This week we’ll 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. **Classification** – dynamic time-warping
3. **Dimensionality reduction** - ?
4. **Recommender systems** – some results from Koren

Next lecture:

1. **Text mining** – “Topics over Time”
2. **Social networks** – densification over time
1. Regression

How can we use features such as product properties and user demographics to make predictions about real-valued outcomes (e.g. star ratings)?

How can we prevent our models from overfitting by favouring simpler models over more complex ones?

How can we assess our decision to optimize a particular error measure, like the MSE?
2. Classification

Next we adapted these ideas to **binary** or **multiclass** outputs.

- **What animal is in this image?**
- **Will I purchase this product?**
- **Will I click on this ad?**

- Combining features using naïve Bayes models
- **Logistic regression**
- **Support vector machines**
3. Dimensionality reduction

Principal component analysis

Community detection
4. Recommender Systems

Rating distributions and the missing-not-at-random assumption

Latent-factor models
Regression for sequence data
Given labeled training data of the form

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

Infer the function

\[ f(data) \rightarrow labels \]
Time-series regression

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

\[(x_1, \ldots, x_n) \in \mathbb{R}^n\]

\[f(x_1, \ldots, x_n) \rightarrow x_{n+1}\]
Time-series regression

Method 1: maintain a “moving average” using a window of some fixed length

\[ f(x_1, \ldots, x_m) = \frac{\sum_{k=0}^{K} x_{n-k}}{K} \]
Time-series regression

**Method 1:** maintain a “moving average” using a window of some fixed length

- This can be computed efficiently via dynamic programming:

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

\[ O(m + k) \]
Time-series regression

Also useful to plot data:

![Scatterplot](BeerAdvocate, ratings over time)

![Sliding window](Sliding window (K=10000))

Code on:

[http://jmcauley.ucsd.edu/cse255/code/week10.py](http://jmcauley.ucsd.edu/cse255/code/week10.py)
Method 2: weight the points in the moving average by age

\[ f(x_1, \ldots, x_m) = \frac{1}{\binom{k}{2} \sum_{k=0}^{K} (K-k)x_{n-k}} \]

\[ = \frac{1}{\binom{k}{2} \sum_{k=0}^{K} (K-k)x_{n-k}} \]
Method 3: weight the most recent points exponentially higher

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

Method 1: Sliding window
Method 2: Linear decay
Method 3: Exponential decay

\[ f(x_1, \ldots, x_n) = O_0 x^n + O_1 x^{n-1} + \ldots \]

1: \[ O_k \]
\[ \begin{array}{c}
  \text{k} \\
  \text{K} \\
\end{array} \]

2: \[ O_k \]
\[ \begin{array}{c}
  \text{k} \\
  \text{k} \\
\end{array} \]

3: \[ O_k \]
\[ \begin{array}{c}
  \text{k} \\
\end{array} \]
Method 4: all of these models are assigning \textit{weights} to previous values using some predefined scheme, why not just \textit{learn} the weights?

\[ f(x_1, \ldots, x_m) = \langle \Theta, (x_n, \ldots, x_{m-K+1}) \rangle \]

\[ \Theta = \text{argmin}_\Theta \sum_{n=1}^{M} \left\| x_n - f(x_1, \ldots, x_{n-1}) \right\|_2^2 \]
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
Classification of sequence data
How can we predict **binary** or **categorical** variables?

Another simple algorithm: nearest neighbo(u)rs
As you recall...
The longest-common subsequence algorithm is a standard dynamic programming problem.
As you recall...
The longest-common subsequence algorithm is a standard dynamic programming problem

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>A</th>
<th>G</th>
<th>C</th>
<th>A</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

- = optimal move is to delete from 1\textsuperscript{st} sequence

\uparrow = optimal move is to delete from 2\textsuperscript{nd} sequence

\downarrow = either deletion is equally optimal

\leftrightarrow = optimal move is a match
Time-series classification

The same type of algorithm is used to find correspondences between time-series data (e.g. speech signals), whose length may vary in time/speed.

\[
\text{DTW\_cost} = \infty \\
\text{for } i \text{ in range}(1,N): \\
\quad \text{for } j \text{ in range}(1,M): \\
\quad \quad d = \text{dist}(s[i], t[j]) \quad \# \text{Distance between sequences } s \text{ and } t \text{ and points } i \text{ and } j \\
\quad \quad \text{DTW}[i,j] = d + \min(\text{DTW}[i-1, j], \text{DTW}[i, j-1], \text{DTW}[i-1, j-1]) \\
\text{return } \text{DTW}[N,M]
\]

Output is a distance between the two sequences.

- skip from seq. 1
- skip from seq. 2
- match
Time-series classification

- This is a simple procedure to infer the similarity between sequences, so we could classify them (for example) using nearest-neighbours (i.e., by comparing a sequence to others with known labels)
- We’ll come back to classification soon when we look at time series using graphical models
Temporal recommender systems
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.
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:

$$\arg\min_{\alpha, \beta, \gamma} \sum_{u,i} (\alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i - R_{u,i})^2 + \lambda \left[ \sum_u \beta_u^2 + \sum_i \beta_i^2 + \sum_i \|\gamma_i\|_2^2 + \sum_u \|\gamma_u\|_2^2 \right]$$

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

To build a reliable system (and to win the Netflix prize!) we need to account for **temporal dynamics**:

So how was this actually done?

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

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:

$$dev_u(t) = \text{sign}(t - t_u) \cdot |t - t_u|^x$$

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

$\text{dev}_u(t)$

$x = 1$

$x = 2$

$0.5$
Start with a simple model of drifting dynamics for users:

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

- **mean rating date for user** $u$
- **hyperparameter** (ended up as $x=0.4$ for Koren)

Before (-1) or after (1) the mean date, days away from mean date.

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**

Graph showing $\text{sign}(x)|x|^{0.4}$
Temporal latent-factor models

Real data

Fitted model

Netflix ratings over time
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
Temporal latent-factor models

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

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

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

- 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.
Temporal latent-factor models

Summary

- Changing the setting, e.g. to model the stages of progression through the symptoms of a disease, or even to model the temporal progression of people’s opinions on beers, means that alternate temporal models are required.

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

**Columns:** review timeline for one user
Further reading:
“Collaborative filtering with temporal dynamics”
Yehuda Koren, 2009