Embedding-based models build embeddings for entities and relations to predict tuples.

Simplest case: relations between two entities. E.g., predicting the rating for a person on a movie (collaborative filtering).

Knowledge graphs: predict subject-verb-object triples.
Learning a relation between two entities

A relation between users and items (movies). From Movielens:

<table>
<thead>
<tr>
<th>User</th>
<th>Item</th>
<th>Rating</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>196</td>
<td>242</td>
<td>3</td>
<td>881250949</td>
</tr>
<tr>
<td>186</td>
<td>302</td>
<td>3</td>
<td>891717742</td>
</tr>
<tr>
<td>22</td>
<td>377</td>
<td>1</td>
<td>878887116</td>
</tr>
<tr>
<td>244</td>
<td>51</td>
<td>2</td>
<td>880606923</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Netflix: 500K users, 17k movies, 100M ratings (withdrawn).
- Movielens: multiple datasets from 100K to 25M ratings, with links to IMDB, plus some user properties

\[ \hat{r}_{ui} = \text{predicted rating of user } u \text{ on item } i \]

\[ Es = \text{set of } (u, i, r) \text{ tuples in the training set (ignoring timestamp)} \]

Minimize sum squares error:

\[ \sum_{(u, i, r) \in Es} (\hat{r}_{ui} - r)^2 \]

© 2023 D. L. Poole and A. K. Mackworth
Learning Relational Models with Latent Variables

- Predict same for all ratings: $\hat{r}_{ui} = \mu$
- Adjust for each user and item: $\hat{r}_{ui} = \mu + b_1[u] + b_2[i]$
  
  Question: $b_1[u]$ could be negative even though all ratings of $u$ are higher than average. How?

- Question: How can this be used to personalize recommendations?

- One latent feature: $f_i$ for each item and $g_u$ for each user

\[
\hat{r}_{ui} = \mu + b_1[u] + b_2[i] + f_1[u] \times f_2[i]
\]

- Positive $f_1[u]$ forms a soft clustering of users.
- Positive $f_2[i]$ forms a soft clustering of items.
- How do these clusterings interact?
- What about negative values?
What is being learned? (Single latent feature)

for each \((u, i, r)\): \(r\) is plotted at point \((f_1[u], f_2[i])\).

```
aipython: relnCollFilt.py
What pattern would you expect?
```
\( k \) latent features (Python notation):

\[
\hat{r}_{ui} = \mu + b_1[u] + b_2[i] + \sum_f E_1[u][f] \times E_2[i][f]
\]

- \( E_1[u] \) is the user embedding, a vector of numbers.
- \( E_2[i] \) is the item embedding, a vector of numbers.
- Regularize parameters except \( \mu \).
L2 regularization, minimize:

\[
\left( \sum_{(u, i, r) \in Es} (\hat{r}_{ui} - r)^2 \right) + \lambda \sum_{\text{parameter } p} p^2
\]
Minimize:
\[
\sum_{(u,i,r) \in E_s} \left( \mu + b_1[u] + b_2[i] + \sum_k E_1[u][f] \ast E_2[i][f] - r \right)^2
\]
\[
+ \lambda \left( \sum_i (b_1[u]^2 + \sum_f E_1[u][f]^2) + \sum_u (b_2[i]^2 + \sum_f E_2[i][f]^2) \right)
\]
where \( \lambda \) is a regularization parameter.

To find minimizing parameters:
- Gradient descent
- Iterative least squares: fix one of \( E_1 \) and \( E_2 \); the problem is ridge regression in the other.
\( \mu := \) average rating

assign \( E_1[u][f], E_2[i][f] \) randomly and assign \( b_1[i], b_2[u] \) arbitrarily

repeat:
  for each \((u, i, r) \in Es:\)
    \( e := \mu + b_1[i] + b_2[u] + \sum_k E_1[u][f] \ast E_2[i][f] - r \)
    \( b_1[i] := b_1[i] - \eta \ast e \)
    \( b_2[u] := b_2[u] - \eta \ast e \)
  for each feature \( f:\)
    \( E_1[u][f] := E_1[u][f] - \eta \ast e \ast E_2[i][f] \)
    \( E_2[i][f] := E_2[i][f] - \eta \ast e \ast E_1[u][f] \)
  for each item \( i:\)
    \( b_1[i] := b_1[i] - \eta \ast \lambda \ast b_1[i] \)
  for each feature \( f:\)
    \( E_1[u][f] := E_1[u][f] - \eta \ast \lambda \ast E_1[u][f] \)
  for each user \( u:\)
    \( b_2[u] := b_2[u] - \eta \ast \lambda \ast b_2[u] \)
  for each feature \( k:\)
    \( E_2[i][f] := E_2[i][f] - \eta \ast \lambda \ast E_2[i][f] \)
Variations

- What is you want to predict Boolean \( rating > 3 \)?
  - Use sigmoid.
  - What should we minimize?
  - How does the algorithm change?

- What if we want to predict \( rated \), where \( rated(u, i) \) is true if \( (u, i, r) \in Es \) for some \( r \)?
  - There are no negative examples!
  - Use \( k \) random examples for each positive example!
    but the average probability is \( 1/k \), which is not derived from the data.
A knowledge graph is defined in terms of triples of the form $(s, r, o)$, with subject $s$, relation (verb) $r$, and object $o$.

The extension of matrix factorization to triples is polyadic decomposition:

$$
\hat{p}((s, r, o)) = \text{sigmoid}(\mu + b_1[s] + b_2[r] + b_3[o] + \sum_f E_1[s][f] * E_2[r][f] * E_3[o][f])
$$

- a global bias $\mu$
- two biases for each entity $e$: $b_1[e]$ used when $e$ is in the first position and $b_3[e]$, for when $e$ in third position
- a bias for each relation $r$, namely $b_2[r]$
- matrixes $E_1$ and $E_3$
  - $E_1[e]$ is subject embedding for entity $e$ → latent properties
  - $E_3[e]$, is object embedding entity $e$
- matrix $E_2$, where $E_2[r]$ is relation embedding for relation $r$.

All embeddings $E_1[e]$, $E_2[r]$ and $E_3[e]$ are the same length.
To optimize log loss with $L_2$ regularization:

- same as previous algorithm with a different predictor and more parameters to tune and regularize.
- Requires negative examples, but knowledge graphs don’t have negative examples.
- Regularize $\mu$ or provide made-up negative examples.
- But, not all relations have same number of negative examples, eg. “married-to” vs “has-streamed”.
Improving Polyadic Decomposition

- Suppose triples are of the form \((u, \text{likes}, m)\) and \((m, \text{directed by}, d)\).
  How can we represent “Sam likes movies directed by Bong Joon-ho”?
The subject and object embeddings for movies are independent of each other, so this cannot be represented or learned.

- Solution: also represent \((m, \text{likes}^{-1}, u)\) and \((d, \text{directed by}^{-1}, m)\).

- Polyadic decomposition with inverses:

  \[
  \hat{p}(h, r, t) = \frac{1}{2}(\hat{pd}(h, r, t) + \hat{pd}(t, r^{-1}, h))
  \]

  where \(\hat{pd}\) is the prediction from the polyadic decomposition.
The polyadic decomposition is fully expressive: it can represent (with error less than any $\epsilon$), any relation.

Initially assume subject and object embeddings are non-negative and bounded.

Each embedding position in the subject/object embedding forms a soft clustering of entities.

For each embedding position, a relation with a high value ($\gg 0$) in that position, the entities in the subject soft cluster are related to the entities in the object soft cluster. (The product is only high when all three are high).

A relation with a value $\ll 0$ in a position forms exceptions.

For possibly negative subject and object embeddings: product of even number of negative values is positive product of odd number of negative values is negative

The addition lets these values be combined.
What does polyadic decomposition learn?

- The polyadic decomposition makes predictions by clustering entities in different ways.
- It learns about each entity; embeddings are used to predict interactions.
- It does not learn general knowledge that can be applied to other populations.