Change point detection in Python

AMF seminar

Charles Truong[1]

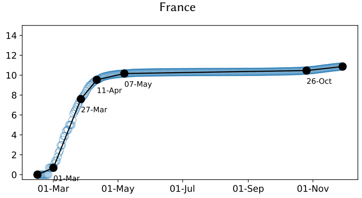
[1]Centre Borelli
Université Paris-Saclay
ENS Paris-Saclay, CNRS

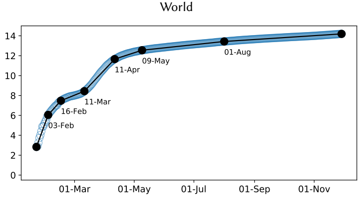
Wednesday 6th January

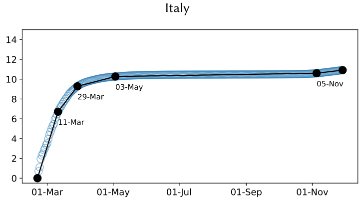
Introduction

- Change point detection is a common task when dealing with non-stationary time series.
- Application example: study of COVID-19 infection curve [Jiang et al., 2020].

- Data from “Our World in Data” (ourworldindata.org).
- Cumulative reported deaths in log-scale.
- Piecewise linear trends (linear spline smoothing with optimal knot selection).
- The slope gives the growth rate (“log-return”).







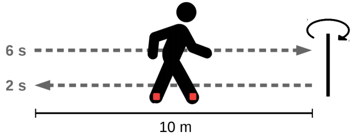
Introduction

- Change point detection is a common task when dealing with non-stationary time series.
- Application example: automatic diagnosis of neurologically impaired patients [Truong et al., 2019a].

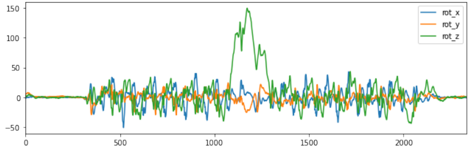
Healthy and pathological subjects underwent a fixed protocol:

- standing still,
- walking 10m,
- turning around,
- walking back,
- standing still.

Protocol schema



Angular velocity (lower back sensor, sampling 100 Hz)



- Can also be applied to finance, industrial monitoring, public health monitoring, etc. [Truong et al., 2020].

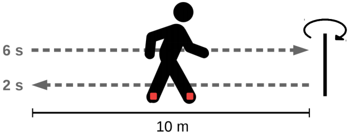
Introduction

- Change point detection is a common task when dealing with non-stationary time series.
- Application example: automatic diagnosis of neurologically impaired patients [Truong et al., 2019a].

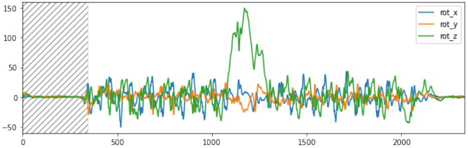
Healthy and pathological subjects underwent a fixed protocol:

- **standing still**,
- walking 10m,
- turning around,
- walking back,
- standing still.

Protocol schema



Angular velocity (lower back sensor, sampling 100 Hz)



- Can also be applied to finance, industrial monitoring, public health monitoring, etc. [Truong et al., 2020].

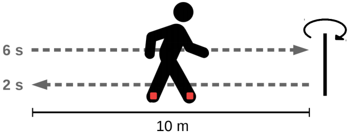
Introduction

- Change point detection is a common task when dealing with non-stationary time series.
- Application example: automatic diagnosis of neurologically impaired patients [Truong et al., 2019a].

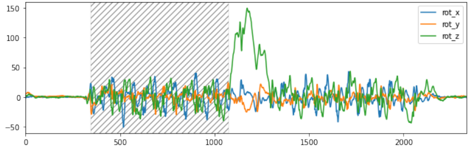
Healthy and pathological subjects underwent a fixed protocol:

- standing still,
- **walking 10m**,
- turning around,
- walking back,
- standing still.

Protocol schema



Angular velocity (lower back sensor, sampling 100 Hz)



- Can also be applied to finance, industrial monitoring, public health monitoring, etc. [Truong et al., 2020].

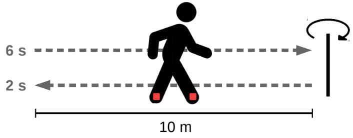
Introduction

- Change point detection is a common task when dealing with non-stationary time series.
- Application example: automatic diagnosis of neurologically impaired patients [Truong et al., 2019a].

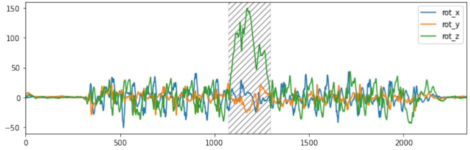
Healthy and pathological subjects underwent a fixed protocol:

- standing still,
- walking 10m,
- **turning around**,
- walking back,
- standing still.

Protocol schema



Angular velocity (lower back sensor, sampling 100 Hz)



- Can also be applied to finance, industrial monitoring, public health monitoring, etc. [Truong et al., 2020].

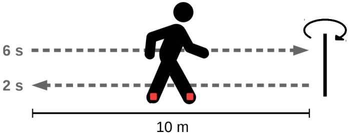
Introduction

- Change point detection is a common task when dealing with non-stationary time series.
- Application example: automatic diagnosis of neurologically impaired patients [Truong et al., 2019a].

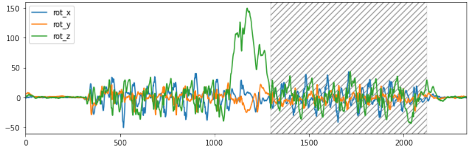
Healthy and pathological subjects underwent a fixed protocol:

- standing still,
- walking 10m,
- turning around,
- **walking back**,
- standing still.

Protocol schema



Angular velocity (lower back sensor, sampling 100 Hz)



- Can also be applied to finance, industrial monitoring, public health monitoring, etc. [Truong et al., 2020].

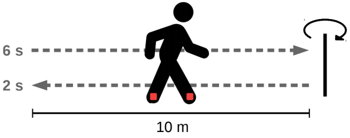
Introduction

- Change point detection is a common task when dealing with non-stationary time series.
- Application example: automatic diagnosis of neurologically impaired patients [Truong et al., 2019a].

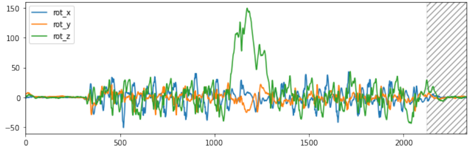
Healthy and pathological subjects underwent a fixed protocol:

- standing still,
- walking 10m,
- turning around,
- walking back,
- **standing still**.

Protocol schema



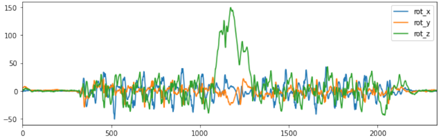
Angular velocity (lower back sensor, sampling 100 Hz)



- Can also be applied to finance, industrial monitoring, public health monitoring, etc. [Truong et al., 2020].

What is change point detection?

- Change point detection consists in finding the temporal boundaries between homogeneous time periods.
- Informally: “multivariate signal $\longrightarrow$ list of change point indexes”

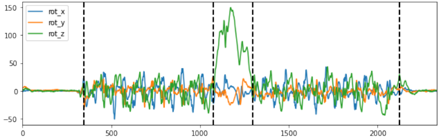


x	y	z
8.139	8.337	-21.055
6.964	9.881	-20.693
4.317	9.993	-19.309
1.752	8.950	-16.941
-0.305	7.356	-13.143
-2.320	7.384	-7.361
-3.312	8.467	-2.530
-3.697	10.891	2.523
-2.622	13.363	5.863
-2.728	11.761	4.473

$$\longrightarrow [t_1, t_2, t_3, \ldots]$$

What is change point detection?

- Change point detection consists in finding the temporal boundaries between homogeneous time periods.
- Informally: "multivariate signal $\longrightarrow$ list of change point indexes"



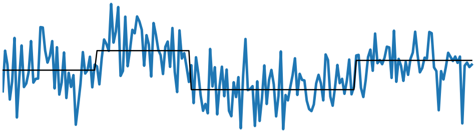
x	y	z
8.139	8.337	-21.055
6.964	9.881	-20.693
4.317	9.993	-19.309
1.752	8.950	-16.941
-0.305	7.356	-13.143
-2.320	7.384	-7.361
-3.312	8.467	-2.530
-3.697	10.891	2.523
-2.622	13.363	5.863
-2.728	11.761	4.473

$$\longrightarrow [t_1, t_2, t_3, \ldots]$$

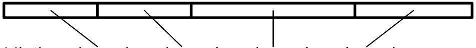
Table of contents

General principle

How to choose a segmentation?



$\mathcal{T} = \{t_1, t_2, t_3\}$



$V(\mathcal{T}) = c(y_{0..t_1}) + c(y_{t_1..t_2}) + c(y_{t_2..t_3}) + c(y_{t_3..T})$

The "best segmentation" is the minimizer, denoted $\widehat{\mathcal{T}}$, of a criterion $V(\mathcal{T})$:

$$V(\mathcal{T}) := \sum_{k=0}^{K} c(y_{t_k..t_{k+1}}).$$

Cost example: $c(y) = \sum_t (y_t - \bar{y})^2$.

Problem 1.
Fixed number K of change points:

$$\widehat{\mathcal{T}} := \arg\min_{\mathcal{T}} \; V(\mathcal{T}) \quad \text{s.t. } |\mathcal{T}| = K.$$

Problem 2.
Unknown number of change points:

$$\widehat{\mathcal{T}} := \arg\min_{\mathcal{T}} \; V(\mathcal{T}) \; + \; \text{pen}(\mathcal{T})$$

where $\text{pen}(\mathcal{T})$ measures the complexity of a segmentation $\mathcal{T}$.

General principle

Detection methods are the combination of three elements [Truong et al., 2020].

Cost function | Search method | Constraint

Criterion $V(\mathcal{T})$ to minimize: $V(\mathcal{T}) := \sum_{k=0}^{K} c(y_{t_k..t_{k+1}})$.

Problem 1.
Fixed number K of change points:

$$\widehat{\mathcal{T}} := \underset{\mathcal{T}}{\arg\min}\ V(\mathcal{T}) \text{ s.t. } |\mathcal{T}| = K.$$

Problem 2.
Unknown number of change points:

$$\widehat{\mathcal{T}} := \underset{\mathcal{T}}{\arg\min}\ V(\mathcal{T}) + \operatorname{pen}(\mathcal{T})$$

where $\operatorname{pen}(\mathcal{T})$ measures the complexity of a segmentation $\mathcal{T}$.

General principle

- A modular architecture.

[319]:

```python
import ruptures as rpt

signal = get_signal(...)  # user defined
```

First import and data loading.

[329]:

```python
# cost function
c = rpt.costs.CostL2()
```

Choosing the cost function. Here, $c(y) = \sum_t (y_t - \bar{y})^2$.

[330]:

```python
# search method
algo = rpt.Binseg(jump=5, min_size=10, custom_cost=c)
```

Choosing the search method. Here, binary segmentation.

[331]:

```python
# fit algo
algo.fit(signal)
```

Fitting the algorithm.

[332]:

```python
# predict change points
# fixed number of changes
bkps = algo.predict(n_bkps=10)
# or penalized detection
bkps = algo.predict(pen=50)
```

Choosing the constraint. Then detecting the change points (“predict”).

[333]:

```python
from ruptures.metrics import hausdorff

error = hausdorff(true_bkps, bkps)
```

Measuring the detection accuracy.

A discrete optimization problem

Minimize the sum of cost over all segmentations:

$$\min_{t_1,t_2,\dots,t_K} \sum_{k=0}^{K} c(y_{t_k..t_{k+1}}).$$

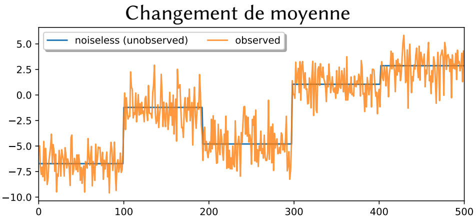
or

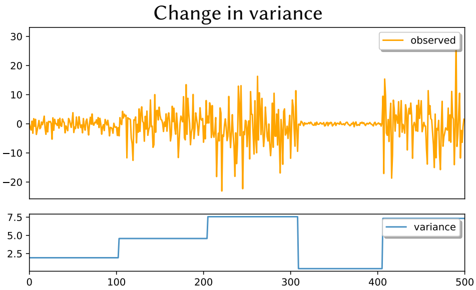
$$\min_{t_1,t_2,\dots,t_K} \sum_{k=0}^{K} c(y_{t_k..t_{k+1}}) + \beta K.$$

- A naive implementation is prohibitive ($\binom{T}{K}$ segmentations).
- The problem is solved recursively using Bellman's dynamic programming.
- For most cost functions, the complexity is $\mathcal{O}(T^2)$ in operations and $\mathcal{O}(T)$.
- Heuristics to approximately solve this problem exist: binary segmentation (with variants) and window-sliding. Complexity in $\mathcal{O}(T)$.

Table of contents

Change in mean and variance (1-D)





Cost function:

$$c(y_{a..b}) = \sum_{t=a}^{b-1} (y_t - \bar{y}_{a..b})^2$$

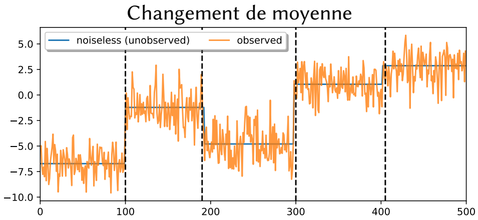
where $\bar{y}_{a..b}$ is the empirical mean of $y_{a..b}$.

Cost function:

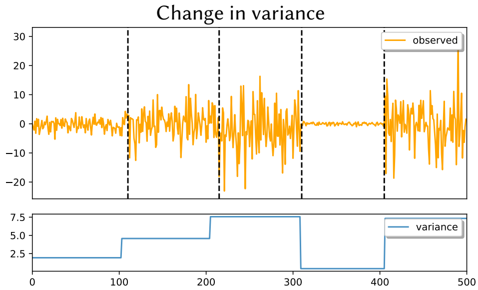
$$c(y_{a..b}) = (b - a) \log(\hat{\sigma}_{a..b})$$

where $\hat{\sigma}_{a..b}$ the empirical standard-deviation $y_{a..b}$.

Change in mean and variance (1-D)



(Error: 3 samples.)



(Error: 10 samples.)

Cost function:

$$c(y_{a..b}) = \sum_{t=a}^{b-1} (y_t - \bar{y}_{a..b})^2$$

where $\bar{y}_{a..b}$ is the empirical mean of $y_{a..b}$.

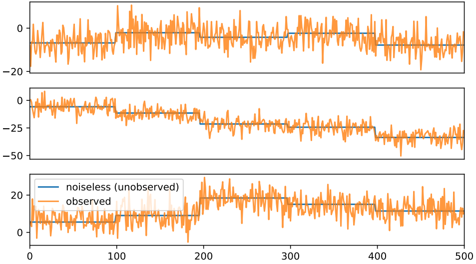
Cost function:

$$c(y_{a..b}) = (b-a)\log(\hat{\sigma}_{a..b})$$

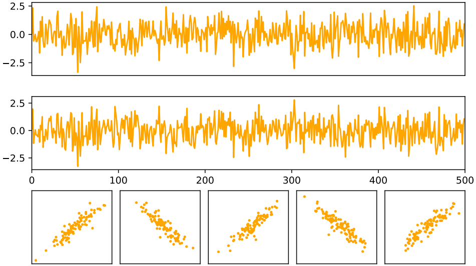
where $\hat{\sigma}_{a..b}$ the empirical standard-deviation $y_{a..b}$.

Change in mean and variance (n-D)

Change in mean



Change in variance/covariance



Cost function:

$$c(y_{a..b}) = \sum_{t=a}^{b-1} \|y_t - \bar{y}_{a..b}\|^2$$

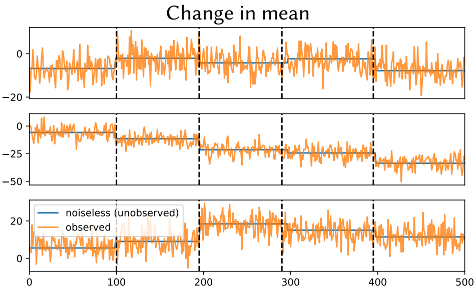
where $\bar{y}_{a..b}$ is the empirical mean of $y_{a..b}$.

Cost function:

$$c(y_{a..b}) = (b-a) \log \det \hat{\Sigma}_{a..b}$$

where $\hat{\sigma}_{a..b}$ is the empirical covariance matrix of $y_{a..b}$.

Change in mean and variance (n-D)

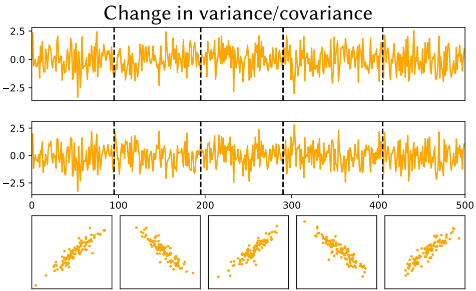


(Error: 7 samples.)

Cost function:

$$c(y_{a..b}) = \sum_{t=a}^{b-1} \|y_t - \bar{y}_{a..b}\|^2$$

where $\bar{y}_{a..b}$ is the empirical mean of $y_{a..b}$.



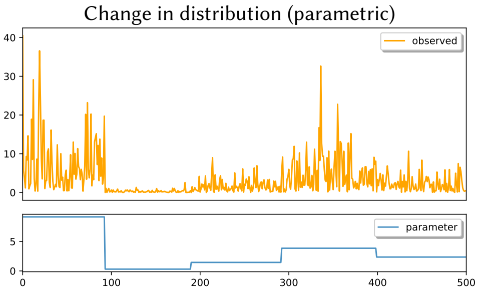
(Error: 3 samples.)

Cost function:

$$c(y_{a..b}) = (b-a) \log \det \hat{\Sigma}_{a..b}$$

where $\hat{\sigma}_{a..b}$ is the empirical covariance matrix of $y_{a..b}$.

Change in distribution (parametric)



Cost function:

$$c(y_{a..b}) = -\max_{\theta} \log f_{\theta}(y_{a..b})$$

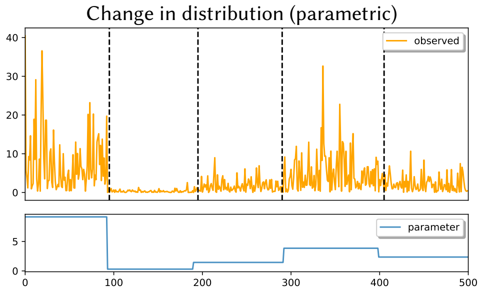
where f_θ is the density of the chosen distribution, parametrized by θ.

Important fact. The estimated change points converge to the true changes.

[Lavielle, 1999, Detection of multiples changes in a sequence of dependant variables. Stochastic Processes and Their Applications, 83(1), 79–102.]

- Not necessarily i.i.d. observations.
- Can be strongly dependant (but stationary).
- …

Change in distribution (parametric)



(exponential distribution, Error: 6 samples.)

Cost function:

$$c(y_{a..b}) = -\max_{\theta} \log f_{\theta}(y_{a..b})$$

where f_θ is the density of the chosen distribution, parametrized by θ.

Important fact. The estimated change points converge to the true changes.

[Lavielle, 1999, Detection of multiples changes in a sequence of dependant variables. Stochastic Processes and Their Applications, 83(1), 79–102.]

- Not necessarily i.i.d. observations.
- Can be strongly dependant (but stationary).
- ...

Change in distribution (non-parametric)

When the underlying distribution is unknown:

- [Arlot et al., 2019, A kernel multiple change-point algorithm via model selection. Journal of Machine Learning Research, 20(162), 1–56.]
- [Matteson and James, 2014, A nonparametric approach for multiple change point analysis of multivariate data. Journal of the American Statistical Association, 109(505), 334–345.]
- [Ross and Adams, 2012, Two nonparametric control charts for detecting arbitrary distribution changes. Journal of Quality Technology, 44(2), 102–117.]

The kernel approach is particularly interesting because it can deal with non-numerical data: symbolic signals, texts, functional time series,...

General principle:

- The signal is mapped to a high-dimensional space: $y_t \longrightarrow \phi(y_t)$.
- Detection of change in the mean of the $\phi(y_t)$.

Cost function:

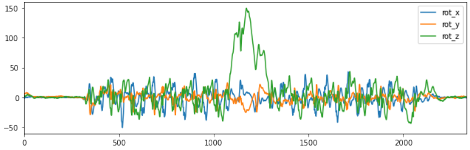
$$c(y_{a..b}) = \sum_{t=a}^{b-1} \|\phi(y_t) - \bar{\mu}\|_{\mathcal{H}}^2$$

where $\bar{\mu}$ is the empirical mean of $\{\phi(y_t)\}_{a..b}$. (Computed using the kernel trick.)

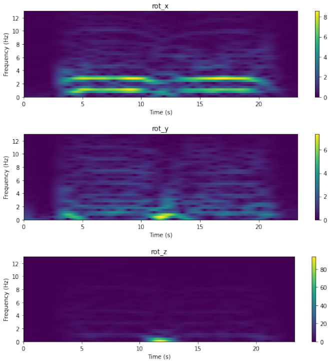
Gait analysis

- To simplify the detection task, the signal is transformed (here, short-term Fourier transform).
- Then mean-shifts are detected.

Angular velocity (lower back sensor, sampling 100 Hz)



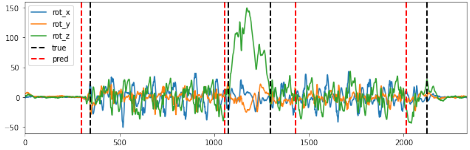
Signal transformation



Gait analysis

- To simplify the detection task, the signal is transformed (here, short-term Fourier transform).
- Then mean-shifts are detected.

Angular velocity (lower back sensor, sampling 100 Hz)



[246]:

```python
# cost for multivariate mean-shifts
cost = rpt.costs.CostL2()
# search method: binary segmentation
algo = rpt.Binseg(custom_cost=cost)
# fit
algo.fit(X)
# predict
prediction = algo.predict(4)
# error
error = hausdorff(true_bkps, prediction)/100
# print results
print(f"True change points:\t\t{true_bkps}")
print(f"Predicted change points:\t{prediction}")
print(f"Max error: {error:.2f} sec")
```

```
True change points:          [347, 1075, 1297, 2123, 2332]
Predicted change points:     [300, 1055, 1430, 2015, 2332]
Max error: 1.33 sec
```

Signal transformation

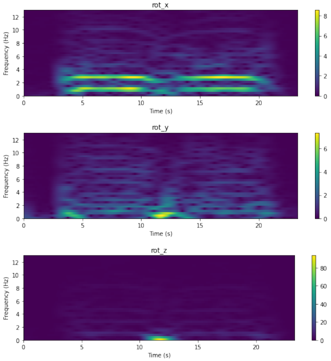


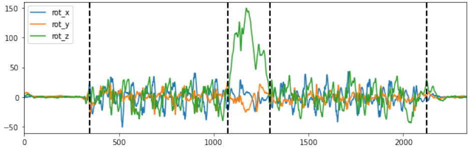
Table of contents

Supervised change point detection

General principle

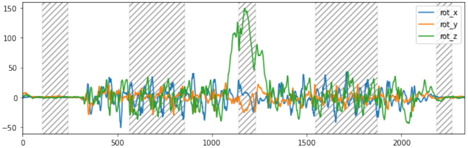
- How to integrate expert knowledge to calibrate the change point detection? [Truong et al., 2019b]
- The expert provides the target segmentation: either full or partial label.
- Labels are hard to collect. The easier for the clinicians, the better.

Full label



The exact change point locations are provided.

Partial label



Only homogeneous periods (hatched areas) are provided (weakly supervised).

Learn the representation

Labels are transformed into constraints. Intuitively, the problem is:

- Learn a transformation Ψ such that
  $d(\Psi(x_t), \Psi(x_s)) \leq u$ if x_t and x_s similar
  $d(\Psi(x_t), \Psi(x_s)) \geq l$ if x_t and x_s dissimilar

($u > 0$ and $l > 0$)

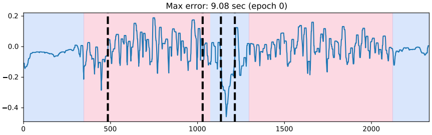
Two samples are *similar* if they belong to the same regime.
Two samples are *dissimilar* if they belong to consecutive regimes.

This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

[256]:
```
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Epoch by epoch (epoch 0)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

Learn the representation

Labels are transformed into constraints. Intuitively, the problem is:

- Learn a transformation Ψ such that
  $d(\Psi(x_t), \Psi(x_s)) \leq u$ if x_t and x_s similar
  $d(\Psi(x_t), \Psi(x_s)) \geq l$ if x_t and x_s dissimilar

($u > 0$ and $l > 0$)

Two samples are *similar* if they belong to the same regime.
Two samples are *dissimilar* if they belong to consecutive regimes.

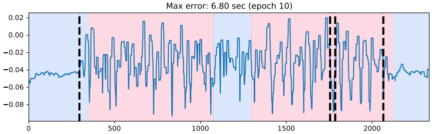
This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

[256]:

```
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Epoch by epoch (epoch 10)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

Learn the representation

Labels are transformed into constraints. Intuitively, the problem is:

- Learn a transformation Ψ such that
  $d(\Psi(x_t), \Psi(x_s)) \leq u$ if x_t and x_s similar
  $d(\Psi(x_t), \Psi(x_s)) \geq l$ if x_t and x_s dissimilar

($u > 0$ and $l > 0$)

Two samples are *similar* if they belong to the same regime.
Two samples are *dissimilar* if they belong to consecutive regimes.

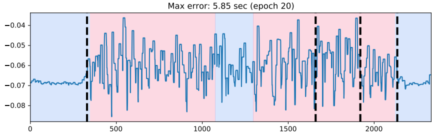
This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

[256]:

```python
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Epoch by epoch (epoch 20)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

Learn the representation

Labels are transformed into constraints. Intuitively, the problem is:

- Learn a transformation Ψ such that
  $d(\Psi(x_t), \Psi(x_s)) \leq u$ if x_t and x_s similar
  $d(\Psi(x_t), \Psi(x_s)) \geq l$ if x_t and x_s dissimilar

($u > 0$ and $l > 0$)

Two samples are *similar* if they belong to the same regime.
Two samples are *dissimilar* if they belong to consecutive regimes.

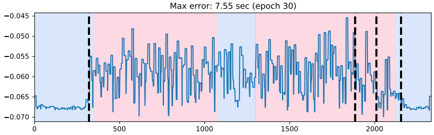
This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

[256]:

```
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Epoch by epoch (epoch 30)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

Learn the representation

Labels are transformed into constraints. Intuitively, the problem is:

- Learn a transformation Ψ such that

$$d(\Psi(x_t), \Psi(x_s)) \leq u \text{ if } x_t \text{ and } x_s \text{ similar}$$

$$d(\Psi(x_t), \Psi(x_s)) \geq l \text{ if } x_t \text{ and } x_s \text{ dissimilar}$$

$(u > 0$ and $l > 0)$

Two samples are *similar* if they belong to the same regime.
Two samples are *dissimilar* if they belong to consecutive regimes.

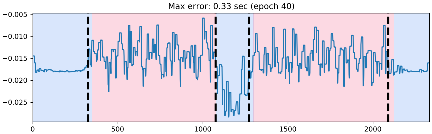
This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

[256]:

```
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Epoch by epoch (epoch 40)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

Learn the representation

Labels are transformed into constraints. Intuitively, the problem is:

- Learn a transformation Ψ such that
  $d(\Psi(x_t), \Psi(x_s)) \leq u$ if x_t and x_s similar
  $d(\Psi(x_t), \Psi(x_s)) \geq l$ if x_t and x_s dissimilar

($u > 0$ and $l > 0$)

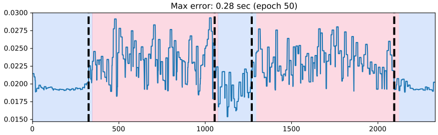
Two samples are *similar* if they belong to the same regime.
Two samples are *dissimilar* if they belong to consecutive regimes.

This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

[256]:
```
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Epoch by epoch (epoch 50)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

Learn the representation

Labels are transformed into constraints. Intuitively, the problem is:

- Learn a transformation Ψ such that
  $d(\Psi(x_t), \Psi(x_s)) \leq u$ if x_t and x_s similar
  $d(\Psi(x_t), \Psi(x_s)) \geq l$ if x_t and x_s dissimilar

($u > 0$ and $l > 0$)

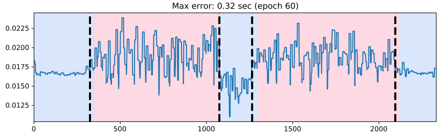
Two samples are *similar* if they belong to the same regime.
Two samples are *dissimilar* if they belong to consecutive regimes.

This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

[256]:
```
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Epoch by epoch (epoch 60)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

Learn the representation

Labels are transformed into constraints. Intuitively, the problem is:

- Learn a transformation Ψ such that
  $d(\Psi(x_t), \Psi(x_s)) \leq u$ if x_t and x_s similar
  $d(\Psi(x_t), \Psi(x_s)) \geq l$ if x_t and x_s dissimilar

($u > 0$ and $l > 0$)

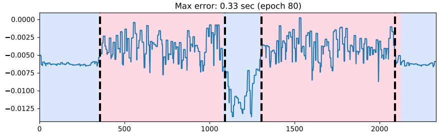
Two samples are *similar* if they belong to the same regime.
Two samples are *dissimilar* if they belong to consecutive regimes.

This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

[256]:
```
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Epoch by epoch (epoch 80)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

Learn the representation

Labels are transformed into constraints. Intuitively, the problem is:

- Learn a transformation Ψ such that
  $d(\Psi(x_t), \Psi(x_s)) \leq u$ if x_t and x_s similar
  $d(\Psi(x_t), \Psi(x_s)) \geq l$ if x_t and x_s dissimilar

($u > 0$ and $l > 0$)

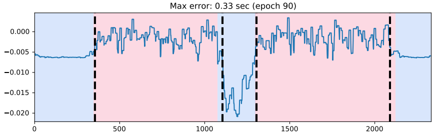
Two samples are *similar* if they belong to the same regime.
Two samples are *dissimilar* if they belong to consecutive regimes.

This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

[256]:
```
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Epoch by epoch (epoch 90)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

Learn the representation

Labels are transformed into constraints. Intuitively, the problem is:

- Learn a transformation Ψ such that

$$d(\Psi(x_t), \Psi(x_s)) \leq u \text{ if } x_t \text{ and } x_s \text{ similar}$$
$$d(\Psi(x_t), \Psi(x_s)) \geq l \text{ if } x_t \text{ and } x_s \text{ dissimilar}$$

($u > 0$ and $l > 0$)

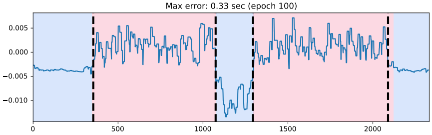
Two samples are *similar* if they belong to the same regime.
Two samples are *dissimilar* if they belong to consecutive regimes.

This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

[256]:
```python
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Epoch by epoch (epoch 100)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

Conclusion

- Code for those experiments will be available on my GitHub `github.com/deepcharles`.
- New methods are frequently implemented in ruptures.
- Extensions to graph/network data soon.
- Differentiable dynamic programming for end-to-end unsupervised representation learning.

References

Arlot, S., Celisse, A., and Harchaoui, Z. (2019).
A kernel multiple change-point algorithm via model selection.
Journal of Machine Learning Research, 20(162):1–56.

Jiang, F., Zhao, Z., and Shao, X. (2020).
Time series analysis of COVID-19 infection curve: a change-point perspective.
Journal of Econometrics.

Lavielle, M. (1999).
Detection of multiples changes in a sequence of dependant variables.
Stochastic Processes and their Applications, 83(1):79–102.

Matteson, D. S. and James, N. A. (2014).
A nonparametric approach for multiple change point analysis of multivariate data.
Journal of the American Statistical Association, 109(505):334–345.

Ross, G. J. and Adams, N. M. (2012).
Two nonparametric control charts for detecting arbitrary distribution changes.
Journal of Quality Technology, 44(2):102–117.

References

Truong, C., Barrois-Müller, R., Moreau, T., Provost, C., Vienne-Jumeau, A., Moreau, A., Vidal, P.-P., Vayatis, N., Buffat, S., Yelnik, A., Ricard, D., and Oudre, L. (2019a).
A data set for the study of human locomotion with inertial measurements units.
Image Processing On Line, 9.

Truong, C., Oudre, L., and Vayatis, N. (2019b).
Supervised kernel change point detection with partial annotations.
In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5, Brighton, UK.

Truong, C., Oudre, L., and Vayatis, N. (2020).
Selective review of offline change point detection methods.
Signal Processing, 167.