Learning in the Model Space for Temporal Data

Peter Tiňo

School of Computer Science
University of Birmingham, UK
Special thanks to ...

- Barbara Hammer
- Huanhuan Chen
- Alessio Micheli
- Michal Čerňanský
- Luba Beňúšková
- Jort van Mourik
- Igor Farkaš
- Ali Rodan
- Jochen Steil
- Xin Yao
- Nick Gianniotis
- Dan Conford
- Manfred Opper
- Fengzhen Tang, ...
Learning in the data or model space?

Complex data, e.g.
- long (possibly) multivariate time series (video, sensor data, etc.) of variable length
- fMRI data measured on normalized cognitive tasks
- time series of mass spectra
- ...

We kind-of know what to do for fix-dimensional vectorial data, but how to represent such complex data in order to usefully "learn" from it?

The critical notion of "smoothness" when learning from historic data - "close" things should lead to the same response "closeness" - distance, dot product, ...

How to represent complex data items in the light of the "smoothness criterion"?
Learning in the Model Space Framework

- Each data item is represented by a model that "explains" it

- Learning is formulated in the space of models (function space)

- Parametrization-free notions of "input similarities": similarity quantifications between the models not their (arbitrary) parametrizations!

- Model class
  - flexible enough to represent variety of data items
  - sufficiently constrained to avoid overfitting
NARMA task - illustrative example

NARMA sequences - orders 10, 20 and 30 - represented as state space models
Representing Sequential Data

We will concentrate on sequential data

Each sequence will be represented by a dedicated model that captures everything important about the sequence in a compact manner.

If we’d like to change focus on different aspects of the sequences, we need to change the base model accordingly.

What should our base model be?

Many possible choices, but here we’ll consider State space models - a very general and powerful model class capturing temporal structure through the notion of information processing state (IPS).

IPS at time $t$ codes for the entire history of time series items we have seen up to time $t$. 
Discrete Time State Space Model

States of the DS form IPS

These states code the entire history of sequence elements we have seen so far.
A State Space Model

- $N$-dimensional continuous state space $[-1, 1]^N$

$$\begin{align*}
\mathbf{x}(t) &= f(\mathbf{x}(t-1), \mathbf{u}(t)) = \sigma(R\mathbf{x}(t-1) + V\mathbf{u}(t) + \mathbf{b}), \\
\mathbf{y}(t) &= h(\mathbf{x}) = \kappa(W\mathbf{x}(t) + \mathbf{a}),
\end{align*}$$

- $\mathbf{x}(t) \in [-1, 1]^N$ - state vector at time $t$
- $\mathbf{u}(t)$ - input time series element at time $t$
- $f(\cdot)$ - state-transition function
- $\mathbf{y}(t)$ - output of the linear readout from the state space.
- $h(\cdot)$ - output function
"Grammatical"/class 1: odd number of b’s
"Non-Grammatical"/class 0: even number of b’s (including none)
translate input streams over \( \{a, b\} \) to sequences over \( \{x, y\} \)
Picking the Right Base Model

Appealing feature of the previous two examples - the state space models were perfectly tailored for their intended purpose.

Of course, in our general scenario we may not know (and typically this is the case!) the right state space structure that extracts from the data (time series) what is needed for the particular task:
- what are the appropriate IPS?
- What is their transition structure?

Since we are ML people, the answer seems to be obvious: Learn everything!
... easier said than done ...
Picking the Right Base Model

Things are actually much worse!
If we adopted a general parameterized model

\[
\begin{align*}
    \mathbf{x}(t) &= f(\mathbf{x}(t-1), \mathbf{u}(t)) = \sigma(R \mathbf{x}(t-1) + V \mathbf{u}(t) + \mathbf{b}), \\
    \mathbf{y}(t) &= h(\mathbf{x}) = \kappa(W \mathbf{x}(t) + \mathbf{a}),
\end{align*}
\]

we may have problems fitting it to individual sequences (information latching problem)

Worse still - Even if we managed to do perfect model fitting so that each sequence is nicely represented
- how to define distance between state space models - driven dynamical systems?

Highly non-trivial - as we know from the work in machine vision and video processing communities (low-dim dynamical mostly linear dynamical systems).
Interpret the state space model as a "dynamical kernel machine".

\[ x(t) = f(x(t-1), u(t)) = \sigma(Rx(t-1) + Vu(t) + b), \]
\[ y(t) = h(x) = Wx(t) + a, \]

At time \( t \) the input item \( u(t) \) is mapped to the feature space - state space of the dynamic system - its feature space image is the state \( x(t) \)

The feature mapping is dynamic - e.g. besides providing an image \( x(t) \) of \( u(t) \) it also reflects the history of inputs up to time \( t \)

Once in the feature space - the representations are rich enough to allow for a convenient use of linear tools!
Look Back at Kernel Machines - Static Data

1. Initially - formulate and learn potentially complex non-linear models directly in the data space.
2. Later - "kernel trick"
   - map data to a high dimensional feature space (linear subspace of the Hilbert space defined by the kernel)
   - the feature space is general and "redundant" - it represents many potentially useful features in the data - not all of them need to be used for every task to the same degree of relevance - but they are there to be potentially picked up by the task dependent "readout" mapping
   - the mechanism for picking up useful features (readout) from the general feature space" - i.e. the model operating on the features can be linear - all the "hard" work has been done in the feature space creation.
In the Temporal Context...

- Would be good to come up with a "dynamic kernel trick"
- fixed, high-dim and "general purpose" state space formulations
- state (feature) space is general and "redundant"
- task dependent linear readout mapping
- all the "hard" work will be done in the feature space creation.

How can the idea of "general purpose" dynamic feature space be realized?

Let’s get an inspiration from information theory ("general purpose" information sources).
The simplest construction of IPS is based on concentrating on the very recent (finite) past
e.g. definite memory machines, finite memory machines, Markov models

Example:
Only the last 5 input symbols matter when reasoning about what symbol comes next
...
  1 2 1 1 2 3 2 1 2 1 2 4 3 2 1 1
  2 1 1 1 1 3 4 4 4 4 1 4 3 2 1 1
  3 3 2 2 1 2 1 2 2 1 3 4 3 2 1 1

All three sequences belong to the same IPS “43211”
Probabilistic framework - Markov model (MM)

finite context-conditional next-symbol distributions

... 1 2 1 1 2 3 2 1 2 1 2 4 3 2 1 1 | ?
... 2 1 1 1 1 3 4 4 4 4 1 4 3 2 1 1 | ?
... 3 3 2 2 1 2 1 2 2 1 3 4 3 2 1 1 | ?

\[ P(s \mid 11111) \]
\[ P(s \mid 11112) \]
\[ P(s \mid 11113) \]
...
\[ P(s \mid 11121) \]
...
\[ P(s \mid 43211) \]
...
\[ P(s \mid 44444) \]

\[ s \in \{1, 2, 3, 4\} \]
Markov model

MM are intuitive and simple, but ...

**Number of IPS (prediction contexts) grows exponentially fast with memory length**

**Large alphabets and long memories are infeasible:**

- computationally demanding and
- very long sequences are needed to obtain sound statistical estimates of free parameters
Variable memory length MM (VLMM)

Sophisticated implementation of potentially high-order MM

Takes advantage of subsequence structure in data

Saves resources by making memory depth context dependent

Use deep memory only when it is needed

Natural representation of IPS in form of Prediction Suffix Trees

Closely linked to the idea of universal simulation of information sources.
Prediction suffix tree (PST)

IFS are organized on nodes of PST
Traverse the tree from root towards leaves in reversed order
Complex and potentially time-consuming training

Each node $i$ has associated next-symbol probability distribution $P(\cdot|\cdot i)$
Contractive DS lead to Markovian IPS
Contractive DS lead to Markovian IPS

Mapping symbolic sequences into Euclidean space

Coding strategy is Markovian

Sequences with shared histories of recently observed symbols have close images

The longer is the common suffix of two sequences, the closer are they mapped to each other

Add a simple output-generating readout on top of the fixed state dynamics!
More Formally...

Given an input symbol $s \in \mathcal{A}$ at time $t$, and activations of recurrent units from the previous step, $\mathbf{x}(t - 1)$,

$$\mathbf{x}(t) = f_s(\mathbf{x}(t - 1)).$$

This notation can be extended to sequences over $\mathcal{A}$ the recurrent activations after $t$ time steps are

$$\mathbf{x}(t) = f_{s_1 \ldots s_t}(\mathbf{x}(0)).$$

If maps $f_s$ are contractions with Lip. constant $0 < C < 1$,

$$\| f_{vq}(\mathbf{x}) - f_{wq}(\mathbf{x}) \| \leq C^{|q|} \cdot \| f_v(\mathbf{x}) - f_w(\mathbf{x}) \| \leq C^{|q|} \cdot diam(\mathcal{X}).$$
Theoretical grounding

Theorem (Hammer & Tiño):

- Recurrent networks with contractive transition function can be approximated arbitrarily well on input sequences of unbounded length by a definite memory machine. Conversely,

- every definite memory machine can be simulated by a recurrent network with contractive transition function.

Hence initialization with small weights induces an architectural bias into learning with recurrent neural networks.
Learnability (PAC framework)

The architectural bias emphasizes one possible region of the weight space where generalization ability can be formally proved.

Standard RNN are not distribution independent learnable in the PAC sense if arbitrary precision and inputs are considered.

Theorem (Hammer & Tiňo):

- Recurrent networks with contractive transition function with a fixed contraction parameter fulfill the distribution independent UCED property and so

- unlike general recurrent networks, are distribution independent PAC-learnable.
State space complexity ↔ sequence complexity

Complexity of the driving input stream (topological entropy) is directly reflected in the complexity of the state space activations (fractal dimension).

Theorem (Tiño & Hammer):
Recurrence activations inside contractive RNN form fractal clusters the dimension of which can be bounded by the scaled entropy of the underlying driving source. The scaling factors are fixed and are given by the RNN parameters.
Fix the DS to a contractive system!

Fix the DS part to a randomly constructed contraction.

Only train the simple linear readout output map. Can be done efficiently.

Reservoir computation models.
Simple, non-randomized reservoirs
Cycle reservoir with regular jumps

still a simple deterministic construction...

(A)

(B)

N=18

Learning in the Model Space for Temporal Data – p.29/66
**Reservoir characterizations**

**Memory capacity** - [Jaeger] A measure correlating past events in an i.i.d. input stream with the network output.

Assume the network is driven by a univariate stationary input signal $u(t)$.

For a given delay $k > 0$, construct the network with optimal parameters for the task of outputting $u(t - k)$ after seeing the input stream ...$u(t - 1)u(t)$ up to time $t$.

Goodness of fit - squared correlation coefficient between the desired output $u(t - k)$ and the observed network output $y(t)$
Memory capacity

\[ MC_k = \frac{Cov^2(u(t - k), y(t))}{Var(u(t)) \cdot Var(y(t))}, \]

where \( Cov \) denotes the covariance and \( Var \) the variance operators. The short term memory (STM) capacity is then given by:

\[ MC = \sum_{k=1}^{\infty} MC_k. \]

[Jaeger] For any recurrent neural network with \( N \) recurrent neurons, under the assumption of i.i.d. input stream, the STM capacity cannot exceed \( N \).
[Tiño & Rodan] Under the assumption of zero-mean i.i.d. input stream, the memory capacity of linear SCR architecture with \( N \) reservoir units can be made arbitrarily close to \( N \).

**Theorem (Tiño & Rodan):** Consider a linear SCR network with reservoir weight \( 0 < r < 1 \) and an input weight vector such that the reservoir activations are full rank. Then, the SCR network memory capacity is equal to

\[
MC = N - (1 - r^{2N}).
\]
Reservoir characterizations

Eigen-decomposition of reservoir matrix

![Graphs showing ESN, SCR, CRJ, and CRHJ](image-url)
Reservoir characterizations

Memory capacity

(A) (B) (C) (D)
Reservoir characterizations

Pseudo-Lyapunov exponents

![Graphs showing pseudo-Lyapunov exponents for NARMA and Laser tasks with scaling parameter on the x-axis and lyapunov exponent on the y-axis. The graphs compare performance of ESN, SCR, and CRJ models.]
1. For all sequences - the same state transition (reservoir) part;
2. For each time series train the linear readout on prediction task
3. Gaussian kernel in the readout function space.
Distance in the Readout Function Space

- Two readouts $h_1$ and $h_2$ obtained on two sequences:

\[
L_2(h_1, h_2) = \left( \int_C \| h_1(x) - h_2(x) \|^2 \, d\mu(x) \right)^{1/2}
\]

where $\mu(x)$ is the probability density measure on the input domain, and $C$ is the integral range.

- $C = [-1, +1]^N$ as we use recurrent neural network reservoir formulation with $tanh$ transfer function.
Model Distance with Uniform Measure

Consider two readouts from the same reservoir

\[ h_1(x) = W_1 x + a_1, \]
\[ h_2(x) = W_2 x + a_2. \]

Since the sigmoid activation function is employed in the domain of the readout, \( C \in [-1, 1]^N \).

\[
L_2(h_1, h_2) = \left( \int_C \| h_1(x) - h_2(x) \|^2 \, d\mu(x) \right)^{1/2}
= \left( \frac{2^N}{3} \sum_{j=1}^N \sum_{i=1}^O w_{i,j}^2 + 2^N \| a \|^2 \right)^{1/2}
\]

where \( w_{i,j} \) is the \((i, j)\)-th element of \( W_1 - W_2 \), \( a = a_1 - a_2 \).
Model Distance vs. Parameter Distance

- Scaling of the squared model distance by

\[
\frac{1}{3} \sum_{j=1}^{N} \sum_{i=1}^{O} w_{i,j}^2 + \|a\|^2,
\]

- Squared Euclidean distance of the readout parameters

\[
\sum_{j=1}^{N} \sum_{i=1}^{O} w_{i,j}^2 + \|a\|^2,
\]

more importance is given to the 'offset' than 'orientation' of the readout mapping.
Non-uniform State Distribution

- For $K$-component Gaussian mixture model

$$\mu(x) = \sum_{i=1}^{K} \alpha_i \mu_i(x | \eta_i, \Sigma_i)$$

The model distance can be obtained as follows:

$$L_2^2(h_1, h_2) = \sum_{i=1}^{K} \alpha_i \left\{ \begin{array}{c} \text{trace}(W^T W \Sigma_k) + a^T a \\ + \eta_k^T W^T W \eta_k + 2a^T W \eta_k \end{array} \right\}$$

- Sampling approximation

$$L_2^2(h_1, h_2) \approx \frac{1}{m} \sum_{i=1}^{m} \|h_1(x(i)) - h_2(x(i))\|^2.$$
Three Time Series Kernels

- Uniform state distribution
  - Reservoir Kernel: $RV$

- Non-uniform state distribution
  - Reservoir kernel by sampling: $\text{SamplingRV}$
  - Reservoir kernel by Gaussian mixture models: $\text{GMMRV}$

\[
\mathcal{K}(h_i, h_j) = \exp \left\{ -\gamma \cdot L_2^2(h_i, h_j) \right\}
\]
Fisher Kernel

- Single model for all data items.
- Based on Information Geometry

Fisher vector (score functions): $U(s) = \nabla_\lambda \log P(s | \lambda)$

Metric tensor - Fisher Information Matrix $\mathcal{F}$

$K(s_i, s_j) = (U(s_i))^T \mathcal{F} U(s_j)$. 
Endowing the readout with a noise model yields a generative time series model of the form:

\[ x(t) = f(u(t), x(t - 1)), \]
\[ u(t + 1) = W x(t) + a + \varepsilon(t). \]

Assume the noise model follows a Gaussian distribution

\[ \varepsilon(t) = \mathcal{N}(0, \sigma^2 I). \]
Fisher Kernel by Reservoir Models

- partial derivatives of log likelihood $\log p(u(1..\ell))$

$$U = \frac{\partial \log p(u(1..\ell))}{\partial W} = \frac{\partial}{\partial W} \prod_{t=1}^{\ell} P(u(t) | u(1 \cdots t - 1))$$

$$= \sum_{t=1}^{\ell} \frac{(u(t) - \alpha) x(t - 1)^T - W x(t - 1) x(t - 1)^T}{\sigma^2}.$$

- Fisher kernel (simplified in the usual way - Id matrix instead of $\mathcal{F}$) for two time series $s_i$ and $s_j$ with scores $U_i$ and $U_j$:

$$\mathcal{K}(s_i, s_j) = \sum_{o=1}^{O} \sum_{n=1}^{N} (U_i \circ U_j)_{o,n}.$$
Other time series kernels

Kernel based on Dynamic time warping (DTW)

DTW tries to ‘warp’ the time axis of one (or both) sequences to achieve a better alignment

DTW can generate un-intuitive alignments by mapping a single point on one time series onto a large subsection of another time series, leading to inferior results

efficient versions of DTW based kernels available
Other time series kernels

Autoregressive kernel

VAR model class of a given order is used to generate an infinite family of features from the time series.

For a given time series $s$, the likelihood profile $p_\theta(s)$ across all possible parameter setting (under a matrix normal-inverse Wishart prior $\omega(\theta)$) forms a representation of $s$.

$$\mathcal{K}_{AR}(s_i, s_j) = \int_\theta p_\theta(s_i)p_\theta(s_j)\omega(d\theta)$$
Experimental Comparisons

Compared Kernels:

- Autoregressive Kernel: **AR**
- Fisher Kernel with Hidden Markov Model: **Fisher**
- Dynamic time wrapping: **DTW**

Our Kernels

- Reservoir Kernel: **RV**
- Reservoir kernel by sampling: **SamplingRV**
- Reservoir kernel by Gaussian mixture models: **GMMRV**
- Fisher kernel with reservoir models: **FisherRV**
Synthetic Data: NARMA

\[ s(t + 1) = 0.3s(t) + 0.05s(t) \sum_{i=0}^{9} s(t - i) + 1.5u(t - 9)u(t) + 0.1 \]

\[ s(t + 1) = \tanh(0.3s(t) + 0.05s(t) \sum_{i=0}^{19} s(t - i) + 1.5u(t - 19)u(t) + 0.01) + 0.2 \]

\[ s(t + 1) = 0.2s(t) + 0.004s(t) \sum_{i=0}^{29} s(t - i) + 1.5u(t - 29)u(t) + 0.201 \]
Illustration of MDS on the model distance among reservoir weights
Illustration of the performance of compared kernels with different noise levels
## Benchmark Data

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Length</th>
<th>Classes</th>
<th>Train</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbols</td>
<td>398</td>
<td>6</td>
<td>25</td>
<td>995</td>
</tr>
<tr>
<td>OSULeaf</td>
<td>427</td>
<td>6</td>
<td>200</td>
<td>242</td>
</tr>
<tr>
<td>Oliveoil</td>
<td>570</td>
<td>4</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Lighting2</td>
<td>637</td>
<td>2</td>
<td>60</td>
<td>61</td>
</tr>
<tr>
<td>Beef</td>
<td>470</td>
<td>6</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Car</td>
<td>576</td>
<td>4</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>Fish</td>
<td>463</td>
<td>8</td>
<td>175</td>
<td>175</td>
</tr>
<tr>
<td>Coffee</td>
<td>286</td>
<td>2</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>Adiac</td>
<td>176</td>
<td>37</td>
<td>390</td>
<td>391</td>
</tr>
</tbody>
</table>
## Results: Classification Accuracy

<table>
<thead>
<tr>
<th>Dataset</th>
<th>DTW</th>
<th>AR</th>
<th>Fisher</th>
<th>RV</th>
<th>FisherRV</th>
<th>GMMRV</th>
<th>SamplingRV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbols</td>
<td>94.77</td>
<td>91.15</td>
<td>94.42</td>
<td><strong>98.08</strong></td>
<td>95.96</td>
<td>97.31</td>
<td>95.77</td>
</tr>
<tr>
<td>OSULeaf</td>
<td><strong>74.79</strong></td>
<td>56.61</td>
<td>54.96</td>
<td>69.83</td>
<td>64.59</td>
<td>56.55</td>
<td>63.33</td>
</tr>
<tr>
<td>Oliveoil</td>
<td>83.33</td>
<td>73.33</td>
<td>56.67</td>
<td>86.67</td>
<td>83.33</td>
<td>84.00</td>
<td><strong>90.00</strong></td>
</tr>
<tr>
<td>Lighting2</td>
<td>64.10</td>
<td>77.05</td>
<td>64.10</td>
<td>77.05</td>
<td>75.41</td>
<td>78.69</td>
<td><strong>80.33</strong></td>
</tr>
<tr>
<td>Beef</td>
<td>66.67</td>
<td>78.69</td>
<td>58.00</td>
<td>80.00</td>
<td>68.00</td>
<td>79.67</td>
<td><strong>86.67</strong></td>
</tr>
<tr>
<td>Car</td>
<td>58.85</td>
<td>60.00</td>
<td>65.00</td>
<td>76.67</td>
<td>72.33</td>
<td>78.33</td>
<td><strong>86.67</strong></td>
</tr>
<tr>
<td>Fish</td>
<td>69.86</td>
<td>60.61</td>
<td>57.14</td>
<td>79.00</td>
<td>74.29</td>
<td>78.00</td>
<td><strong>85.71</strong></td>
</tr>
<tr>
<td>Coffee</td>
<td>85.71</td>
<td><strong>100.00</strong></td>
<td>81.43</td>
<td><strong>100.00</strong></td>
<td>92.86</td>
<td>96.43</td>
<td><strong>100.00</strong></td>
</tr>
<tr>
<td>Adiac</td>
<td>65.47</td>
<td>64.45</td>
<td>68.03</td>
<td>72.63</td>
<td>71.61</td>
<td>74.94</td>
<td><strong>76.73</strong></td>
</tr>
</tbody>
</table>
## Results: CPU Time (s)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>DTW</th>
<th>AR</th>
<th>Fisher</th>
<th>RV</th>
<th>FisherRV</th>
<th>GMMRV</th>
<th>SamplingRV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbols</td>
<td>1,318</td>
<td>2,868</td>
<td>2,331</td>
<td>202</td>
<td>236</td>
<td>374</td>
<td>808</td>
</tr>
<tr>
<td>OSULeaf</td>
<td>6,030</td>
<td>1,375</td>
<td>3,264</td>
<td>98</td>
<td>111</td>
<td>186</td>
<td>447</td>
</tr>
<tr>
<td>Oliveoil</td>
<td>295</td>
<td>113</td>
<td>832</td>
<td>11</td>
<td>19</td>
<td>27</td>
<td>43</td>
</tr>
<tr>
<td>Lighting2</td>
<td>918</td>
<td>151</td>
<td>1,143</td>
<td>33</td>
<td>46</td>
<td>61</td>
<td>95</td>
</tr>
<tr>
<td>Beef</td>
<td>107</td>
<td>54</td>
<td>87</td>
<td>10</td>
<td>17</td>
<td>23</td>
<td>40</td>
</tr>
<tr>
<td>Car</td>
<td>679</td>
<td>442</td>
<td>902</td>
<td>27</td>
<td>42</td>
<td>50</td>
<td>84</td>
</tr>
<tr>
<td>Fish</td>
<td>3,353</td>
<td>495</td>
<td>1,998</td>
<td>81</td>
<td>96</td>
<td>159</td>
<td>286</td>
</tr>
<tr>
<td>Coffee</td>
<td>21</td>
<td>25</td>
<td>145</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td>19</td>
</tr>
<tr>
<td>Adiac</td>
<td>550</td>
<td>8131</td>
<td>1,122</td>
<td>201</td>
<td>213</td>
<td>394</td>
<td>699</td>
</tr>
</tbody>
</table>
Performance on increasing length of time series

- Generalization Accuracy
- CPU Time
Multivariate Time Series

- Our kernels can be applicable for
  - multivariate time series
  - of variable length

<table>
<thead>
<tr>
<th>Dataset</th>
<th>dim</th>
<th>length</th>
<th>classes</th>
<th>train</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libras</td>
<td>2</td>
<td>45</td>
<td>15</td>
<td>360</td>
<td>585</td>
</tr>
<tr>
<td>handwritten</td>
<td>3</td>
<td>60-182</td>
<td>20</td>
<td>600</td>
<td>2258</td>
</tr>
<tr>
<td>AUSLAN</td>
<td>22</td>
<td>45-136</td>
<td>95</td>
<td>600</td>
<td>1865</td>
</tr>
</tbody>
</table>
Multivariate Time Series: Results

Generalization Accuracy

CPU Time
Online Kernel Construction

- Reservoir readouts can be trained in an on-line fashion, e.g. using Recursive Least Squares. This enables us to construct and refine reservoir kernels.

\[ w_{opt} = w_0 + k(b - a^T w_0) \]

\[ k = \frac{1}{\lambda + a^T (A_0^T A_0)^{-1} a} (A_0^T A_0)^{-1} a \]

\[ (A_0^T A_0)^{-1} = [\lambda^{-1} I - k a^T] (A_0^T A_0)^{-1} \]

\( \lambda \): forgetting factor
Long Time series

- PEMS-SF is 15 months (Jan. 1st 2008 to Mar. 30th 2009) of daily data from the California Department of Transportation PEMS.

- The data describes the occupancy rate of different car lanes of San Francisco bay area freeways.

- PEMS-SF with 440 time series of length 138,672.
Results

The computational complexity of our kernels is $O(l)$, where $l$ is the length of time series: Scale lineally with the length of time series.

1. RV kernel achieves 86.13% accuracy.

2. The best accuracy reported are 82% 83% for AR, global alignment, spline smoothing and Bag of vectors kernels.
Incremental One Class Learning

1. “Normal data” - apply reservoir model in the sliding window (size \(m\)) in the first \(t\) steps

2. Kernel matrix

\[
K_\sigma(f_i, f_j) = \exp \left\{ -\sigma \cdot L_2(f_i, f_j) \right\},
\]

train a one-class SVM

3. If needed, incrementally create new clusters of models (signal ‘regimes’).
NARMA task

NARMA sequences - orders 10, 20 and 30
## Experiments

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Classes</th>
<th>Precision</th>
<th>Recall</th>
<th>Specificity</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBscan</td>
<td>4</td>
<td>0.6690*</td>
<td>0.7650*</td>
<td>0.8825*</td>
</tr>
<tr>
<td>ARMAX-OCS</td>
<td>5</td>
<td>0.9354*</td>
<td>0.9229*</td>
<td>0.9615*</td>
</tr>
<tr>
<td>RC-OCS</td>
<td>3</td>
<td>0.9637*</td>
<td>0.9615*</td>
<td>0.9808*</td>
</tr>
<tr>
<td>DRC-OCS(Sampling)</td>
<td>3</td>
<td>0.9683*</td>
<td>0.9692*</td>
<td>0.9914</td>
</tr>
<tr>
<td>DRC-OCS</td>
<td>3</td>
<td>0.9861</td>
<td>0.9858</td>
<td>0.9929</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Classes</td>
<td>Precision</td>
<td>Recall</td>
<td>Specificity</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------</td>
<td>-----------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>DBscan</td>
<td>10</td>
<td>0.7629*</td>
<td>0.6842*</td>
<td>0.8018*</td>
</tr>
<tr>
<td>ARMAX-OCS</td>
<td>2</td>
<td>0.4309*</td>
<td>0.4880*</td>
<td>0.7868*</td>
</tr>
<tr>
<td>RC-OCS</td>
<td>6</td>
<td>0.9606*</td>
<td>0.9583*</td>
<td>0.9861</td>
</tr>
<tr>
<td>DRC-OCS (Sampling)</td>
<td>5</td>
<td>0.9617*</td>
<td>0.9726</td>
<td>0.9819*</td>
</tr>
<tr>
<td>DRC-OCS</td>
<td>5</td>
<td><strong>0.9736</strong></td>
<td><strong>0.9731</strong></td>
<td><strong>0.9910</strong></td>
</tr>
</tbody>
</table>

*Note: The metrics are likely subject to statistical significance (e.g., * denotes statistical significance).
## Experiments

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Classes</th>
<th>Precision</th>
<th>Recall</th>
<th>Specificity</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBscan</td>
<td>14</td>
<td>0.8742*</td>
<td>0.7561*</td>
<td>0.9253*</td>
</tr>
<tr>
<td>ARMAX-OCS</td>
<td>5</td>
<td>0.9914*</td>
<td>0.9923</td>
<td>0.9984</td>
</tr>
<tr>
<td>RC-OCS</td>
<td>9</td>
<td>0.9182*</td>
<td>0.8788*</td>
<td>0.9596*</td>
</tr>
<tr>
<td>DRC-OCS(Sampling)</td>
<td>7</td>
<td>0.9940</td>
<td>0.9949*</td>
<td>0.9988</td>
</tr>
<tr>
<td>DRC-OCS</td>
<td>10</td>
<td>0.9931</td>
<td>0.9931</td>
<td>0.9977</td>
</tr>
</tbody>
</table>
## Experiments

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Classes</th>
<th>Precision</th>
<th>Recall</th>
<th>Specificity</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBscan</td>
<td>61</td>
<td>0.8019*</td>
<td>0.7326*</td>
<td>0.8654*</td>
</tr>
<tr>
<td>ARMAX-OCS</td>
<td>57</td>
<td>0.7826*</td>
<td>0.7419*</td>
<td>0.8237*</td>
</tr>
<tr>
<td>RC-OCS</td>
<td>44</td>
<td>0.8913*</td>
<td>0.8942*</td>
<td>0.9263*</td>
</tr>
<tr>
<td>DRC-OCS(Sampling)</td>
<td>39</td>
<td>0.9219*</td>
<td>0.9310*</td>
<td>0.9513*</td>
</tr>
<tr>
<td>DRC-OCS</td>
<td>48</td>
<td><strong>0.9538</strong></td>
<td><strong>0.9640</strong></td>
<td><strong>0.9871</strong></td>
</tr>
</tbody>
</table>

Barcelona Water (32 classes)
Further reading