Learning Time Series
Today’s lecture

• Doing machine learning on time series

• Dynamic Time Warping

• Hidden Markov Models
Our view of data so far

• Data are points in a high-dimensional space
What time series are

• A sequence of samples, can be thought of as a point in a very very high-D space
  • Often a bad idea
  • Remember the curse!!
Shift variance

- Time series have shift variance
- Are these two points close?
Time warp variance

- Slight changes in timing are not relevant
- Are these two points close?
Noise/filtering variance

- Small changes can look serious
- How about these two points?
A real-world case

- Spoken digits
What now?

- Our models so far were too simple
- How do we incorporate time?
- How to get around all these problems?
A small case study

- How to recognize words
  - e.g. yes/no, or spoken digits

- Build reliable features
  - Invariant to minor differences in inputs

- Build a classifier that can deal with a time index
  - Invariant to temporal differences in inputs
Example data
Going from fine to coarse

• Small differences are not important
  • Find features that obscure them
Frequency domain

- Look at the magnitude Fourier transform
Time/Frequency features

- A more robust representation
- Bypassing small waveform differences
A new problem

• What about time warping?
Time warping

- There is a “warped” time map
- How do we find it?
Matching warped series

- Represent the warping with a path on a graph

\[ r(i), i = 1, 2, \ldots, 6 \quad t(j), j = 1, 2, \ldots, 5 \]
Finding the overall “distance”

- Each node will have a cost
  - e.g., $d(i, j) = \|r(i) - t(j)\|$
- Overall path cost is:
  $$D = \sum_{k} d(i_k, j_k)$$
- Optimal path $(i_k, j_k)$ defines the “distance” between two given sequences
- But how do we find the optimal path?
  - Big search space ...
Bellman’s optimality principle

- For an optimal path passing through

\[(i, j) : (i_0, j_0)^{opt} \rightarrow (i_f, j_f)\]

- Then:

\[(i_0, j_0)^{opt} \rightarrow (i_f, j_f) = \begin{cases} (i_0, j_0)^{opt} \rightarrow (i, j), (i, j)^{opt} \rightarrow (i_f, j_f) \end{cases}\]
Finding an optimal path

- Optimal path to \((i_k, j_k)\):

\[
D_{\text{min}}(i_k, j_k) = \min_{i_{k-1}, j_{k-1}} D_{\text{min}}(i_{k-1}, j_{k-1}) + d(i_k, j_k | i_{k-1}, j_{k-1})
\]

- Smaller search!

- Local/global constraints
  - Limited transitions
  - Nodes we never visit
And adding some constraints

- Global constraints
  - Can only visit these
    - Bold dots

- Local constraints
  - Can only transition using them
    - Black lines

- Optimal path
  - Blue line
Making this work for speech recognition

- Define a distance function
- Define local constraints
- Define global constraints
Distance function

• Given our robust feature we can use a simple measure like Euclidean distance: \( d(i, j) = \| f_1(i) - f_2(j) \| \)
Global constraints

- Define time ratios that make sense
  - e.g. no sequence can be half as slow as the other one
Local constraints

- Monotonicity:
  \[ i_{k-1} \leq i_k \quad j_{k-1} \leq j_k \]
  - Repeat, but don’t go backwards in time

- This enforces time order
  - Don’t match “ban” with “banana”
More local constraints

- Define acceptable local subpaths
  - Consider only these patterns
  - Application dependent
Toy data run

Local Constraint

Cost matrix

Distance matrix

Input 1

Input 2
Speech example with identical utterance
Ditto with similar utterance
Ditto with different utterance

Distance matrix

Cost matrix, $d = 3.0415$
A basic speech recognizer

• Collect template spoken words $T_i(t)$
• Get their DTW distances from input $x(t)$
  • Smallest distance wins
Example case

**DTW-derived distances**

Comparison with template 1

Comparison with template 2

Comparison with template 3

Comparison with template 4

Comparison with template 5

Input class
A real-world application of DTW

- Simplifying ADR
  - Costs time and money
- Automatically align audio tracks
Defining a distance for time series

- Dynamic Time Warping computes time series distances
  - Dependent on the constraints, etc

- Having a time series distance gives us options
  - Nearest-neighbor classifiers (just did that)
  - Clustering of time series
  - Manifold embeddings of time series
  - ...

What’s not quite right so far?

• Something I keep constantly complaining about ...
We prefer soft models

- The paths are hard decisions
- We’d rather have soft assignments
- Also obtaining probabilities would help
Starting with the GMM

- GMMs are “time agnostic”
- We don’t have a notion of a sequence
What if there is time order?

- Both plots are the same data
- But we see time structure on the left
Representing time

- **Hidden Markov Model**
  - Gaussian variant for now

- **Each state is represented by a Gaussian distribution**

- **Transition probabilities between all states**
  - Only the last state matters

*Initial Probabilities: $P(A), P(B), P(C)$*
Problems to solve

- Evaluation
  - Given a model, evaluate likelihoods

- Decoding
  - Given a model, find state sequences

- Learning
  - Given the data, find the model parameters
• Propagate likelihoods using the *forward algorithm*:

\[
\alpha_i(1) = P_1(i)P(x_1 \mid i)
\]

\[
\alpha_i(t + 1) = P(x_{t+1} \mid i) \sum_j \alpha_j(t) P(i \mid j)
\]
Evaluation

• Forward pass provides probability of a specific state and time, given past inputs

\[ \alpha_i(t) = P(x_t \mid i) \sum_j \alpha_j(t-1)P(i \mid j) \]

• Terminal time point values provide the overall likelihood of model given an input

\[ P(x) = \sum_i \alpha_i(T) \]
Decoding

- The forward pass provides us with the state likelihoods
  - “soft” decisions

- With a backwards pass we can find most likely transitions
  - Just as we did with DTW

- The Viterbi algorithm
The Viterbi algorithm

• Similar to forward algorithm, but with a hard decision:

\[ v_i(1) = P_1(i)P(x_1 | i) \]

\[ v_i(t + 1) = P(x_{t+1} | i) \max \left( P(j | i)v_j(t - 1) \right) \]

• Probability of most likely path up to time \( t + 1 \) that ends in state \( i \)

• For every step remember most likely transition (as with DTW)

• Backwards pass

  • Get most likely terminal state
  • Work backwards using the most likely transitions

\[
\begin{align*}
\text{Time} \\
\text{State} \\
t=1 \quad t=2 \quad t=3
\end{align*}
\]
Learning

• An EM-type approach
  • Known as Baum-Welch training

• E-step
  • Find likelihood of each data point being associated with each state

• M-step
  • Estimate each state’s parameters based on the associations from the E-step
E-step: Forward-backward pass

- Forward pass provides probability of a specific state and time given past inputs $\alpha_i(t)$

- A backward pass will provide probability of given state and time given future inputs $\beta_i(t)$
  - Same as forward pass, but computed backwards in time

- The product of these will be the probability of a state and time given all inputs $\gamma_i(t) = \alpha_i(t) \beta_i(t)$
M-step: Parameter estimation

- State models
  - Use $\gamma_i(t)$ as weights to compute state parameters, e.g.:
    
    $$
    \mu_i = \frac{\sum_t x_t(t) \gamma_i(t)}{\sum_t \gamma_i(t)}
    $$
    
    $$
    \Sigma_i = \frac{\sum_t \gamma_i(t) (x_t - \mu_i)(x_t - \mu_i)^T}{\sum_t \gamma_i(t)}
    $$

- Transition matrix and initial probabilities
  - Count potential transitions between states
Noteworthy things

• HMM learning works with multiple training sequences

• Use log probabilities
  • What’s a state likelihood after a million time points? Can you represent it?

\[ \alpha_i(t) = P(x_t | i) \sum_j \alpha_j(t-1)P(i | j) \]

• Use an arbitrary state model
  • e.g. a neural net, or a GMM
Learning an HMM model

Log Likelihood: -83.83

Initial/Transition probabilities

Input

State sequence
Back to speech

• Learning an HMM for a word
  • Each state should correspond to a coherent part
    • e.g. a syllable, a phoneme, etc
Fully-connected model

Log Likelihood: $1.35125.74 	imes 10^4$

Epoch

Input

Initial/Transition probabilities

State sequence

Time step
Left-to-right model

Log Likelihood: 144358.50

Initial/Transition probabilities

State sequence
Digit recognition results

Model log likelihoods

Input class

Distance

Likelihood 1

Distance

Likelihood 2

Distance

Likelihood 3

Distance

Likelihood 4

Distance

Likelihood 5
Speech recognition in a nutshell

- Small scale speech recognition

- Large scale systems are another beast ...

\[
\text{the} \rightarrow \text{dog} \rightarrow \text{was} \rightarrow \text{here}
\]
Elaborations

- Denser interconnections
  - e.g., second order time ties
- Markov random fields
Recap

- Learning with time series
- Dynamic Time Warping
- Hidden Markov Models
- Some basic speech recognition
Next lecture

- Missing data
  - Using temporal dynamics to fill in missing values

- Dynamical systems
  - Learning continuous time-series
Reading

- Textbook chapters 8, 9
- A tutorial on HMMs
Final projects!

• Time to start thinking about projects
  • Send us (me/Minje) an informal proposal by end of next week
    • 1-2 paragraphs

• You have to be part of a team
  • $2 \leq N \leq 3$, no solo projects, $N = 4$ if you have a good excuse

• Deliverables
  • A poster presentation/demo and a 3-4 page writeup
Some project advice

• Work on a domain you have expertise (if any)
  • Bring your own (signal) data, don’t stick to generic projects

• Show me that you learned something in class
  • Demonstrate proper use of the tools we cover, use some signals-fu

• Propose multiple objectives, see how far you can go
  • One objective will be really easy to do (you pass the class)
  • Another will be tough, but doable (you get a potential paper out of it)
  • Final should be an amazing result (here’s your dissertation!)