CSE 190 – Lecture 17
Data Mining and Predictive Analytics

Temporal data mining
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 \textit{temporal} information

1. \textbf{Regression} – sliding windows and autoregression
2. \textbf{Classification} – dynamic time-warping
3. \textbf{Dimensionality reduction} - ?
4. \textbf{Graphical models} – Hidden Markov Models
5. \textbf{Recommender systems} – some results from Koren

Next lecture:

1. \textbf{Text mining} – “Topics over Time”
2. \textbf{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. Graphical models

Directed and undirected models

Inference via graph cuts
5. Recommender Systems!!!

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

Latent-factor models
CSE 190 – Lecture 17
Data Mining and Predictive Analytics

Regression for sequence data
Given **labeled training data** of the form

\[\{(\text{data}_1, \text{label}_1), \ldots, (\text{data}_n, \text{label}_n)\}\]

Infer the function

\[f(\text{data}) \rightarrow \text{labels}\]
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

\[
\frac{1}{u-v+1} \sum_{t=u}^{v} \left( f_t(x_1, \ldots, x_{u-1}) - x_t \right)^2
\]
Time-series regression

**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**

**BeerAdvocate, ratings over time**

- Scatterplot
- Sliding window (K=10000)
- Seasonal effects
- Long-term trends

Code on:

http://jmcauley.ucsd.edu/cse190/code/week10.py
Time-series regression

Method 2: weight the points in the moving average by age

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

newest points have the highest weight

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

\[ f(x_1) = x_1 \]

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

most recent point has weight \( \alpha \)

previous prediction has weight \( 1 - \alpha \)
Method 4: all of these models are assigning weights to previous values using some predefined scheme, why not just learn the weights?

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

\[
= \alpha + \langle \theta, (x_{m-(K-1)}, \ldots, x_m) \rangle
\]

• 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?

\[ f(\text{data}) \rightarrow \text{labels} \]

\{0,1\}, \{True, False\}

\{1, \ldots, N\}

Another simple algorithm: nearest neighbor(u)rs
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 1st sequence

↑ = optimal move is to delete from 2nd sequence

↑ = either deletion is equally optimal

↔ = optimal move is a match
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.

```
DTW_cost = infty
for i in range(1,N):
    for j in range(1,M):
        d = dist(s[i], t[j])  # Distance between sequences s and t and points i and j
        DTW[i,j] = d + min(DTW[i-1, j], DTW[i, j-1], DTW[i-1, j-1])
return DTW[N,M]
```

Output is a **distance** between the two sequences.
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
CSE 190 – Lecture 17
Data Mining and Predictive Analytics

Temporal dynamics with graphical models
Graphical models & Interdependent variables

How can we solve predictive tasks when

• There are multiple unknowns to infer simultaneously
• There are **dependencies** between the unknowns
• In other words, what can we do when...

\[ p(label_1, label_2 | data) \neq p(label_1 | data)p(label_2 | data) \]
We’ve already seen (at least in theory) how **graphical models** could be used to handle sequence data, e.g. the weather between subsequent days.
In Week 4 we showed how to fit very simple models of this type, e.g.:

\[
p(b|a) = \begin{pmatrix} p(\text{dry}|\text{dry}) & p(\text{dry}|\text{rain}) \\ p(\text{rain}|\text{dry}) & p(\text{rain}|\text{rain}) \end{pmatrix}
\]

\[
= \begin{pmatrix} 0.8 & 0.4 \\ 0.2 & 0.6 \end{pmatrix}
\]

Q: But how could we adapt these models to handle longer-term effects, periodic behavior, etc.
A: Let’s assume that there’s some type of (possibly unknown) **hidden state** underlying the whole process.
Each of these states is also associated with an **emission probability**, which is what generates our observations:

- stormy
- cloudy
- snowy
- clear

- If we know the sequence of hidden states, then we can easily compute the probability of a particular sequence of observations
- But of course this really becomes interesting when the hidden states and emission probabilities are **unknown**!
This is quite a flexible model that is capable of capturing long-term effects, periodic effects, etc.

gradual patterns

e.g. stages of progression in beer expertise (week 5)!
A few problems of interest:

1. Compute the probability of a sequence of outputs given an **observed** sequence of hidden states.
2. Compute the sequence of **hidden states** that was most likely to have generated the observed output.
3. Infer the **transition probabilities** between some set of hidden states in order to model time-evolving event sequences.
A few problems of interest:
2. Compute the sequence of **hidden states** that was most likely to have generated the observed output.

This is not particularly easy, but this was the problem we discussed in **week 4** – inferring sequences of unknowns **given** the transition probabilities between them.

**remember!**: infer the marginal probability of $g$

$$= \sum_{a,b,c,d,e,f} p(a)p(b|a)p(c|b)p(d|c)p(e|d)p(f|e)p(g|f)$$
Time-series with graphical models

A few problems of interest:
3. Infer the **transition probabilities** between some set of hidden states in order to model time-evolving event sequences

Use the following **iterative** procedure:
1. Start with some random initial hidden states/transition probabilities
2. Infer values for the hidden states given the current estimates of the transition probabilities (previous slides/week 4)
3. Infer the transition probabilities given the current estimates for the hidden states
   - This is known as the Baum-Welch algorithm (see refs. for details)
Ultimately what we have is the following:

- For **known** transition probabilities, a procedure to estimate the hidden states most likely to have generated a sequence.
- This allows us to associate a **probability** with a particular model having generated a sequence of observed events.
  - We can use this idea for **classification**:
  1. Fit transition probabilities for sequences with known classes, e.g. for heart-rate data of patients with known normal/abnormal beat patterns.
  2. Given a new time series, compute the most likely sequence of hidden states under each model.
  3. Return the class label corresponding to the sequence of hidden states with higher probability.
Further reading:
Baum-Welch algorithm (inferring hidden states and transition probabilities):
(countless refs, but this one is good):
“A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models” (Bilmes, 1998)
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.

- Preference toward “action”
- Preference toward “special effects”
- HP’s (item) “properties”
- Are the special effects good?
- Is the movie action-heavy?

**Compatibility**
Week 4/5

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:

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

(error) (regularizer)

(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**: 

![Netflix ratings over time](image1)

(Netflix changed their interface)

![Netflix ratings by movie age](image2)

(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)
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

*note: this is the opposite of what we saw in Week 5 when modeling the temporal dynamics of opinions on beers!
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:

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

- **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**
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:

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

- 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

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

- This is now a reasonably flexible model, but still only captures \textit{gradual drift}, i.e., it can’t handle sudden changes (e.g. a user simply having a bad day)

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

• 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:

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

**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
Temporal latent-factor models

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.
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