M2 Data & Knowledge: Data Stream Mining

Jesse Read

Time Series and Sequential Data

Outline

Reminder: Data Streams

In a data stream, we assume that data arrives i.d.[1]

$$(\mathbf{x}_t, y_t) \sim p_t(\mathcal{X}, \mathcal{Y})$$

over time $t = 1, \ldots, \infty$ (p_t is the generating process, i.e., the concept). A model is given test instance $\mathbf{x}_t$ and is required to make a prediction at time t:

$$\hat{y}_t = h_t(\mathbf{x}_t)$$

- The computational time (training + testing) spent per instance must be less that the rate of arrival of new instances (i.e., the real clock time between time steps $t-1$ and t).
- A usual assumption: true label y_{t-1} available at time t (can use to update the model)

[1] Independently but not identically distributed

Concept Drift

$$\mathbf{x}_t, y_t \sim p_t(\mathcal{X}, \mathcal{Y})$$

where $p_t \neq p_{t-1}$.

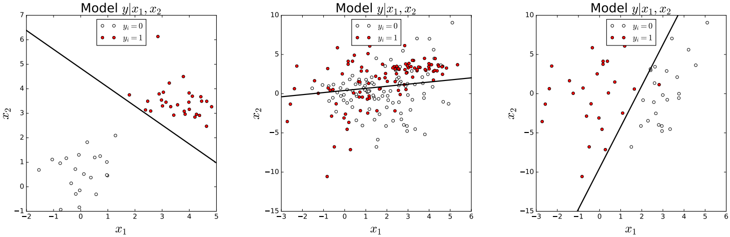


Figure: Data and decision boundary [concept] with a window of examples; before (left), during (centre), and after (right) concept drift.

Concept Drift

$$\mathbf{x}_t, y_t \sim p_t(\mathcal{X}, \mathcal{Y})$$

where $p_t \neq p_{t-1}$.

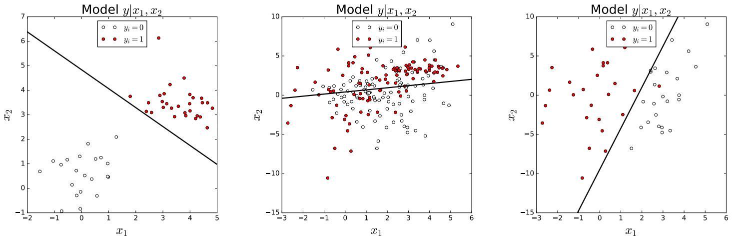


Figure: Data and decision boundary [concept] with a window of examples; before (left), during (centre), and after (right) concept drift.

- Model becomes invalid when the concept drifts
- Multi-label concept drift involves also the label variables.

Dealing with Concept Drift

1. Just ignore it (assume that models will adapt),
   - kNN
   - SGD
   - Weighted batch-ensembles/fading factor

   If drift is not too major – it won't affect accuracy too much.
2. Monitor the stream and detect it, with a change detector, e.g., monitor
   - a predictive performance statistic
   - the distribution

   then reset/recalibrate models. **If detected correctly, we get streams with i.i.d. distributed data!**

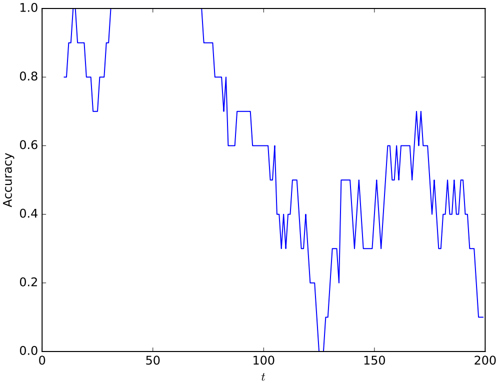


Figure: Accuracy through concept drift ($t = 50, \ldots, 150$).

In multi-label classification, we have a multi-dimensional error:

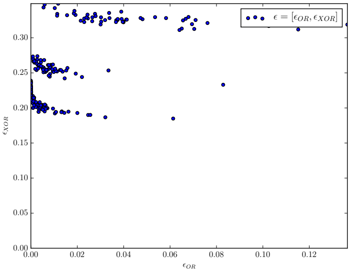


Figure: Error distribution for two labels.

If the shape of this distribution changes, we may need to:

- Assume that concept drift has occurred
- Assume that conditional label dependence has changed
- Consider a new structure for the multi-label model

Outline

Reminder: Data Streams

In a data stream, we assume that data arrives i.d.[2]

$$(\mathbf{x}_t, y_t) \sim p_t(\mathcal{X}, \mathcal{Y})$$

[2] Independently but not identically distributed

Reminder: Data Streams

In a data stream, we assume that data arrives i.d.[2]

$$(\mathbf{x}_t, y_t) \sim p_t(\mathcal{X}, \mathcal{Y})$$

Is it really i.d., though?

[2] Independently but not identically distributed

Reminder: Data Streams

In a data stream, we assume that data arrives i.d.[2]

$$(\mathbf{x}_t, y_t) \sim p_t(\mathcal{X}, \mathcal{Y})$$

Is it really i.d., though?

Many data streams, by their very temporal nature, exhibit temporal dependence!

[2] Independently but not identically distributed

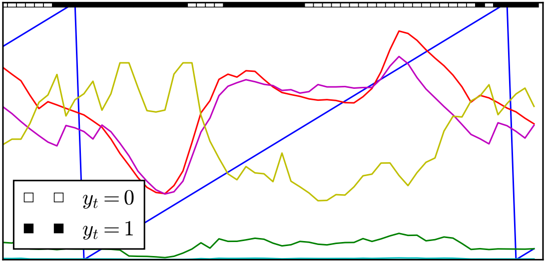
Reminder: Data Streams

In a data stream, we assume that data arrives i.d.[2]

$$(\mathbf{x}_t, y_t) \sim p_t(\mathcal{X}, \mathcal{Y})$$

Is it really i.d., though?

Many data streams, by their very temporal nature, exhibit temporal dependence!



Electricity data: Instances $\mathbf{x}_t \in \mathbb{R}^5$ and class labels $y_t \in \{0, 1\}$.

[2] Independently but not identically distributed

Temporal dependence

The auto-correlation function (basically Pearson's correlation coefficient of a variable with itself, lagged $+1$),

$$\rho_{Y_t, Y_{t+1}} = \frac{\mathrm{Cov}(Y_t, Y_{t+1})}{\mathrm{Std}(Y_t)\mathrm{Std}(Y_{t+1})} \tag{1}$$

$$= \frac{\sum_{t=1}^{T-1}[(y_t - \bar{y})(y_{t+1} - \bar{y})]}{\sqrt{\sum_{t=1}^{T-1}(y_t - \bar{y})^2 \sum_{t=2}^{T}(y_t - \bar{y})^2}} \tag{2}$$

$$\approx \frac{\sum_{t=1}^{T-1}[(y_t - \bar{y})(y_{t+1} - \bar{y})]}{\sum_{t=2}^{T}(y_t - \bar{y})^2} \tag{3}$$

(For large T, the difference in the mean of $Y_1, \ldots, Y_{T-1}$ and of $Y_2, \ldots, Y_T$ can be ignored, hence Eq. (3).)

Temporal dependence

The auto-correlation function (basically Pearson's correlation coefficient of a variable with itself, lagged $+1$),

$$\rho_{Y_t,Y_{t+1}} = \frac{\mathrm{Cov}(Y_t, Y_{t+1})}{\mathrm{Std}(Y_t)\mathrm{Std}(Y_{t+1})} \tag{1}$$

$$= \frac{\sum_{t=1}^{T-1}[(y_t - \bar{y})(y_{t+1} - \bar{y})]}{\sqrt{\sum_{t=1}^{T-1}(y_t - \bar{y})^2 \sum_{t=2}^{T}(y_t - \bar{y})^2}} \tag{2}$$

$$\approx \frac{\sum_{t=1}^{T-1}[(y_t - \bar{y})(y_{t+1} - \bar{y})]}{\sum_{t=2}^{T}(y_t - \bar{y})^2} \tag{3}$$

(For large T, the difference in the mean of $Y_1, \ldots, Y_{T-1}$ and of $Y_2, \ldots, Y_T$ can be ignored, hence Eq. (3).)

We can generalise to $\rho(k)$ to consider the correlation from y_t and y_{t+k} for any lag $\pm k$.

Temporal dependence

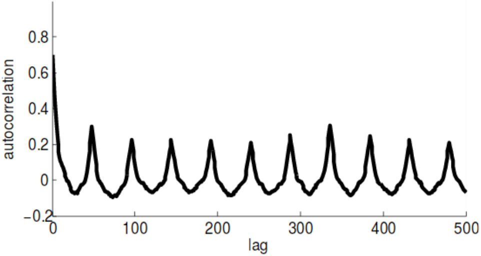


Figure: Auto-correlation function on the Electricity dataset, for $k = 1, 2, \ldots, 500$; source: Indrė Žliobaitė arXiv:1301.3524v1, Jan 2015.

Dealing with Temporal Dependence

1. Remove it
2. Model it

Dealing with Temporal Dependence

1. Remove it
2. Model it

To remove temporal dependence, build a stream of instances

$$\mathbf{x}'_t = [\mathbf{x}_{t-k}, \mathbf{x}_{t-k-1}, \ldots, \mathbf{x}_t]$$

selecting k such that

$$p(y_t|\mathbf{x}'_t) = p(y_t|\mathbf{x}'_t, \mathbf{x}'_{t-1})$$

(a sliding window large enough to create independence!)

Dealing with Temporal Dependence

1. Remove it
2. Model it

To remove temporal dependence, build a stream of instances

$$\mathbf{x}'_t = [\mathbf{x}_{t-k}, \mathbf{x}_{t-k-1}, \ldots, \mathbf{x}_t]$$

selecting k such that

$$p(y_t|\mathbf{x}'_t) = p(y_t|\mathbf{x}'_t, \mathbf{x}'_{t-1})$$

(a sliding window large enough to create independence!)

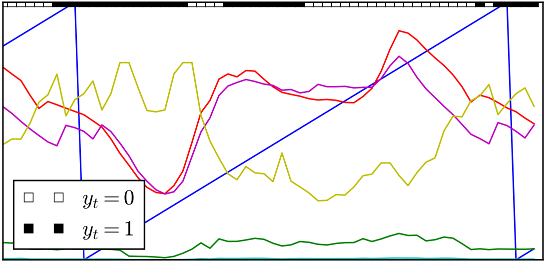
- Time complexity increases by factor of k.
- Does not take into account temporal structure.
- Resulting model may be difficult to interpret.

Outline

Time Series

$$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, \ldots$$

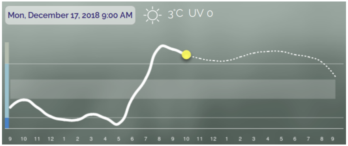
Measurements may be continuous, $\mathbf{x}_t \in \mathbb{R}^D$ or discrete, $\mathbf{x}_t \in \mathbb{N}_+^D$; across time t. May be associated with secondary (possibly unobserved) signal $\mathbf{y}_t$ (e.g., labels).



Time series $\mathbf{x}_t \in \mathbb{R}^5$ associated with state $y_t \in \{0, 1\}$.

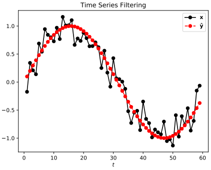
Examples of Time Series Data

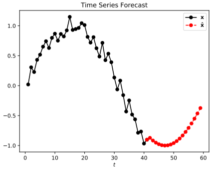
- Electricity demand for a city
- Sensor measurements on equipment in an aircraft
- Number of calls to an insurance service
- Light-sensor measurements (and movement through a room)
- Smartphone GPS and signal strength measurements of urban travellers (and their predicted trajectory)
- EEG and ECG signals obtained during sleep
- Cellular growth in trees
- Environmental measurements (temperature, humidity)

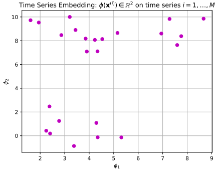


Time Series Tasks

- **Filtering**: (*estimate*) target/labels from observations
- **Forecasting**: (*predict*) the future of a series
- **Embedding**: Describe a time series as a *fixed-length* vector
- Clustering
- Classification
- Motif discovery/extraction
- Novelty/anomaly detection
- Query by content
- Segmentation (e.g., change-point/concept-drift detection)







Outline

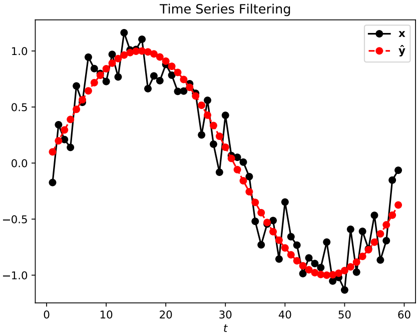
Filtering

Given observations (time series)

$$\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, \ldots\}$$

we want a model f to predict corresponding

$$\{\widehat{\mathbf{y}}_1, \widehat{\mathbf{y}}_2, \ldots, \widehat{\mathbf{y}}_t, \ldots\}$$



where $\widehat{\mathbf{y}}_t$ can be a categorical label, or continuous value.

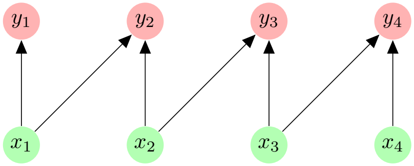
Outline

Traditional Methods

- Moving average, exponential smoothing (i.e., low-pass filters), e.g.,

$$\hat{y}_t = f(w_1 x_t + \cdots + w_k x_{t-k-1})$$

  with some weights $\mathbf{w} = [w_1, \ldots, w_k]$ (window size k). This is a convolution with kernel/filter $\mathbf{w}$.
  Can be seen as a particular instance of a finite impulse response (FIR) filter (time-delay neural network):



Basic models are

- Robust and well-understood
- Need to be hand-crafted, calibration by domain expert
- else not suitable for multiple dimensions; complex problems

Machine Learning for Filtering

Given training data, we can design a machine learning approach (e.g., artificial neural networks, decision trees, ...), on, e.g.,

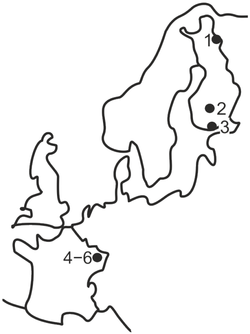
X_{t-4}	X_{t-3}	X_{t-2}	X_{t-1}	X_t	Y_t
1	A	2.3	1.8	-3	-24
A	2.3	1.8	-3	4	-28
2.3	1.8	-3	4	B	-32
1.8	-3	4	B	3	-43
...	...	...	...	...	...
T	39	3	4	0.1	?

i.e., model

$$y_t = f(x_{t-4}, \ldots, x_t; \theta) + \epsilon$$

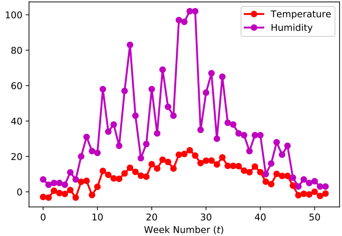
The decision making and interpretation is relegated to the learner.

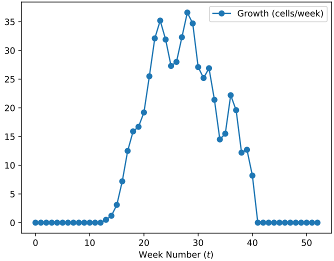
Example: Predicting Celular Growth in Scots Pine



- 6 sites in Finland and France, of Scots pine
- Interested in modelling cellular growth under different latitude, altitude, . . .
- Models must be carefully crafted, parametrized, and adjusted by domain experts, *per site*.

Example: Predicting Celular Growth in Scots Pine

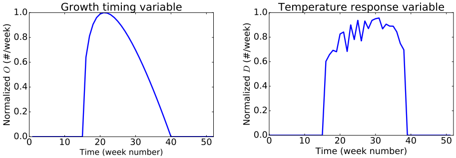




- Environmental measurements (temperature, humidity, ...).
- Some cell-growth data (from micro-core samples and counts during growth season) over 3–4 years

i.e., input $\mathbf{x}_t = [\text{temp}, \text{humidity}, \text{week-number}, \ldots]$ at week t, output y_t.

- Domain experts were using hand-crafted functions, e.g., growth timing variable (left) and heat sum (right),

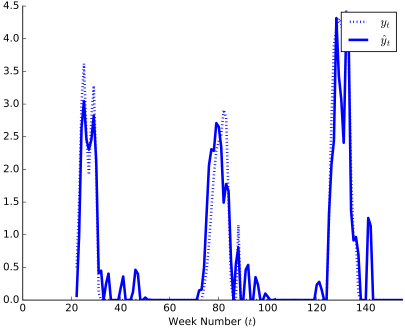


e.g., where τ_t = temperature and week t,

$$z_t = 1_{\tau_t > c}\left[\frac{1}{1 + \exp(-\beta \tau_t)}\right]$$

and c, β are per-site parameters.

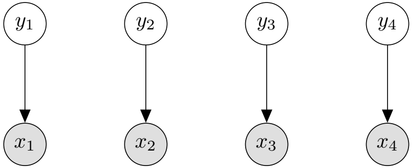
- Assembled into a differential equation
- About 4-5 parameters to be hand-selected *per site*



- Data-driven approach; parametrized expert functions = features
- Learning via regularized SGD and decision tree learners (for interpretation)
- Black-box experiments to calibrate history (length of ‘sliding window’ – but note that some features may embed history)

Outline

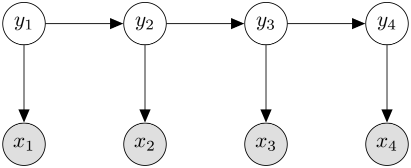
Bayesian Filtering



$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})}$$

Bayesian inference is often naturally online: The posterior can be treated as a prior for the next batch, and updated recursively.

Bayesian Filtering



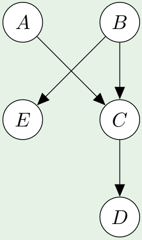
$$p(y_t|\mathbf{x}_t) = \frac{p(\mathbf{x}_t|y_t)p(y_t|y_{t-1})}{p(\mathbf{x}_t)}$$

Bayesian inference is often naturally online: The posterior can be treated as a prior for the next batch, and updated recursively.

Bayesian Networks

aka Belief Networks[3]:

- Probabilistic graphical models
- They are [probabilistic] directed acyclic graphs, i.e., DAGs
- Structure represents conditional dependence between variables (*not necessarily* causal; $P(A|B) \Leftrightarrow$ '*B influences A*')
- Specifies factorization of the joint probability distribution.



$$P(A, B, C, D, E) = P(A)P(B)P(C|A, B)P(D|C)P(E|B)$$

[3] Otherwise, not necessarily a commitment to Bayesian statistics – can use frequentist methods to estimate the conditional probability distributions

Bayesian Networks

aka Belief Networks[3]:

- Probabilistic graphical models
- They are [probabilistic] directed acyclic graphs, i.e., DAGs
- Structure represents conditional dependence between variables (*not necessarily* causal; $P(A|B) \Leftrightarrow$ '*B influences A*')
- Specifies factorization of the joint probability distribution.

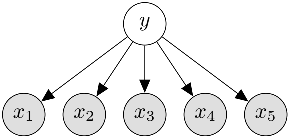
Factorization

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|\mathrm{pa}(X_i))$$

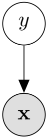
where $\mathrm{pa}(X_i) :=$ parents of X_i.

[3]Otherwise, not necessarily a commitment to Bayesian statistics – can use frequentist methods to estimate the conditional probability distributions

Recall: Naive Bayes



or, abbreviated:



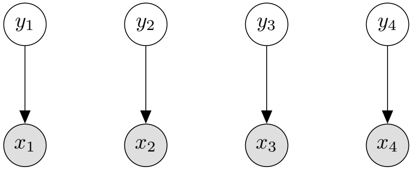
$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}, y)}{P(\mathbf{x})} = \frac{P(y)P(\mathbf{x}|y)}{P(\mathbf{x})}$$

Maximum a-posteriori probability (MAP) task:

$$\hat{y} = h(\mathbf{x}) = \operatorname*{argmax}_{y \in \{0,1\}} P(y)P(\mathbf{x}|y) \quad \bullet \; P(y|\mathbf{x}) \propto P(\mathbf{x}, y)$$

$$= \operatorname*{argmax}_{y \in \{0,1\}} P(y) \prod_{d=1}^{D} P(x_d|y)$$

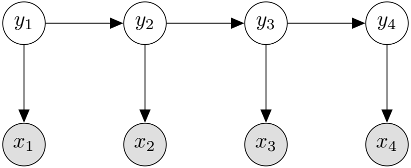
Naive Bayes in a Stream (Temporal Independence)



$$P(y_t|x_t) = \frac{P(x_t|y_t)P(y_t)}{P(x_t)}$$

This could be seen as 'filtering' $\{x_t\}$.

Hidden Markov Models: Filtering



In the filtering task, at time t we want a prediction

$$\hat{y}_t = \operatorname*{argmax}_{y_t} P(y_t | x_{1:t})$$

Recall: $P(y_t | x_{1:t}) \propto P(y_t, x_{1:t})$. We factorize this joint distribution according to the model ...

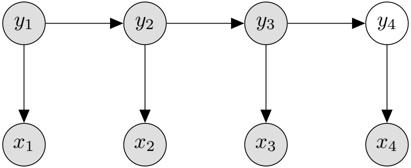
$$
\begin{aligned}
P(y_t = k | x_{1:t}) &\propto P(y_t = k, x_{1:t}) \\
&= \sum_{y_{1:t-1}} P(y_t = k, y_{1:t-1}, x_{1:t}) \quad \triangleright \text{ marginalize out} \\
&= \sum_{y_{1:t-1}} P(x_t|y_t) P(y_t|y_{t-1}) \cdot P(x_{t-1}|y_{t-1}) P(y_{t-1}|y_{t-2}) \cdots \\
&= \sum_{y_{t-1}} P(x_t|y_t) P(y_t|y_{t-1}) \cdot \sum_{y_{1:t-2}} P(x_{t-1}|y_{t-1}) P(y_{t-1}|y_{t-2}) \cdots \\
= \mu_{t,k} &= \sum_{y_{t-1}} \underbrace{P(x_t|y_t)}_{\boldsymbol{\phi}} \underbrace{P(y_t|y_{t-1})}_{\boldsymbol{\theta}} \underbrace{P(y_{t-1}, x_{1:t-1})}_{\mu_{t-1,k}} \quad \triangleright \text{ N.B. Recursive!} \\
&= \phi_{x_t|k} \sum_{y_{t-1}} \theta_{k|y_{t-1}} \mu_{t-1,k}
\end{aligned}
$$

Recursion stops at

$$
\begin{aligned}
\mu_{1,k} &= P(x_1|y_1) P(y_0) \\
&= \phi_{x_1|k} \pi_k
\end{aligned}
$$

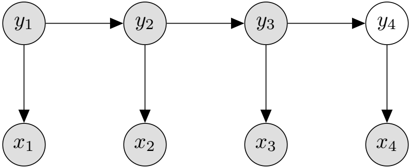
where $\pi_k \approx P(y_0 = k)$, $\phi_{v|j} \approx P(x_t = v | y_t = j)$,
$\theta_{j|k} \approx P(y_t = j | y_{t-1} = k)$ estimated from training data (counts).

Good news: in data streams, recursion stops early, since we have new evidence y_{t-1} already at time t. Thus



$$\begin{aligned}\hat{y}_t &= \operatorname*{argmax}_{y_t \in \{0,1\}} P(y_t, x_{1:t}) \\ &= \operatorname*{argmax}_{y_t \in \{0,1\}} \theta_{y_t|y_{t-1}} \phi_{x_t|y_t}\end{aligned}$$

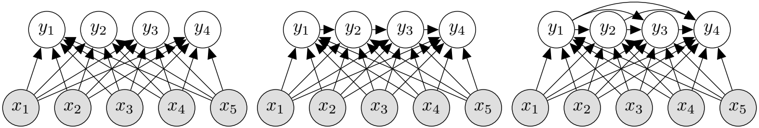
Good news: in data streams, recursion stops early, since we have new evidence y_{t-1} already at time t. Thus



$$
\begin{aligned}
\hat{y}_t &= \underset{y_t \in \{0,1\}}{\operatorname{argmax}} P(y_t, x_{1:t}) \\
&= \underset{y_t \in \{0,1\}}{\operatorname{argmax}} \theta_{y_t | y_{t-1}} \phi_{x_t | y_t}
\end{aligned}
$$

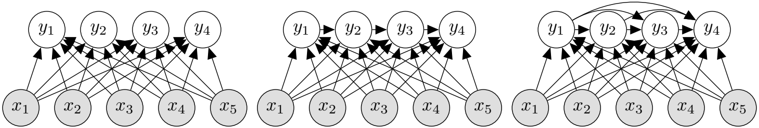
Except in delayed and semi-supervised (partially-labelled) settings!

Connection to Multi-label Classification



- Labels indices can correspond to steps in time (or space)
- Many existing multi-label methodologies can be applied
- Time dependence (= label dependence!)

Connection to Multi-label Classification

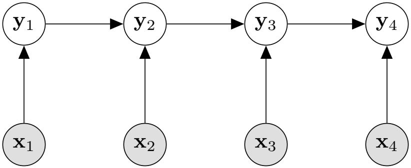


- Labels indices can correspond to steps in time (or space)
- Many existing multi-label methodologies can be applied
- Time dependence ($=$ label dependence!)

Some differences in sequential data:

- Input observation may be different to each label
- Specific domain assumptions and features

Across Labels *and* Time



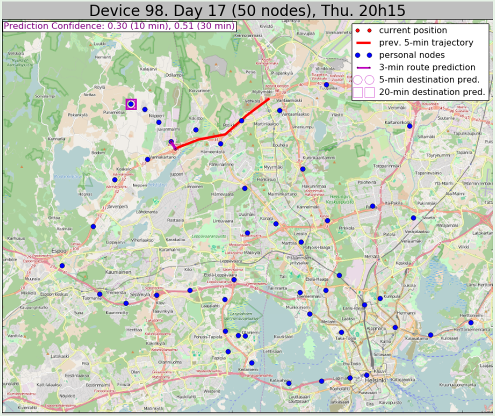
Each index is time, containing multiple labels:

$$\mathbf{y}_t = [y^{(1)}, \dots, y^{(L)}]$$

for L labels and time $t = 1, \dots, T$.

Across Labels *and* Time

Position Estimation



- Predict $\hat{\mathbf{y}}_t | t = 1, \ldots$ for L travellers.

HMMs: Continuous Observation Space

If observations are continuous values, $x_t \in \mathbb{R}$, for example sensor readings, we choose an appropriate distribution to approximate p, for example:

$$\begin{aligned} \phi_{x_t|y_t=k} &\approx p(x_t|y_t = k) \\ &= \mathcal{N}(x_t; \mu_k, \sigma_k) \end{aligned}$$

where $k \in \{1, \ldots, K\}$. And we continue as before.

- When $D > 1$, univariate mixture or multi-variate Gaussian.
- Machine learning approach: Build $\mathcal{N}$ from training data
- Other distributions are possible (exponential, Poisson, . . .)

HMMs: Continuous Observation Space

If observations are continuous values, $x_t \in \mathbb{R}$, for example sensor readings, we choose an appropriate distribution to approximate p, for example:

$$\begin{aligned}\phi_{x_t|y_t=k} &\approx p(x_t|y_t = k)\\ &= \mathcal{N}(x_t; \mu_k, \sigma_k)\end{aligned}$$

where $k \in \{1, \ldots, K\}$. And we continue as before.

- When $D > 1$, univariate mixture or multi-variate Gaussian.
- Machine learning approach: Build $\mathcal{N}$ from training data
- Other distributions are possible (exponential, Poisson, ...)

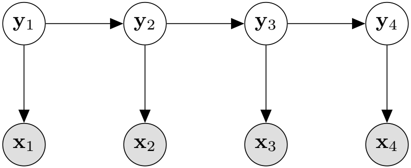
And what if the state/label space is continuous?

Continuous State Space

If we want to filter (e.g., track a target) through continuous space (e.g., $\mathbf{y}_t \in \mathbb{R}^D$), we now have[4]

$$p(\mathbf{y}_t|\mathbf{x}_{1:t}) = \int p(\mathbf{y}_t, \mathbf{y}_{1:t-1}|\mathbf{x}_{1:t}) \, \mathrm{d}\mathbf{y}_{1:t-1} \tag{4}$$

$$\begin{aligned} p(\mathbf{y}_t, \mathbf{x}_{1:t}) &= \int p(\mathbf{y}_t, \mathbf{y}_{1:t-1}, \mathbf{x}_{1:t}) \, \mathrm{d}\mathbf{y}_{1:t-1} \\ &= \int p(\mathbf{x}_t|\mathbf{y}_t) p(\mathbf{y}_t|\mathbf{y}_{t-1}) p(\mathbf{y}_{1:t-2}, \mathbf{x}_{1:t-1}) \, \mathrm{d}\mathbf{y}_{1:t-1} \end{aligned}$$



How to deal with the integral?

[4] N.B. In signal processing literature, often **x** denotes the *state*!

Outline

Kalman Filter

A linear Gaussian state space model.

To make Eq. (4) tractable, all latent variables have Gaussian distribution.

$$p(\mathbf{y}_t|\mathbf{y}_{t-1}) = \mathcal{N}(\mathbf{A}\mathbf{y}_t, \mathbf{\Sigma}_x) \quad \triangleright \text{ emission model}$$
$$p(\mathbf{x}_t|\mathbf{y}_t) = \mathcal{N}(\mathbf{B}\mathbf{y}_t, \mathbf{\Sigma}_y) \quad \triangleright \text{ transition model}$$

with transition and observation matrices **A** and **B**.

The product of two Gaussians is a Gaussian, so Eq. (4) becomes a multi-variate Gaussian for all t:

$$\mathcal{N}(\boldsymbol{\mu}_t, \mathbf{\Sigma}_t)$$

(See, e.g., David Barber *Bayesian Reasoning and Machine Learning* for full derivation).

[5] Approximations to non-lin. exist (*extended*, *unscented*); used widely

Kalman Filter

A linear Gaussian state space model.

To make Eq. (4) tractable, all latent variables have Gaussian distribution.

$$
\begin{aligned}
p(\mathbf{y}_t|\mathbf{y}_{t-1}) &= \mathcal{N}(\mathbf{A}\mathbf{y}_t, \mathbf{\Sigma}_x) && \triangleright \text{ emission model} \\
p(\mathbf{x}_t|\mathbf{y}_t) &= \mathcal{N}(\mathbf{B}\mathbf{y}_t, \mathbf{\Sigma}_y) && \triangleright \text{ transition model}
\end{aligned}
$$

with transition and observation matrices **A** and **B**.

The product of two Gaussians is a Gaussian, so Eq. (4) becomes a multi-variate Gaussian for all t:

$$\mathcal{N}(\boldsymbol{\mu}_t, \mathbf{\Sigma}_t)$$

(See, e.g., David Barber *Bayesian Reasoning and Machine Learning* for full derivation).

- Works well for e.g., tracking via GPS signal; but
- limited in many scenarios (e.g., where distributions are not uni-modal)[5].

---

[5] Approximations to non-lin. exist (*extended*, *unscented*); used widely

Outline

Particle Filter

A sequential Monte Carlo method.

- Similar goal to Kalman Filters, but very different derivation
- No Gaussianity assumption; use $f \approx p(\mathbf{x}_t|\mathbf{y}_t)$, $q \approx p(\mathbf{y}_t|\mathbf{y}_{t-1})$.

Eq. (4) is now intractable. We *approximate it* with a sum of weighted samples (i.e., particles):

$$p(\mathbf{y}_t|\mathbf{x}_{1:t}) \Leftrightarrow \{\bar{w}_t^{(m)}\mathbf{y}_t^{(m)}\}_{m=1}^{M} \tag{5}$$

where $\mathbf{y}^{(m)}$ is the m-th particle (a sample), and $\bar{w}^{(m)}$ is a normalized weight (i.e., $\sum_m \bar{w}^{(m)} = 1$).
In other words, Monte Carlo integration; we replace the integral with a weighted sum over samples:

$$\hat{\mathbf{y}}_t = \sum_{m=1}^{M} \bar{w}_t^{(m)}\mathbf{y}_t^{(m)} \approx \mathbb{E}[\mathbf{y}_t|\mathbf{x}_{1:t}] \tag{6}$$

(Recall: to minimize $(y_t - \hat{y}_t)^2$ we estimate $\mathbb{E}[y]$, not MAP).

Particle Filter

Initialize $\mathbf{y}_0^{(i)} \sim q(\mathbf{y}_0)$ and $\{w^{(i)} = \frac{1}{M}\}_{i=1}^{M}$, then for $t = 1, 2, \ldots$:

1. Particle moves (Predict)

$$\mathbf{y}_t^{(i)} \sim q(\mathbf{y}|\mathbf{y}_{t-1})$$

2. Weight update $\propto$ likelihood (Update)

$$w_t^{(i)} \propto w_{t-1}^{(i)} \cdot f(\mathbf{x}_t; \mathbf{y}_t^{(i)})$$

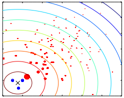
3. Resample (e.g., weighted Bagging).

$$\{\mathbf{y}_t^{(i)}\}_{i=1}^{M} \sim \{\mathbf{y}_t^{(i)}, w_t^{(i)}\}_{i=1}^{M}$$

The algorithm returns $\{\mathbf{y}_t^{(i)}, w_t^{(i)}\}_{i=1}^{M}$ which approximates the predictive density that we are interested in.

Resampling is fundamental to the success of the particle filter:

- Particles 'get lost' in dead/flat parts of the density (curse of dimensionality) – particle *degeneracy*
- The unnormalized weights w_t go towards 0
- We must regather the particles closer to the [estimation of the] target



(size $\propto w^{(i)}$)

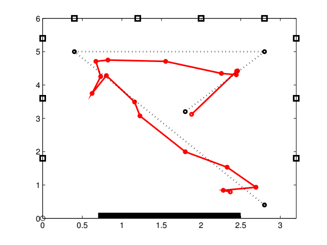
There are several different varieties[6]. Typically, we monitor the effective sample size,

$$N_{\mathrm{eff},t} = \frac{1}{\sum_{m=1}^{M} (\bar{w}_t^{(m)})^2}$$

and resample when it falls below a threshold.

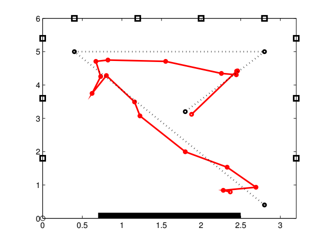
[6] See, e.g., Li, Bolić, Djurić, Resampling Methods for Particle Filtering, 2015

An example. See the Demo ▸ Link :



- 4-dimensional target (2D position + 2D velocity)
- 10-dimensional observation (light-sensor readings)
- 10×4 MHz *motes* for processing
- Tracking done in a real time stream

An example. See the Demo ▸ Link :



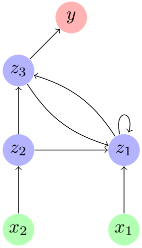
- 4-dimensional target (2D position + 2D velocity)
- 10-dimensional observation (light-sensor readings)
- 10×4 MHz *motes* for processing
- Tracking done in a real time stream

Many applications in, e.g.,

- Robotics (localization and tracking)
- Tracking (RADAR, GPS, Video, weights and errors . . .)

Outline

Recurrent Neural Networks (RNNs)



represented by weight matrices[7], e.g.,

$$\mathbf{W}_{IH} = \begin{array}{c c} & \begin{array}{ccc} z_1 & z_2 & z_3 \end{array} \\ \begin{array}{c} x_1 \\ x_2 \end{array} & \begin{pmatrix} -3 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix} \end{array}, \mathbf{W}_{HH} = \begin{array}{c c} & \begin{array}{ccc} z_1 & z_2 & z_3 \end{array} \\ \begin{array}{c} z_1 \\ z_2 \\ z_3 \end{array} & \begin{pmatrix} 0.1 & 0 & -0.4 \\ -0.3 & 0 & 0.2 \\ -2.0 & 0 & 0 \end{pmatrix} \end{array}, \mathbf{W}_{HO} = \begin{array}{c c} & y \\ \begin{array}{c} z_1 \\ z_2 \\ z_3 \end{array} & \begin{pmatrix} 0 \\ 0 \\ 2.0 \end{pmatrix} \end{array},$$

and, for inputs $\mathbf{x}_t = [x_1, x_2]^\top$ at time t, we propagate forward:

$$\mathbf{z}_t = f_H(\mathbf{W}_{IH}^\top \mathbf{x}_t + \mathbf{W}_{HH}^\top \mathbf{z}_{t-1})$$
$$\mathbf{y}_t = f_O(\mathbf{W}_{HO}^\top \mathbf{z}_t)$$

[7] Bias nodes not shown. We are modelling $\mathbf{y} = h(\mathbf{x})$.

Recurrent Neural Networks vs Bayesian Filtering

Recurrent neural networks . . .

- Deterministic, not stochastic.
- No assumption on Gaussianity, or linearity.
- Relax Markov assumption, condition on the full history
- Can model any non-linear dynamical system (they are universal approximators)
- Can be stacked in multiple layers (deep models)
- Naturally multi-label/multi-output capable
- Can generate *complex* sequences (music, language, markup text, images, . . .)
- Need intensive training and adaptations to work well

HMMs and particle filters struggle on tasks with complex dynamics. Limited information is available for predicting the next state. RNNs have memory – as big as we want / can afford.

Applications of RNNs

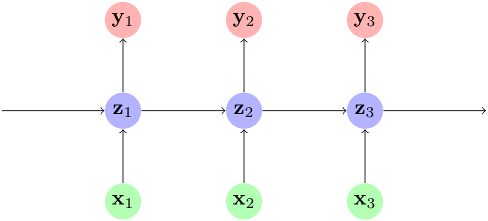
RNNs have been used in many applications (most can be considered structured-output prediction); for example:

- time series prediction
- anomaly detection
- music composition
- sequence representation
- speech recognition
- machine translation
- robot control (a problem in RL)

Various examples: Link Link Link .

Unrolling a RNN

We can unroll a RNN across time, forming a network of $t = 1, \ldots, T$ layers; each layer using the same weights:



where the weight matrices are on the edges. Note that $\mathbf{x}_t = [x_1, \ldots, x_D]$ is the input, etc.

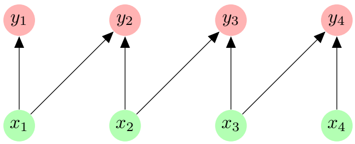
Training RNNs

. . . is very difficult!

- Backpropagation? Can apply it but may not work in practice;
  - extreme sensitivity, the gradient explodes or vanishes
  - (the gradient is not 'squashed' by a non-linearity on the way back down the network)
  - MLPs only have a few layers, and things do not get too bad (and even then – sometimes they do!)
  - More recent developments help overcome this problem:
    - LSTMs – Long Short Term Memory
    - GRUs – Gated Recurrent Units
- Extended Kalman filters in the *weight* space
- Evolutionary algorithms
- Use a simplified RNN, e.g.,
  - Time Delay Neural Networks
- Or we don't bother training them at all, e.g.,
  - Echo State Networks & Reservoir computing

Time Delay Neural Network

- A time-delay neural network maps $x_{t-k:t}$ into output y_t.
- Not actually a RNN (it's a MLP), but it unrolls like one.



$$y_t = h(x_t, x_{t-1}, x_{t-2}, \ldots, x_{t-k}) = f(\mathbf{W}^\top \mathbf{x}_{t-k:t})$$

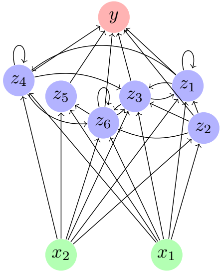
We just need to learn weights $\mathbf{W}$ (i.e., model h). It's simple, but we can still do a lot with this network, e.g.,

- time-series forecasting
- tracking
- speech recognition

Note also: Elman net ($\mathbf{z}_{t-1}$ as inputs); Jordan net ($\mathbf{y}_{t-1}$ as inputs)

Echo State Networks (ESNs)

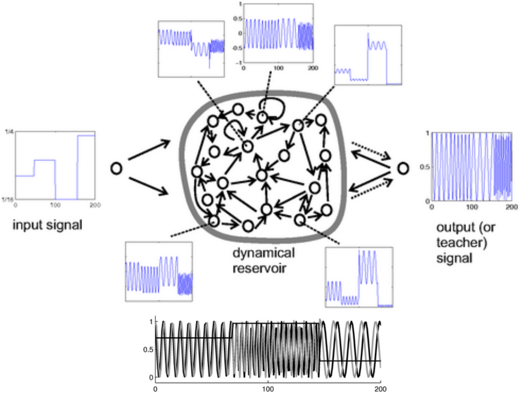
A random hidden recurrent layer; i.e., a dynamic reservoir[8] of nodes.



- $\mathbf{W}_{IH}$ is *random*
- $\mathbf{W}_{HH}$ is *random*, very large, and usually very sparse
- $\mathbf{W}_{HO}$ is typically a linear layer or simple (non-recurrent) regressor/classifier – this is the only training component

[8] Hence why this is often called reservoir computing

- Very fast (especially if can take advantage of sparsity)
- Easy application in online learning like data streams
- Non-linear expansion
- Has memory – directly proportional to the number of hidden units, but can store information *from any point in time*.
- Can be used to turn time series into i.i.d. stream!



source: Scholarpedia: trained to generate specified frequency

Practical considerations of ESNs:

- Careful initialization of weights (scaling, sparsity, ...)
- Regularization on the output layer is crucial
- Echo state property: the initial information in the reservoir must asymptotically disappear (eventually, the network state is uniquely determined by the training data)
  - Keep the spectral radius $\rho(\mathbf{W}_{HH}) \leq 1$ (normalize by $\lambda_{\max}$) (this is a recommendation; $\rho(\mathbf{W}_{HH}) > 1$ is possible also)
- Memory: more hidden units is better
- Leaking rate $\alpha \in (0, 1]$:

$$\mathbf{z}_t = (1 - \alpha)\mathbf{z}_{t-1} + \alpha f_H(\mathbf{W}_{IH}^\top \mathbf{x}_t + \mathbf{W}_{HH}^\top \mathbf{z}_{t-1})$$

  (where $\alpha = 1$ is a special case) – it determines the speed of the reservoir update (should match the speed of the dynamics).
- Typically $f_H = \tanh$, but other activation functions are possible

See, for example: M. Lukoševičius *A Practical Guide to Applying Echo State Networks*

A collection of *random functions* helps with learning? Yes, we do this all the time in machine learning;

- Feature creation and basis function expansion, especially,
- Radial Basis Function networks and “E.L.M.s” (random MLPs)
- Kernels (similarity function)

We can think of the reservoir as a large recurrent basis function expansion!

- Nice way to represent sequences.
- It provides more predictive power via non-linearity and memory.
- Will not solve advanced applications (i.e., high-dimensional data such as in video or speech), but can be very useful!

Outline

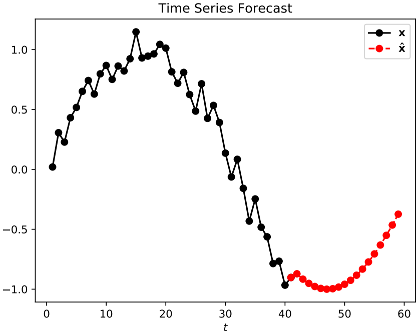
Time-Series Forecasting (Prediction)

Given

$$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t$$

we want a model f to predict

$$\widehat{\mathbf{x}}_{t+1}, \widehat{\mathbf{x}}_{t+2}, \ldots, \widehat{\mathbf{x}}_{t+k}$$



Outline

Simple Methods

- Naive Forecasting (rain today = rain tomorrow),

$$\hat{x}_{t+1} = x_t$$

- Moving average (mean of last k observations)
- Auto-regressive linear fit on previous k points, and extrapolate.

Machine Learning for Forecasting

Given training data, we can design a machine learning approach (e.g., artificial neural networks, decision trees, ...), on, e.g.,

X_{t-k}	$X_{\ldots}$	X_{t-2}	X_{t-1}	X_t	X_{t+1}
1	A	2.3	1.8	-3	4
A	2.3	1.8	-3	4	B
2.3	1.8	-3	4	B	3
1.8	-3	4	B	3	-7
...	...	...	...	...	...
T	39	3	4	0.1	?

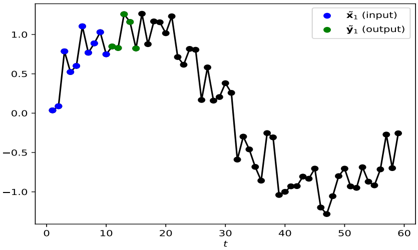
i.e., model

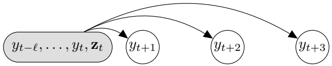
$$\hat{x}_{t+1} = f(x_{t-k}, \ldots, x_t; \theta)$$

The decision making and interpretation is relegated to the learner. We can plug in $\hat{x}_{t+1}$ (as a feature) and propagate; or estimate a window directly by adding more columns (i.e., multi-output):

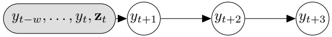
$$\hat{x}_{t+1}, \ldots, \hat{x}_{t+k} = f(x_{t-4}, \ldots, x_t)$$

Multi-Step-Ahead Forecasting

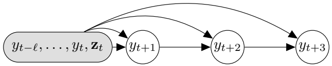




Direct

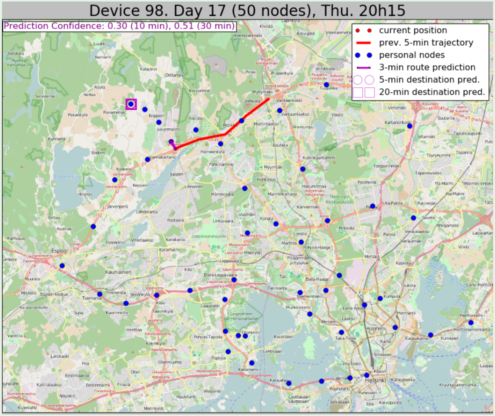


Iterated



Classifier/Regressor Chain cascade

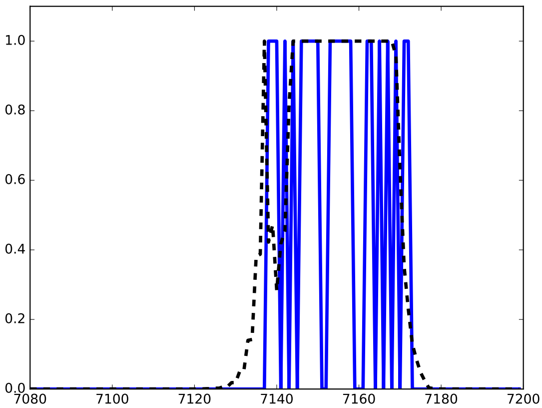
Trajectory Prediction



- Predict $\widehat{\mathbf{y}}_{t+1}, \widehat{\mathbf{y}}_{t+2}, \ldots$ for L travellers.
- i.e., predict future trajectory given, GPS coordinates, signal strength, battery level, current time, ...

Predictive Maintenance

- Sensor readings from aircraft and textual description of observations
- Predict warnings/required replacement of components

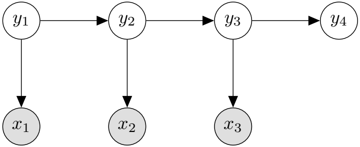


Outline

HMMs for Forecasting

In the forecasting task, at time t we want a prediction

$$\hat{y}_{t+1} = \operatorname*{argmax}_{y} P(y|\mathbf{x}, y_{1:t})$$



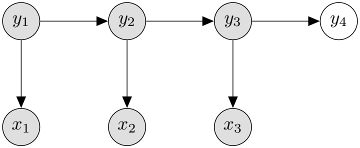
Similarly to filtering task (one extra term):

$$\begin{aligned}
P(y_{t+1} = k|x_{1:t}) &\propto P(y_{t+1} = k, x_{1:t}) \\
&= \sum_{y_{1:t}} P(y_{t+1}|y_t) P(x_t|y_t) P(y_t|y_{t-1}) \cdot P(x_{t-1}|y_{t-1}) P(y_{t-1}|y_{t-2}) \cdots \\
&= \ldots
\end{aligned}$$

HMMs for Forecasting

In the forecasting task, at time t we want a prediction

$$\hat{y}_{t+1} = \underset{y}{\operatorname{argmax}} P(y|\mathbf{x}, y_{1:t})$$



Similarly to filtering task (one extra term):

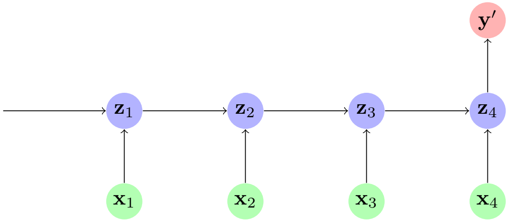
$$\begin{aligned}
P(y_{t+1} = k|x_{1:t}) &\propto P(y_{t+1} = k, x_{1:t}) \\
&= \sum_{y_{1:t}} {\color{red}P(y_{t+1}|y_t)} P(x_t|y_t) P(y_t|y_{t-1}) \cdot P(x_{t-1}|y_{t-1}) P(y_{t-1}|y_{t-2}) \cdots \\
&= \ldots
\end{aligned}$$

(In streams, only that term is necessary.)

Outline

RNNs for Forecasting

The forecasting task for RNNs involves an extra link without input, e.g., where $\mathbf{y} := \mathbf{x}_5$:



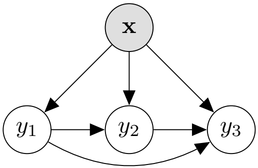
Unrolled RNN suitable for forecasting: nodes $\mathbf{y}_1, \ldots, \mathbf{y}_3$ not shown/necessary.

$$\mathbf{z}_t = f_H(\mathbf{W}_{IH}^\top \mathbf{x}_t + \mathbf{W}_{HH}^\top \mathbf{z}_{t-1})$$
$$\mathbf{y}_t = f_O(\mathbf{W}_{HO}^\top \mathbf{z}_t)$$

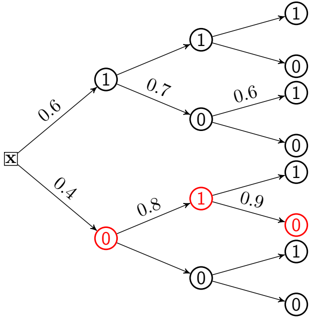
Only need relevant calculations in forward/back-propagation.

Outline

Probabilistic Classifier Chains



For example, where each $y_t \in \{0,1\}$



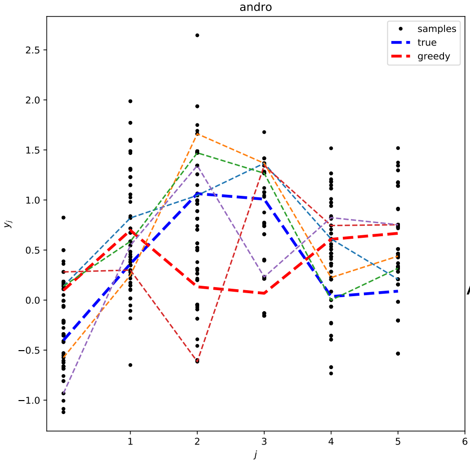
- Predictions become input, across a cascade/chain
- Efficient
- Probabilistic interpretation:

$$P(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{T} P(y_t|\mathbf{x}, y_1, \ldots, y_{t-1})$$

$$\hat{\mathbf{y}} = f(\mathbf{x}) = \operatorname*{argmax}_{\mathbf{y}\in\{0,1\}^3} P(\mathbf{y}|\mathbf{x})$$

- Search probability tree (for best prediction) with AI-search techniques (Monte-Carlo search, beam search, A* search, . . .)
- Explore structure

Regressor Chains



e.g., where $\mathbf{y} \in \mathbb{R}^6$,

- Sample down the chain

$$y_{t+1} \sim p(y_{t+1} | y_1, \ldots, y_t, \mathbf{x})$$

- More samples = more hypotheses
- Consider different loss functions

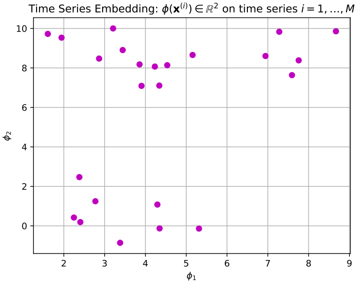
Applications:

- Multi-output regression
- Tracking
- Forecasting
- Anomaly detection and interpretation
- Imputation of missing data

Outline

Embedding Time Series

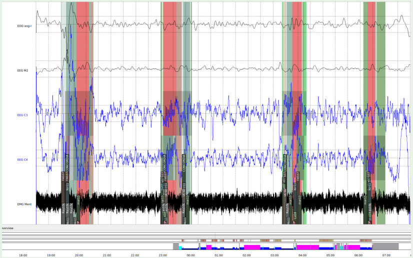
We seek to turn variable-length time series $\{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$ into fixed-length vectors $\phi^{(t)} = [\phi_1, \ldots, \phi_D]$.



This lets us compare and cluster time series/look for anomalies, (and classify, if we have the label): measure similarity/distance between $\phi(\mathbf{x}^{(i)})$ and $\phi(\mathbf{x}^{(2)})$, etc. (i.e., proceed 'as normal').

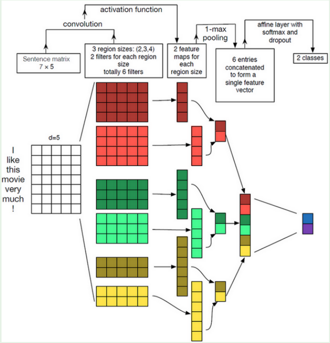
Signal Embedding (Expert Driven)

- Patients suffering from insomnia are monitored overnight (6-channel EEG)
- Certain signals are of interest: Spindles, α-waves, β-waves, ...
- We can filter the signal for certain frequencies (using e.g., wavelet transform)
- Send through simple hand-crafted pooling functions (`average`, `max`, ...)



Raw Text Classification (Data Driven)

- We want to classify positive vs negative movie reviews
- Incoming reviews $\mathbf{x}_t = [x_1, \ldots, x_{D_t}]$ are of variable length
- we embed them into fixed-length ϕ_D and proceed 'normally'
- Similar to a 'simple' embedding, but data-driven.
- Modern solutions employ deep neural networks (RNNs, CNNs) propagate all the way back through the input filter.



Outline

Reminder: One-Step Decision Theory

Under uncertainty, we wish to assign $y^* = f^*(\mathbf{x})$, the best label/hypothesis, $y^* \in \mathcal{Y}$, given $\mathbf{x} \in \mathbb{R}^D$.

Minimizing conditional expected loss

$$f^* = \operatorname*{argmin}_{f \in \mathcal{F}} \underbrace{\sum_{y \in \mathcal{Y}} \ell(f(\mathbf{x}), y) P(y|\mathbf{x})}_{\mathbb{E}_{Y \sim P(Y|\mathbf{x})}[\ell(\hat{y}, Y)|\mathbf{x}]}$$

aka risk.

under loss function ℓ, which describes our preferences.
In the case of 0/1 loss (1 if $y \neq \hat{y}$, else 0),

Maximum a Posteriori

$$y^* = \operatorname*{argmax}_{y \in \mathcal{Y}} p(\mathbf{x}|y)P(y) = \operatorname*{argmax}_{y \in \{0,1\}} P(y|\mathbf{x})$$

We can estimate P from the training data.

An intelligent agent wishes to make a decision to achieve a goal. The decision which involves the least risk. Another way of looking at the problem: utility. A rational agent maximizes their expected utility, not necessarily a simple *payoff* (e.g., amount of money):

Expected Utility

$$U(y) = \sum_{y \in \mathcal{Y}} u(y)p(y)$$

with satisfaction/utility $u(y)$ for outcome y. Different agents may have different utility functions, even when 'payoff' is the same item. Instead of labels given input, we can deal with actions given evidence and belief.

- A risk-prone agent will tend to gamble higher stakes
- A conservative (risk-adverse) agent will not
- A risk-neutral agent only cares about payoff y directly

An intelligent agent wishes to make a decision to achieve a goal. The decision which involves the least risk. Another way of looking at the problem: utility. A rational agent maximizes their expected utility, not necessarily a simple *payoff* (e.g., amount of money):

Expected Utility

$$U(y) = \sum_{y \in \mathcal{Y}} u(y)p(y)$$

with satisfaction/utility $u(y)$ for outcome y. Different agents may have different utility functions, even when 'payoff' is the same item. Instead of labels given input, we can deal with actions given evidence and belief.

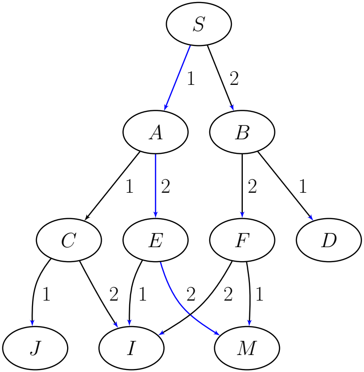
- A risk-prone agent will tend to gamble higher stakes
- A conservative (risk-adverse) agent will not
- A risk-neutral agent only cares about payoff y directly

What about sequential decisions?

In a Deterministic Environment

(e.g., board games: chess, etc.)

- The state space, e.g., $\{A, B, \ldots, M\}$
- An initial state, e.g., S
- A goal state, e.g., M
- A set of actions, e.g., $\{1, 2\}$
- A cost for each branch, e.g., $\text{Cost}(S, 1) = 0$

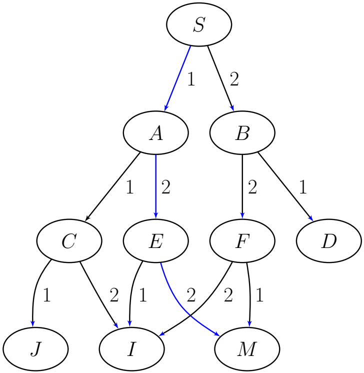


It's just a search! AI-search techniques applicable (DFS, A^*, ...).

In a Deterministic Environment

(e.g., board games: chess, etc.)

- The state space, e.g., $\{A, B, \ldots, M\}$
- An initial state, e.g., S
- A goal state, e.g., M
- A set of actions, e.g., $\{1, 2\}$
- A cost for each branch, e.g., $\mathrm{Cost}(S, 1) = 0$



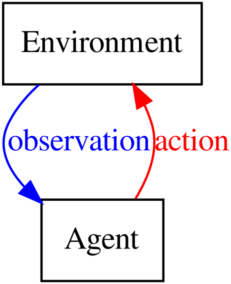
It's just a search! AI-search techniques applicable (DFS, A^*, ...).

What if environment is stochastic?

Markov Decision Processes (MDP)

MDPs are models that seek to provide optimal solutions for stocastic sequential decision problems.

> MDP = Markov Chain+One-step Decision Theory



Now we have a model with

- $\mathcal{P}(s'|s,a)$ transition function
- $\mathcal{R}(s',a,s)$ reward function

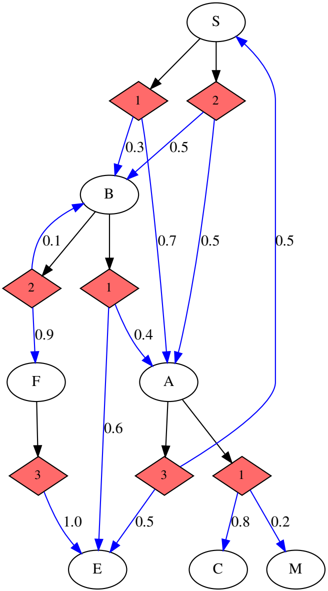
Objective: obtain a policy

$$\pi : \mathcal{S} \mapsto \mathcal{A}$$

which maximizes expected reward:

$$\mathbb{E}[R_0|s_0 = s] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t)\right]$$

solution can be found via dynamic programming! Just need the model . . .



Now we have a model with

- $\mathcal{P}(s'|s,a)$ transition function
- $\mathcal{R}(s',a,s)$ reward function

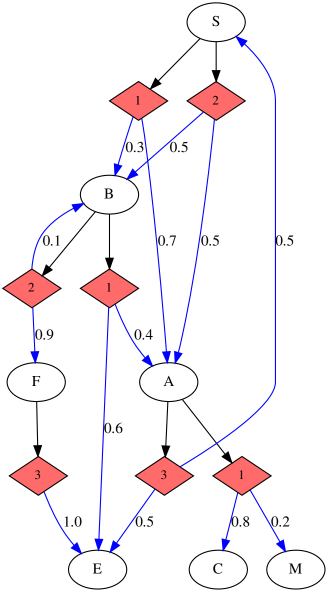
Objective: obtain a policy

$$\pi : \mathcal{S} \mapsto \mathcal{A}$$

which maximizes expected reward:

$$\mathbb{E}[R_0|s_0 = s] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t)\right]$$

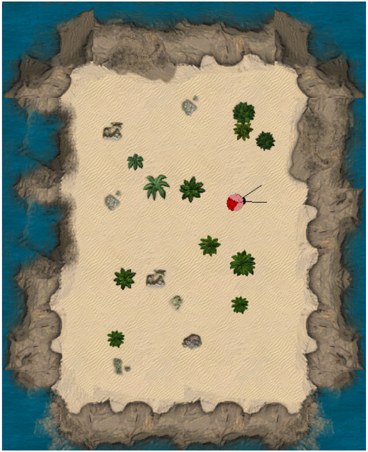
solution can be found via dynamic programming! Just need the model ...

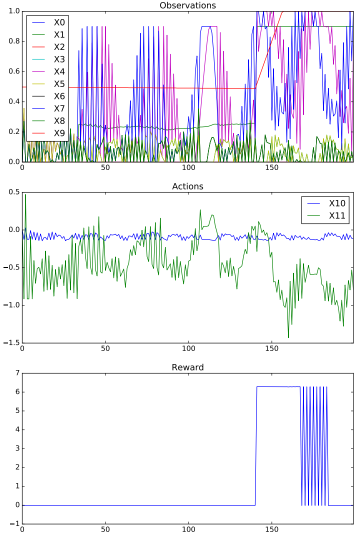


If we don't have it?

Reinforcement Learning

- Don't have transition/reward functions.
- No input-output training pairs, just reward signal.
- The agent needs to experiment! Exploration vs exploitation.
- Deep neural net can learn a model
- ... over millions of iterations.
- Emerging applications:
  - Gameplay
  - Robotics (usually trained in simulation)
  - Parameter-tuning, etc. (as a tool)
- Transfer learning is promising




Observations
X0
X1
X2
X3
X4
X5
X6
X7
X8
X9
Actions
X10
X11
Reward

Outline

Summary

M2 Data & Knowledge: Data Stream Mining

Jesse Read

Time Series and Sequential Data