$ a new table $\widetilde{R}$ by replacing all $r^*(j,i)$ in $\overline{R}$ with values $\widetilde{r}(j,i) = r^*(j,i) - v_i - \mu_j$ where $v's$ and $\mu's$ were obtained by our regularised least squares fit.

- Having removed the systematic biases of users and trendiness of movies, we are now ready to estimate similarities of users and similarities of movies.

Neighbourhood Method

- Unfortunately, Least Squares fits usually suffer from overfitting: they often minimise the objective function by choosing excessively large values for the variables, relying on m[illegible]elations of positive and negative terms.
- The solution to this pr[illegible]lled *regularisation*: we introduce a term which penalises for lar[illegible] the variables.
- Thus, instead, we mini[illegible]um

$$S(\vec{\upsilon}, \vec{\mu}, \lambda) = \sum_{(j,i)\in R} (r^*(j,i) - \upsilon_i - \mu_j)^2 + \lambda \left(\sum_i \upsilon_i^2 + \sum_j \mu_j^2 \right)$$

where λ is a suitably chosen small positive constant, usually $10^{-10} \leq \lambda \leq 10^{-2}$.

- Optimal value of λ can be "learned" in a way to be described later.
- We now obtain from table $\overline{R}$ a new table $\widetilde{R}$ by replacing all $r^*(j,i)$ in $\overline{R}$ with values $\widetilde{r}(j,i) = r^*(j,i) - \upsilon_i - \mu_j$ where $\upsilon's$ and $\mu's$ were obtained by our regularised least squares fit.

- Having removed the systematic biases of users and trendiness of movies, we are now ready to estimate similarities of users and similarities of movies.

Neighbourhood Method

- Unfortunately, Least Squares fits usually suffer from overfitting: they often minimise the objective function by choosing excessively large values for the variables, relying on mutual cancellations of positive and negative terms.
- The solution to this problem is called *regularisation*: we introduce a term which penalises for large values of the variables.
- Thus, instead, we minimise the sum

$$S(\vec{\upsilon}, \vec{\mu}, \lambda) = \sum_{(j,i)\in R} (r^*(j,i) - \upsilon_i - \mu_j)^2 + \lambda \left(\sum_i \upsilon_i^2 + \sum_j \mu_j^2 \right)$$

where λ is a suitably chosen small positive constant, usually $10^{-10} \leq \lambda \leq 10^{-2}$.

- Optimal value of λ can be "learned" in a way to be described later.
- We now obtain from table $\overline{R}$ a new table $\widetilde{R}$ by replacing all $r^*(j,i)$ in $\overline{R}$ with values $\widetilde{r}(j,i) = r^*(j,i) - \upsilon_i - \mu_j$ where $\upsilon's$ and $\mu's$ were obtained by our regularised least squares fit.
- Having removed the systematic biases of users and trendiness of movies, we are now ready to estimate similarities of users and similarities of movies.

Neighbourhood Method - similarity of users

- One of the most frequently used measure of similarity of users is the *cosine similarity measure.*
- Let us first compare two users U_i and U_k. We find all movies that both users have ranked and delete all other entries $\widetilde{r}(j,i)$ and $\widetilde{r}(j',k)$ in the corresponding columns of these two users in the partial table $\widetilde{R}$ (thus, we remove ratings of movies which only one of the two users have seen and all the blank spaces).
- In this way we obtain two column vectors $\vec{u}_i$ and $\vec{u}_k$ such that the coordinates of vector $\vec{u}_i$ are the rankings of user U_i and the coordinates of vector $\vec{u}_k$ are rankings of user U_k of all the movies seen by both users.
- The similarity of the two users is measured by the cosine of the angle between these two vectors.
- Intuitively, these two users have similar tastes if the two vectors point in "similar directions".
- Recall that

$$\cos(u_i, u_k) = \frac{\langle \vec{u}_i, \vec{u}_k \rangle}{\|\vec{u}_i\| \cdot \|\vec{u}_k\|}$$

where $\langle \vec{u}_i, \vec{u}_k \rangle = \sum_p (\vec{u}_i)_p (\vec{u}_k)_p$ is the scalar product of vectors $\vec{u}_i$ and $\vec{u_k}$ and $\|\vec{u}_k\| = \sqrt{\sum_p (u_k)_p^2}$ is the norm (the "length") of vector $\vec{u}_k$.

Neighbourhood Method - similarity of users

- One of the most frequently used measure of similarity of users is the *cosine similarity measure.*
- Let us first compare tw and U_k. We find all movies that both users have ranked and delete entries $\widetilde{r}(j,i)$ and $\widetilde{r}(j',k)$ in the corresponding columns of these two u partial table $\widetilde{R}$ (thus, we remove ratings of movies which only one users have seen and all the blank spaces).
- In this way we obtain two column vectors $\vec{u}_i$ and $\vec{u}_k$ such that the coordinates of vector $\vec{u}_i$ are the rankings of user U_i and the coordinates of vector $\vec{u}_k$ are rankings of user U_k of all the movies seen by both users.
- The similarity of the two users is measured by the cosine of the angle between these two vectors.
- Intuitively, these two users have similar tastes if the two vectors point in "similar directions".
- Recall that

$$\cos(u_i, u_k) = \frac{\langle \vec{u}_i, \vec{u}_k \rangle}{\|\vec{u}_i\| \cdot \|\vec{u}_k\|}$$

where $\langle \vec{u}_i, \vec{u}_k \rangle = \sum_p (\vec{u}_i)_p (\vec{u}_k)_p$ is the scalar product of vectors $\vec{u}_i$ and $\vec{u_k}$ and $\|\vec{u}_k\| = \sqrt{\sum_p (u_k)_p^2}$ is the norm (the "length") of vector $\vec{u}_k$.

- One of the most frequently used measure of similarity of users is the *cosine similarity measure.*
- Let us first compare tw[illegible] and U_k. We find all movies that both users have ranked and delete [illegible] entries $\widetilde{r}(j,i)$ and $\widetilde{r}(j',k)$ in the corresponding columns of these two u[illegible] partial table $\widetilde{R}$ (thus, we remove ratings of movies which only one [illegible] users have seen and all the blank spaces).
- In this way we obtain two column vectors $\vec{u}_i$ and $\vec{u}_k$ such that the coordinates of vector $\vec{u}_i$ are the rankings of user U_i and the coordinates of vector $\vec{u}_k$ are rankings of user U_k of all the movies seen by both users.
- The similarity of the two users is measured by the cosine of the angle between these two vectors.
- Intuitively, these two users have similar tastes if the two vectors point in "similar directions".
- Recall that

$$\cos(u_i, u_k) = \frac{\langle \vec{u}_i, \vec{u}_k \rangle}{\|\vec{u}_i\| \cdot \|\vec{u}_k\|}$$

where $\langle \vec{u}_i, \vec{u}_k \rangle = \sum_p (\vec{u}_i)_p (\vec{u}_k)_p$ is the scalar product of vectors $\vec{u}_i$ and $\vec{u}_k$ and $\|\vec{u}_k\| = \sqrt{\sum_p (u_k)_p^2}$ is the norm (the "length") of vector $\vec{u}_k$.

- One of the most frequently used measure of similarity of users is the *cosine similarity measure.*
- Let us first compare tw[illegible] and U_k. We find all movies that both users have ranked and delete [illegible] entries $\widetilde{r}(j,i)$ and $\widetilde{r}(j',k)$ in the corresponding columns of these two u[illegible] partial table $\widetilde{R}$ (thus, we remove ratings of movies which only one [illegible] users have seen and all the blank spaces).
- In this way we obtain two column vectors $\vec{u}_i$ and $\vec{u}_k$ such that the coordinates of vector $\vec{u}_i$ are the rankings of user U_i and the coordinates of vector $\vec{u}_k$ are rankings of user U_k of all the movies seen by both users.
- The similarity of the two users is measured by the cosine of the angle between these two vectors.
- Intuitively, these two users have similar tastes if the two vectors point in "similar directions".
- Recall that

$$\cos(u_i, u_k) = \frac{\langle \vec{u}_i, \vec{u}_k \rangle}{\|\vec{u}_i\| \cdot \|\vec{u}_k\|}$$

where $\langle \vec{u}_i, \vec{u}_k \rangle = \sum_p (\vec{u}_i)_p (\vec{u}_k)_p$ is the scalar product of vectors $\vec{u}_i$ and $\vec{u_k}$ and $\|\vec{u}_k\| = \sqrt{\sum_p (u_k)_p^2}$ is the norm (the "length") of vector $\vec{u}_k$.

- One of the most freque[illegible] of users is the *cosine similarity measure.*
- Let us first compare tw[illegible] and U_k. We find all movies that both users have ranked and delete[illegible]ntries $\widetilde{r}(j,i)$ and $\widetilde{r}(j',k)$ in the corresponding columns of these two u[illegible] partial table $\widetilde{R}$ (thus, we remove ratings of movies which only one [illegible] users have seen and all the blank spaces).
- In this way we obtain two column vectors $\vec{u}_i$ and $\vec{u}_k$ such that the coordinates of vector $\vec{u}_i$ are the rankings of user U_i and the coordinates of vector $\vec{u}_k$ are rankings of user U_k of all the movies seen by both users.
- The similarity of the two users is measured by the cosine of the angle between these two vectors.
- Intuitively, these two users have similar tastes if the two vectors point in "similar directions".
- Recall that

$$\cos(u_i, u_k) = \frac{\langle \vec{u}_i, \vec{u}_k \rangle}{\|\vec{u}_i\| \cdot \|\vec{u}_k\|}$$

where $\langle \vec{u}_i, \vec{u}_k \rangle = \sum_p (\vec{u}_i)_p (\vec{u}_k)_p$ is the scalar product of vectors $\vec{u}_i$ and $\vec{u}_k$ and $\|\vec{u}_k\| = \sqrt{\sum_p (u_k)_p^2}$ is the norm (the "length") of vector $\vec{u}_k$.

- One of the most frequently used measure of similarity of users is the *cosine similarity measure.*
- Let us first compare tw… and U_k. We find all movies that both users have ranked and delete… entries $\widetilde{r}(j,i)$ and $\widetilde{r}(j',k)$ in the corresponding columns of these two u… partial table $\widetilde{R}$ (thus, we remove ratings of movies which only one… users have seen and all the blank spaces).
- In this way we obtain two column vectors $\vec{u}_i$ and $\vec{u}_k$ such that the coordinates of vector $\vec{u}_i$ are the rankings of user U_i and the coordinates of vector $\vec{u}_k$ are rankings of user U_k of all the movies seen by both users.
- The similarity of the two users is measured by the cosine of the angle between these two vectors.
- Intuitively, these two users have similar tastes if the two vectors point in "similar directions".
- Recall that

$$\cos(u_i, u_k) = \frac{\langle \vec{u}_i, \vec{u}_k \rangle}{\|\vec{u}_i\| \cdot \|\vec{u}_k\|}$$

where $\langle \vec{u}_i, \vec{u}_k \rangle = \sum_p (\vec{u}_i)_p (\vec{u}_k)_p$ is the scalar product of vectors $\vec{u}_i$ and $\vec{u_k}$ and $\|\vec{u}_k\| = \sqrt{\sum_p (u_k)_p^2}$ is the norm (the "length") of vector $\vec{u}_k$.

- Thus we define the similarity of users U_i and U_k as

$$[illegible], U_k) = \frac{\langle \vec{u}_i, \vec{u}_k \rangle}{\|\vec{u}_i\| \cdot \|\vec{u}_k\|}$$

- To explain why we div[illegible]ar product $\langle \vec{u}_i, \vec{u}_k \rangle$ by the product $\|\vec{u}_i\| \cdot \|\vec{u}_k\|$ of the norms of the tw[illegible]ote that these norms are likely to depend on the dimension of vectors $\vec{u}_i$ and $\vec{u}_k$, which in turn depends on the number of the movies these two users have both seen.

- This is not a good feature; $\text{sim}(U_i, U_k)$ should depend only on the "intrinsic similarity" of tastes of the two users and thus it should not depend on irrelevant things such as the number of movies they have both seen.

- Dividing the scalar product of the two vectors by the product of their norms results in a quantity depending only the angle between the two vectors, which more properly reflects similarity of the tastes of the two users.

- Determining the values of $\text{sim}(U_i, U_k)$ for every pair of users is a "preprocessing" step which can be updated every few days as new ratings from users are received.

- We can now predict the rating a user U_i would give to a movie M_j which U_i has not seen as follows.

Neighbourhood Method - similarity of users

- Thus we define the similarity of users U_i and U_k as

$$\mathrm{sim}(U_i, U_k) = \frac{\langle \vec{u}_i, \vec{u}_k \rangle}{\|\vec{u}_i\| \cdot \|\vec{u}_k\|}$$

- To explain why we divide the scalar product $\langle \vec{u}_i, \vec{u}_k \rangle$ by the product $\|\vec{u}_i\| \cdot \|\vec{u}_k\|$ of the norms of the two vectors, note that these norms are likely to depend on the dimension of vectors $\vec{u}_i$ and $\vec{u}_k$, which in turn depends on the number of the movies these two users have both seen.

- This is not a good feature; $\mathrm{sim}(U_i, U_k)$ should depend only on the "intrinsic similarity" of tastes of the two users and thus it should not depend on irrelevant things such as the number of movies they have both seen.

- Dividing the scalar product of the two vectors by the product of their norms results in a quantity depending only the angle between the two vectors, which more properly reflects similarity of the tastes of the two users.

- Determining the values of $\mathrm{sim}(U_i, U_k)$ for every pair of users is a "preprocessing" step which can be updated every few days as new ratings from users are received.

- We can now predict the rating a user U_i would give to a movie M_j which U_i has not seen as follows.

- Thus we define the similarity of users U_i and U_k as

$$\mathrm{sim}(U_i, U_k) = \frac{\langle \vec{u}_i, \vec{u}_k \rangle}{\|\vec{u}_i\| \cdot \|\vec{u}_k\|}$$

- To explain why we divide the scalar product $\langle \vec{u}_i, \vec{u}_k \rangle$ by the product $\|\vec{u}_i\| \cdot \|\vec{u}_k\|$ of the norms of the two vectors, note that these norms are likely to depend on the dimension of vectors $\vec{u}_i$ and $\vec{u}_k$, which in turn depends on the number of the movies these two users have both seen.

- This is not a good feature; $\mathrm{sim}(U_i, U_k)$ should depend only on the "intrinsic similarity" of tastes of the two users and thus it should not depend on irrelevant things such as the number of movies they have both seen.

- Dividing the scalar product of the two vectors by the product of their norms results in a quantity depending only the angle between the two vectors, which more properly reflects similarity of the tastes of the two users.

- Determining the values of $\mathrm{sim}(U_i, U_k)$ for every pair of users is a "preprocessing" step which can be updated every few days as new ratings from users are received.

- We can now predict the rating a user U_i would give to a movie M_j which U_i has not seen as follows.

Neighbourhood Method - similarity of users

- Thus we define the similarity of users U_i and U_k as

$$\mathrm{sim}(U_i, U_k) = \frac{\langle \vec{u}_i, \vec{u}_k \rangle}{\|\vec{u}_i\| \cdot \|\vec{u}_k\|}$$

- To explain why we divide the scalar product $\langle \vec{u}_i, \vec{u}_k \rangle$ by the product $\|\vec{u}_i\| \cdot \|\vec{u}_k\|$ of the norms of the two vectors, note that these norms are likely to depend on the dimension of vectors $\vec{u}_i$ and $\vec{u}_k$, which in turn depends on the number of the movies these two users have both seen.
- This is not a good feature; $\mathrm{sim}(U_i, U_k)$ should depend only on the "intrinsic similarity" of tastes of the two users and thus it should not depend on irrelevant things such as the number of movies they have both seen.
- Dividing the scalar product of the two vectors by the product of their norms results in a quantity depending only the angle between the two vectors, which more properly reflects similarity of the tastes of the two users.
- Determining the values of $\mathrm{sim}(U_i, U_k)$ for every pair of users is a "preprocessing" step which can be updated every few days as new ratings from users are received.
- We can now predict the rating a user U_i would give to a movie M_j which U_i has not seen as follows.

Neighbourhood Method - similarity of users

- Thus we define the similarity of users U_i and U_k as

$$[illegible], U_k) = \frac{\langle \vec{u}_i, \vec{u}_k \rangle}{\|\vec{u}_i\| \cdot \|\vec{u}_k\|}$$

- To explain why we div[illegible]ar product $\langle \vec{u}_i, \vec{u}_k \rangle$ by the product $\|\vec{u}_i\| \cdot \|\vec{u}_k\|$ of the norms of the tw[illegible]ote that these norms are likely to depend on the dimension of vectors $\vec{u}_i$ and $\vec{u}_k$, which in turn depends on the number of the movies these two users have both seen.
- This is not a good feature; $\mathrm{sim}(U_i, U_k)$ should depend only on the "intrinsic similarity" of tastes of the two users and thus it should not depend on irrelevant things such as the number of movies they have both seen.
- Dividing the scalar product of the two vectors by the product of their norms results in a quantity depending only the angle between the two vectors, which more properly reflects similarity of the tastes of the two users.
- Determining the values of $\mathrm{sim}(U_i, U_k)$ for every pair of users is a "preprocessing" step which can be updated every few days as new ratings from users are received.
- We can now predict the rating a user U_i would give to a movie M_j which U_i has not seen as follows.

Neighbourhood Method - similarity of users

- Thus we define the similarity of users U_i and U_k as

$$\mathrm{sim}(U_i, U_k) = \frac{\langle \vec{u}_i, \vec{u}_k \rangle}{\|\vec{u}_i\| \cdot \|\vec{u}_k\|}$$

- To explain why we divide the scalar product $\langle \vec{u}_i, \vec{u}_k \rangle$ by the product $\|\vec{u}_i\| \cdot \|\vec{u}_k\|$ of the norms of the two vectors, note that these norms are likely to depend on the dimension of vectors $\vec{u}_i$ and $\vec{u}_k$, which in turn depends on the number of the movies these two users have both seen.

- This is not a good feature; $\mathrm{sim}(U_i, U_k)$ should depend only on the "intrinsic similarity" of tastes of the two users and thus it should not depend on irrelevant things such as the number of movies they have both seen.

- Dividing the scalar product of the two vectors by the product of their norms results in a quantity depending only the angle between the two vectors, which more properly reflects similarity of the tastes of the two users.

- Determining the values of $\mathrm{sim}(U_i, U_k)$ for every pair of users is a "preprocessing" step which can be updated every few days as new ratings from users are received.

- We can now predict the rating a user U_i would give to a movie M_j which U_i has not seen as follows.

Neighbourhood Method - similarity of users

- Among all users who have seen movie M_j pick L many users U_{k_l} with L largest values of $|\mathrm{sim}(U_i, U_{k_l})|$.
- Note that we pick not [illegible] U_k which are the most similar (with a large positive $\mathrm{sim}(U_i, U_k)$) b[illegible] the most **dissimilar ones** (with negative $\mathrm{sim}(U_i, U_k)$).
- We now predict the rating user U_i would give to movie M_j as

$$\mathrm{pred}(j, i) = \overline{r} + v_i + \mu_j + \frac{\sum_{1 \le l \le L} \mathrm{sim}(U_i, U_{k_l})\, \widetilde{r}(j, k_l)}{\sum_{1 \le l \le L} |\mathrm{sim}(U_i, U_{k_l})|}$$

- We then recommend to user U_i movie M_j for which the predicted rating $\mathrm{pred}(j, i)$ is the highest.
- Note that "the hype factor" μ_j is brought back into the equation when deciding what to recommend.
- Factor v_i is constant across movies so it is insignificant; adding it is mostly for test purposes because it will more realistically predict the possible rating of user U_i of movie M_j allowing easy comparison in tests.

Neighbourhood Method - similarity of users

- Among all users who have seen movie M_j pick L many users U_{k_l} with L largest values of $|\mathrm{sim}(U_i, U_{k_l})|$.
- Note that we pick not only the users U_k which are the most similar (with a large positive $\mathrm{sim}(U_i, U_k)$) but also the most **dissimilar ones** (with negative $\mathrm{sim}(U_i, U_k)$).
- We now predict the rating user U_i would give to movie M_j as

$$\mathrm{pred}(j,i) = \overline{r} + v_i + \mu_j + \frac{\sum_{1\le l\le L} \mathrm{sim}(U_i, U_{k_l})\, \widetilde{r}(j, k_l)}{\sum_{1\le l\le L} |\mathrm{sim}(U_i, U_{k_l})|}$$

- We then recommend to user U_i movie M_j for which the predicted rating $\mathrm{pred}(j,i)$ is the highest.
- Note that "the hype factor" μ_j is brought back into the equation when deciding what to recommend.
- Factor v_i is constant across movies so it is insignificant; adding it is mostly for test purposes because it will more realistically predict the possible rating of user U_i of movie M_j allowing easy comparison in tests.

Neighbourhood Method - similarity of users

- Among all users who have seen movie M_j pick L many users U_{k_l} with L largest values of $|\mathrm{sim}(U_i, U_{k_l})|$.
- Note that we pick not only users U_k which are the most similar (with a large positive $\mathrm{sim}(U_i, U_k)$) but also the most **dissimilar ones** (with negative $\mathrm{sim}(U_i, U_k)$).
- We now predict the rating user U_i would give to movie M_j as

$$\mathrm{pred}(j, i) = \overline{r} + v_i + \mu_j + \frac{\sum_{1 \le l \le L} \mathrm{sim}(U_i, U_{k_l})\, \widetilde{r}(j, k_l)}{\sum_{1 \le l \le L} |\mathrm{sim}(U_i, U_{k_l})|}$$

- We then recommend to user U_i movie M_j for which the predicted rating $\mathrm{pred}(j, i)$ is the highest.
- Note that “the hype factor” μ_j is brought back into the equation when deciding what to recommend.
- Factor v_i is constant across movies so it is insignificant; adding it is mostly for test purposes because it will more realistically predict the possible rating of user U_i of movie M_j allowing easy comparison in tests.

Neighbourhood Method - similarity of users

- Among all users who have seen movie M_j pick L many users U_{k_l} with L largest values of $|\mathrm{sim}(U_i, U_{k_l})|$.
- Note that we pick not only users U_k which are the most similar (with a large positive $\mathrm{sim}(U_i, U_k)$) but also the most **dissimilar ones** (with negative $\mathrm{sim}(U_i, U_k)$).
- We now predict the rating user U_i would give to movie M_j as

$$\mathrm{pred}(j, i) = \overline{r} + v_i + \mu_j + \frac{\sum_{1 \leq l \leq L} \mathrm{sim}(U_i, U_{k_l})\, \widetilde{r}(j, k_l)}{\sum_{1 \leq l \leq L} |\mathrm{sim}(U_i, U_{k_l})|}$$

- We then recommend to user U_i movie M_j for which the predicted rating $\mathrm{pred}(j, i)$ is the highest.
- Note that "the hype factor" μ_j is brought back into the equation when deciding what to recommend.
- Factor v_i is constant across movies so it is insignificant; adding it is mostly for test purposes because it will more realistically predict the possible rating of user U_i of movie M_j allowing easy comparison in tests.

Neighbourhood Method - similarity of users

- Among all users who have seen movie M_j pick L many users U_{k_l} with L largest values of $|\text{sim}(U_i, U_{k_l})|$.
- Note that we pick not only U_k which are the most similar (with a large positive $\text{sim}(U_i, U_k)$) but also the most **dissimilar ones** (with negative $\text{sim}(U_i, U_k)$).
- We now predict the rating user U_i would give to movie M_j as

$$\text{pred}(j, i) = \overline{r} + v_i + \mu_j + \frac{\sum_{1 \le l \le L} \text{sim}(U_i, U_{k_l})\, \widetilde{r}(j, k_l)}{\sum_{1 \le l \le L} |\text{sim}(U_i, U_{k_l})|}$$

- We then recommend to user U_i movie M_j for which the predicted rating $\text{pred}(j, i)$ is the highest.
- Note that "the hype factor" μ_j is brought back into the equation when deciding what to recommend.

- Factor v_i is constant across movies so it is insignificant; adding it is mostly for test purposes because it will more realistically predict the possible rating of user U_i of movie M_j allowing easy comparison in tests.

Neighbourhood Method - similarity of users

- Among all users who have seen movie M_j pick L many users U_{k_l} with L largest values of $|\mathrm{sim}(U_i, U_{k_l})|$.
- Note that we pick not only users U_k which are the most similar (with a large positive $\mathrm{sim}(U_i, U_k)$) but also the most **dissimilar ones** (with negative $\mathrm{sim}(U_i, U_k)$).
- We now predict the rating user U_i would give to movie M_j as

$$\mathrm{pred}(j,i) = \overline{r} + v_i + \mu_j + \frac{\sum_{1\le l\le L} \mathrm{sim}(U_i, U_{k_l})\, \widetilde{r}(j, k_l)}{\sum_{1\le l\le L} |\mathrm{sim}(U_i, U_{k_l})|}$$

- We then recommend to user U_i movie M_j for which the predicted rating $\mathrm{pred}(j,i)$ is the highest.

- Note that “the hype factor” μ_j is brought back into the equation when deciding what to recommend.

- Factor v_i is constant across movies so it is insignificant; adding it is mostly for test purposes because it will more realistically predict the possible rating of user U_i of movie M_j allowing easy comparison in tests.

Neighbourhood Method - similarity of movies

- We can in a similar way estimate similarity of movies, working on columns of table $\widetilde{R}$ (instead of rows).

- For any two movies M_j [illegible] M_n consider all users which have rated both movies and form two v[illegible] and $\vec{m}_n$ with coordinates which are the ratings of the form $\widetilde{r}(j$ [illegible] $l)$ where l ranges over all users who rated both movies.
- We can now define the [illegible]ilarity between these two movies as

$$\mathrm{sim}(M_j, M_n) = \frac{\langle \vec{m}_j, \vec{m}_n \rangle}{\|\vec{m}_j\| \cdot \|\vec{m}_n\|}$$

- If we now want to predict how a user U_i would rank a movie M_j we would pick among all the movies h[illegible] L many of them for which $|\mathrm{sim}(M_j, M_{n_l})|$ are the largest.

- We now predict the rating user U_i would give to movie M_j as

$$\mathrm{pred}(j,i) = \text{[illegible]} \frac{\sum_{1\leq l\leq L} \mathrm{sim}(M_j, M_{n_l})\, \widetilde{r}(n_l, i)}{\sum_{1\leq l\leq L} |\sum_{1\leq l\leq L} \mathrm{sim}(M_j, M_{n_l})|}$$

- Again, we would recommend the movie M_j with the highest predicted value $\mathrm{pred}(j,i)$.

Neighbourhood Method - similarity of movies

- We can in a similar way estimate similarity of movies, working on columns of table $\widetilde{R}$ (instead of rows).
- For any two movies M_j and M_n consider all users which have rated both movies and form two vectors $\vec{m}_j$ and $\vec{m}_n$ with coordinates which are the ratings of the form $\tilde{r}(j, l)$ and $\tilde{r}(n, l)$ where l ranges over all users who rated both movies.
- We can now define the similarity between these two movies as

$$\mathrm{sim}(M_j, M_n) = \frac{\langle \vec{m}_j, \vec{m}_n \rangle}{\|\vec{m}_j\| \cdot \|\vec{m}_n\|}$$

- If we now want to predict how a user U_i would rank a movie M_j we would pick among all the movies he has seen L many of them for which $|\mathrm{sim}(M_j, M_{n_l})|$ are the largest.
- We now predict the rating user U_i would give to movie M_j as

$$\mathrm{pred}(j, i) = \bar{r}(j) + \frac{\sum_{1 \le l \le L} \mathrm{sim}(M_j, M_{n_l})\, \tilde{r}(n_l, i)}{\sum_{1 \le l \le L} |\mathrm{sim}(M_j, M_{n_l})|}$$

- Again, we would recommend the movie M_j with the highest predicted value $\mathrm{pred}(j, i)$.

- We can in a similar way estimate similarity of movies, working on columns of table $\widetilde{R}$ (instead of rows).
- For any two movies M_j and M_n consider all users which have rated both movies and form two vectors $\vec{m}_j$ and $\vec{m}_n$ with coordinates which are the ratings of the form $\tilde{r}(j,l)$ and $\tilde{r}(n,l)$ where l ranges over all users who rated both movies.
- We can now define the cosine similarity between these two movies as

$$\mathrm{sim}(M_j, M_n) = \frac{\langle \vec{m}_j, \vec{m}_n \rangle}{\|\vec{m}_j\| \cdot \|\vec{m}_n\|}$$

- If we now want to predict how a user U_i would rank a movie M_j we would pick among all the movies he has seen L many of them for which $|\mathrm{sim}(M_j, M_{n_l})|$ are the largest.

- We now predict the rating user U_i would give to movie M_j as

$$\mathrm{pred}(j,i) = \bar{r}(j) + \frac{\sum_{1\le l\le L} \mathrm{sim}(M_j, M_{n_l})\, \tilde{r}(n_l, i)}{\sum_{1\le l\le L} |\mathrm{sim}(M_j, M_{n_l})|}$$

- Again, we would recommend the movie M_j with the highest predicted value $\mathrm{pred}(j,i)$.

Neighbourhood Method - similarity of movies

- We can in a similar way estimate similarity of movies, working on columns of table $\widetilde{R}$ (instead of rows).
- For any two movies M[illegible] consider all users which have rated both movies and form two v[illegible] and $\vec{m}_n$ with coordinates which are the ratings of the form $\widetilde{r}(j$[illegible]$, l)$ where l ranges over all users who rated both movies.
- We can now define the [illegible]ilarity between these two movies as

$$\mathrm{sim}(M_j, M_n) = \frac{\langle \vec{m}_j, \vec{m}_n \rangle}{\|\vec{m}_j\| \cdot \|\vec{m}_n\|}$$

- If we now want to predict how a user U_i would rank a movie M_j we would pick among all the movies he has seen L many of them for which $|\mathrm{sim}(M_j, M_{n_l})|$ are the largest.

- We now predict the rating user U_i would give to movie M_j as

$$\mathrm{pred}(j,i) = \frac{\sum_{1 \le l \le L} \mathrm{sim}(M_j, M_{n_l})\, \widetilde{r}(n_l, i)}{\sum_{1 \le l \le L} |\sum_{1 \le l \le L} \mathrm{sim}(M_j, M_{n_l})|}$$

- Again, we would recommend the movie M_j with the highest predicted value $\mathrm{pred}(j,i)$.

Neighbourhood Method - similarity of movies

- We can in a similar way estimate similarity of movies, working on columns of table $\widetilde{R}$ (instead of rows).
- For any two movies M[illegible]consider all users which have rated both movies and form two v[illegible]and $\vec{m}_n$ with coordinates which are the ratings of the form $\widetilde{r}(j$[illegible]$, l)$ where l ranges over all users who rated both movies.
- We can now define the [illegible]ilarity between these two movies as

$$\mathrm{sim}(M_j, M_n) = \frac{\langle \vec{m}_j, \vec{m}_n \rangle}{\|\vec{m}_j\| \cdot \|\vec{m}_n\|}$$

- If we now want to predict how a user U_i would rank a movie M_j we would pick among all the movies he has seen L many of them for which $|\mathrm{sim}(M_j, M_{n_l})|$ are the largest.
- We now predict the rating user U_i would give to movie M_j as

$$\mathrm{pred}(j,i) = \bar{r} + \nu_i + \mu_j + \frac{\sum_{1\leq l\leq L} \mathrm{sim}(M_j, M_{n_l})\, \widetilde{r}(n_l, i)}{\sum_{1\leq l\leq L} |\sum_{1\leq l\leq L} \mathrm{sim}(M_j, M_{n_l})|}$$

- Again, we would recommend the movie M_j with the highest predicted value $\mathrm{pred}(j,i)$.

- We can in a similar way estimate similarity of movies, working on columns of table $\widetilde{R}$ (instead of rows).
- For any two movies M_j and M_n consider all users which have rated both movies and form two vectors $\vec{m}_j$ and $\vec{m}_n$ with coordinates which are the ratings of the form $\widetilde{r}(j,l)$ and $\widetilde{r}(n,l)$ where l ranges over all users who rated both movies.
- We can now define the cosine similarity between these two movies as

$$\text{sim}(M_j, M_n) = \frac{\langle \vec{m}_j, \vec{m}_n \rangle}{\|\vec{m}_j\| \cdot \|\vec{m}_n\|}$$

- If we now want to predict how a user U_i would rank a movie M_j we would pick among all the movies he has seen L many of them for which $|\text{sim}(M_j, M_{n_l})|$ are the largest.

- We now predict the rating user U_i would give to movie M_j as

$$\text{pred}(j,i) = \bar{r} + \nu_i + \mu_j + \frac{\sum_{1\leq l\leq L} \text{sim}(M_j, M_{n_l})\, \widetilde{r}(n_l, i)}{\sum_{1\leq l\leq L} |\sum_{1\leq l\leq L} \text{sim}(M_j, M_{n_l})|}$$

- Again, we would recommend the movie M_j with the highest predicted value $\text{pred}(j,i)$.

- A very different commonly used method is the **Latent Factor Method**.
- Heuristics behind the method:
  - One can argue that there is only a relatively small number (up to a few hundreds) of features a movie might posses to various extents which appeal to users' tastes and which determine how much a particular user would like such a movie.
  - Examples of such features are "action movie", "romantic movie", "famous actors", "special effects", "violence", "humour", etc.
  - Let us enumerate all of these features as $f_1, f_2, \ldots, f_N$ where N is of the order of a few tens to a few hundreds.
  - A movie can have each of these features say f_i to an extent e_i, where e_i is, say, between 0 and 10.
  - Thus, to each movie M_j there corresponds a vector $\vec{e}^{\,j}$ of length N such that its i^{th} coordinate $(\vec{e}^{\,j})_i$ represents the extent to which movie M_j has feature f_i.
  - We can now form a matrix F such that rows of F correspond to movies M_j and columns correspond to features f_i.

Latent Factor Method

- A very different commonly used method is the **Latent Factor Method**.
- Heuristics behind the method:
  - One can argue t[illegible] is only a relatively small number (up to a few hundreds) of [illegible] a movie might posses to various extents which appeal to [illegible] tastes and which determine how much a particular user would like such a movie.
  - Examples of such features are "action movie", "romantic movie", "famous actors", "special effects", "violence", "humour", etc.
  - Let us enumerate all of these features as $f_1, f_2, \ldots, f_N$ where N is of the order of a few tens to a few hundreds.
  - A movie can have some of these features, say f_i to an extent e_i, where e_i is, say, between 0 and 10.
  - Thus, to each movie M_j there corresponds a vector $\vec{e}^{\,j}$ of length N such that its i^{th} coordinate $(\vec{e}^{\,j})_i$ represents the extent to which movie M_j has feature f_i.
  - We can now form a matrix F such that rows of F correspond to movies M_j and columns correspond to features f_i.

- A very different commonly used method is the **Latent Factor Method.**
- Heuristics behind the method:
  - One can argue that there is only a relatively small number (up to a few hundreds) of features which a movie might posses to various extents which appeal to users' tastes and which determine how much a particular user would like such a movie.
  - Examples of such features are "action movie", "romantic movie", "famous actors", "special effects", "violence", "humour", etc.
  - Let us enumerate all of these features as $f_1, f_2, \ldots, f_N$ where N is of the order of a few tens to a few hundreds.
  - A movie can have each of these features, say f_i to an extent e_i, where e_i is, say, between 0 and 10.
  - Thus, to each movie M_j there corresponds a vector $\vec{e}^j$ of length N such that its i^{th} coordinate $(\vec{e}^j)_i$ represents the extent to which movie M_j has feature f_i.
  - We can now form a matrix F such that rows of F correspond to movies M_j and columns correspond to features f_i.

Latent Factor Method

- A very different commonly used method is the **Latent Factor Method**.
- Heuristics behind the method:
  - One can argue that there is only a relatively small number (up to a few hundreds) of features which a movie might posses to various extents which appeal to users' tastes and which determine how much a particular user would like such a movie.
  - Examples of such features are "action movie", "romantic movie", "famous actors", "special effects", "violence", "humour", etc.
  - Let us enumerate all of these features as $f_1, f_2, \ldots, f_N$ where N is of the order of a few tens to a few hundreds.
  - A movie can have each of these features, say f_i to an extent e_i, where e_i is, say, between 0 and 10.
  - Thus, to each movie M_j there corresponds a vector $\vec{e}^{\,j}$ of length N such that its i^{th} coordinate $(\vec{e}^{\,j})_i$ represents the extent to which movie M_j has feature f_i.
  - We can now form a matrix F such that rows of F correspond to movies M_j and columns correspond to features f_i.

- A very different commonly used method is the **Latent Factor Method**.
- Heuristics behind the method:
  - One can argue t[illegible] is only a relatively small number (up to a few hundreds) c[illegible] a movie might posses to various extents which appeal to [illegible] tastes and which determine how much a particular user would like such a movie.
  - Examples of such features are "action movie", "romantic movie", "famous actors", "special effects", "violence", "humour", etc.
  - Let us enumerate all of these features as $f_1, f_2, \ldots, f_N$ where N is of the order of a few tens to a few hundreds.
  - A movie can have each of these features, say f_i to an extent e_i, where e_i is, say, between 0 and 10.
  - Thus, to each movie M_j there corresponds a vector $\vec{e}^{\,j}$ of length N such that its i^{th} coordinate $(\vec{e}^{\,j})_i$ represents the extent to which movie M_j has feature f_i.
  - We can now form a matrix F such that rows of F correspond to movies M_j and columns correspond to features f_i.

- A very different commonly used method is the **Latent Factor Method**.
- Heuristics behind the method:
  - One can argue t[illegible] is only a relatively small number (up to a few hundreds) c[illegible] a movie might posses to various extents which appeal to [illegible] tastes and which determine how much a particular user would like such a movie.
  - Examples of such features are "action movie", "romantic movie", "famous actors", "special effects", "violence", "humour", etc.
  - Let us enumerate all of these features as $f_1, f_2, \ldots, f_N$ where N is of the order of a few tens to a few hundreds.
  - A movie can have each of these features, say f_i to an extent e_i, where e_i is, say, between 0 and 10.
  - Thus, to each movie M_j there corresponds a vector $\vec{e}^j$ of length N such that its i^{th} coordinate $(\vec{e}^j)_i$ represents the extent to which movie M_j has feature f_i.
  - We can now form a matrix F such that rows of F correspond to movies M_j and columns correspond to features f_i.

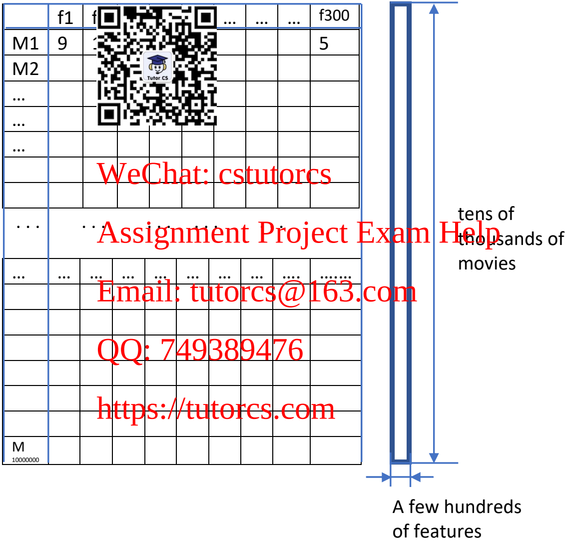
Latent Factor Method

- A very different commonly used method is the **Latent Factor Method**.
- Heuristics behind the method:
  - One can argue t[illegible] is only a relatively small number (up to a few hundreds) o[illegible] a movie might posses to various extents which appeal to [illegible] tastes and which determine how much a particular user would like such a movie.
  - Examples of such features are "action movie", "romantic movie", "famous actors", "special effects", "violence", "humour", etc.
  - Let us enumerate all of these features as $f_1, f_2, \ldots, f_N$ where N is of the order of a few tens to a few hundreds.
  - A movie can have each of these features, say f_i to an extent e_i, where e_i is, say, between 0 and 10.
  - Thus, to each movie M_j there corresponds a vector $\vec{e^j}$ of length N such that its i^{th} coordinate $(\vec{e^j})_i$ represents the extent to which movie M_j has feature f_i.
  - We can now form a matrix F such that rows of F correspond to movies M_j and columns correspond to features f_i.

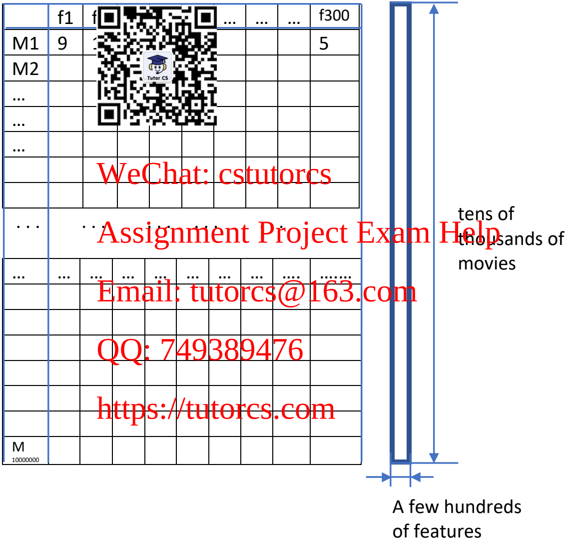
- A very different commonly used method is the **Latent Factor Method**.
- Heuristics behind the method:
  - One can argue t[illegible] is only a relatively small number (up to a few hundreds) c[illegible] a movie might posses to various extents which appeal to [illegible] tastes and which determine how much a particular user would like such a movie.
  - Examples of such features are "action movie", "romantic movie", "famous actors", "special effects", "violence", "humour", etc.
  - Let us enumerate all of these features as $f_1, f_2, \ldots, f_N$ where N is of the order of a few tens to a few hundreds.
  - A movie can have each of these features, say f_i to an extent e_i, where e_i is, say, between 0 and 10.
  - Thus, to each movie M_j there corresponds a vector $\vec{e^j}$ of length N such that its i^{th} coordinate $(\vec{e^j})_i$ represents the extent to which movie M_j has feature f_i.
  - We can now form a matrix F such that rows of F correspond to movies M_j and columns correspond to features f_i.

Latent Factor Method

- Thus, if feature f_1 is “action movie” and if $F(1, 1) = 9$ this would mean that the first movie on our list has a very significant action component.



- Thus, if feature f_1 is "action movie" and if $F(1,1) = 9$ this would mean that the first movie on our list has a very significant action component.



Latent Factor Method

- We can also associate [illegible]ser U_i a column vector $\vec{l^i}$ such that its m^{th} coordinate $(\vec{l^i})_m$ is a n[illegible]e same range of, say, 0 to 10, which tells us how much user U_i like[illegible]ature f_m in a movie.
- Thus, for example, if f[illegible] "action movie" and if for user U_1 the value of $(\vec{l^1})_1$ is 9, this would mean that user U_1 likes very much movies with a lot of action.
- On the other hand, if feature f_2 is "romantic" and the value of $(\vec{l^1})_2$ is 1, this would mean that user U_1 does not like very much movies with lots of romance.
- We can now form a matrix L whose rows correspond to features and columns correspond to users.
- If feature f_m is "special effects" and entry $L(m,i)$ in m^{th} row and i^{th} column is, say, 5, this would mean that user U_i is ambivalent towards feature f_m: he neither likes nor dislikes movies which have lots of special effects.

- We can also associate [illegible] user U_i a column vector $\vec{l^i}$ such that its m^{th} coordinate $(\vec{l^i})_m$ is a n[illegible] the same range of, say, 0 to 10, which tells us how much user U_i like[illegible] ature f_m in a movie.
- Thus, for example, if f[illegible] f_1 is "action movie" and if for user U_1 the value of $(\vec{l^1})_1$ is 9, this would mean that user U_1 likes very much movies with a lot of action.

- On the other hand, if feature f_2 is "romantic" and the value of $(\vec{l^1})_2$ is 1, this would mean that user U_1 does not like very much movies with lots of romance.

- We can now form a matrix L whose rows correspond to features and columns correspond to users.

- If feature f_m is "special effects" and entry $L(m, i)$ in m^{th} row and i^{th} column is, say, 5, this would mean that user U_i is ambivalent towards feature f_m: he neither likes nor dislikes movies which have lots of special effects.

Latent Factor Method

- We can also associate [illegible]ser U_i a column vector $\vec{l^i}$ such that its m^{th} coordinate $(\vec{l^i})_m$ is a n[illegible]e same range of, say, 0 to 10, which tells us how much user U_i like[illegible]ature f_m in a movie.
- Thus, for example, if feature f_1 is "action movie" and if for user U_1 the value of $(\vec{l^1})_1$ is 9, this would mean that user U_1 likes very much movies with a lot of action.
- On the other hand, if feature f_2 is "romantic" and the value of $(\vec{l^1})_2$ is 1, this would mean that user U_1 does not like very much movies with lots of romance.
- We can now form a matrix L whose rows correspond to features and columns correspond to users.
- If feature f_m is "special effects" and entry $L(m, i)$ in m^{th} row and i^{th} column is, say, 5, this would mean that user U_i is ambivalent towards feature f_m: he neither likes nor dislikes movies which have lots of special effects.

- We can also associate [illegible] user U_i a column vector $\vec{l^i}$ such that its m^{th} coordinate $(\vec{l^i})_m$ is a n[illegible]e same range of, say, 0 to 10, which tells us how much user U_i like[illegible]ature f_m in a movie.
- Thus, for example, if fe[illegible] f_1 is "action movie" and if for user U_1 the value of $(\vec{l^1})_1$ is 9, this would mean that user U_1 likes very much movies with a lot of action.

- On the other hand, if feature f_2 is "romantic" and the value of $(\vec{l^1})_2$ is 1, this would mean that user U_1 does not like very much movies with lots of romance.

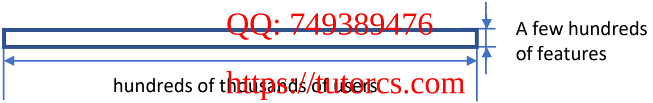
- We can now form a matrix L whose rows correspond to features and columns correspond to users.

- If feature f_m is "special effects" and entry $L(m,i)$ in m^{th} row and i^{th} column is, say, 5, this would mean that user U_i is ambivalent towards feature f_m: he neither likes nor dislikes movies which have lots of special effects.

- We can also associate ser U_i a column vector $\vec{l^i}$ such that its m^{th} coordinate $(\vec{l^i})_m$ is a n e same range of, say, 0 to 10, which tells us how much user U_i like ature f_m in a movie.
- Thus, for example, if fe "action movie" and if for user U_1 the value of $(\vec{l^1})_1$ is 9, this would mean that user U_1 likes very much movies with a lot of action.
- On the other hand, if feature f_2 is "romantic" and the value of $(\vec{l^1})_2$ is 1, this would mean that user U_1 does not like very much movies with lots of romance.
- We can now form a matrix L whose rows correspond to features and columns correspond to users.
- If feature f_m is "special effects" and entry $L(m,i)$ in m^{th} row and i^{th} column is, say, 5, this would mean that user U_i is ambivalent towards feature f_m: he neither likes nor dislikes movies which have lots of special effects.

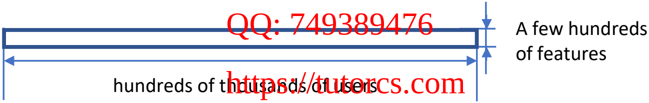
- If feature f_1 is "action movie" and feature f_2 is "romantic movie" and if $L(1, 1) = 9$ and $L(2, 1) = 1$, this would mean that the first user on our list likes movies with lots of action but does not like movies with lots of romance.

.	U1	U2	…	…	…					…	…	…	…	…	…	…	…	…	….	U 10000000
f1	9									…										
f2	1									…										
f3	7									…										
…	…									…										
fm	5									…										
…	…									…										
f300										…										



- If feature f_1 is "action d feature f_2 is "romantic movie" and if $L(1,1) = 9$ and $L(2,1)$ ould mean that the first user on our list likes movies with lots of act es not like movies with lots of romance.

.	U1	U2	…	…	…					…	…	…	…	…	…	…	…	…	….	U 10000000
f1	9									…										
f2	1									…										
f3	7									…										
…	…									…										
fm	5									…										
…	…									…										
f300										…										



- **Assume for a moment** that somehow we have access to matrix F which specifies for each movie M_j to what degree it has each feature f_m and matrix L which specifies for each user U_i how important each feature f_m is.
- Let us fix a movie M_j and its feature content vector $\vec{e}^j$.
- Thus, for every feature f_m the coordinate $(\vec{e}^j)_m$ of $\vec{e}^j$ specifies how much of feature f_m the movie M_j has.
- Let us also fix a user U_i and its feature importance vector $\vec{l}^i$.
- Thus, for every feature f_m the coordinate $(\vec{l}^i)_m$ of $\vec{l}^i$ specifies how important is that a movie has feature f_m in order for U_i to like it.
- Then for every user U_i and every movie M_j it would be easy to predict how much U_i would like M_j by evaluating the expression

$$E(j,i) = \sum_{1\le m\le N} (\vec{e}^j)_m\,(\vec{l}^i)_m = \langle \vec{e}^j, \vec{l}^i\rangle.$$

- But note that $E(j,i)$ is precisely the entry of the matrix $E = F \times L$ in j^{th} row and i^{th} column:

- **Assume for a moment** that somehow we have access to matrix F which specifies for each movie M_j to what degree it has each feature f_m and matrix L which specifies for each user U_i how important each feature f_m is.
- Let us fix a movie M_j and its feature content vector $\vec{e}^j$.
- Thus, for every feature f_m the coordinate $(\vec{e}^j)_m$ of $\vec{e}^j$ specifies how much of feature f_m the movie M_j has.
- Let us also fix a user U_i and its feature importance vector $\vec{l}^i$.

- Thus, for every feature f_m the coordinate $(\vec{l}^i)_m$ of $\vec{l}^i$ specifies how important is that a movie has feature f_m in order for U_i to like it.

- Then for every user U_i and every movie M_j it would be easy to predict how much U_i would like M_j by evaluating the expression

$$E(j,i) = \sum_{1 \le m \le N} (\vec{e}^j)_m \, (\vec{l}^i)_m = \langle \vec{e}^j, \vec{l}^i \rangle.$$

- But note that $E(j,i)$ is precisely the entry of the matrix $E = F \times L$ in j^{th} row and i^{th} column:

Latent Factor Method

- **Assume for a moment** that somehow we have access to matrix F which specifies for each movie M_j to what degree it has each feature f_m and matrix L which specifies for each user U_i how important each feature f_m is.
- Let us fix a movie M_j and its feature content vector $\vec{e}^j$.
- Thus, for every feature f_m the coordinate $(\vec{e}^j)_m$ of $\vec{e}^j$ specifies how much of feature f_m the movie M_j has.
- Let us also fix a user U_i and its feature importance vector $\vec{l}^i$.

- Thus, for every feature f_m the coordinate $(\vec{l}^i)_m$ of $\vec{l}^i$ specifies how important is that a movie has feature f_m in order for U_i to like it.

- Then for every user U_i and every movie M_j it would be easy to predict how much U_i would like M_j by evaluating the expression

$$E(j,i) = \sum_{1 \le m \le N} (\vec{e}^j)_m (\vec{l}^i)_m = \langle \vec{e}^j, \vec{l}^i \rangle.$$

- But note that $E(j,i)$ is precisely the entry of the matrix $E = F \times L$ in j^{th} row and i^{th} column:

Latent Factor Method

- **Assume for a moment** that somehow we have access to matrix F which specifies for each movie M_j to what degree it has each feature f_m and matrix L which specifies for e[illegible] how important each feature f_m is.
- Let us fix a movie M_j [illegible]ture content vector $\vec{e^j}$.
- Thus, for every feature [illegible]ordinate $(\vec{e^j})_m$ of $\vec{e^j}$ specifies how much of feature f_m the movie M_i has.
- Let us also fix a user U_i and its feature importance vector $\vec{l^i}$.

- Thus, for every feature f_m the coordinate $(\vec{l^i})_m$ of $\vec{l^i}$ specifies how important is that a movie has featu[illegible]

- Then for every user U_i and every movie M_j it would be easy to predict how much U_i would like M_j [illegible]

$$E(j,i) = \sum_{1 \le m \le N} (\vec{e^j})_m \, (\vec{l^i})_m = \langle \vec{e^j}, \vec{l^i} \rangle.$$

- But note that $E(j,i)$ is precisely the entry of the matrix $E = F \times L$ in j^{th} row and i^{th} column:

- **Assume for a moment** that somehow we have access to matrix F which specifies for each movie M_j to what degree it has each feature f_m and matrix L which specifies for each user U_i how important each feature f_m is.
- Let us fix a movie M_j and its feature content vector $\vec{e^j}$.
- Thus, for every feature f_m the coordinate $(\vec{e^j})_m$ of $\vec{e^j}$ specifies how much of feature f_m the movie M_j has.
- Let us also fix a user U_i and its feature importance vector $\vec{l^i}$.

- Thus, for every feature f_m the coordinate $(\vec{l^i})_m$ of $\vec{l^i}$ specifies how important is that a movie has feature f_m in order for U_i to like it.

- Then for every user U_i and every movie M_j it would be easy to predict how much U_i would like M_j by evaluating the expression

$$E(j,i) = \sum_{1\leq m\leq N} (\vec{e^j})_m\,(\vec{l^i})_m = \langle \vec{e^j}, \vec{l^i}\rangle.$$

- But note that $E(j,i)$ is precisely the entry of the matrix $E = F \times L$ in j^{th} row and i^{th} column:

- **Assume for a moment** that somehow we have access to matrix F which specifies for each movie M_j to what degree it has each feature f_m and matrix L which specifies for each user U_i how important each feature f_m is.
- Let us fix a movie M_j and its feature content vector $\vec{e^j}$.
- Thus, for every feature f_m the coordinate $(\vec{e^j})_m$ of $\vec{e^j}$ specifies how much of feature f_m the movie M_j has.
- Let us also fix a user U_i and its feature importance vector $\vec{l^i}$.
- Thus, for every feature f_m the coordinate $(\vec{l^i})_m$ of $\vec{l^i}$ specifies how important is that a movie has feature f_m in order for U_i to like it.
- Then for every user U_i and every movie M_j it would be easy to predict how much U_i would like M_j by evaluating the expression

$$E(j,i) = \sum_{1 \leq m \leq N} (\vec{e^j})_m \, (\vec{l^i})_m = \langle \vec{e^j}, \vec{l^i} \rangle.$$

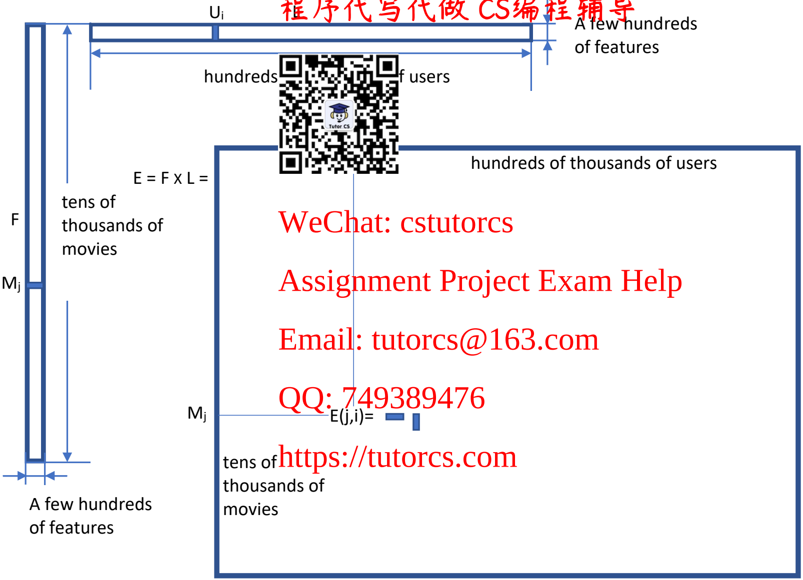
- But note that $E(j,i)$ is precisely the entry of the matrix $E = F \times L$ in j^{th} row and i^{th} column:

- **Assume for a moment** that somehow we have access to matrix F which specifies for each movie M_j to what degree it has each feature f_m and matrix L which specifies for each user U_i how important each feature f_m is.
- Let us fix a movie M_j and its feature content vector $\vec{e^j}$.
- Thus, for every feature f_m the coordinate $(\vec{e^j})_m$ of $\vec{e^j}$ specifies how much of feature f_m the movie M_j has.
- Let us also fix a user U_i and its feature importance vector $\vec{l^i}$.
- Thus, for every feature f_m the coordinate $(\vec{l^i})_m$ of $\vec{l^i}$ specifies how important is that a movie has feature f_m in order for U_i to like it.
- Then for every user U_i and every movie M_j it would be easy to predict how much U_i would like M_j by evaluating the expression

$$E(j,i) = \sum_{1 \le m \le N} (\vec{e^j})_m \, (\vec{l^i})_m = \langle \vec{e^j}, \vec{l^i} \rangle.$$

- But note that $E(j,i)$ is precisely the entry of the matrix $E = F \times L$ in j^{th} row and i^{th} column:

Latent Factor Method



Latent Factor Method

- However, there is a ve[illegible]roblem with such an approach to prediction how much would a use[illegible] movie M_j.
- How can we determine [illegible] the relevant few dozens to few hundreds of features needed to describe a movie exhaustively?
- Who would assess each movie objectively according to how much of each feature such amovie has?
- Even worse, how would we determine objectively how much each feature is important to each user?
- **Solution**: all of these should be "learned" from the partial table of the existing ratings of movies!
- We even do not need to know what the features are or what they mean.
- These features should also "emerge" from the partial table of user ratings R!

Latent Factor Method

- However, there is a ve[illegible]roblem with such an approach to prediction how much would a use[illegible] movie M_j.
- How can we determine [illegible] the relevant few dozens to few hundreds of features needed to describe a movie exhaustively?
- Who would assess each movie objectively according to how much of each feature such amovie has?
- Even worse, how would we determine objectively how much each feature is important to each user?
- **Solution**: all of these should be "learned" from the partial table of the existing ratings of movies!
- We even do not need to know what the features are or what they mean.
- These features should also "emerge" from the partial table of user ratings R!

Latent Factor Method

- However, there is a ve[illegible]roblem with such an approach to prediction how much would a use[illegible] movie M_j.
- How can we determine [illegible]he relevant few dozens to few hundreds of features needed to describe a movie exhaustively?
- Who would assess each movie objectively according to how much of each feature such amovie has?

- Even worse, how would we determine objectively how much each feature is important to each user?

- **Solution**: all of these should be "learned" from the partial table of the existing ratings of movies!

- We even do not need to know what the features are or what they mean.

- These features should also "emerge" from the partial table of user ratings R!

- However, there is a ve[illegible]roblem with such an approach to prediction how much would a use[illegible] movie M_j.
- How can we determine [illegible]he relevant few dozens to few hundreds of features needed to describe a movie exhaustively?
- Who would assess each movie objectively according to how much of each feature such amovie has?
- Even worse, how would we determine objectively how much each feature is important to each user?
- **Solution**: all of these should be "learned" from the partial table of the existing ratings of movies!
- We even do not need to know what the features are or what they mean.
- These features should also "emerge" from the partial table of user ratings R!

- However, there is a ve[illegible]roblem with such an approach to prediction how much would a use[illegible] movie M_j.
- How can we determine [illegible]he relevant few dozens to few hundreds of features needed to describe a movie exhaustively?
- Who would assess each movie objectively according to how much of each feature such amovie has?
- Even worse, how would we determine objectively how much each feature is important to each user?
- **Solution**: all of these should be "learned" from the partial table of the existing ratings of movies!
- We even do not need to know what the features are or what they mean.
- These features should also "emerge" from the partial table of user ratings R!

- However, there is a ve[illegible]roblem with such an approach to prediction how much would a use[illegible] movie M_j.
- How can we determine [illegible]he relevant few dozens to few hundreds of features needed to describe a movie exhaustively?
- Who would assess each movie objectively according to how much of each feature such amovie has?
- Even worse, how would we determine objectively how much each feature is important to each user?
- **Solution**: all of these should be "learned" from the partial table of the existing ratings of movies!
- We even do not need to know what the features are or what they mean.
- These features should also "emerge" from the partial table of user ratings R!

Latent Factor Method

- However, there is a ve[illegible]roblem with such an approach to prediction how much would a use[illegible] movie M_j.
- How can we determine [illegible]he relevant few dozens to few hundreds of features needed to describe a movie exhaustively?
- Who would assess each movie objectively according to how much of each feature such amovie has?
- Even worse, how would we determine objectively how much each feature is important to each user?
- **Solution**: all of these should be "learned" from the partial table of the existing ratings of movies!
- We even do not need to know what the features are or what they mean.
- These features should also "emerge" from the partial table of user ratings R!

- Let N be the number of "features" we want to let emerge (with no meaning assigned whatsoever). In applications N ranges between 20 and up to 200.
- Let $\#M$ be the number of movies in the database and $\#U$ be the number of users.
- **Idea:** Fill matrices F of size $\#M \times N$ and L of size $N \times \#U$ with variables $F(j,m)$ and $L(m,i)$ which yet have to be determined.
- Solve the following least squares problem in the variables $\{F(j,m) : 1 \leq j \leq \#M; 1 \leq m \leq N\} \cup \{L(m,i) : 1 \leq m \leq N; 1 \leq i \leq \#U\}$:

minimize

$$S(\vec{F}, \vec{L}) = \sum_{(j,i):R(j,i)\text{exists}} \left(\sum_{1 \leq m \leq N} F(j,m) \cdot L(m,i) - R(j,i) \right)^2$$

- Note that the total number of variables is $(\#M + \#U) \times N$.
- So N should be chosen so that $(\#M + \#U) \times N$ is a fraction of the total number of existing entries in the partially filled table R of user's ratings.
- Note that if we manage to find $F(j,m)$'s and $L(m,i)$'s which "optimally model" data, we have no way of figuring out what are the "features" these numbers are representing; they simply "emerged" from the data.

Latent Factor Method

- Let N be the number of "features" we want to let emerge (with no meaning assigned whatsoever). In applications N ranges between 20 and up to 200.
- Let $\#M$ be the number of movies in the database and $\#U$ be the number of users.
- **Idea:** Fill matrices F of size $\#M \times N$ and L of size $N \times \#U$ with variables $F(j,m)$ and $L(m,i)$ whose values yet have to be determined.
- Solve the following least squares problem in the variables $\{F(j,m) : 1 \leq j \leq \#M; 1 \leq m \leq N\} \cup \{L(m,i) : 1 \leq m \leq N; 1 \leq i \leq \#U\}$:

minimize

$$S(\vec{F}, \vec{L}) = \sum_{(j,i):R(j,i)\text{exists}} \left(\sum_{1 \leq m \leq N} F(j,m) \cdot L(m,i) - R(j,i) \right)^2$$

- Note that the total number of variables is $(\#M + \#U) \times N$.
- So N should be chosen so that $(\#M + \#U) \times N$ is a fraction of the total number of existing entries in the partially filled table R of user's ratings.
- Note that if we manage to find $F(j,m)$'s and $L(m,i)$'s which "optimally model" data, we have no way of figuring out what are the "features" these numbers are representing; they simply "emerged" from the data.

Latent Factor Method

- Let N be the number of "features" we want to let emerge (with no meaning assigned whatsoever). In applications N ranges between 20 and up to 200.
- Let $\#M$ be the number of movies in the database and $\#U$ be the number of users.
- **Idea:** Fill matrices F of size $\#M \times N$ and L of size $N \times \#U$ with variables $F(j,m)$ and $L(m,i)$ which yet have to be determined.
- Solve the following least squares problem in the variables $\{F(j,m) : 1 \leq j \leq \#M; 1 \leq m \leq N\} \cup \{L(m,i) : 1 \leq m \leq N; 1 \leq i \leq \#U\}$:

minimize

$$S(\vec{F}, \vec{L}) = \sum_{(j,i):R(j,i)\text{exists}} \left(\sum_{1 \leq m \leq N} F(j,m) \cdot L(m,i) - R(j,i) \right)^2$$

- Note that the total number of variables is $(\#M + \#U) \times N$.
- So N should be chosen so that $(\#M + \#U) \times N$ is a fraction of the total number of existing entries in the partially filled table R of user's ratings.
- Note that if we manage to find $F(j,m)$'s and $L(m,i)$'s which "optimally model" data, we have no way of figuring out what are the "features" these numbers are representing; they simply "emerged" from the data.

Latent Factor Method

- Let N be the number of "features" we want to let emerge (with no meaning assigned whatsoever). In applications N ranges between 20 and up to 200.
- Let $\#M$ be the number of movies in the database and $\#U$ be the number of users.
- **Idea:** Fill matrices F of size $\#M \times N$ and L of size $N \times \#U$ with variables $F(j,m)$ and $L(m,i)$ which yet have to be determined.
- Solve the following least squares problem in the variables $\{F(j,m) : 1 \leq j \leq \#M; 1 \leq m \leq N\} \cup \{L(m,i) : 1 \leq m \leq N; 1 \leq i \leq \#U\}$:

minimize

$$S(\vec{F}, \vec{L}) = \sum_{(j,i):R(j,i)\text{exists}} \left(\sum_{1 \leq m \leq N} F(j,m) \cdot L(m,i) - R(j,i) \right)^2$$

- Note that the total number of variables is $(\#M + \#U) \times N$.
- So N should be chosen so that $(\#M + \#U) \times N$ is a fraction of the total number of existing entries in the partially filled table R of user's ratings.
- Note that if we manage to find $F(j,m)$'s and $L(m,i)$'s which "optimally model" data, we have no way of figuring out what are the "features" these numbers are representing; they simply "emerged" from the data.

Latent Factor Method

- Let N be the number of "features" we want to let emerge (with no meaning assigned whatsoever). In applications N ranges between 20 and up to 200.
- Let $\#M$ be the number of movies in the database and $\#U$ be the number of users.
- **Idea:** Fill matrices F of size $\#M \times N$ and L of size $N \times \#U$ with variables $F(j, m)$ and $L(m, i)$ whose values yet have to be determined.
- Solve the following least squares problem in the variables $\{F(j, m) : 1 \leq j \leq \#M; 1 \leq m \leq N\} \cup \{L(m, i) : 1 \leq m \leq N; 1 \leq i \leq \#U\}$:

minimize

$$S(\vec{F}, \vec{L}) = \sum_{(j,i):R(j,i)\text{exists}} \left(\sum_{1 \leq m \leq N} F(j, m) \cdot L(m, i) - R(j, i) \right)^2$$

- Note that the total number of variables is $(\#M + \#U) \times N$.
- So N should be chosen so that $(\#M + \#U) \times N$ is a fraction of the total number of existing entries in the partially filled table R of user's ratings.
- Note that if we manage to find $F(j, m)$'s and $L(m, i)$'s which "optimally model" data, we have no way of figuring out what are the "features" these numbers are representing; they simply "emerged" from the data.

Latent Factor Method

- Let N be the number of "features" we want to let emerge (with no meaning assigned whatsoever). In applications N ranges between 20 and up to 200.
- Let $\#M$ be the number of movies in the database and $\#U$ be the number of users.
- **Idea:** Fill matrices F of size $\#M \times N$ and L of size $N \times \#U$ with variables $F(j,m)$ and $L(m,i)$ which yet have to be determined.
- Solve the following least squares problem in the variables $\{F(j,m) : 1 \leq j \leq \#M; 1 \leq m \leq N\} \cup \{L(m,i) : 1 \leq m \leq N; 1 \leq i \leq \#U\}$:

minimize

$$S(\vec{F}, \vec{L}) = \sum_{(j,i):R(j,i)\text{exists}} \left(\sum_{1 \leq m \leq N} F(j,m) \cdot L(m,i) - R(j,i) \right)^2$$

- Note that the total number of variables is $(\#M + \#U) \times N$.
- So N should be chosen so that $(\#M + \#U) \times N$ is a fraction of the total number of existing entries in the partially filled table R of user's ratings.
- Note that if we manage to find $F(j,m)$'s and $L(m,i)$'s which "optimally model" data, we have no way of figuring out what are the "features" these numbers are representing; they simply "emerged" from the data.

Latent Factor Method

- Let N be the number of "features" we want to let emerge (with no meaning assigned whatsoever). In applications N ranges between 20 and up to 200.
- Let $\#M$ be the number of movies in the database and $\#U$ be the number of users.
- **Idea:** Fill matrices F [illegible] $\times N$ and L of size $N \times \#U$ with variables $F(j,m)$ and $L(m,i)$ w[illegible] yet have to be determined.
- Solve the following least squares problem in the variables $\{F(j,m) : 1 \leq j \leq \#M; 1 \leq m \leq N\} \cup \{L(m,i) : 1 \leq m \leq N; 1 \leq i \leq \#U\}$:

minimize

$$S(\vec{F}, \vec{L}) = \sum_{(j,i):R(j,i)\text{exists}} \left(\sum_{1 \leq m \leq N} F(j,m) \cdot L(m,i) - R(j,i) \right)^2$$

- Note that the total number of variables is $(\#M + \#U) \times N$.
- So N should be chosen so that $(\#M + \#U) \times N$ is a fraction of the total number of existing entries in the partially filled table R of user's ratings.
- Note that if we manage to find $F(j,m)$'s and $L(m,i)$'s which "optimally model" data, we have no way of figuring out what are the "features" these numbers are representing; they simply "emerged" from the data.

- However, there is a serious problem with this approach: setting the partial derivatives of the objective $S(\vec{F},\vec{L})$ with respect to all variables to zero results in the following system of equations:

$$\begin{aligned}
\frac{\partial}{\partial F(j,m)} S(\vec{F},\vec{L}) &= \frac{\partial}{\partial F(j,m)} \sum_{(j,i):R(j,i)\ \text{exists}} \left(\sum_{1\le m\le N} F(j,m)\cdot L(m,i) - R(j,i) \right)^2 \\
&= 2 \sum_{i:R(j,i)\ \text{exists}} \left(\sum_{1\le m\le N} F(j,m)\cdot L(m,i) - R(j,i) \right) L(m,i) = 0;
\end{aligned}$$

$$\begin{aligned}
\frac{\partial}{\partial L(m,i)} S(\vec{F},\vec{L}) &= 2 \sum_{j:R(j,i)\ \text{exists}} \left(\sum_{1\le m\le N} F(j,m)\cdot L(m,i) - R(j,i) \right) F(j,m) = 0.
\end{aligned}$$

- However, there is a serious problem with this approach: setting the partial derivatives of the objective $S(\vec{F},\vec{L})$ with respect to all variables to zero results in the following system of equations:

$$\frac{\partial}{\partial F(j,m)} S(\vec{F},\vec{L}) = \frac{\partial}{\partial F(j,m)} \sum_{(j,i):R(j,i)\ \text{exists}} \left(\sum_{1\leq m\leq N} F(j,m)\cdot L(m,i) - R(j,i) \right)^2$$

$$= 2 \sum_{i:R(j,i)\ \text{exists}} \left(\sum_{1\leq m\leq N} F(j,m)\cdot L(m,i) - R(j,i) \right) L(m,i) = 0;$$

$$\frac{\partial}{\partial L(m,i)} S(\vec{F},\vec{L}) = 2 \sum_{j:R(j,i)\ \text{exists}} \left(\sum_{1\leq m\leq N} F(j,m)\cdot L(m,i) - R(j,i) \right) F(j,m) = 0.$$

Latent Factor Method

- This is a huge system of cubic equations and cannot be solved feasibly.
- Worse, such an optimis[illegible] problem is even not convex, so search for the optimal solution can end up in a local minimum.
- We apply an iterative [illegible] find an approximate solution.
- Note that we apply su[illegible] to "raw data" - no de-biasing like the one we performed in the N[illegible]od Method.
- Steps:
  - We initially set all variables $F(j,m)$ to the same value $F^{(0)}(j,m)$, say a median va[illegible]
  - we now solve the following Least Squares problem in variables $\{L(m,i) : 1 \leq m$ [illegible] only

$$\textit{minimize} \quad \sum_{(j,i):R(j,i)\ \text{exists}} \left(\sum_{1 \leq m \leq N} F^{(0)}(j,m) \cdot L(m,i) - R(j,i) \right)^2$$

  - Note that, since $F^{(0)}(j,m)$ are concrete numbers rather than variables, such a Least Squares problem does reduce to a system of linear equations after we find the partials and set them to zero.
  - This Least Squares can also be regularised just as previously.

Latent Factor Method

- This is a huge system of cubic equations and cannot be solved feasibly.
- Worse, such an optimisation problem is even not convex, so search for the optimal solution can end up in a local minimum.
- We apply an iterative [illegible] find an approximate solution.
- Note that we apply su[illegible]d to "raw data" - no de-biasing like the one we performed in the N[illegible]od Method.
- Steps:
  - We initially set all variables $F(j,m)$ to the same value $F^{(0)}(j,m)$, say a median va[illegible]
  - we now solve the following Least Squares problem in variables $\{L(m,i) : 1 \leq m \leq N, 1 \leq i \leq \#I\}$ only:

$$\textit{minimize} \quad \sum_{(j,i):R(j,i)\text{ exists}} \left(\sum_{1 \leq m \leq N} F^{(0)}(j,m) \cdot L(m,i) - R(j,i) \right)^2$$

  - Note that, since $F^{(0)}(j,m)$ are concrete numbers rather than variables, such a Least Squares problem does reduce to a system of linear equations after we find the partials and set them to zero.
  - This Least Squares can also be regularised just as previously.

- This is a huge system of cubic equations and cannot be solved feasibly.
- Worse, such an optimisation problem is even not convex, so search for the optimal solution can end up in a local minimum.
- We apply an iterative [illegible] find an approximate solution.
- Note that we apply su[illegible]d to "raw data" - no de-biasing like the one we performed in the N[illegible]od Method.
- Steps:
  - We initially set all variables $F(j,m)$ to the same value $F^{(0)}(j,m)$, say a median va[illegible]
  - we now solve the following Least Squares problem in variables $\{L(m,i) : 1 \leq m \leq N, 1 \leq i \leq \#I\}$ only:

minimize

$$\sum_{(j,i):R(j,i)\text{ exists}} \left(\sum_{1 \leq m \leq N} F^{(0)}(j,m) \cdot L(m,i) - R(j,i) \right)^2$$

  - Note that, since $F^{(0)}(j,m)$ are concrete numbers rather than variables, such a Least Squares problem does reduce to a system of linear equations after we find the partials and set them to zero.
  - This Least Squares can also be regularised just as previously.

Latent Factor Method

- This is a huge system of cubic equations and cannot be solved feasibly.
- Worse, such an optimisation problem is even not convex, so search for the optimal solution can end up in a local minimum.
- We apply an iterative … find an approximate solution.
- Note that we apply su… d to "raw data" - no de-biasing like the one we performed in the N… od Method.
- Steps:
  - We initially set all variables $F(j,m)$ to the same value $F^{(0)}(j,m)$, say a median va…
  - we now solve the following Least Squares problem in variables $\{L(m,i) : 1 \leq m$ …

$$\text{minimize} \quad \sum_{(j,i):R(j,i)\ \text{exists}} \left(\sum_{1\leq m\leq N} F^{(0)}(j,m)\cdot L(m,i) - R(j,i) \right)^2$$

  - Note that, since $F^{(0)}(j,m)$ are concrete numbers rather than variables, such a Least Squares problem does reduce to a system of linear equations after we find the partials and set them to zero.
  - This Least Squares can also be regularised just as previously.

- This is a huge system of cubic equations and cannot be solved feasibly.
- Worse, such an optimisation problem is even not convex, so search for the optimal solution can end up in a local minimum.
- We apply an iterative ... find an approximate solution.
- Note that we apply su... d to "raw data" - no de-biasing like the one we performed in the N... od Method.
- Steps:
  - We initially set all variables $F(j,m)$ to the same value $F^{(0)}(j,m)$, say a median va...
  - we now solve the following Least Squares problem in variables $\{L(m,i) : 1 \leq m \leq N, 1 \leq i \leq \#I\}$ only

$$\text{minimize} \quad \sum_{(j,i):R(j,i)\text{ exists}} \left(\sum_{1 \leq m \leq N} F^{(0)}(j,m) \cdot L(m,i) - R(j,i) \right)^2$$

  - Note that, since $F^{(0)}(j,m)$ are concrete numbers rather than variables, such a Least Squares problem does reduce to a system of linear equations after we find the partials and set them to zero.
  - This Least Squares can also be regularised just as previously.

Latent Factor Method

- This is a huge system of cubic equations and cannot be solved feasibly.
- Worse, such an optimisation problem is even not convex, so search for the optimal solution can end up in a local minimum.
- We apply an iterative [illegible] find an approximate solution.
- Note that we apply su[illegible]d to “raw data” - no de-biasing like the one we performed in the N[illegible]od Method.
- Steps:
  - We initially set all variables $F(j,m)$ to the same value $F^{(0)}(j,m)$, say a median value 5.
  - we now solve the following Least Squares problem in variables $\{L(m,i) : 1 \leq m \leq N, 1 \leq i \leq \#I\}$ only:

$$\text{minimize} \quad \sum_{(j,i):R(j,i)\text{ exists}} \left(\sum_{1 \leq m \leq N} F^{(0)}(j,m) \cdot L(m,i) - R(j,i) \right)^2$$

  - Note that, since $F^{(0)}(j,m)$ are concrete numbers rather than variables, such a Least Squares problem does reduce to a system of linear equations after we find the partials and set them to zero.
  - This Least Squares can also be regularised just as previously.

Latent Factor Method

- This is a huge system of cubic equations and cannot be solved feasibly.
- Worse, such an optimisation problem is even not convex, so search for the optimal solution can end up in a local minimum.
- We apply an iterative find an approximate solution.
- Note that we apply su d to "raw data" - no de-biasing like the one we performed in the N od Method.
- Steps:
  - We initially set all variables $F(j,m)$ to the same value $F^{(0)}(j,m)$, say a median value 5.
  - we now solve the following Least Squares problem in variables $\{L(m,i) : 1 \leq m \leq N; 1 \leq i \leq \#U\}$ only:

$$\textit{minimize} \quad \sum_{(j,i):R(j,i)\text{ exists}} \left(\sum_{1 \leq m \leq N} F^{(0)}(j,m) \cdot L(m,i) - R(j,i) \right)^2$$

  - Note that, since $F^{(0)}(j,m)$ are concrete numbers rather than variables, such a Least Squares problem does reduce to a system of linear equations after we find the partials and set them to zero.
  - This Least Squares can also be regularised just as previously.

- This is a huge system of cubic equations and cannot be solved feasibly.
- Worse, such an optimisation problem is even not convex, so search for the optimal solution can end up in a local minimum.
- We apply an iterative find an approximate solution.
- Note that we apply su d to “raw data” - no de-biasing like the one we performed in the N od Method.
- Steps:
  - We initially set all variables $F(j,m)$ to the same value $F^{(0)}(j,m)$, say a median value 5.
  - we now solve the following Least Squares problem in variables $\{L(m,i) : 1 \leq m \leq N, 1 \leq i \leq \#U\}$ only:

$$\text{minimize} \sum_{(j,i):R(j,i)\ \text{exists}} \left(\sum_{1 \leq m \leq N} F^{(0)}(j,m) \cdot L(m,i) - R(j,i) \right)^2$$

  - Note that, since $F^{(0)}(j,m)$ are concrete numbers rather than variables, such a Least Squares problem does reduce to a system of linear equations after we find the partials and set them to zero.
  - This Least Squares can also be regularised just as previously.

Latent Factor Method

- This is a huge system of cubic equations and cannot be solved feasibly.
- Worse, such an optimisation problem is even not convex, so search for the optimal solution can end up in a local minimum.
- We apply an iterative [illegible] find an approximate solution.
- Note that we apply su[illegible]d to "raw data" - no de-biasing like the one we performed in the N[illegible]od Method.
- Steps:
  - We initially set all variables $F(j,m)$ to the same value $F^{(0)}(j,m)$, say a median value 5.
  - we now solve the following Least Squares problem in variables $\{L(m,i) : 1 \leq m \leq N, 1 \leq i \leq \#U\}$ only:

minimize

$$\sum_{(j,i):R(j,i)\text{ exists}} \left(\sum_{1 \leq m \leq N} F^{(0)}(j,m) \cdot L(m,i) - R(j,i) \right)^2$$

  - Note that, since $F^{(0)}(j,m)$ are concrete numbers rather than variables, such a Least Squares problem does reduce to a system of linear equations after we find the partials and set them to zero.
  - This Least Squares can also be regularised just as previously.

- Steps (continued):
  - Let $L^{(0)}(m,i)$ be the solutions to such a Least Squares problem.
  - We now solve the following Least Squares problem in variables $\{F(j,m) : 1 \leq m \leq N; 1 \leq i \leq \#U\}$ only:

$$\text{minimize} \sum_{(j,i):R(j,i)\text{ exists}} \left(\sum_{1 \leq m \leq N} F(j,m) \cdot L^{(0)}(m,i) - R(j,i) \right)^2$$

  - Note that, since $L^{(0)}(m,i)$ are constants (numbers) (obtained as the solutions of the previous Least Squares problem) rather than variables, such a Least Squares problem again reduces to a system of linear equations after we find the partials and set them to zero.
  - Let $F^{(1)}(j,m)$ be the solutions to such a Least Squares problem; we now use these values to solve the following Least Squares problem in variables $\{L(m,i) : 1 \leq m \leq N; 1 \leq i \leq \#U\}$ only:

Latent Factor Method

- Steps (continued):
  - Let $L^{(0)}(m,i)$ be the solutions to such a Least Squares problem.
  - We now solve the following Least Squares problem in variables $\{F(j,m) : 1 \leq$ [illegible] $i \leq \#U\}$ only:

    minimize

$$\sum_{(j,i):R(j,i)\ [illegible]} \left(\sum_{[illegible]} F(j,m) \cdot L^{(0)}(m,i) - R(j,i) \right)^2$$

  - Note that, since $L^{(0)}(m,i)$ are constant numbers (obtained as the solutions of the previous Least Squares problem) rather than variables, such a Least Squares problem again reduces to a system of linear equations after we find the partials and set them to zero.
  - Let $F^{(1)}(j,m)$ be the solutions to such a Least Squares problem; we now use these values to solve the following Least Squares problem in variables $\{L(m,i) : 1 \leq m \leq N; 1 \leq i \leq \#U\}$ only:

- Steps (continued):
  - Let $L^{(0)}(m,i)$ be the solutions to such a Least Squares problem.
  - We now solve the following Least Squares problem in variables $\{F(j,m) : 1 \leq m \leq N; 1 \leq i \leq \#U\}$ only:

minimize

$$\sum_{(j,i):R(j,i)\text{ exists}} \left(\sum_{1\leq m\leq N} F(j,m) \cdot L^{(0)}(m,i) - R(j,i) \right)^2$$

  - Note that, since $L^{(0)}(m,i)$ are constant numbers (obtained as the solutions of the previous Least Squares problem) rather than variables, such a Least Squares problem again reduces to a system of linear equations after we find the partials and set them to zero.
  - Let $F^{(1)}(j,m)$ be the solutions to such a Least Squares problem; we now use these values to solve the following Least Squares problem in variables $\{L(m,i) : 1 \leq m \leq N; 1 \leq i \leq \#U\}$ only:

- Steps (continued):
  - Let $L^{(0)}(m,i)$ be the solutions to such a Least Squares problem.
  - We now solve the following Least Squares problem in variables $\{F(j,m) : 1 \leq m \leq N; 1 \leq i \leq \#U\}$ only:

    minimize

$$\sum_{(j,i):R(j,i)\ \text{exists}} \left(\sum_{1 \leq m \leq N} F(j,m) \cdot L^{(0)}(m,i) - R(j,i) \right)^2$$

  - Note that, since $G^{(0)}(m,i)$ are concrete numbers (obtained as the solutions of the previous Least Squares problem) rather than variables, such a Least Squares problem again reduces to a system of linear equations after we find the partials and set them to zero.
  - Let $F^{(1)}(j,m)$ be the solutions to such a Least Squares problem; we now use these values to solve the following Least Squares problem in variables $\{L(m,i) : 1 \leq m \leq N; 1 \leq i \leq \#U\}$ only:

Latent Factor Method

- Steps (continued):
  - Let $L^{(0)}(m,i)$ be the solutions to such a Least Squares problem.
  - We now solve the following Least Squares problem in variables $\{F(j,m) : 1 \leq m \leq N; 1 \leq i \leq \#U\}$ only:

minimize

$$\sum_{(j,i):R(j,i)\ \text{exists}} \left(\sum_{1 \leq m \leq N} F(j,m) \cdot L^{(0)}(m,i) - R(j,i) \right)^2$$

  - Note that, since $G^{(0)}(m,i)$ are concrete numbers (obtained as the solutions of the previous Least Squares problem) rather than variables, such a Least Squares problem again reduces to a system of linear equations after we find the partials and set them to zero.
  - Let $F^{(1)}(j,m)$ be the solutions to such a Least Squares problem; we now use these values to solve the following Least Squares problem in variables $\{L(m,i) : 1 \leq m \leq N; 1 \leq i \leq \#U\}$ only:

- Steps (continued):

  - *minimize*

$$\sum_{(j,i):R(j,i)} \left(\sum_{m\le N} F^{(1)}(j,m)\cdot L(m,i) - R(j,i)\right)^2$$

  - We keep alternating between taking either $\{F(j,m) : 1 \leq m \leq N; 1 \leq j \leq \#U\}$ or $\{L(m,i) : 1 \leq m \leq N; 1 \leq i \leq \#U\}$ as free variables, fixing the values of the other ones from the previously obtained solution to the corresponding Least Squares problem.
  - This method is sometimes called "*Method of Alternating Projections*".
  - We stop such iterations when

$$\sum_{(j,m)} (F^{(k)}(j,m) - F^{(k-1)}(j,m))^2 + \sum_{(i,m)} (L^{(k)}(m,i) - L^{(k-1)}(m,i))^2$$

becomes smaller than an accuracy threshold $\varepsilon > 0$.

- Steps (continued):

  - *minimize*

$$\sum_{(j,i):R(j,i)} \left(\sum_{1\le m\le N} F^{(1)}(j,m) \cdot L(m,i) - R(j,i) \right)^2$$

  - We keep alternating between taking either $\{F(j,m) : 1 \le m \le N; 1 \le i \le \#U\}$ or $\{L(m,i) : 1 \le m \le N; 1 \le i \le \#U\}$ as free variables, fixing the values of the other set from the previously obtained solution to the corresponding Least Squares problem.

  - This method is sometimes called "*Method of Alternating Projections*".
  - We stop such iterations when

$$\sum_{(j,m)} (F^{(k)}(j,m) - F^{(k-1)}(j,m))^2 + \sum_{(i,m)} (L^{(k)}(m,i) - L^{(k-1)}(m,i))^2$$

becomes smaller than an accuracy threshold $\varepsilon > 0$.

Latent Factor Method

- Steps (continued):
  - *minimize*

$$\sum_{(j,i):R(j,i)} \left(\sum_{1 \le m \le N} F^{(1)}(j,m) \cdot L(m,i) - R(j,i) \right)^2$$

  - We keep alternating between taking either $\{F(j,m) : 1 \le m \le N; 1 \le j \le \#U\}$ or $\{L(m,i) : 1 \le m \le N; 1 \le i \le \#U\}$ as free variables, fixing the values of the other set from the previously obtained solution to the corresponding Least Squares problem.
  - This method is sometimes called "*Method of Alternating Projections*".
  - We stop such iterations when

$$\sum_{(j,m)} (F^{(k)}(j,m) - F^{(k-1)}(j,m))^2 + \sum_{(i,m)} (L^{(k)}(m,i) - L^{(k-1)}(m,i))^2$$

becomes smaller than an accuracy threshold $\varepsilon > 0$.

- Steps (continued):
  - *minimize*

$$\sum_{(j,i):R(j,i)} \left(\sum_{m\le N} F^{(1)}(j,m)\cdot L(m,i) - R(j,i)\right)^2$$

  - We keep alternating between taking either $\{F(j,m) : 1 \le m \le N; 1 \le i \le \#U\}$ or $\{L(m,i) : 1 \le m \le N; 1 \le i \le \#U\}$ as free variables, fixing the values of the other set from the previously obtained solution to the corresponding Least Squares problem.
  - This method is sometimes called "*Method of Alternating Projections*".
  - We stop such iterations when

$$\sum_{(j,m)} (F^{(k)}(j,m) - F^{(k-1)}(j,m))^2 + \sum_{(i,m)} (L^{(k)}(m,i) - L^{(k-1)}(m,i))^2$$

becomes smaller than an accuracy threshold $\varepsilon > 0$.

Latent Factor Method

- Steps (continued):

  - After we obtain the values $F^{(k)}(j,m)$ and $L^{(k)}(m,i)$ from the last iteration k, we f[illegible]rresponding matrices F of size $\#M \times N$ and L of size N[illegible]

$$\widetilde{F} = \Big([illegible] \Big) : 1 \leq j \leq \#M; \;\; 1 \leq m \leq N \Big);$$

$$\widetilde{L} = \Big(L^{(k)}(m,i) : 1 \leq m \leq N; \;\; 1 \leq i \leq \#U; \Big).$$

  - We finally set $E = F \times L$ as the final matrix of predicted ratings of all movies by al[illegible] where $E(j,i)$ is the prediction of the rating of movie M_j by user U_i.

- Each of N "features" [illegible]hich $F(j,m)$ is supposed to "measure" in a movie M_j is a "latent factor" which we have no way of describing.
- Some computer scienti[illegible] but the recommender systems based on the Latent Factor Method perform remarkably well in many domains.

- Most likely this is because they are able to leverage the "global information", based on the relationship of ALL ratings, more effectively than the Neighbourhood Methods which use ratings in a more "localised way".

Latent Factor Method

- Steps (continued):
  - After we obtain the values $F^{(k)}(j,m)$ and $L^{(k)}(m,i)$ from the last iteration k, we form the corresponding matrices F of size $\#M \times N$ and L of size $N \times \#U$:

$$\widetilde{F} = \left(F^{(k)}(j,m) : 1 \leq j \leq \#M; \quad 1 \leq m \leq N\right);$$

$$\widetilde{L} = \left(L^{(k)}(m,i) : 1 \leq m \leq N; \quad 1 \leq i \leq \#U;\right).$$

  - We finally set $E = \widetilde{F} \times \widetilde{L}$ as the final matrix of predicted ratings of all movies by all users, where $E(j,i)$ is the prediction of the rating of movie M_j by user U_i.
- Each of N "features" f_m of movie M_j which $F(j,m)$ is supposed to "measure" in a movie M_j is a "latent factor" which we have no way of describing.
- Some computer scientists find this troubling, but the recommender systems based on the Latent Factor Method perform remarkably well in many domains.
- Most likely this is because they are able to leverage the "global information", based on the relationship of ALL ratings, more effectively than the Neighbourhood Methods which use ratings in a more "localised way".

- Steps (continued):

  - After we obtain the values $F^{(k)}(j,m)$ and $L^{(k)}(m,i)$ from the last iteration k, we f… …orresponding matrices F of size $\#M \times N$ and L of size N…

$$\widetilde{F} = \Big(\ldots \Big) : 1 \leq j \leq \#M; \;\; 1 \leq m \leq N \Big);$$

$$\widetilde{L} = \Big(L^{(k)}(m,i) : 1 \leq m \leq N; \;\; 1 \leq i \leq \#U; \Big).$$

  - We finally set $E = \widetilde{F} \times \widetilde{L}$ as the final matrix of predicted ratings of all movies by all users, where $E(j,i)$ is the prediction of the rating of movie M_j by user U_i.

- Each of N "features" … which $F(j,m)$ is supposed to "measure" in a movie M_j is a "latent factor" which we have no way of describing.
- Some computer scienti… but the recommender systems based on the Latent Factor Method perform remarkably well in many domains.
- Most likely this is because they are able to leverage the "global information", based on the relationship of ALL ratings, more effectively than the Neighbourhood Methods which use ratings in a more "localised way".

Latent Factor Method

- Steps (continued):
  - After we obtain the values $F^{(k)}(j, m)$ and $L^{(k)}(m, i)$ from the last iteration k, we form the corresponding matrices F of size $\#M \times N$ and L of size $N \times \#U$:

$$\widetilde{F} = \left(F^{(k)}(j, m) : 1 \leq j \leq \#M; \;\; 1 \leq m \leq N \right);$$

$$\widetilde{L} = \left(L^{(k)}(m, i) : 1 \leq m \leq N; \;\; 1 \leq i \leq \#U; \right).$$

  - We finally set $E = \widetilde{F} \times \widetilde{L}$ as the final matrix of predicted ratings of all movies by all users, where $E(j, i)$ is the prediction of the rating of movie M_j by user U_i.
- Each of N "features" f_m of movies M_j which $F(j, m)$ is supposed to "measure" in a movie M_j is a "latent factor" which we have no way of describing.
- Some computer scientists find this troubling, but the recommender systems based on the Latent Factor Method perform remarkably well in many domains.
- Most likely this is because they are able to leverage the "global information", based on the relationship of ALL ratings, more effectively than the Neighbourhood Methods which use ratings in a more "localised way".

- Steps (continued):
  - After we obtain the values $F^{(k)}(j,m)$ and $L^{(k)}(m,i)$ from the last iteration k, we [illegible]orresponding matrices F of size $\#M \times N$ and L of size N[illegible]

$$\widetilde{F} = \text{[illegible]}) : 1 \leq j \leq \#M; \;\; 1 \leq m \leq N \Big);$$

$$\widetilde{L} = \Big(L^{(k)}(m,i) : 1 \leq m \leq N; \;\; 1 \leq i \leq \#U; \Big).$$

  - We finally set $E = \widetilde{F} \times \widetilde{L}$ as the final matrix of predicted ratings of all movies by all users, where $E(j,i)$ is the prediction of the rating of movie M_j by user U_i.
- Each of N "features" f_m of movies M_j which $F(j,m)$ is supposed to "measure" in a movie M_j is a "latent factor" which we have no way of describing.
- Some computer scientists find this troubling, but the recommender systems based on the Latent Factor Method perform remarkably well in many domains.
- Most likely this is because they are able to leverage the "global information", based on the relationship of ALL ratings, more effectively than the Neighbourhood Methods which use ratings in a more "localised way".

Latent Factor Method

- Steps (continued):
  - After we obtain the values $F^{(k)}(j,m)$ and $L^{(k)}(m,i)$ from the last iteration k, we form the corresponding matrices F of size $\#M \times N$ and L of size $N \times \#U$:

$$\widetilde{F} = \left(F^{(k)}(j,m) : 1 \leq j \leq \#M; \;\; 1 \leq m \leq N \right);$$

$$\widetilde{L} = \left(L^{(k)}(m,i) : 1 \leq m \leq N; \;\; 1 \leq i \leq \#U; \right).$$

  - We finally set $E = \widetilde{F} \times \widetilde{L}$ as the final matrix of predicted ratings of all movies by all users, where $E(j,i)$ is the prediction of the rating of movie M_j by user U_i.
- Each of N "features" f_m of movies M_j which $F(j,m)$ is supposed to "measure" in a movie M_j is a "latent factor" which we have no way of describing.
- Some computer scientists find this troubling, but the recommender systems based on the Latent Factor Method perform remarkably well in many domains.
- Most likely this is because they are able to leverage the "global information", based on the relationship of ALL ratings, more effectively than the Neighbourhood Methods which use ratings in a more "localised way".

- So we presented two kinds of recommender systems:
  - the Neighbourhood Method (in two flavours, one based on the similarity of users, the other based on similarity of movies)
  - the Latent Factor Method, which can be deployed with different number N of "latent factors" (in applications usually between 20 and 200)
- So how do we decide which one we should use in a particular application?
- How can we evaluate how effective a particular choice of a recommender system is?
- **Idea:** We use real existing data. As an example we use the *Netflix Challenge* competition.
- Netflix provided approximately 100 million actual ratings of 480, 000 users, rating 17, 770 movies.
- The competition was to stay open till a submission was able to beat the Netflix's own recommender system *Cinematch* by more than 10% and then all the competitors had 30 days to submit an algorithm which was the final entry.
- The team with the best performing algorithm would get a prize of 1 million US dollars.

Recommender Systems - conclusions

- So we presented two kinds of recommender systems:
  - the Neighbourhood Method (in two flavours, one based on the similarity of users, the other based on similarity of movies)
  - the Latent Factor Method which can be deployed with different number N of "latent factors" (in applications usually between 20 and 200)
- So how do we decide which one we should use in a particular application?
- How can we evaluate how effective a particular choice of a recommender system is?
- **Idea:** We use real existing data. As an example we use the *Netflix Challenge* competition.
- Netflix provided approximately 100 million actual ratings of $480,000$ users, rating $17,770$ movies.
- The competition was to stay open till a submission was able to beat the Netflix's own recommender system *Cinematch* by more than 10% and then all the competitors had 30 days to submit an algorithm which was the final entry.
- The team with the best performing algorithm would get a prize of 1 million US dollars.

- So we presented two kinds of recommender systems:
  - the Neighbourhood Method (in two flavours, one based on the similarity of use[illegible]other based on similarity of movies)
  - the Latent Fact[illegible] which can be deployed with different number N of “l[illegible]rs” (in applications usually between 20 and 200)
- So how do we decide which one we should use in a particular application?
- How can we evaluate how effective a particular choice of a recommender system is?
- **Idea:** We use real existing data. As an example we use the *Netflix Challenge* competition.
- Netflix provided approximately 100 million actual ratings of $480,000$ users, rating $17,770$ movies.
- The competition was to stay open till a submission was able to beat the Netflix's own recommender system *Cinematch* by more than 10% and then all the competitors had 30 days to submit an algorithm which was the final entry.
- The team with the best performing algorithm would get a prize of 1 million US dollars.

Recommender Systems - conclusions

- So we presented two kinds of recommender systems:
  - the Neighbourhood Method (in two flavours, one based on the similarity of users, the other based on similarity of movies)
  - the Latent Factor Method which can be deployed with different number N of "latent factors" (in applications usually between 20 and 200)
- So how do we decide which one we should use in a particular application?

- How can we evaluate how effective a particular choice of a recommender system is?
- **Idea:** We use real existing data. As an example we use the *Netflix Challenge* competition.
- Netflix provided approximately 100 million actual ratings of $480,000$ users, rating $17,770$ movies.
- The competition was to stay open till a submission was able to beat the Netflix's own recommender system *Cinematch* by more than 10% and then all the competitors had 30 days to submit an algorithm which was the final entry.
- The team with the best performing algorithm would get a prize of 1 million US dollars.

Recommender Systems - conclusions

- So we presented two kinds of recommender systems:
  - the Neighbourhood Method (in two flavours, one based on the similarity of users, the other based on similarity of movies)
  - the Latent Factor Method which can be deployed with different number N of "latent factors" (in applications usually between 20 and 200)
- So how do we decide which one we should use in a particular application?
- How can we evaluate how effective a particular choice of a recommender system is?
- **Idea:** We use real existing data. As an example we use the *Netflix Challenge* competition.
- Netflix provided approximately 100 million actual ratings of $480,000$ users, rating $17,770$ movies.
- The competition was to stay open till a submission was able to beat the Netflix's own recommender system *Cinematch* by more than 10% and then all the competitors had 30 days to submit an algorithm which was the final entry.
- The team with the best performing algorithm would get a prize of 1 million US dollars.

Recommender Systems - conclusions

- So we presented two kinds of recommender systems:
  - the Neighbourhood Method (in two flavours, one based on the similarity of use[illegible]other based on similarity of movies)
  - the Latent Fact[illegible] which can be deployed with different number N of “l[illegible]rs” (in applications usually between 20 and 200)
- So how do we decide which one we should use in a particular application?

- How can we evaluate how effective a particular choice of a recommender system is?

- **Idea:** We use real existing data. As an example we use the *Netflix Challenge* competition.

- Netflix provided approximately 100 million actual ratings of 480, 000 users, rating 17, 770 movies.

- The competition was to stay open till a submission was able to beat the Netflix's own recomme[illegible]*h* by more than 10% and then all the competitors had 30 days to submit an algorithm which was the final entry.

- The team with the best performing algorithm would get a prize of 1 million US dollars.

Recommender Systems - conclusions

- So we presented two kinds of recommender systems:
  - the Neighbourhood Method (in two flavours, one based on the similarity of users, the other based on similarity of movies)
  - the Latent Factor Method, which can be deployed with different number N of "latent factors" (in applications usually between 20 and 200)
- So how do we decide which one we should use in a particular application?

- How can we evaluate how effective a particular choice of a recommender system is?

- **Idea:** We use real existing data. As an example we use the *Netflix Challenge* competition.

- Netflix provided approximately 100 million actual ratings of $480,000$ users, rating $17,770$ movies.

- The competition was to stay open till a submission was able to beat the Netflix's own recommender system *Cinematch* by more than 10% and then all the competitors had 30 days to submit an algorithm which was the final entry.

- The team with the best performing algorithm would get a prize of 1 million US dollars.

Recommender Systems - conclusions

- So we presented two kinds of recommender systems:
  - the Neighbourhood Method (in two flavours, one based on the similarity of users, the other based on similarity of movies)
  - the Latent Factor Method which can be deployed with different number N of “latent factors” (in applications usually between 20 and 200)
- So how do we decide which one we should use in a particular application?
- How can we evaluate how effective a particular choice of a recommender system is?
- **Idea:** We use real existing data. As an example we use the *Netflix Challenge* competition.
- Netflix provided approximately 100 million actual ratings of $480,000$ users, rating $17,770$ movies.
- The competition was to stay open till a submission was able to beat the Netflix’s own recommender system *Cinematch* by more than 10% and then all the competitors had 30 days to submit an algorithm which was the final entry.
- The team with the best performing algorithm would get a prize of 1 million US dollars.

Recommender Systems - conclusions

- So we presented two kinds of recommender systems:
  - the Neighbourhood Method (in two flavours, one based on the similarity of users, the other based on similarity of movies)
  - the Latent Factor Method which can be deployed with different number N of "latent factors" (in applications usually between 20 and 200)
- So how do we decide which one we should use in a particular application?
- How can we evaluate how effective a particular choice of a recommender system is?
- **Idea:** We use real existing data. As an example we use the *Netflix Challenge* competition.
- Netflix provided approximately 100 million actual ratings of $480,000$ users, rating $17,770$ movies.
- The competition was to stay open till a submission was able to beat the Netflix's own recommender system *Cinematch* by more than 10% and then all the competitors had 30 days to submit an algorithm which was the final entry.
- The team with the best performing algorithm would get a prize of 1 million US dollars.

Recommender Systems - conclusions

- But how was the performance of the proposed algorithms measured?
- 100 million ratings that were made available was to serve as the training data set R for the algorithms.
- The test consisted of about 1.4 million ratings which were NOT included in the 100 million ratings training data set R and not available to the teams.
- However, all the users and movies of these 1.4 million ratings were included in the 100 million ratings made available (but with these users rating different movies in the training data set than the movies they rated in these 1.4 million test ratings).
- The algorithms had to predict these 1.4 million ratings on the basis of such 100 million ratings training data set.
- The accuracy was measured by the RMS (Root Mean Square) error

$$rms\_error = \sqrt{\frac{\sum_{(j,i)\in T}(T(j,i) - P_a(j,i))^2}{1.4 \times 10^6}}$$

- Here $T(j,i)$ are the actual ratings included in the test set T (but not included in the training set R made available to the competitors)
- $P_a(j,i)$ were the predictions of algorithm a, made on the basis of the massive training data set R which contained other ratings of the users involved in the test set T as well as ratings of other users of the movies involved in T (as well as many other ratings by users and of movies not involved in T).

Recommender Systems - conclusions

- But how was the performance of the proposed algorithms measured?
- 100 million ratings that were made available was to serve as the training data set R for the algorithms.
- The test consisted of a[illegible]4 million ratings which were NOT included in the 100 million ratings [illegible]ning data set R and not available to the teams.
- However, all the users [illegible] movies of these 1.4 million ratings were included in the 100 million ratings made available (but with these users rating different movies in the training data set than the movies they rated in these 1.4 million test ratings).
- The algorithms had to predict these 1.4 million ratings on the basis of such 100 million ratings training data set.
- The accuracy was measured by the RMS (Root Mean Square) error

$$rms\_error = \sqrt{\frac{\sum_{(j,i)\in T}(T(j,i) - P_a(j,i))^2}{1.4 \times 10^6}}$$

- Here $T(j,i)$ are the actual ratings included in the test set T (but not included in the training set R made available to the competitors)
- $P_a(j,i)$ were the predictions of algorithm a, made on the basis of the massive training data set R which contained other ratings of the users involved in the test set T as well as ratings of other users of the movies involved in T (as well as many other ratings by users and of movies not involved in T).

Recommender Systems - conclusions

- But how was the performance of the proposed algorithms measured?
- 100 million ratings that were made available was to serve as the training data set R for the algorithms.
- The test consisted of a 1.4 million ratings which were NOT included in the 100 million ratings training data set R and not available to the teams.
- However, all the users and movies of these 1.4 million ratings were included in the 100 million ratings made available (but with these users rating different movies in the training data set than the movies they rated in these 1.4 million test ratings).
- The algorithms had to predict these 1.4 million ratings on the basis of such 100 million ratings training data set.
- The accuracy was measured by the RMS (Root Mean Square) error

$$rms\_error = \sqrt{\frac{\sum_{(j,i)\in T}(T(j,i) - P_a(j,i))^2}{1.4 \times 10^6}}$$

- Here $T(j,i)$ are the actual ratings included in the test set T (but not included in the training set R made available to the competitors)
- $P_a(j,i)$ were the predictions of algorithm a, made on the basis of the massive training data set R which contained other ratings of the users involved in the test set T as well as ratings of other users of the movies involved in T (as well as many other ratings by users and of movies not involved in T).

Recommender Systems - conclusions

- But how was the performance of the proposed algorithms measured?
- 100 million ratings that were made available was to serve as the training data set R for the algorithms.
- The test consisted of about 1.4 million ratings which were NOT included in the 100 million ratings in the training data set R and not available to the teams.
- However, all the users and all the movies of these 1.4 million ratings were included in the 100 million ratings made available (but with these users rating different movies in the training data set than the movies they rated in these 1.4 million test ratings).
- The algorithms had to predict these 1.4 million ratings on the basis of such 100 million ratings training data set.
- The accuracy was measured by the RMS (Root Mean Square) error

$$rms\_error = \sqrt{\frac{\sum_{(j,i)\in T}(T(j,i) - P_a(j,i))^2}{1.4 \times 10^6}}$$

- Here $T(j,i)$ are the actual ratings included in the test set T (but not included in the training set R made available to the competitors)
- $P_a(j,i)$ were the predictions of algorithm a, made on the basis of the massive training data set R which contained other ratings of the users involved in the test set T as well as ratings of other users of the movies involved in T (as well as many other ratings by users and of movies not involved in T).

Recommender Systems - conclusions

- But how was the performance of the proposed algorithms measured?
- 100 million ratings that were made available was to serve as the training data set R for the algorithms.
- The test consisted of a[illegible]4 million ratings which were NOT included in the 100 million ratings [illegible]ning data set R and not available to the teams.
- However, all the users [illegible] movies of these 1.4 million ratings were included in the 100 million ratings made available (but with these users rating different movies in the training data set than the movies they rated in these 1.4 million test ratings).
- The algorithms had to predict these 1.4 million ratings on the basis of such 100 million ratings training data set.
- The accuracy was measured by the RMS (Root Mean Square) error

$$rms\_error = \sqrt{\frac{\sum_{(j,i)\in T}(T(j,i) - P_a(j,i))^2}{1.4 \times 10^6}}$$

- Here $T(j,i)$ are the actual ratings included in the test set T (but not included in the training set R made available to the competitors)
- $P_a(j,i)$ were the predictions of algorithm a, made on the basis of the massive training data set R which contained other ratings of the users involved in the test set T as well as ratings of other users of the movies involved in T (as well as many other ratings by users and of movies not involved in T).

- But how was the performance of the proposed algorithms measured?
- 100 million ratings that were made available was to serve as the training data set R for the algorithms.
- The test consisted of a[illegible]4 million ratings which were NOT included in the 100 million ratings[illegible]ning data set R and not available to the teams.
- However, all the users [illegible] movies of these 1.4 million ratings were included in the 100 million ratings made available (but with these users rating different movies in the training data set than the movies they rated in these 1.4 million test ratings).
- The algorithms had to predict these 1.4 million ratings on the basis of such 100 million ratings training data set.
- The accuracy was measured by the RMS (Root Mean Square) error

$$rms\_error = \sqrt{\frac{\sum_{(j,i)\in T}(T(j,i) - P_a(j,i))^2}{1.4 \times 10^6}}$$

- Here $T(j,i)$ are the actual ratings included in the test set T (but not included in the training set R made available to the competitors)
- $P_a(j,i)$ were the predictions of algorithm a, made on the basis of the massive training data set R which contained other ratings of the users involved in the test set T as well as ratings of other users of the movies involved in T (as well as many other ratings by users and of movies not involved in T).

Recommender Systems - conclusions

- But how was the performance of the proposed algorithms measured?
- 100 million ratings that were made available was to serve as the training data set R for the algorithms.
- The test consisted of a [illegible]4 million ratings which were NOT included in the 100 million ratings [illegible]ning data set R and not available to the teams.
- However, all the users [illegible] movies of these 1.4 million ratings were included in the 100 million ratings made available (but with these users rating different movies in the training data set than the movies they rated in these 1.4 million test ratings).
- The algorithms had to predict these 1.4 million ratings on the basis of such 100 million ratings training data set.
- The accuracy was measured by the RMS (Root Mean Square) error

$$rms\_error = \sqrt{\frac{\sum_{(j,i)\in T}(T(j,i) - P_a(j,i))^2}{1.4 \times 10^6}}$$

- Here $T(j,i)$ are the actual ratings included in the test set T (but not included in the training set R made available to the competitors)
- $P_a(j,i)$ were the predictions of algorithm a, made on the basis of the massive training data set R which contained other ratings of the users involved in the test set T as well as ratings of other users of the movies involved in T (as well as many other ratings by users and of movies not involved in T).

Recommender Systems - conclusions

- But how was the performance of the proposed algorithms measured?
- 100 million ratings that were made available was to serve as the training data set R for the algorithms.
- The test consisted of a[illegible]4 million ratings which were NOT included in the 100 million ratings[illegible]ning data set R and not available to the teams.
- However, all the users [illegible] movies of these 1.4 million ratings were included in the 100 million ratings made available (but with these users rating different movies in the training data set than the movies they rated in these 1.4 million test ratings).
- The algorithms had to predict these 1.4 million ratings on the basis of such 100 million ratings training data set.
- The accuracy was measured by the RMS (Root Mean Square) error

$$rms\_error = \sqrt{\frac{\sum_{(j,i)\in T}(T(j,i) - P_a(j,i))^2}{1.4 \times 10^6}}$$

- Here $T(j,i)$ are the actual ratings included in the test set T (but not included in the training set R made available to the competitors)
- $P_a(j,i)$ were the predictions of algorithm a, made on the basis of the massive training data set R which contained other ratings of the users involved in the test set T as well as ratings of other users of the movies involved in T (as well as many other ratings by users and of movies not involved in T).

Recommender Systems - conclusions

- But instead of picking one recommender system over another one, we can also combine several recommender systems as follows.
- Let $P_k(j,i)$ be the predicted ratings of a recommender system S_k, $1 \leq k \leq B$, where we have B many recommender systems.
- We can now form a combined prediction as a weighted average

$$= \sum_{1 \leq k \leq B} w_k P_k(j,i)$$

where $\sum_{1 \leq k \leq B} w_k = 1$ are positive weight factors.

- But how do we determine optimal weights w_k, and also optimal values of other parameters such as the regularisation factors λ and the number N of Latent Factors?
- The answer is pretty mundane: by an arduous trial and error procedure:
- If we have a massive training data set as in the case of the Netflix prize, we can remove quite a few smaller testing subsets T_q of ratings and then use the algorithm with different values of the parameters to predict these removed test ratings.
- We can then measure the RMS error of the predictions on these test data sets T_q with different values of the parameters, trying to tweak the parameters till we get as smaller error as possible, but making sure that we do not overfit, by using reasonably diverse and numerous tests sets T_q.

Recommender Systems - conclusions

- But instead of picking one recommender system over another one, we can also combine several recommender systems as follows.
- Let $P_k(j,i)$ be the predicted ratings of a recommender system S_k, $1 \leq k \leq B$, where we have B many recommender systems.
- We can now form a combined prediction as a weighted average

$$= \sum_{1\leq k\leq B} w_k P_k(j,i)$$

where $\sum_{1\leq k\leq B} w_k = 1$ are positive weight factors.

- But how do we determine optimal weights w_k, and also optimal values of other parameters such as the regularisation factors λ and the number N of Latent Factors?
- The answer is pretty mundane: by an arduous trial and error procedure:
- If we have a massive training data set as in the case of the Netflix prize, we can remove quite a few smaller testing subsets T_q of ratings and then use the algorithm with different values of the parameters to predict these removed test ratings.
- We can then measure the RMS error of the predictions on these test data sets T_q with different values of the parameters, trying to tweak the parameters till we get as smaller error as possible, but making sure that we do not overfit, by using reasonably diverse and numerous tests sets T_q.

Recommender Systems - conclusions

- But instead of picking one recommender system over another one, we can also combine several recommender systems as follows.
- Let $P_k(j,i)$ be the predicted ratings of a recommender system S_k, $1 \leq k \leq B$, where we have B many recommender systems.
- We can now form a combined prediction as a weighted average

$$\quad = \sum_{1 \leq k \leq B} w_k P_k(j,i)$$

where $\sum_{1 \leq k \leq B} w_k = 1$ are positive weight factors.

- But how do we determine optimal weights w_k, and also optimal values of other parameters such as the regularisation factors λ and the number N of Latent Factors?
- The answer is pretty mundane: by an arduous trial and error procedure:
- If we have a massive training data set as in the case of the Netflix prize, we can remove quite a few smaller testing subsets T_q of ratings and then use the algorithm with different values of the parameters to predict these removed test ratings.
- We can then measure the RMS error of the predictions on these test data sets T_q with different values of the parameters, trying to tweak the parameters till we get as smaller error as possible, but making sure that we do not overfit, by using reasonably diverse and numerous tests sets T_q.

Recommender Systems - conclusions

- But instead of picking one recommender system over another one, we can also combine several recommender systems as follows.
- Let $P_k(j,i)$ be the predicted ratings of a recommender system S_k, $1 \leq k \leq B$, where we have B many recommender systems.
- We can now form a combined prediction as a weighted average

$$\ldots = \sum_{1 \leq k \leq B} w_k P_k(j,i)$$

where $\sum_{1 \leq k \leq B} w_k = 1$ are positive weight factors.

- But how do we determine optimal weights w_k, and also optimal values of other parameters such as the regularisation factors λ and the number N of Latent Factors?
- The answer is pretty mundane: by an arduous trial and error procedure:
- If we have a massive training data set as in the case of the Netflix prize, we can remove quite a few smaller testing subsets T_q of ratings and then use the algorithm with different values of the parameters to predict these removed test ratings.
- We can then measure the RMS error of the predictions on these test data sets T_q with different values of the parameters, trying to tweak the parameters till we get as smaller error as possible, but making sure that we do not overfit, by using reasonably diverse and numerous tests sets T_q.

Recommender Systems - conclusions

- But instead of picking one recommender system over another one, we can also combine several recommender systems as follows.
- Let $P_k(j,i)$ be the predicted ratings of a recommender system S_k, $1 \leq k \leq B$, where we have B many recommender systems.
- We can now form a combined prediction as a weighted average

$$P(j,i) = \sum_{1 \leq k \leq B} w_k P_k(j,i)$$

where $\sum_{1 \leq k \leq B} w_k = 1$ are positive weight factors.

- But how do we determine optimal weights w_k, and also optimal values of other parameters such as the regularisation factors λ and the number N of Latent Factors?
- The answer is pretty mundane: by an arduous trial and error procedure:
- If we have a massive training data set as in the case of the Netflix prize, we can remove quite a few smaller testing subsets T_q of ratings and then use the algorithm with different values of the parameters to predict these removed test ratings.
- We can then measure the RMS error of the predictions on these test data sets T_q with different values of the parameters, trying to tweak the parameters till we get as smaller error as possible, but making sure that we do not overfit, by using reasonably diverse and numerous tests sets T_q.

- But instead of picking one recommender system over another one, we can also combine several recommender systems as follows.
- Let $P_k(j,i)$ be the predicted ratings of a recommender system S_k, $1 \leq k \leq B$, where we have B many recommender systems.
- We can now form a combined prediction as a weighted average

$$= \sum_{1 \leq k \leq B} w_k P_k(j,i)$$

where $\sum_{1 \leq k \leq B} w_k = 1$ are positive weight factors.

- But how do we determine optimal weights w_k, and also optimal values of other parameters such as the regularisation factors λ and the number N of Latent Factors?

- The answer is pretty mundane: by an arduous trial and error procedure:
- If we have a massive training data set as in the case of the Netflix prize, we can remove quite a few smaller testing subsets T_q of ratings and then use the algorithm with different values of the parameters to predict these removed test ratings.

- We can then measure the RMS error of the predictions on these test data sets T_q with different values of the parameters, trying to tweak the parameters till we get as smaller error as possible, but making sure that we do not overfit, by using reasonably diverse and numerous tests sets T_q.

Recommender Systems - conclusions

- But instead of picking one recommender system over another one, we can also combine several recommender systems as follows.
- Let $P_k(j,i)$ be the predicted ratings of a recommender system S_k, $1 \leq k \leq B$, where we have B man[illegible]der systems.
- We can now form a co[illegible]ediction as a weighted average

$$[illegible] = \sum_{1 \leq k \leq B} w_k P_k(j,i)$$

where $\sum_{1 \leq k \leq B} w_k = 1$ are positive weight factors.
- But how do we determine optimal weights w_k, and also optimal values of other parameters such as the regularisation factors λ and the number N of Latent Factors?
- The answer is pretty mundane: by an arduous trial and error procedure:
- If we have a massive training data set as in the case of the Netflix prize, we can remove quite a few smaller testing subsets T_q of ratings and then use the algorithm with different values of the parameters to predict these removed test ratings.
- We can then measure the RMS error of the predictions on these test data sets T_q with different values of the parameters, trying to tweak the parameters till we get as smaller error as possible, but making sure that we do not overfit, by using reasonably diverse and numerous tests sets T_q.

Recommender Systems - conclusions

- In fact, the best performing algorithms at the Netflix competition were combinations of dozens of components with empirically tuned parameters.
- Further improvements in performance can be achieved by giving lower weights to older ratings of movies, thus also introducing the temporal dimension.
- **Conclusion:**
- The Recommender Systems, just as the Google PageRank algorithm, exemplify a design paradigm:
- The ingredient "baseline" algorithms have a sound basis employing increasingly sophisticated mathematical concepts and theorems.
- However, the final product is an empirically obtained "tweak" of such component algorithms.
- Unlike Physics, Computer Science cannot seek "definitive", exact methods and theories, especially for applications which involve subjective human factors such as taste or human opinion.
- We look for good approximations of complex and "noisy" reality, obtained from mathematically based components through empirical testing and tweaking.
- In most of engineering fields the only real criterion of the success of a new design is the commercial impact of such a design!

Recommender Systems - conclusions

- In fact, the best performing algorithms at the Netflix competition were combinations of dozens of components with empirically tuned parameters.
- Further improvements [illegible]ance can be achieved by giving lower weights to older ratings of mov[illegible]so introducing the temporal dimension.
- **Conclusion:**
- The Recommender Systems, just as the Google PageRank algorithm, exemplify a design paradigm:
- The ingredient "baseline" algorithms have a sound basis employing increasingly sophisticated mathematical concepts and theorems.
- However, the final product is an empirically obtained "tweak" of such component algorithms.
- Unlike Physics, Computer Science cannot seek "definitive", exact methods and theories, especially for applications which involve subjective human factors such as taste or human opinion.
- We look for good approximations of complex and "noisy" reality, obtained from mathematically based components through empirical testing and tweaking.
- In most of engineering fields the only real criterion of the success of a new design is the commercial impact of such a design!

Recommender Systems - conclusions

- In fact, the best performing algorithms at the Netflix competition were combinations of dozens of components with empirically tuned parameters.
- Further improvements [illegible]ance can be achieved by giving lower weights to older ratings of mov[illegible]so introducing the temporal dimension.
- **Conclusion:**
- The Recommender Systems, just as the Google PageRank algorithm, exemplify a design paradigm:
- The ingredient "baseline" algorithms have a sound basis employing increasingly sophisticated mathematical concepts and theorems.
- However, the final product is an empirically obtained "tweak" of such component algorithms.
- Unlike Physics, Computer Science cannot seek "definitive", exact methods and theories, especially for applications which involve subjective human factors such as taste or human opinion.
- We look for good approximations of complex and "noisy" reality, obtained from mathematically based components through empirical testing and tweaking.
- In most of engineering fields the only real criterion of the success of a new design is the commercial impact of such a design!

- In fact, the best performing algorithms at the Netflix competition were combinations of dozens of components with empirically tuned parameters.
- Further improvements [illegible]ance can be achieved by giving lower weights to older ratings of mov[illegible]so introducing the temporal dimension.
- **Conclusion:**
- The Recommender Systems, just as the Google PageRank algorithm, exemplify a design paradigm:

- The ingredient "baseline" algorithms have a sound basis employing increasingly sophisticated mathematical concepts and theorems.

- However, the final product is an empirically obtained "tweak" of such component algorithms.

- Unlike Physics, Computer Science cannot seek "definitive", exact methods and theories, especially for applications which involve subjective human factors such as taste or human opinion.

- We look for good approximations of complex and "noisy" reality, obtained from mathematically based components through empirical testing and tweaking.

- In most of engineering fields the only real criterion of the success of a new design is the commercial impact of such a design!

Recommender Systems - conclusions

- In fact, the best performing algorithms at the Netflix competition were combinations of dozens of components with empirically tuned parameters.
- Further improvements ... ance can be achieved by giving lower weights to older ratings of mov... so introducing the temporal dimension.
- **Conclusion:**
- The Recommender Systems, just as the Google PageRank algorithm, exemplify a design paradigm:
- The ingredient "baseline" algorithms have a sound basis employing increasingly sophisticated mathematical concepts and theorems.
- However, the final product is an empirically obtained "tweak" of such component algorithms.
- Unlike Physics, Computer Science cannot seek "definitive", exact methods and theories, especially for applications which involve subjective human factors such as taste or human opinion.
- We look for good approximations of complex and "noisy" reality, obtained from mathematically based components through empirical testing and tweaking.
- In most of engineering fields the only real criterion of the success of a new design is the commercial impact of such a design!

Recommender Systems - conclusions

- In fact, the best performing algorithms at the Netflix competition were combinations of dozens of components with empirically tuned parameters.
- Further improvements [illegible]ance can be achieved by giving lower weights to older ratings of mov[illegible]so introducing the temporal dimension.
- **Conclusion:**
- The Recommender Systems, just as the Google PageRank algorithm, exemplify a design paradigm:
- The ingredient "baseline" algorithms have a sound basis employing increasingly sophisticated mathematical concepts and theorems.
- However, the final product is an empirically obtained "tweak" of such component algorithms.
- Unlike Physics, Computer Science cannot seek "definitive", exact methods and theories, especially for applications which involve subjective human factors such as taste or human opinion.
- We look for good approximations of complex and "noisy" reality, obtained from mathematically based components through empirical testing and tweaking.
- In most of engineering fields the only real criterion of the success of a new design is the commercial impact of such a design!

- In fact, the best performing algorithms at the Netflix competition were combinations of dozens of components with empirically tuned parameters.
- Further improvements in performance can be achieved by giving lower weights to older ratings of movies, thus also introducing the temporal dimension.
- **Conclusion:**
- The Recommender Systems, just as the Google PageRank algorithm, exemplify a design paradigm:
- The ingredient "baseline" algorithms have a sound basis employing increasingly sophisticated mathematical concepts and theorems.
- However, the final product is an empirically obtained "tweak" of such component algorithms.
- Unlike Physics, Computer Science cannot seek "definitive", exact methods and theories, especially for applications which involve subjective human factors such as taste or human opinion.
- We look for good approximations of complex and "noisy" reality, obtained from mathematically based components through empirical testing and tweaking.
- In most of engineering fields the only real criterion of the success of a new design is the commercial impact of such a design!

Recommender Systems - conclusions

- In fact, the best performing algorithms at the Netflix competition were combinations of dozens of components with empirically tuned parameters.
- Further improvements [...]ance can be achieved by giving lower weights to older ratings of mov[...]so introducing the temporal dimension.
- **Conclusion:**
- The Recommender Systems, just as the Google PageRank algorithm, exemplify a design paradigm:

- The ingredient "baseline" algorithms have a sound basis employing increasingly sophisticated mathematical concepts and theorems.

- However, the final product is an empirically obtained "tweak" of such component algorithms.

- Unlike Physics, Computer Science cannot seek "definitive", exact methods and theories, especially for applications which involve subjective human factors such as taste or human opinion.

- We look for good approximations of complex and "noisy" reality, obtained from mathematically based components through empirical testing and tweaking.

- In most of engineering fields the only real criterion of the success of a new design is the commercial impact of such a design!

Recommender Systems - conclusions

- In fact, the best performing algorithms at the Netflix competition were combinations of dozens of components with empirically tuned parameters.
- Further improvements [illegible]ance can be achieved by giving lower weights to older ratings of mov[illegible]so introducing the temporal dimension.
- **Conclusion:**
- The Recommender Systems, just as the Google PageRank algorithm, exemplify a design paradigm:
- The ingredient "baseline" algorithms have a sound basis employing increasingly sophisticated mathematical concepts and theorems.
- However, the final product is an empirically obtained "tweak" of such component algorithms.
- Unlike Physics, Computer Science cannot seek "definitive", exact methods and theories, especially for applications which involve subjective human factors such as taste or human opinion.
- We look for good approximations of complex and "noisy" reality, obtained from mathematically based components through empirical testing and tweaking.
- In most of engineering fields the only real criterion of the success of a new design is the commercial impact of such a design!