Web Mining and Recommender Systems

Temporal data mining: Regression for Sequence Data
Temporal models

This topic will look back on some of the topics already covered in this class, and see how they can be adapted to make use of temporal information

1. **Regression** – sliding windows and autoregression
2. **Recommender systems** – temporal recommendation and the Netflix Prize
3. Sequential Recommendation
Previously – Regression

Given **labeled training data** of the form

\[ \{(data_1, label_1), \ldots, (data_n, label_n)\} \]

Infer the function

\[ f(data) \overset{?}{\rightarrow} labels \]
Time-series regression

Here, we’d like to predict sequences of **real-valued** events as accurately as possible.

\[ x_1, \ldots, x_m \rightarrow x_{m+1} \rightarrow x_{m+2} \]

- **Features**
- **Label**
Here, we’d like to predict sequences of \textbf{real-valued} events as accurately as possible.

Given: a time series:

\[(x_1, \ldots, x_N) \in \mathbb{R}^N\]

Suppose we’d like to minimize the MSE (as usual!) of the final part of some continuous portion of the sequence

\[
\frac{1}{u-v+1} \sum_{t=u}^{v} (f_t(x_1, \ldots, x_{u-1}) - x_t)^2
\]
**Method 1:** maintain a “moving average” using a window of some fixed length

\[ f(x_1, \ldots, x_m) = \frac{X_m + x_{m-1} + \ldots + x_{m-K+1}}{K} \]

\[ = \sum_{k=0}^{K-1} x_{m-k} \]
**Method 1:** maintain a “moving average” using a window of some fixed length

- This can be computed efficiently via dynamic programming:

\[ f(x_1, \ldots, x_{m+1}) = K f(x_1, \ldots, x_m) - x_{m-K} + x_{m+1} \]
**Method 1:** maintain a “moving average” using a window of some fixed length

\[
f(x_1, \ldots, x_m) = \frac{1}{K} \sum_{k=0}^{K-1} x_{m-k}
\]

- This can be computed efficiently via dynamic programming:

\[
f(x_1, \ldots, x_{m+1}) = \frac{1}{K} (K \cdot f(x_1, \ldots, x_m) - x_{m-k} + x_{m+1})
\]

“peel-off” the oldest point  
add the newest point
Time-series regression

Also useful to plot data:

BeerAdvocate, ratings over time

Scatterplot

BeerAdvocate, ratings over time

Sliding window (K=10000)

long-term trends

seasonal effects

Code on course webpage
Method 2: weight the points in the moving average by age

\[ f(x_1, \ldots, x_m) = \frac{Kx_1 + (K-1)x_{n-1} + \ldots + 1x_{n-K+1}}{1+2+\ldots+K} \]
Method 2: weight the points in the moving average by age

\[ f(x_1, \ldots, x_m) = \sum_{k=0}^{K-1} \binom{K}{k} x_{m-k} \]

newest points have the highest weight
weight decays to zero after K points
Method 3: weight the most recent points exponentially higher

\[
\begin{align*}
    f(x_1) &= x_1 \\
    f(x_1, \ldots, x_m) &= \alpha f(x_1, \ldots, x_{m-1}) + (1-\alpha)x_m
\end{align*}
\]
Methods 1, 2, 3

Method 1: Sliding window
Method 2: Linear decay
Method 3: Exponential decay
Time-series regression

**Method 4:** all of these models are assigning weights to previous values using some predefined scheme, why not just learn the weights?

\[
f(x_1, \ldots, x_m) = \sum_{k=0}^{K-1} \Theta_k x_{m-k} + \Theta_0 x_m + \Theta_1 x_{m-1} + \ldots + \Theta_{K-1} x_{m-K+1}
\]

\[= \sum_{k=0}^{K-1} \Theta_k x_{m-k} \quad \text{subject to} \quad \Theta \in \arg\min \sum_{m} (f(x_1, \ldots, x_{m-1}) - x_m)^2\]
Method 4: all of these models are assigning *weights* to previous values using some predefined scheme, why not just *learn* the weights?

- We can now fit this model using least-squares
- This procedure is known as *autoregression*
- Using this model, we can capture *periodic* effects, e.g. that the traffic of a website is most similar to its traffic 7 days ago
Web Mining and Recommender Systems

Temporal recommender systems: the Netflix Prize
Extensions of latent-factor models

Lots of reasons preferences could change over time...

Figure from Koren: “Collaborative Filtering with Temporal Dynamics” (KDD 2009)
Extensions of latent-factor models

Lots of reasons preferences could change over time...

Figure from Koren: “Collaborative Filtering with Temporal Dynamics” (KDD 2009)

People tend to give higher ratings to older movies
Extensions of latent-factor models

Lots of reasons preferences could change over time...

A few temporal effects from beer reviews
Extensions of latent-factor models

Lots of reasons preferences could change over time...

There are a number of reasons why rating data might be subject to temporal effects...

- Changes in the interface
- People give higher ratings to older movies (or, people who watch older movies are a biased sample)
- The community’s preferences gradually change over time
- My girlfriend starts using my Netflix account one day
- I binge watch all 144 episodes of Buffy one week and then revert to my normal behavior
- I become a “connoisseur” of a certain type of movie
- Anchoring, public perception, seasonal effects, etc.
- More in textbook!

- e.g. “Collaborative filtering with temporal dynamics”
  Koren, 2009

- e.g. “Sequential & temporal dynamics of online opinion”
  Godes & Silva, 2012

- e.g. “Temporal recommendation on graphs via long- and short-term preference fusion”
  Xiang et al., 2010

- e.g. “Modeling the evolution of user expertise through online reviews”
  McAuley & Leskovec, 2013
Temporal recommender systems: simple heuristics
How can simple recommenders be adapted to handle temporal factors?

Previously we saw simple recommenders that predict ratings based on similarity functions:

\[
r(u, i) = \frac{1}{\sum_{j \in I_u \setminus \{i\}} r(u, j) \cdot S_{ui}(i, j)}
\]
Simple temporal models

How can this type of model be adjusted to account for temporal factors?

\[ r(u, i) = \frac{1}{Z} \sum_{j \in I_u \setminus \{i\}} c(u, j) s_m(i, j) \times f(t_{uj}) \]

\[ f(t_{uj}) = e^{-At_{uj}} \]
Simple temporal models

How can this type of model be adjusted to account for temporal factors?
Simple temporal models

How can this type of model be adjusted to account for temporal factors?

\[ r(u, i) = \frac{\sum_{j \in I_u} R_{u,j} \cdot \text{Sim}(i, j) \cdot f(t_{u,j})}{\sum_{j \in I_u} \text{Sim}(i, j) \cdot f(t_{u,j})} \]

e.g. \[ f(t) = e^{-\lambda \cdot t} \]
Simple temporal models

(see more in textbook)

- recency
- sessions
Web Mining and Recommender Systems

Temporal recommender systems: the Netflix Prize
Previously...

**Recommender Systems** go beyond the methods we’ve seen so far by trying to model the relationships between people and the items they’re evaluating.

- Preference toward "action"
- Preference toward "special effects"
- m: user’s "preferences"
- y: HP’s "properties"
- Compatibility
- Is the movie action-heavy?
- Are the special effects good?

\[ \alpha + \beta m_i + \beta y_i + \epsilon_i \]
Previously...

Predict a user’s rating of an item according to:

\[ f(u, i) = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i \]

By solving the optimization problem:

\[
\arg \min_{\alpha, \beta, \gamma} \sum_{u,i} (\alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i - R_{u,i})^2 + \lambda \left[ \sum_u \beta_u^2 + \sum_i \beta_i^2 + \sum_i \|\gamma_i\|^2 + \sum_u \|\gamma_u\|^2 \right]
\]

(error) \quad (regularizer)

(e.g. using stochastic gradient descent)
Extensions of latent-factor models

Different definitions of temporal evolution demand slightly different model assumptions (we’ll see some in more detail later!) but the basic idea is the following:

Start with our original model:

\[ f(u, i) = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i \]

And define some of the parameters as a function of time, e.g.:

\[ f(u, i, t) = \alpha + \beta_u(t) + \beta_i(t) + \gamma_u(t) \cdot \gamma_i \]
To build a reliable system (and to win the Netflix prize!) we need to account for **temporal dynamics**:

![Netflix ratings by movie age](image)

(Netflix changed their interface)

(People tend to give higher ratings to older movies)

So how was this actually done?

Figure from Koren: “Collaborative Filtering with Temporal Dynamics” (KDD 2009)
To start with, let’s just assume that it’s only the **bias** terms that explain these types of temporal variation (which, for the examples on the previous slides, is potentially enough)

\[ b_{u,i}(t) = \alpha + \beta_u(t) + \beta_i(t) \]

**Idea:** temporal dynamics for *items* can be explained by long-term, gradual changes, whereas for users we’ll need a different model that allows for “bursty”, short-lived behavior.
Temporal latent-factor models

temporal bias model:
\[ b_{u,i}(t) = \alpha + \beta_u(t) + \beta_i(t) \]

For item terms, just separate the dataset into (equally sized) bins:

\[ \beta_i(t) = \beta_i + \beta_i,\text{Bin}(t) \]

*in Koren’s paper they suggested ~30 bins corresponding to about 10 weeks each for Netflix

or bins for periodic effects (e.g. the day of the week):
\[ \beta_i(t) = \beta_i + \beta_i,\text{Bin}(t) + \beta_i,\text{period}(t) \]

What about user terms?
- We need something much finer-grained
- **But** – for most users we have far too little data to fit very short term dynamics
Temporal latent-factor models

Start with a simple model of drifting dynamics for users:

\[ \text{dev}_u(t) = \text{sign}(t - t_u) \cdot |t - t_u|^x \]

- \text{mean} rating date for user \( u \)
- \text{hyperparameter} (ended up as \( x=0.4 \) for Koren)
- days away from mean date
  - before \((-1)\) or after \((1)\) the mean date
Temporal latent-factor models

Start with a simple model of drifting dynamics for users:

\[ \text{dev}_u(t) = \text{sign}(t - t_u) \cdot |t - t_u| \times \]

- \textbf{mean} rating date for user u
- before (-1) or after (1) the mean date
- days away from mean date
- hyperparameter (ended up as \(x=0.4\) for Koren)

Time-dependent user bias can then be defined as:

\[ \beta_u^{(1)}(t) = \beta_u + \alpha_u \cdot \text{dev}_u(t) \]

- overall user bias
- sign and scale for deviation term

\[ \text{sign}(x)|x|^{0.4} \]
Temporal latent-factor models

 Netflix ratings over time

Real data

Fitted model
Temporal latent-factor models

time-dependent user bias can then be defined as:

\[ \beta_u^{(1)}(t) = \beta_u + \alpha_u \cdot \text{dev}_u(t) \]

- Requires only two parameters per user and captures some notion of temporal “drift” (even if the model found through cross-validation is (to me) completely unintuitive)
- To develop a slightly more expressive model, we can interpolate smoothly between biases using splines

\[ \text{sign}(x)|x|^{0.4} \]
Temporal latent-factor models

\[ \beta_{u}^{(2)}(t) = \beta_{u} + \sum_{l=1}^{k_u} e^{-\gamma |t - t_{u}^{l}|} \frac{b_{u}^{l}}{\sum_{l=1}^{k_u} e^{-\gamma |t - t_{u}^{l}|}} \]

- number of control points for this user \( (k_u = n_u^{0.25} \text{ in Koren}) \)
- user bias associated with this control point
- time associated with control point (uniformly spaced)
Temporal latent-factor models

\[ \beta_{u}^{(2)}(t) = \beta_{u} + \sum_{l=1}^{k_{u}} \frac{e^{-\gamma|t-t_{l}|}b_{ul}}{\sum_{l=1}^{k_{u}} e^{-\gamma|t-t_{l}|}} \]

- This is now a reasonably flexible model, but still only captures *gradual drift*, i.e., it can’t handle sudden changes (e.g. a user simply having a bad day)
Temporal latent-factor models

• Koren got around this just by adding a “per-day” user bias:

\[
\beta_{u,t}
\]

bias for a particular day (or session)

• Of course, this is only useful for particular days in which users have a lot of (abnormal) activity

• The final (time-evolving bias) model then combines all of these factors:

\[
\beta_{u,i}(t) = \alpha + \beta_u + \alpha_u \cdot \text{dev}_u(t) + \beta_{u,t} + \beta_i + \beta_{i,Bin}(t)
\]
Finally, we can add a time-dependent scaling factor:

$$\beta_{u,i}(t) = \alpha + \beta_u + \alpha_u \cdot \text{dev}_u(t) + \beta_{u,t} + (\beta_i + \beta_{i,\text{Bin}(t)}) \cdot c_u(t)$$

*also* defined as $c_u + c_{u,t}$

Latent factors can also be defined to evolve in the same way:

$$\gamma_{u,k}(t) = \gamma_{u,k} + \alpha_{u,k} \cdot \text{dev}_u(t) + \gamma_{u,k,t}$$

- factor-dependent user drift
- factor-dependent short-term effects
Summary

• Effective modeling of temporal factors was absolutely critical to this solution outperforming alternatives on Netflix’s data
• In fact, even with only temporally evolving bias terms, their solution was already ahead of Netflix’s previous (“Cinematch”) model

On the other hand...

• Many of the ideas here depend on dynamics that are quite specific to “Netflix-like” settings
• Some factors (e.g. short-term effects) depend on a high density of data per-user and per-item, which is not always available
Moral(s) of the story

How much do these extensions help?

Moral: increasing complexity helps a bit, but changing the model can help a lot.

Figure from Koren: “Collaborative Filtering with Temporal Dynamics” (KDD 2009)
So what actually happened with Netflix?

• The AT&T team “BellKor”, consisting of Yehuda Koren, Robert Bell, and Chris Volinsky were early leaders. Their main insight was how to effectively incorporate temporal dynamics into recommendation on Netflix.
• Before long, it was clear that no one team would build the winning solution, and Frankenstein efforts started to merge. Two frontrunners emerged, “BellKor’s Pragmatic Chaos”, and “The Ensemble”.
• The BellKor team was the first to achieve a 10% improvement in RMSE, putting the competition in “last call” mode. The winner would be decided after 30 days.
• After 30 days, performance was evaluated on the hidden part of the test set.
• Both of the frontrunning teams had the same RMSE (up to some precision) but BellKor’s team submitted their solution 20 minutes earlier and won $1,000,000

For a less rough summary, see the Wikipedia page about the Netflix prize, and the nytimes article about the competition: http://goo.gl/WNpy7o
Moral(s) of the story

Afterword

• Netflix had a class-action lawsuit filed against them after somebody de-anonymized the competition data
• $1,000,000 seems to be incredibly cheap for a company the size of Netflix in terms of the amount of research that was devoted to the task, and the potential benefit to Netflix of having their recommendation algorithm improved by 10%
• Other similar competitions have emerged, such as the Heritage Health Prize ($3,000,000 to predict the length of future hospital visits)

• But... the winning solution never made it into production at Netflix – it’s a monolithic algorithm that is very expensive to update as new data comes in*

*source: a friend of mine told me and I have no actual evidence of this claim
Further reading:
“Collaborative filtering with temporal dynamics”
Yehuda Koren, 2009
Web Mining and Recommender Systems

Markov-Chain based recommendation
So far, our study of temporal recommendation has mostly made use of timestamps; timestamps (or features extracted from them) can tell us:

- Whether users like action movies on Fridays
- Whether older movies gradually get more popular
- Whether certain movies are popular during certain seasons
- (other than movies) whether restaurants are popular on certain occasions
- Etc.
Temporal versus Sequential Recommendation

But a timestamp *can't* tell us (for example):

- I'm likely to watch Harry Potter 2 because I just watched Harry Potter 1
- I'm likely to purchase a memory card because I just purchased a camera
- I'm likely to watch more action movies because I've recently been watching action movies

These dynamics depend on the *ordering* of actions, rather than the *absolute time* they occurred
Markov Chain models

This type of dynamic can be captured via a *Markov Chain*

Precisely: next action is conditionally independent of the interaction history given the previous action

\[
P(i^{(t)}; i^{(t-1)}, \ldots, i^{(1)}) = P(i^{(t)} = i | i^{(t-1)})
\]
Markov Chain models

This type of dynamic can be captured via a *Markov Chain*

Precisely: next action is conditionally independent of the interaction history given the previous action

\[
p(i^{(t+1)} = i \mid i^{(t)} \ldots i^{(1)}) = p(i^{(t+1)} = i \mid i^{(t)})
\]
Markov Chain models

Ideally, this type of dynamic should be personalized to a user, i.e.,

• The next action should match the user's preferences
• It should also be consistent with the previous item

\[ p(i_u^{(t+1)} = i \mid i_u^{(t)} \ldots i_u^{1}) = p(i_u^{(t+1)} = i \mid i_u^{(t)}) \]
Markov Chain models

Most of the time we don't have to worry too much about probabilities and precise mathematical formalism: we just want to fit a function

\[
 f(u, i | i_u^{(t-1)}). \]

given the user’s previous interaction
**Challenge**: whereas our previous models required modeling a UxI matrix, we must now model a UxIxl tensor:
Markov Chain models: FPMC

**Idea:** decompose the tensor into pairwise factors

\[
f(u,i|j) = x_u \cdot x_i + x_u \cdot x_i'' + x_i''' \cdot x_j'''
\]
Markov Chain models: FPMC

**Idea:** decompose the tensor into pairwise factors

\[
f(i|u, j) = \gamma_{u}^{(ui)} \cdot \gamma_{i}^{(iu)} + \gamma_{i}^{(ij)} \cdot \gamma_{j}^{(ji)} + \gamma_{u}^{(uj)} \cdot \gamma_{j}^{(ju)}
\]

This particular model is called *Factorizing Personalized Markov Chains* (FPMC)
Markov Chain models: FPMC

Note: terms cancel when contrasting examples

\[ O\left(\epsilon \hat{u} \cdot \hat{x}_i - \epsilon \hat{u} \cdot \hat{x}_i^-\right) \]
Note: can be implemented using a factorization machine!
More from the original paper:

- Original paper actually looks at sequential *baskets* rather than sequential *items*; but this is mainly an implementation detail.
Factorizing Personalized Markov Chains for Next-Basket Recommendation

F@5

Online-Shopping (sparse)

F-Measure @ Top5

Dimensionality

SBPR-FPMC
SBPR-FMC
SBPR-MF
MC dense
most popular

FMC: not personalized
MF: personalized, but not sequentially-aware
Morals of the story:

• Can improve performance by modeling third order interactions between the user, the item, and the previous item
• This is simpler than temporal models – but makes a big assumption
• Given the blowup in the interaction space, this can be handled by tensor decomposition techniques
Web Mining and Recommender Systems

Case Studies
Case studies (will only cover one or two)

- Playlist prediction and POI recommendation: inner products versus distances
  \[ \langle \mathbf{u}, \mathbf{v} \rangle \text{ vs } d(\mathbf{u}, \mathbf{v}) \]
- Translation-based recommendation (sequential recommendation as a translation operation)
- Modeling user "expertise" over time
Personalized Ranking Metric Embedding for Next New POI Recommendation

1. Markov-like model
2. $x_{t+1}$ vs $p(x_{t+1})$
**Goal:** Can we build better sequential recommendation models by using the idea of *metric embeddings*

\[ \gamma_u \cdot \gamma_i \quad \text{vs.} \quad d(\gamma_u, \gamma_i) \]

\[ -\|\gamma_u - \gamma_i\|_2^2 \]
Personalized Ranking Metric Embedding for Next New POI Recommendation

Why would we expect this to work (or not)?

inner product

movies?

comedy

metric
- POI
- songs?
- price dynamics?

congruence

@ Euler
Otherwise, goal is the same as the previous paper:

\[ p_u(i|j) \]
## Data

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#User</th>
<th>#POI</th>
<th>#Check-in</th>
<th>Time range</th>
</tr>
</thead>
<tbody>
<tr>
<td>FourSquare</td>
<td>1917</td>
<td>2675</td>
<td>155365</td>
<td>08/2010-07/2011</td>
</tr>
<tr>
<td>Gowalla</td>
<td>4996</td>
<td>6871</td>
<td>245157</td>
<td>11/2009-10/2010</td>
</tr>
</tbody>
</table>
Personalized Ranking Metric Embedding for Next New POI Recommendation

Qualitative analysis

![Graph showing the ratio of new POIs over the number of days for FourSquare and Gowalla, with a note indicating cold start.](image)
Personalized Ranking Metric Embedding for Next New POI Recommendation

Qualitative analysis

![Graphs showing qualitative analysis for FourSquare and Gowalla datasets.](image-url)
Basic model (not personalized)

\[ \hat{P}(l_j | l_i) = \frac{e^{-||X(l_j) - X(l_i)||^2}}{Z(l_i)} \]
Personalized Ranking Metric Embedding for Next New POI Recommendation

Basic model (not personalized)

\[ l_i >_{lc} l_j \iff \hat{P}(l_i|l^c) > \hat{P}(l_j|l^c) \]

\[ i > j \rightarrow d(i, c) < d(j, c) \]
Personalized Ranking Metric Embedding for Next New POI Recommendation

**Personalized version**

\[ D_{u,l,c,l} = \alpha D_{u,l}^P + (1 - \alpha) D_{l,c,l}^S \]

\[ \alpha e^{-\| \delta_u - \delta_l \|} + e^{-\| \delta_{e,c} - \delta_{e,l} \|} \]

\[ (1 - \alpha)^{n_{uv}} \]
Personalized Ranking Metric Embedding for Next New POI Recommendation

**Personalized version**

\[
D_{u,lc,l} = \begin{cases} 
D_{u,l}^P & \text{if } \Delta(l, l^c) > \tau \\
\alpha D_{u,l}^P + (1 - \alpha) D_{lc,l}^S & \text{otherwise}
\end{cases}
\]
Personalized Ranking Metric Embedding for Next New POI Recommendation

Learning

\[
P(>_{u,l} | \Theta) = P \left( (D_{u,l_{j}} - D_{u,l_{i}}) > 0 | \Theta \right) \\
= \sigma(D_{u,l_{j}} - D_{u,l_{i}})
\]
Results

(a) Precision on FourSquare  (b) Recall on FourSquare  (c) Precision on Gowalla  (d) Recall on Gowalla
Morals of the story:

• In some applications, metric embeddings might be better than inner products
• Examples could include geographical data, but also others (e.g. playlists?)
Web Mining and Recommender Systems

Playlist Prediction via Metric Embedding
ABSTRACT
Digital storage of personal music collections and cloud-based music services (e.g., Pandora, Spotify) have fundamentally changed how music is consumed. In particular, automatically generated playlists have become an important mode of accessing large music collections. The key goal of automated playlist generation is to provide the user with a coherent listening experience. In this paper, we present Latent Markov Embedding (LME), a machine learning algorithm for generating such playlists. In analogy to matrix factorization methods for collaborative filtering, the algorithm does not require songs to be described by features a priori, but it learns a representation from example playlists. We formulate this problem as a regularized maximum-likelihood embedding of Markov chains in Euclidian space, and show how addition, when using a cloud-based service like Rhapsody or Spotify, the consumer has instant on-demand access to millions of songs. This has created substantial interest in automatic playlist algorithms that can help consumers explore large collections of music. Companies like Apple and Pandora have developed successful commercial playlist algorithms, but relatively little is known about how these algorithms work and how well they perform in rigorous evaluations.

Despite the large commercial demand, comparably little scholarly work has been done on automated methods for playlist generation (e.g., [13, 4, 9, 11]), and the results to date indicate that it is far from trivial to operationally define what makes a playlist coherent. The most comprehensive study was done by [11]. Working under a model where
**Goal:** Build a recommender system that recommends sequences of songs

**Idea:** Might also use a metric embedding (consecutive songs should be “nearby” in some space)
Basic model:

\[
Pr(p^{[i]}|p^{[i-1]}) = \frac{e^{-\|X(p^{[i]}) - X(p^{[i-1]})\|^2_2}}{\sum_{j=1}^{|S|} e^{-\|X(s_j) - X(p^{[i-1]})\|^2_2}}
\]

(compare with metric model from previous section)
Basic model ("single point"): $x_i$ should be nearby
Playlist prediction via Metric Embedding

“Dual-point” model

\[ \|X(s) - X(s')\|_2 \]

\[ \|V(s) - U(s')\|_2 \]

\[ \|X_{i-1} - X_i\| \]
Playlist prediction via Metric Embedding

**Extensions:**

- Popularity biases

\[
Pr(p^{[i]}|p^{[i-1]}) = \frac{e^{-\Delta(p^{[i]},p^{[i-1]})^2 + b_i}}{\sum_j e^{-\Delta(s_j,p^{[i-1]})^2 + b_j}}
\]

basically just $\alpha_i$
Playlist prediction via Metric Embedding

Extensions:

• Personalization

\[
Pr(p^[i] | p^[i-1], u) = \frac{e^{-\Delta(p^[i], p^[i-1])^2 + A(p^[i])^T B(u)}}{\sum_j e^{-\Delta(s_j, p^[i-1])^2 + A(s_j)^T B(u)}}
\]

\[
\|x_i - x_{i-1}\| + \gamma_u \cdot \sigma_i
\]
Extensions:

• Semantic Tags

\[ Pr(X(s)|T(s)) = \mathcal{N} \left( \frac{1}{|T(s)|} \sum_{t \in T(s)} M(t), \frac{1}{2\lambda} I_d \right) \]
Extensions:

• Observable Features

\[ P_r(p[i]|p[i-1]) = \frac{e^{-\Delta(p[i],p[i-1])^2 + O(p[i])^T W O(p[i-1])}}{\sum_j e^{-\Delta(s_j,p[i-1])^2 + O(s_j)^T W O(p[i-1])}} \]
Experiments:

Yes.com playlists
• Dec 2010 – May 2011
“Small” dataset:
• 3,168 songs
• 134,431 playlists + 1,191,279 transitions
“Large” dataset
• 9,775 songs
• 172,510 playlists + 1,602,079 transitions
Playlist prediction via Metric Embedding

Experiments:
Playlist prediction via Metric Embedding

Experiments:

Small

Big

Avg. log likelihood

d

d

single-point LME
dual-point LME
Uniform
Unigram
Bigram
Morals of the story:

• Metric assumption works well in settings other than “geographical” data!
• However, they require some modifications in order to work well (e.g. “start points” and “end points”)
• Effective combination of latent + observed features, as well as metric + inner-product models
Web Mining and Recommender Systems

Temporal Modeling of Reviewer Expertise
Why do Americans like Pale Ale?

**Lupulin threshold shift:**
People become accustomed to hops over time, and can recognize more subtle flavors.

American Pale Ales:
- Hopsecutioner
- Hoptimus Prime
- Smooth Hoperator
- Red Hoptober
- Hoppy ending
- Hoptopus
- Hopsickle
- Tricerahops

---

**Hop cone**

**Lupulin**
Users and products evolve over time

“Classics” are rated better (Koren, 2010); new products cause users to change focus (Koller & Malouf, 2007)

Users influence each other (Ma et al., 2011); communities shift over time (Xiong et al., 2010)

How can we effectively characterize acquired tastes or expertise?

Age of the product

Age (development) of the user

Age (zeitgeist) of the community
Data

ratebeer
3M reviews, 100K beers, 40K users

Beer Advocate
1.5M reviews, 60K beers, 30K users

CellarTracker!
2M reviews, 500K wines, 45K users
Models of user and community evolution

Replace the ‘standard’ model

\[
rec(u, i) = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i
\]

With one whose parameters change as a function of time (t)

\[
rec_t(u, i) = \alpha(t) + \beta_u(t) + \beta_i(t) + \gamma_u(t) \cdot \gamma_i(t)
\]

How we define t determines what type of evolution we model
Models of user and community evolution

user review timelines  stages of community evolution

time →

user review timelines  stages of user evolution

time →
Models of user and community evolution

**Observation:**
People evolve and develop at different rates. We must **learn** the rate of development for each user.
Models of user and community evolution

**Rows:** models of increasingly “experienced” users

**Columns:** review timeline for one user

Each user’s evolution can be thought of as a **monotonic** path through a graph.
Optimization problem & fitting

**Model:**

\[ rec_{u,i}(u, i) = \alpha(e) + \beta_u(e) + \beta_i(e) + \gamma_u(e) \cdot \gamma_i(e) \]

experience at time of review, offset, bias (user/item), and latent factors

**Optimization problem:**

\[
\arg\min_{\Theta, \varepsilon} \frac{1}{|T|} \sum_{r_{u,i} \in T} (rec_{u,i}(u, i) - r_{u,i})^2 + \Omega(\Theta)
\]

model & experience parameters, smoothness & l2 regularity
Optimization problem & fitting

Repeat steps (1) and (2) until convergence:

**Step 1:**
fit expertise progression

(solved using dynamic programming)

**Step 2:**
fit rating models for each expertise level

\[
\arg \min_\Theta \frac{1}{|T|} \sum_{r_{u,i} \in T} (r_{u,i} - \hat{r}_{u,i})^2 + \Omega(\Theta)
\]

solved via gradient ascent using L-BFGS
(see e.g. Koren & Bell, 2011)
Outcomes – applications

**Rating prediction:**
- Beer: 6% improvement over state-of-the-art
- Wine: 13% improvement
- Movies (Amazon): 23% improvement

**User retention:**
What happens to users who **fail** to acquire taste for a product?

![Graph showing user retention](#)

Users who acquire tastes slowly are more likely to quit the community.
Outcomes – understanding

1. Entire categories of products tend to be preferred by experts or beginners.
2. Experts rate the top products more generously, and the bottom products more harshly.
Outcomes – understanding

**What are experts?**

Predictability and agreement are two necessary conditions (Einhorn, 1974)

Experts are more predictable than beginners. They are also more inclined to agree with each other (right).
Web Mining and Recommender Systems

Translation-based recommendation
Translation-based Recommendation

Ruining He
UC San Diego
rthe@cs.ucsd.edu

Wang-Cheng Kang
UC San Diego
wckang@eng.ucsd.edu

Julian McAuley
UC San Diego
jmcauley@cs.ucsd.edu

ABSTRACT

Modeling the complex interactions between users and items as well as amongst items themselves is at the core of designing successful recommender systems. One classical setting is predicting users’ personalized sequential behaviors (or ‘next-item’ recommendation), where the challenges mainly lie in modeling ‘third-order’ interactions between a user, her previously visited item(s), and the next item to consume. Existing methods typically decompose these higher-order interactions into a combination of pairwise relationships, by way of which user preferences (user-item interactions) and sequential patterns (item-item interactions) are captured by separate components. In this paper, we propose a unified method, Translation-based Recommendation, to model such third-order relationships for large-scale sequential prediction. Methodologically, we embed items into a ‘transition space’ where users are modeled as translation vectors. The transition of a user from one item to another is captured by a user-specific translation operation. Figure 1 demonstrates the historical sequences $S_{u1}$, $S_{u2}$, and $S_{u3}$ of three users. Given the same starting point, the movie Mission: Impossible I, $u_1$ went on to watch the whole series, $u_2$ continued to watch drama movies by Tom Cruise, and $u_3$ switched to similiar action movies.

Figure 1: Translation-based Recommendation as a sequential model: Items (movies) are embedded into a ‘transition space’ where each user is modeled by a translation vector. The transition of a user from one item to another is captured by a user-specific translation operation. Here we demonstrate the historical sequences $S_{u1}$, $S_{u2}$, and $S_{u3}$ of three users. Given the same starting point, the movie Mission: Impossible I, $u_1$ went on to watch the whole series, $u_2$ continued to watch drama movies by Tom Cruise, and $u_3$ switched to similar action movies.

1 INTRODUCTION

Modeling and predicting the interactions between users and items, as well as the relationships amongst the items themselves are the main tasks of recommender systems. For instance, in order to predict sequential user actions like the next product to purchase, movie to watch, or place to visit, it is essential and challenging to model the third-order interactions between a user ($u$), the item(s)
Goal: (e.g) which movie is this user going to watch next?

Translation-based Recommendation

Want models that consider
• characteristics/preferences of each user
• local context, i.e., the last consumed item(s)
Translation-based Recommendation

**Compare:** Factorized Personalized Markov Chains (earlier today)

\[
Prob(j \mid u, i) \propto \langle \tilde{M}_u, \tilde{N}_j \rangle + \langle \tilde{P}_i, \tilde{Q}_j \rangle
\]

- user preference
- local context
Translation-based Recommendation

**Compare:** Personalized Ranking Metric Embedding
(earlier today)

\[
Prob(j \mid u, i) \propto - \left( \alpha \cdot \|\vec{M}_u - \vec{N}_j\|_2^2 + (1 - \alpha) \cdot \|\vec{P}_i - \vec{P}_j\|_2^2 \right)
\]

an additional hyperpara. to balance the two components
Detour: Translation models in Knowledge Bases

Data: entities; links (multiple types of relationships)
State-of-the-art method: 'relationships as translations’
Goal: Predict unseen links

Training example: 

Basic idea: \( \vec{h} + \vec{r} \approx \vec{t} \)

E.g. [Bordes et al., 2013], [Wang et al., 2014], [Lin et al., 2015]
Translating-based Recommendation

Embedding space

\{ \begin{align*}
\text{Items as } & \text{points} \\
\text{Users as } & \text{translation vectors}
\end{align*} \}

Training triplet:

\[
\begin{align*}
\mu & \text{ user} \\
& i \text{ previous item} \\
& j \text{ next item}
\end{align*}
\]

Objective:

\[
\vec{\gamma}_i + \vec{t}_u \approx \vec{\gamma}_j
\]
Translation-based Recommendation

Embedding space

Items as points

Users as translation vectors

Translation operation:
prev. item + user \approx next item
Translation-based Recommendation

\[ \text{Prob}(j|u, i) \propto \beta_j - d(\vec{\gamma}_i + \vec{t}_u, \vec{\gamma}_j) \]

- Benefit from using metric embeddings
- Model \((u, i, j)\) with a single component
- Recommendations can be made by a simple NN search
\[ \hat{\Theta} = \arg \max_{\Theta} \ln \prod_{u \in U} \prod_{j \in S^u} \prod_{j' \notin S^u} \text{Prob}(j > u, i, j' | \Theta) \text{Prob}(\Theta) \]

\[ = \arg \max_{\Theta} \sum_{u \in U} \sum_{j \in S^u} \sum_{j' \notin S^u} \ln \sigma(\hat{p}_{u,i,j} - \hat{p}_{u,i,j'}) - \Omega(\Theta), \]
Translation-based Recommendation

Works well with...

Doesn’t work well with...
Morals of the story:

• Today we looked at two main ideas that extend the recommender systems we saw in class:
  1. **Sequential Recommendation:** Most of the dynamics due to time can be captured purely by knowing the sequence of items
  2. **Metric Recommendation:** In some settings, using inner products may not be the correct assumption
A few other approaches to temporal recommendation
• Markov Chains are a "traditional" model for sequence prediction
• Can more advanced sequence prediction models (e.g. Recurrent Neural Networks) be applied in the context of recommendation?
Recurrent Neural Networks

E.g. Recurrent Neural Networks (LSTM, GRU, etc.):
Recurrent Neural Networks

E.g. Recurrent Neural Networks (LSTM, GRU, etc.):

\[
\begin{array}{c}
\text{HP} \\
y_{t-1} \rightarrow h_{t-1} \\
x_{t-1} \\
\text{RH} \\
y_{t} \rightarrow h_{t} \\
x_{t} \\
\text{PB} \\
y_{t+1} \rightarrow h_{t+1} \\
x_{t+1} \\
\text{...} \\
\text{y} \\
h_{t+2} \\
\end{array}
\]
Recurrent Neural Networks

These types of sequential model can (potentially) capture longer-term dynamics in data, compared to what's available in a Markov-Chain model

• Recommendation problems could have longer-term semantics, e.g. as users revisit items across sessions, or the next item they visit is relevant to a combination of recent items
• Could also be very expensive – often simpler approaches are favorable just because we have too little data to train complex models
Recurrent Neural Networks

Such models can be "user-free" (recall FISM, item2vec, etc.)

- Since the network can potentially store "state" about a user across a large number of steps, we don't need to store a representation of a user
- At inference time, just feed the recent interaction history into the model, and it will predict the next interaction
- (relatively easy to deploy!)
Lots of recent state-of-the-art models come from NLP!
- Models like item2vec (word2vec), LTSMs, GRUs etc. have famously been used for sequence modeling tasks in NLP
- Such models can be adapted to build recommenders, and often represent the current state-of-the-art
- See e.g. models based on Transformers, BERT, etc.
- More in textbook!