Fast and Effective Time Series Classification

Daniel F. Schmidt
Work done with Angus Dempster and Geoffrey I Webb

Department of Data Science and AI
Monash University

University of Helsinki
Feb 26, 2024

Outline

Outline

Problem Description

- Time series classification is a specialised type of classification problem
- We have vector of n targets $\mathbf{y} = (y_1, \ldots, y_n)$
  - These are L-class labels, $y_i \in \{1, 2, \ldots, L\}$
- The raw features associated with each target are time series
  - There is of course no specific need for these to be indexed by time
- For simplicity of exposition we will assume
  - There is only a single time series per individual
  - The time series all the same length, say m

  $\Longrightarrow$ Denote these by $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,m})$
- The aim is to classify new individuals based on their time series
- Typical applications:
  - Classification of land patches on satellite imagery
  - Prediction of epileptic episodes from neural measurements

State of the Art Pre-2020

- There are lots of time series classification techniques, e.g.,
- Benchmarking is frequently done on the UCR time series repository
  - 112+ problems of varying size from diverse domains
- Some important techniques (classification accuracy in brackets)
  - Nearest neighbour with dynamic time warping [0.768]
  - TS-CHIEF (proximity forests) [0.876]
  - Deep learning (InceptionTime architecture) [0.872]
  - HVE-COTE2 (hierarchical ensemble classifier) [0.891]
- Unfortunately HIVE-COTE2 is very slow
- Rough wall-clock timings (for calibration)
  - InceptionTime took c. 4 days to run (on GPUs)
  - TS-CHIEF takes c. 2 weeks
  - HIVE-COTE2 took c. 3 weeks
- As a general rule, fast methods tended to perform poorly

State of the Art Pre-2020

- There are lots of time series classification techniques, e.g.,
- Benchmarking is frequently done on the UCR time series repository
  - 112+ problems of varying size from diverse domains
- Some important techniques (classification accuracy in brackets)
  - Nearest neighbour with dynamic time warping [0.768]
  - TS-CHIEF (proximity forests) [0.876]
  - Deep learning (InceptionTime architecture) [0.872]
  - HVE-COTE2 (hierarchical ensemble classifier) [0.891]
- Unfortunately HIVE-COTE2 is very slow
- Rough wall-clock timings (for calibration)
  - InceptionTime took c. 4 days to run (on GPUs)
  - TS-CHIEF takes c. 2 weeks
  - HIVE-COTE2 took c. 3 weeks
- As a general rule, fast methods tended to perform poorly

Outline

Rocket (1)

- In 2019 my collaborators introduced Rocket
  - A shallow (single layer) random convolutional model
- K randomly generated convolutional kernels; each with
  - Random number of coefficients (7, 9, or 11)
  - Coefficients randomly sampled from normal
  - Random dilation (stretching) for multi-scale analysis
- For time series i, for each kernel $\mathbf{c}_j$ compute convolution

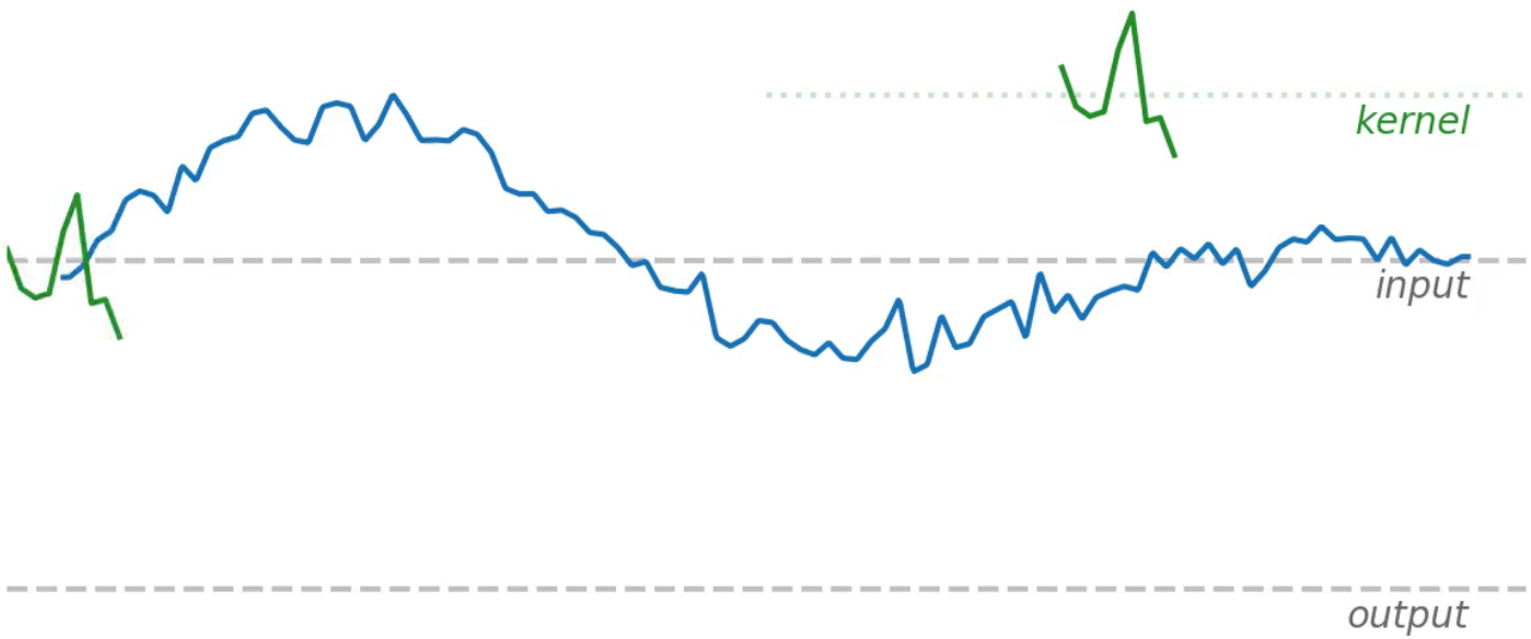
$$\mathbf{v}_{i,j} = \mathbf{x}_i * \mathbf{c}_j$$

- Extract pooling features for each kernel/time series pair
  - Maximum value $\max_k [\mathbf{v}_{i,j}]_k$
  - The proportion of positive values

$$\sum_k I\left\{[\mathbf{v}_{i,j}]_k > b_j\right\}$$

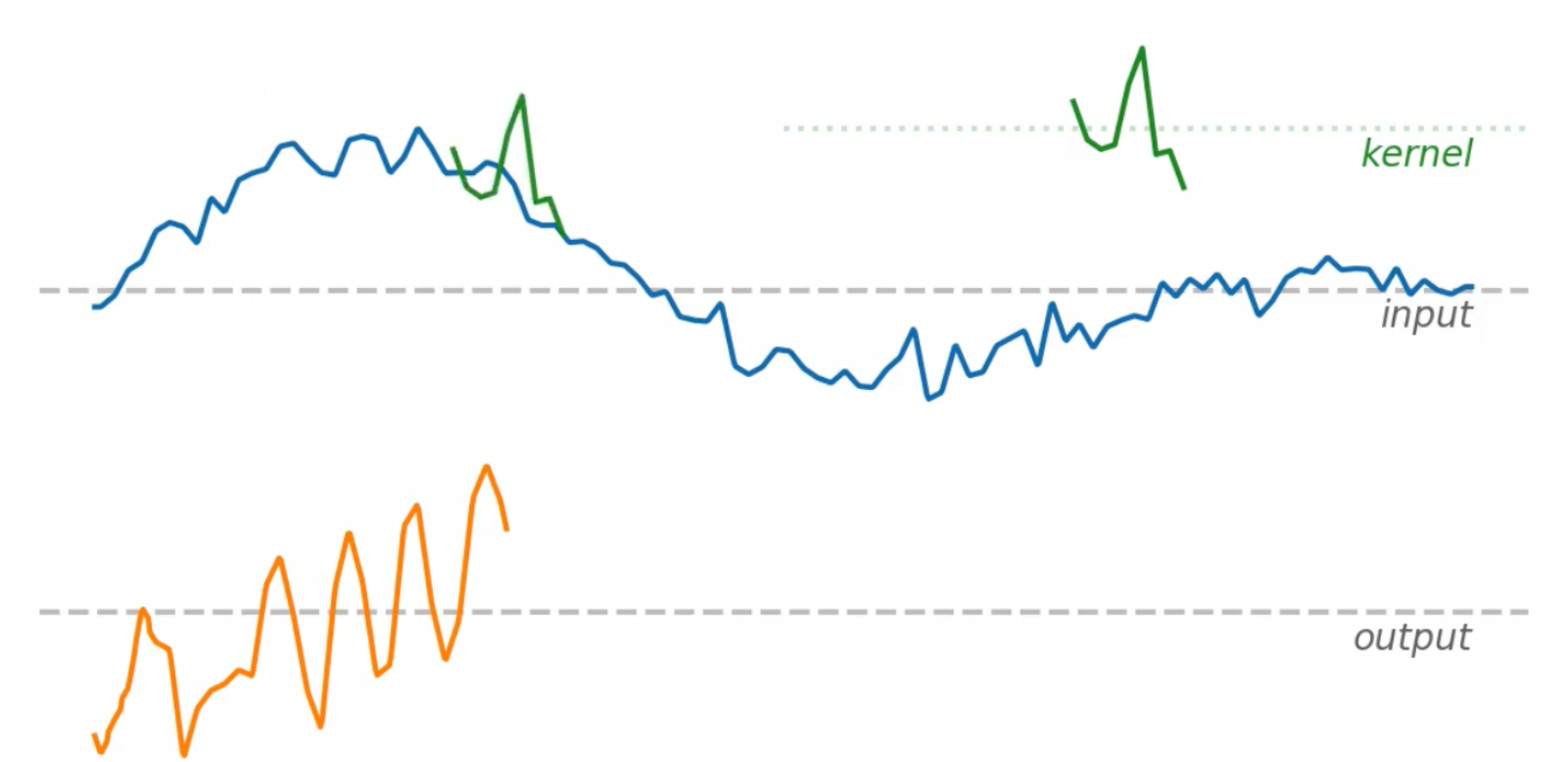
where the biases b_j are also randomly generated

Convolution (1)



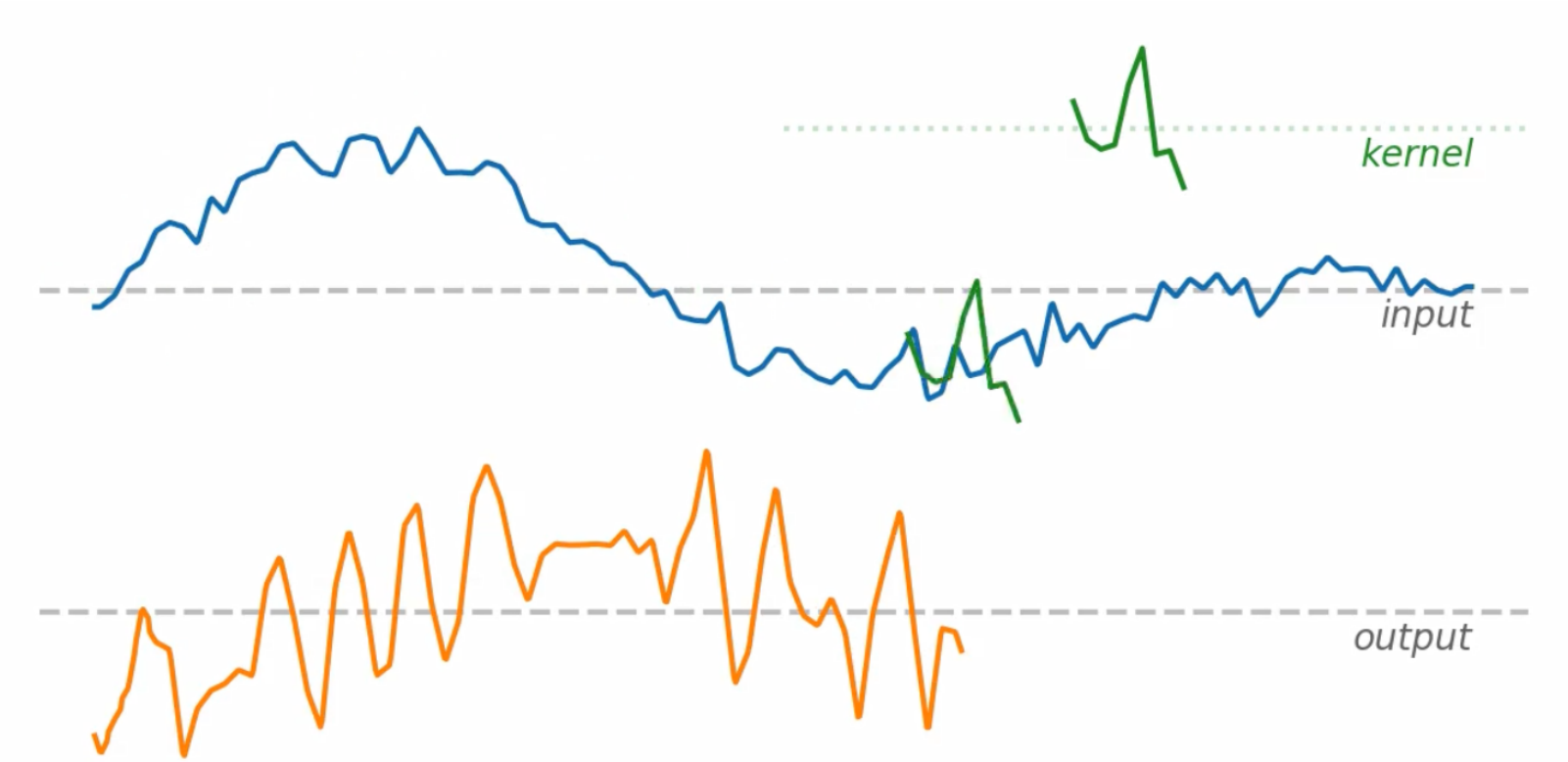
Convolving a moving average filter with a time series involves computing the inner product between each the series and the filter coefficients at each time point. The larger the response, the more correlated the series is with the kernel at that point.

Convolution (2)



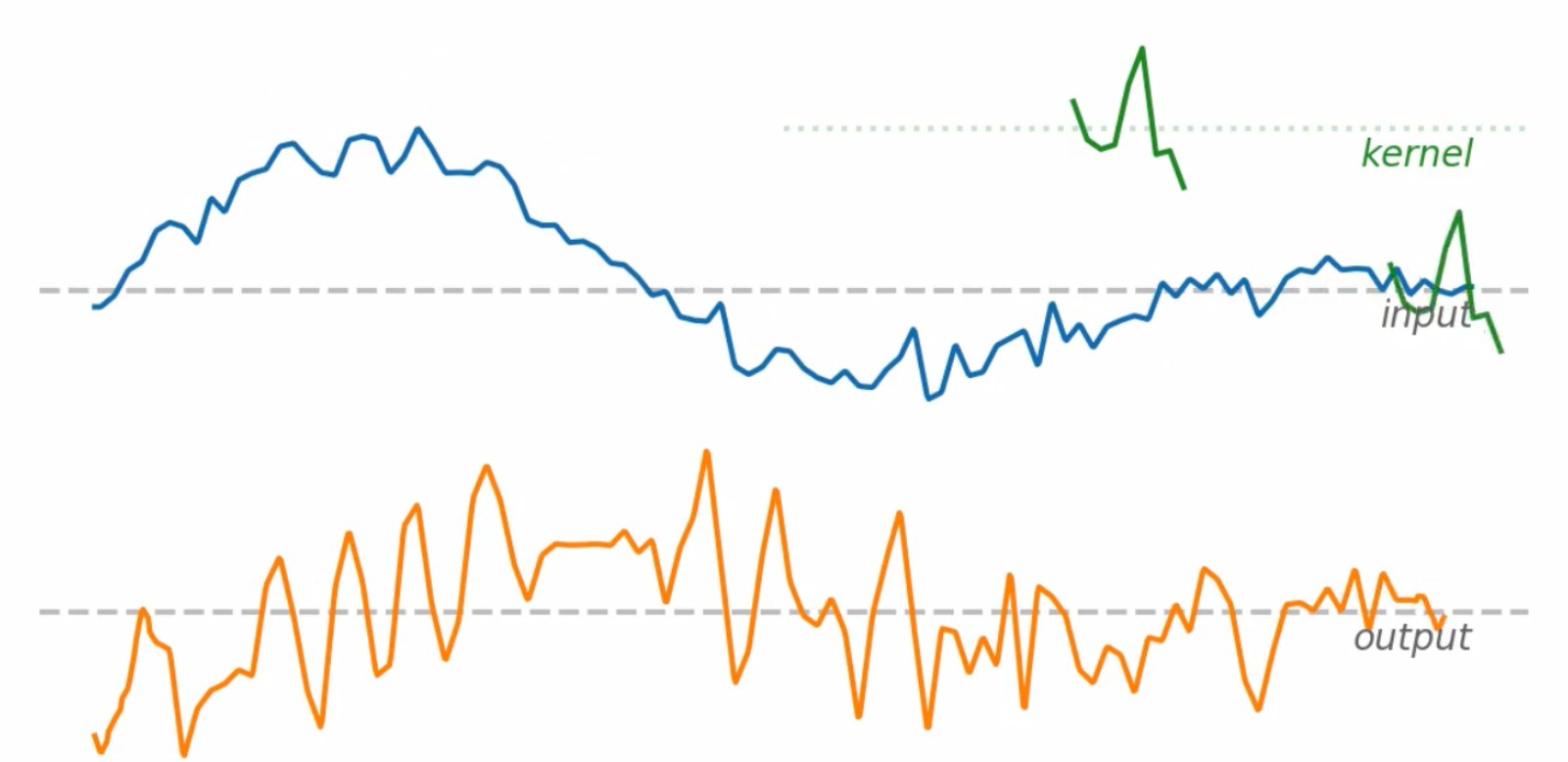
Convolving a moving average filter with a time series involves computing the inner product between each the series and the filter coefficients at each time point. The larger the response, the more correlated the series is with the kernel at that point.

Convolution (3)



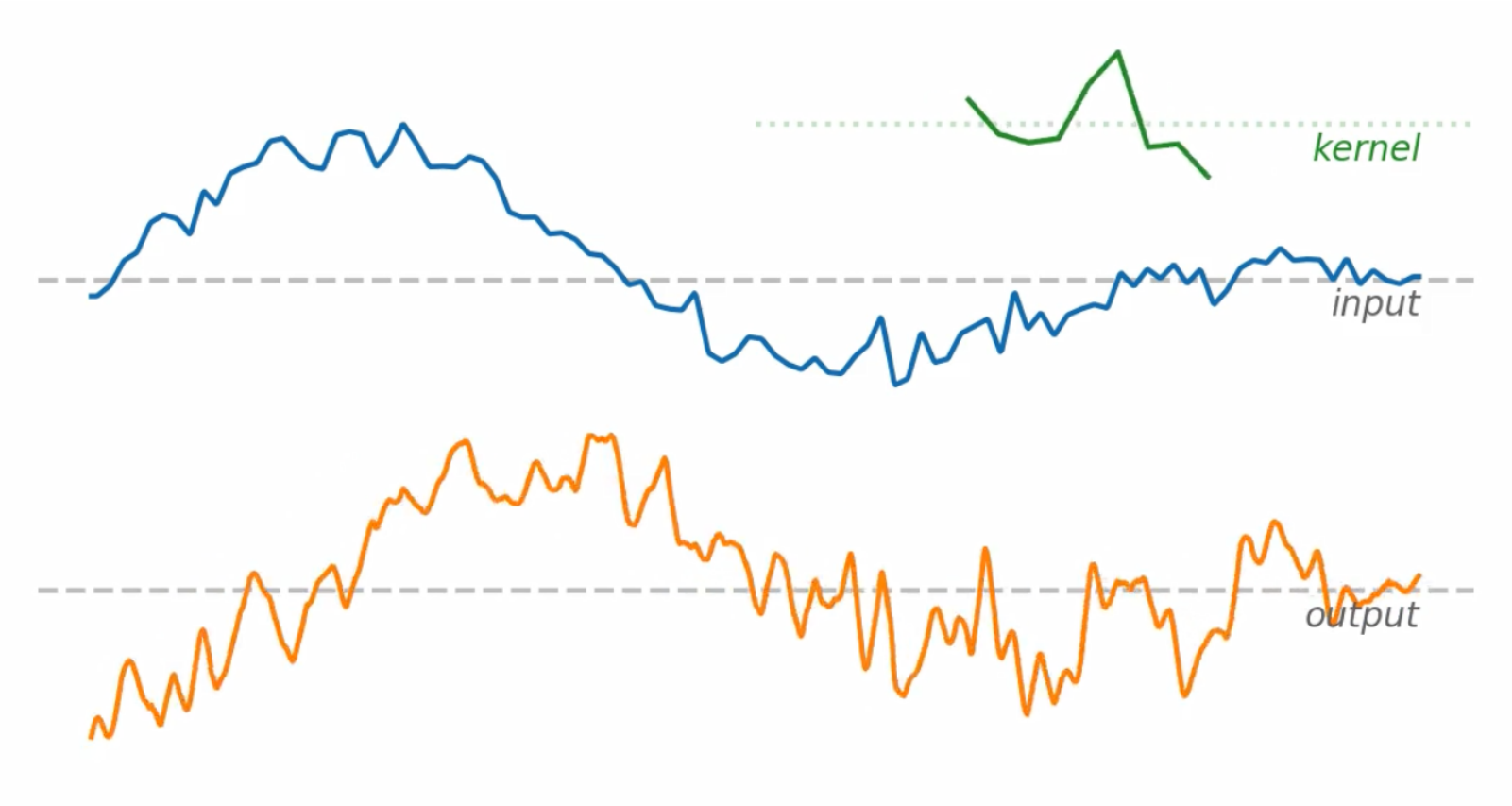
Convolving a moving average filter with a time series involves computing the inner product between each the series and the filter coefficients at each time point. The larger the response, the more correlated the series is with the kernel at that point.

Convolution (4)



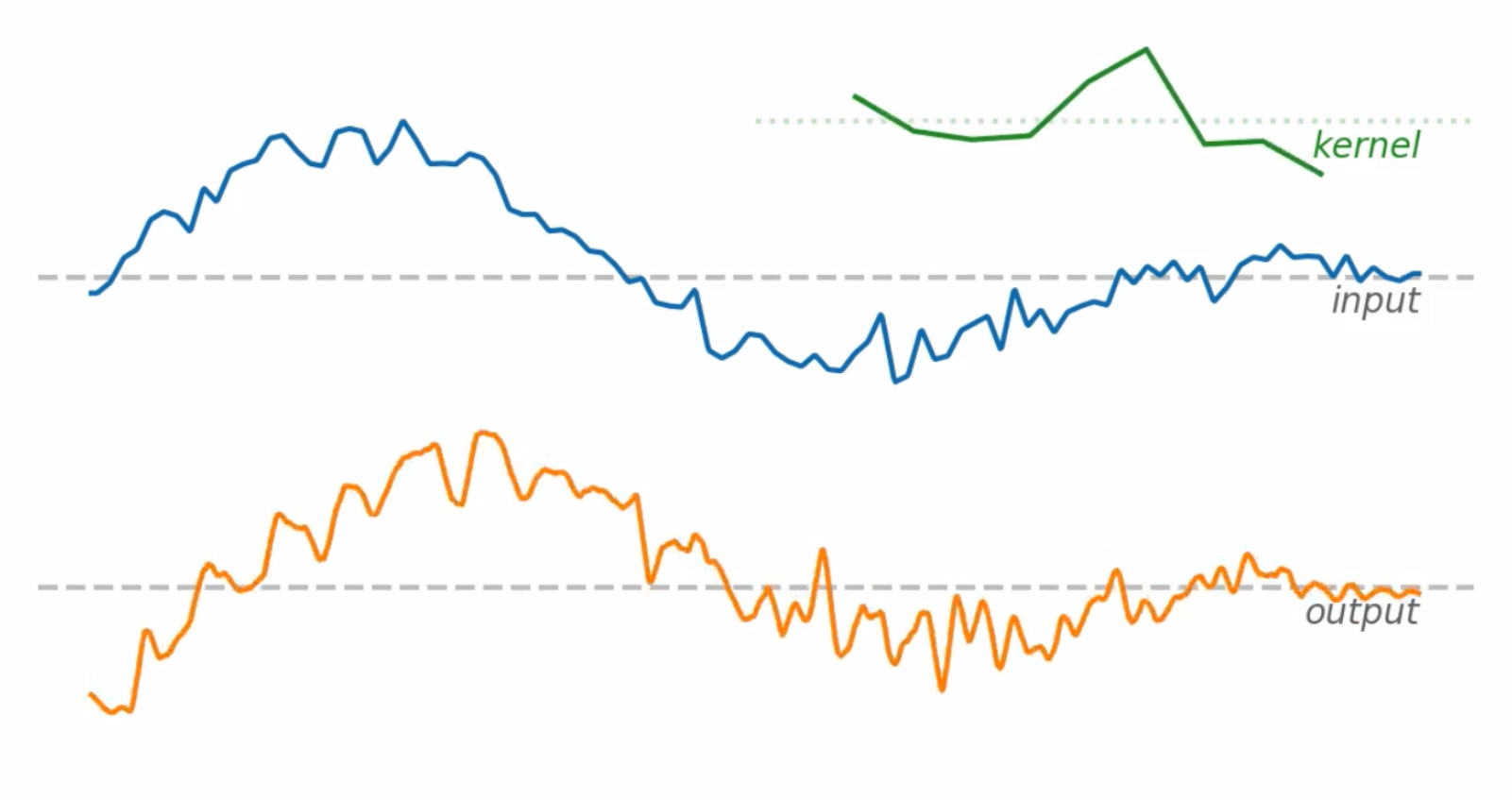
Convolving a moving average filter with a time series involves computing the inner product between each the series and the filter coefficients at each time point. The larger the response, the more correlated the series is with the kernel at that point.

Dilation (1)



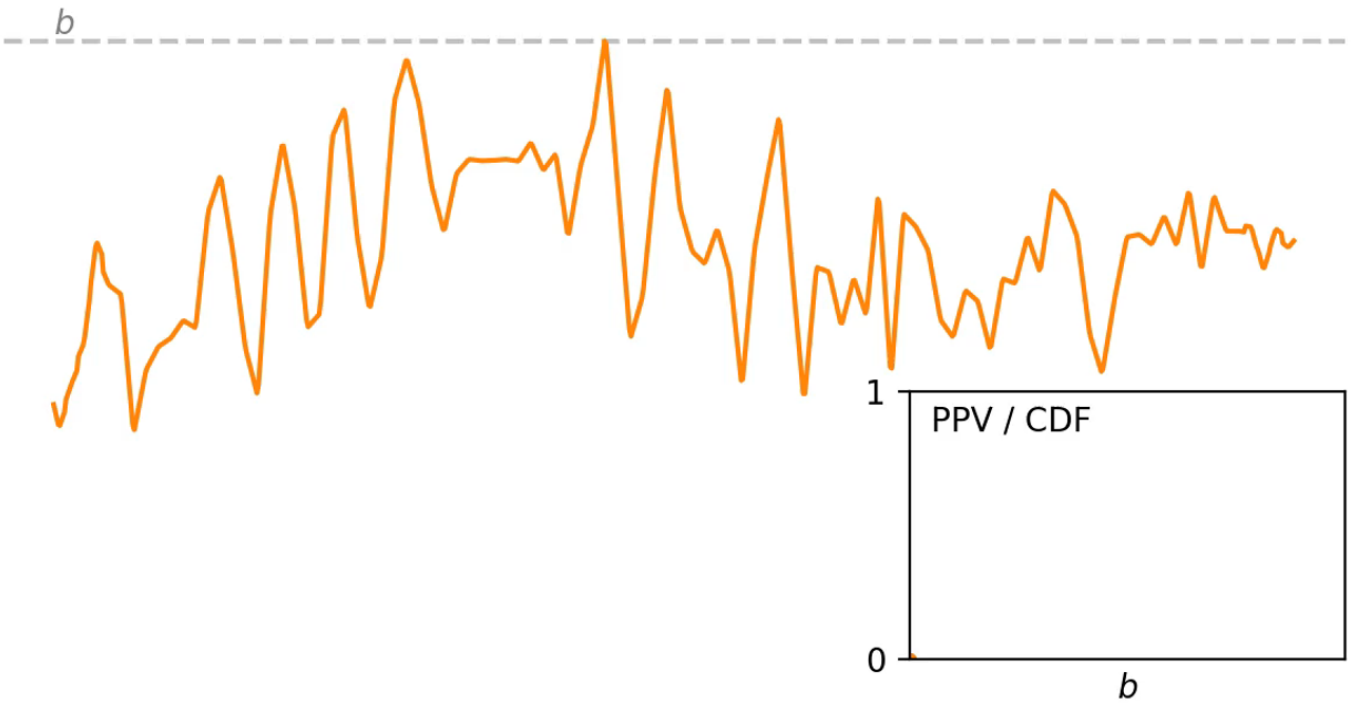
Dilation involves stretching the kernel to capture features at different time scales.

Dilation (2)



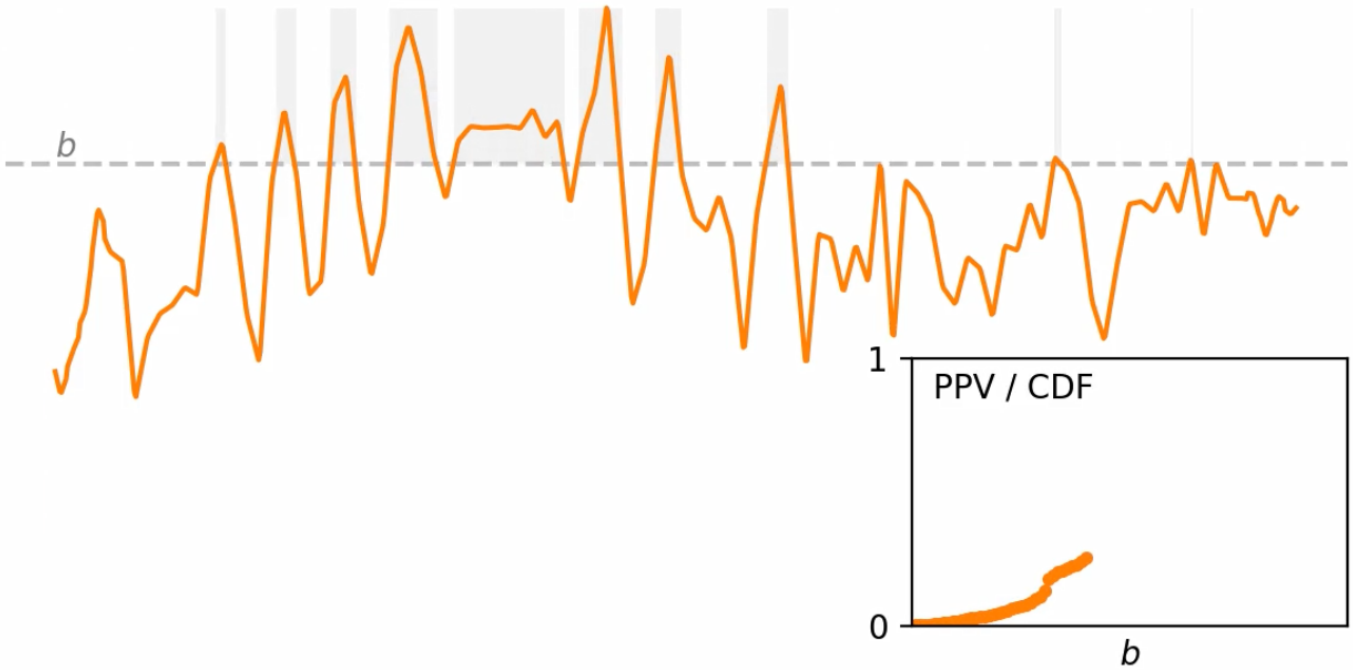
Dilation involves stretching the kernel to capture features at different time scales.

PPV (1)



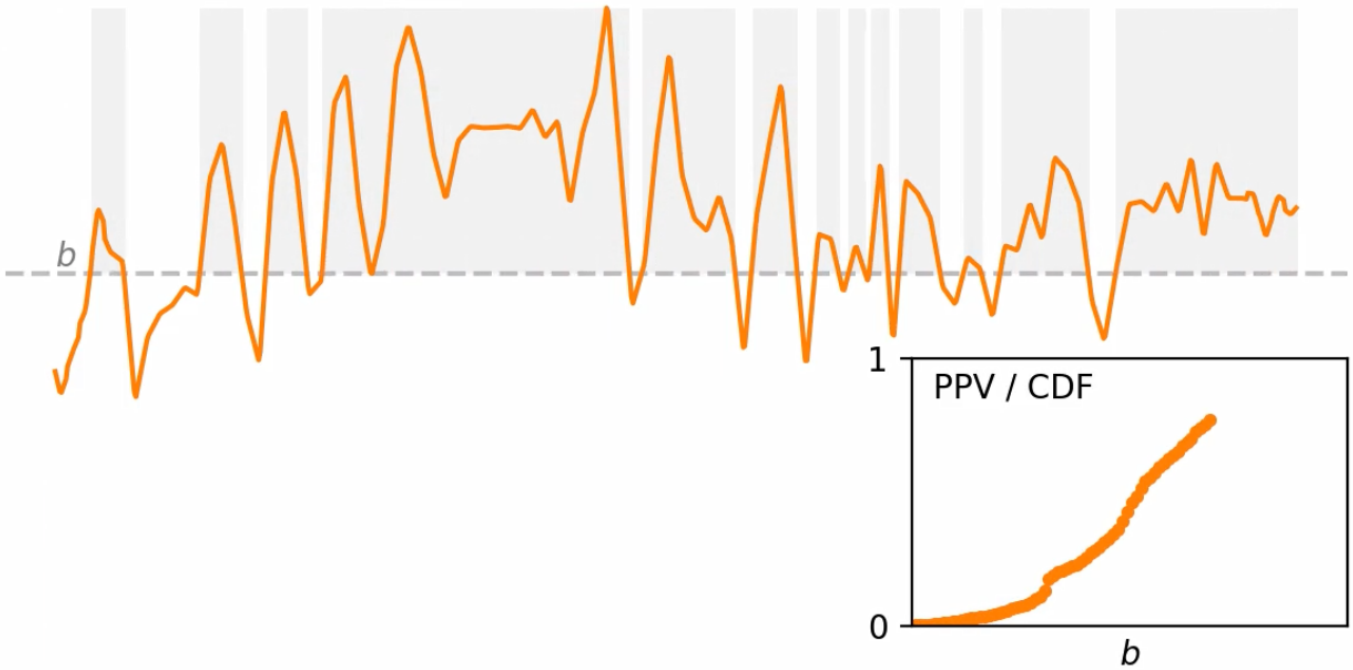
The PPV feature is the proportion of the filtered signal exceeding threshold b.

PPV (2)



The PPV feature is the proportion of the filtered signal exceeding threshold b.

PPV (3)



The PPV feature is the proportion of the filtered signal exceeding threshold b.

Classification

- Total of $2K$ features for each individual
  - Default $K = 10,000$
- Classification is done by fitting a "ridge regression" classifier
  - Squared-error ridge regression based technique
  - Train $L - 1$ linear models in a "one-against-all" manner, recoding targets to 1 and -1 for each class
  - At prediction time, compute linear predictors for each model and choose the class with the largest linear predictor
  - Importantly, ℓ_2 regularisation worked well
- Results on UCR:
  - Prediction performance of 0.865
  - c. 2 hours run time

  $\Longrightarrow$ near-state-of-the-art performance at a fraction of the run-time
- Why might random convolutions work so well?
  - Perhaps not dissimilar from features obtained from a large neural network with random initialisation and several gradient descent updates

Classification

- Total of $2K$ features for each individual
  - Default $K = 10,000$
- Classification is done by fitting a "ridge regression" classifier
  - Squared-error ridge regression based technique
  - Train $L - 1$ linear models in a "one-against-all" manner, recoding targets to 1 and -1 for each class
  - At prediction time, compute linear predictors for each model and choose the class with the largest linear predictor
  - Importantly, ℓ_2 regularisation worked well
- Results on UCR:
  - Prediction performance of 0.865
  - c. 2 hours run time

  $\Longrightarrow$ near-state-of-the-art performance at a fraction of the run-time
- Why might random convolutions work so well?
  - Perhaps not dissimilar from features obtained from a large neural network with random initialisation and several gradient descent updates

Outline

Outstanding Issues with Rocket

- I became involved at this point
- While Rocket worked well, there were several possible issues
  - The randomness of features
  - While much faster than SOTA, felt there was room to speed things up
  - It did not produce probabilistic predictions
- I will first focus on the first two issues

Deterministic Kernels (1)

- Due to light tails of the normal distribution most kernels have weights clustered between (-2,2)
- So, instead of continuously random weights, use kernels based on only two values, say α and β
- Specifically, we proposed to use all permutations of weight vectors with a values of α and b values of β
- For example, if $a = 6$ and $b = 3$, the kernels would all be permutations of

$$(\alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \beta, \beta, \beta)$$

Deterministic Kernels (2)

- How to choose α and β?
- We require the kernels sum to zero (invariance to level shifts), i.e.,

$$a\alpha - b\beta = 0$$

- Taking $\alpha = 1$ (without loss of generality) yields

$$\beta = -\frac{b}{a}$$

- a and b are hyperparameters, but it turns out having one substantially larger than the other helps speed-up
- In our work we settled on $a = 6$ and $b = 3$ (so $\alpha = 1$, $\beta = -2$) for length-9 filters, but the procedure is very insensitive to this choice $\implies$ results in 84 unique kernels
- In total there are then $84 \times \log_2 m$ convolutions (including dilations)

Deterministic Kernels (3)

The 84 unique kernels formed from permutations of $a = 6$ values of $\alpha = 1$ and $b = 3$ values of $\beta = -2$.

Optimisations (1)

- This restriction to this small set of kernels leads to optimisations

1. We quasi-randomly assign $L = 10,000$ biases to generate PPV features
   - Biases are drawn from the convolutional outputs (scale invariance)
   - Multiple PPV features now from the same kernels

   $\Longrightarrow$ we only need to compute the convolutions for each kernel once

2. The weights are integers $\Rightarrow$ no need for multiplication

3. We can compute $a/(a+b)$ of the convolutions for all kernels at a fixed dilation level "in one step"

Optimisations (2)

- Consider a convolution of a signal $\mathbf{x}(t)$ with a k-filter $\mathbf{c}$

$$\mathbf{v}(t) = \mathbf{x}(t) * \mathbf{c} = \sum_{i=0}^{k-1} c_i \, \mathbf{x}(t) \, z^i$$

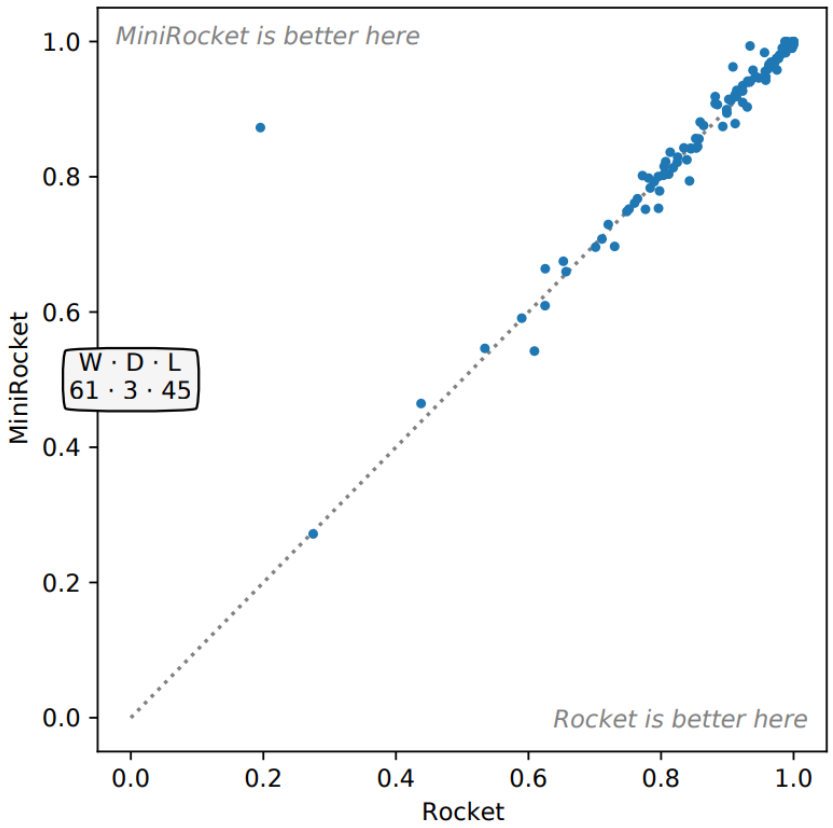
  where z^d denotes a "delay by d time steps"

- Consider the case where $k = 9$, and $\mathbf{c} = (1, 1, 1, 1, 1, 1, -2, -2, -2)$

$$\begin{aligned}
\mathbf{v}(t) &= \sum_{i=0}^{5} \mathbf{x}(t) \, z^i - 2 \sum_{i=6}^{8} \mathbf{x}(t) \, z^i \\
&= \sum_{i=0}^{5} \mathbf{x}(t) \, z^i - 2 \sum_{i=6}^{8} \mathbf{x}(t) \, z^i + \sum_{i=6}^{8} \mathbf{x}(t) \, z^i - \sum_{i=6}^{8} \mathbf{x}(t) \, z^i \\
&= \sum_{i=0}^{9} \mathbf{x}(t) \, z^i - 3 \sum_{i=6}^{8} \mathbf{x}(t) \, z^i
\end{aligned}$$

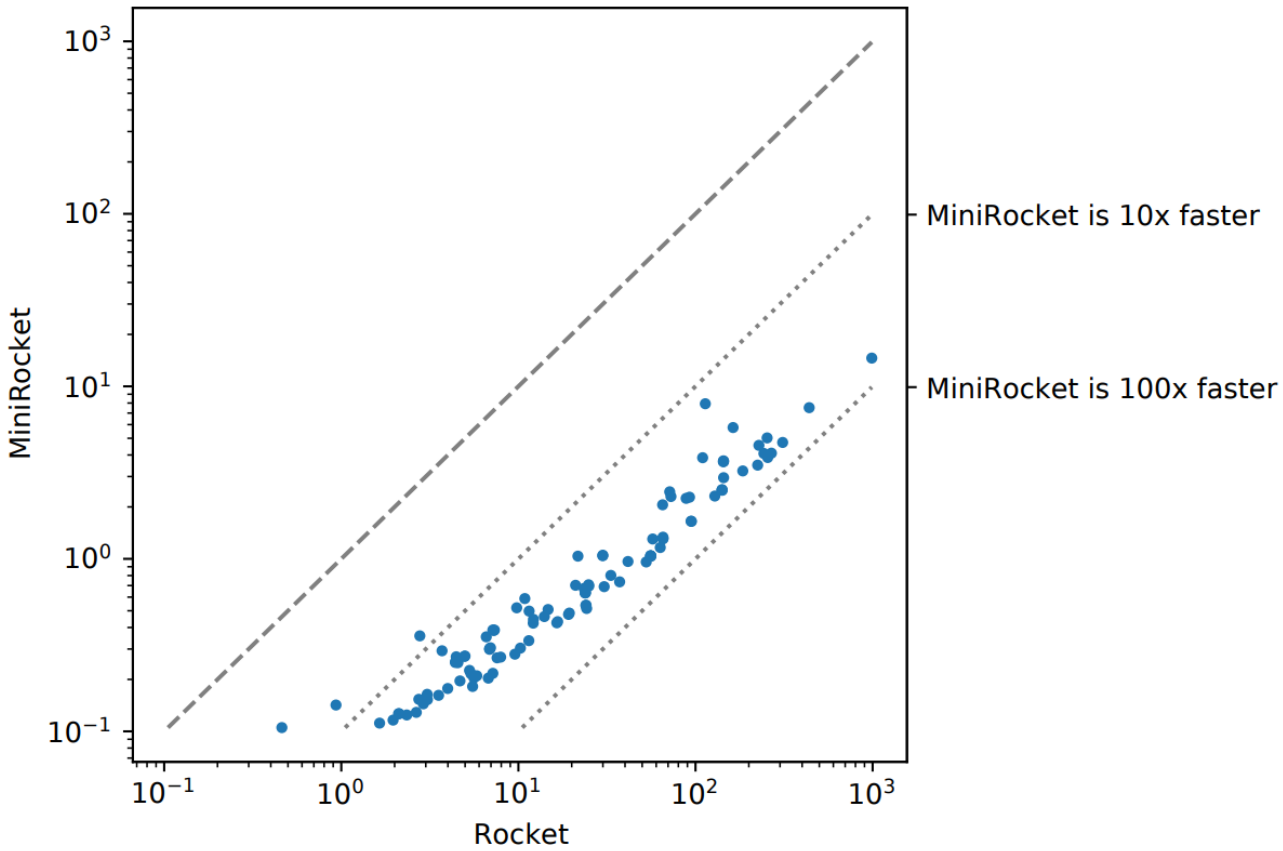
  $\Longrightarrow$ all convolutions are differences from the first term

Results (1)



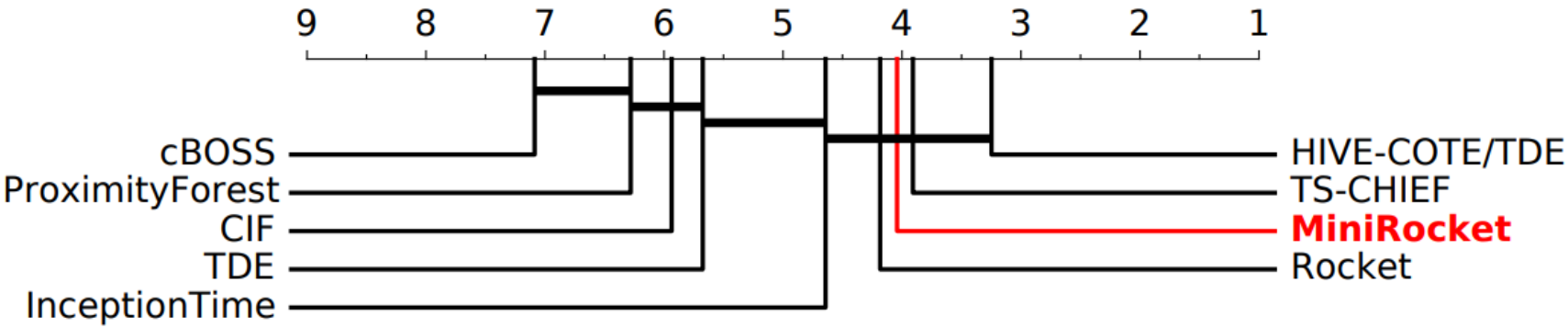
Pairwise accuracy of MiniRocket vs Rocket on UCR benchmark.
Average classification accuracy of 0.872 (0.864 for Rocket)

Results (2)



Timing comparisons between MiniRocket and Rocket.
Run time of MiniRocket is c. 8 minutes (c. 2 hours for Rocket)
(InceptionTime was c. 4 days, HIVE-COTE2 was c. 3 weeks)

Results (3)



Mean rank of MiniRocket vs other SOTA methods on 30 resamples of UCR. Dark lines indicate "cliques" of statistical indistinguishable methods.

Results (4)

	Accuracy		Training Time	
	ROCKET	MINIROCKET	ROCKET	MINIROCKET
Fruit	0.9491	0.9568	2h 36m 40s	2m 22s
Insect	0.7796	0.7639	26m 44s	37s
Mosquito	0.8271	0.8165	15h 34m 58s	12m 32s

Comparison of Rocket and MiniRocket on larger datasets.

- *FruitFlies* (17,250 series, each length 5,000)
- *InsectSound* (25,000 series, each length 600)
- *MosquitoSound* (140,000 series, each length 3,750)

Outline

Multinomial Regression (1)

- Consider targets $\mathbf{y} \in \{1, \ldots, L\}^n$ and feature matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$
- The standard linear classifier is the multinomial logistic regression
- Let $\boldsymbol{\beta}^{(i)}$ denote the coefficients associated with class i; let

$$\boldsymbol{\eta} = \mathbf{X}\boldsymbol{\beta}^{(i)}$$

be the linear predictor associated with class i, and form

$$\mathbf{H} = (\boldsymbol{\eta}^{(1)}, \ldots, \boldsymbol{\eta}^{(L)})$$

- Under the multinomial logistic regression model

$$\mathbb{P}(Y_i = j) = \frac{e^{\eta_i^{(j)}}}{\sum_k e^{\eta_i^{(k)}}}$$

- The negative log-likelihood of $\mathbf{y}$, given $\mathbf{H}$ is then

$$l(\mathbf{H}) = \sum_{i=1}^{n} \left[-\eta_i^{(y_i)} + \log \sum_{j=1}^{L} \exp \eta_i^{(j)} \right]$$

Multinomial Regression (2)

- Fitting is usually done through (penalized) maximised likelihood, i.e.,

$$\underset{\boldsymbol{\beta}^{(1)},\ldots,\boldsymbol{\beta}^{(L)}}{\arg\max} \left\{ l(\mathbf{H}(\boldsymbol{\beta}^{(1)},\ldots,\boldsymbol{\beta}^{(L)})) + \lambda \sum_{j=1}^{L} ||\boldsymbol{\beta}^{(j)}||^2 \right\}$$

  is ℓ_2-penalized maximum likelihood
- This is difficult/slow for large p as it must be solved numerically
- Usually, λ is found through K-fold cross-validation
  - Even more expensive, as multiple re-fittings must be done for each fold
- If all we are interested in is class predictions, we can speed it up by using approximate solutions based on penalized least-squares

Ridge Regression Classifier (1)

- To use squared error to fit a classifier we need recode the targets
- We make L new column vectors $\tilde{\mathbf{y}}^{(j)}$, one for each class with entries

$$\tilde{y}_i^{(j)} = \begin{cases} 1 & \text{if } y_i = j, \\ -1 & \text{otherwise} \end{cases}$$

- We can then fit L ridge regressions to these new targets

$$\hat{\boldsymbol{\beta}}^{(j)} = \arg\min_{\boldsymbol{\beta}} \left\{ ||\tilde{\mathbf{y}}^{(j)} - \mathbf{X}\boldsymbol{\beta}||^2 + \lambda_j ||\boldsymbol{\beta}||^2 \right\}$$

  which have closed form solutions

$$\hat{\boldsymbol{\beta}}^{(j)} = \left(\mathbf{X}^T\mathbf{X} + \lambda_j \mathbf{I}_p \right)^{-1} \mathbf{X}^T \tilde{\mathbf{y}}^{(j)}$$

- The regularisation parameter λ_j can be found efficiently via LOOCV
- Given a feature vector $\mathbf{x} = (x_1, \ldots, x_p)$, we can classify it using

$$c = \arg\max_j \left\{ \mathbf{x}^T \hat{\boldsymbol{\beta}}^{(j)} \right\}$$

Ridge Regression Classifier (2)

- This unfortunately only provides predictions of classes, and not estimates of probabilites
- We might try to calibrate this model directly; form

$$\hat{\mathbf{H}}_\lambda = \left(\mathbf{X}\hat{\boldsymbol{\beta}}^{(1)}, \ldots, \mathbf{X}\hat{\boldsymbol{\beta}}^{(L)}\right)$$

  and pass it through the logistic transformation

$$\hat{\kappa} = \arg\min_{\kappa} l(\kappa \cdot \hat{\mathbf{H}}_\lambda)$$

  where κ is a scaling factor to "calibrate" the linear predictor
- Unfortunately, if p is large this does not work well
  - The linear models predict the classes of future data well, but they do fit the training data perfectly ("benign overfitting")
- So $\hat{\kappa}$ is generally chosen to be very large, and the resulting model now predicts extreme probabilities for new data

Prevalidated Ridge Regression Classifier (1)

- The problem is that the regularisation is done with respect to the squared error fits, not the log-loss
- To fix this, we calibrate the cross validated linear predictions
- Specifically, for a given λ, we form a matrix $\tilde{\mathbf{H}}_\lambda$ composed of the LOO cross-validation predictions for each individual and class
  - These are the predictions for $\tilde{y}_i^{(j)}$ formed by dropping $\tilde{y}_i^{(j)}$ from $\mathbf{y}^{(j)}$, solving the ridge solution on this reduced dataset, and then predicting onto $\tilde{y}_i^{(j)}$
  - These are quantities used in the jack-knife (a linearised form of bootstrap)
- We then calibrate these predictions by solving

$$\hat{\kappa}, \hat{\lambda} = \arg\min_{\kappa,\lambda} l(\kappa \cdot \tilde{\mathbf{H}}_\lambda)$$

- We call these predictions "pre-validated"

Prevalidated Ridge Regression Classifier (2)

- Why should this work?
  - The matrix $\tilde{\mathbf{H}}_\lambda$ is a perturbed version of the predictions $\hat{\mathbf{H}}_\lambda$; it no longer provide perfect fits to the training data
  - The regularisation parameter is now chosen with respect to overall log-loss, rather than the squared-error of each of the regressions
- Why is it fast?
  - Let $m = \max(n, p)$ and $r = \min(n, p)$
  - We note that each linear predictor uses the same feature matrix $\mathbf{X}$
  - If we perform an SVD on this matrix (time complexity $O(mr^2)$, we perform dimensionality reduction and orthogonalisation
  - The ridge solutions for a λ can be computed in $O(r)$ time given an orthogonal design
  - Using the LOOCV "shortcut", all the LOOCV predictions can be computed in $O(r^2)$ time

  $\implies$ Overall time complexity not greater than a single ridge fit

Prevalidated Ridge Regression Classifier (2)

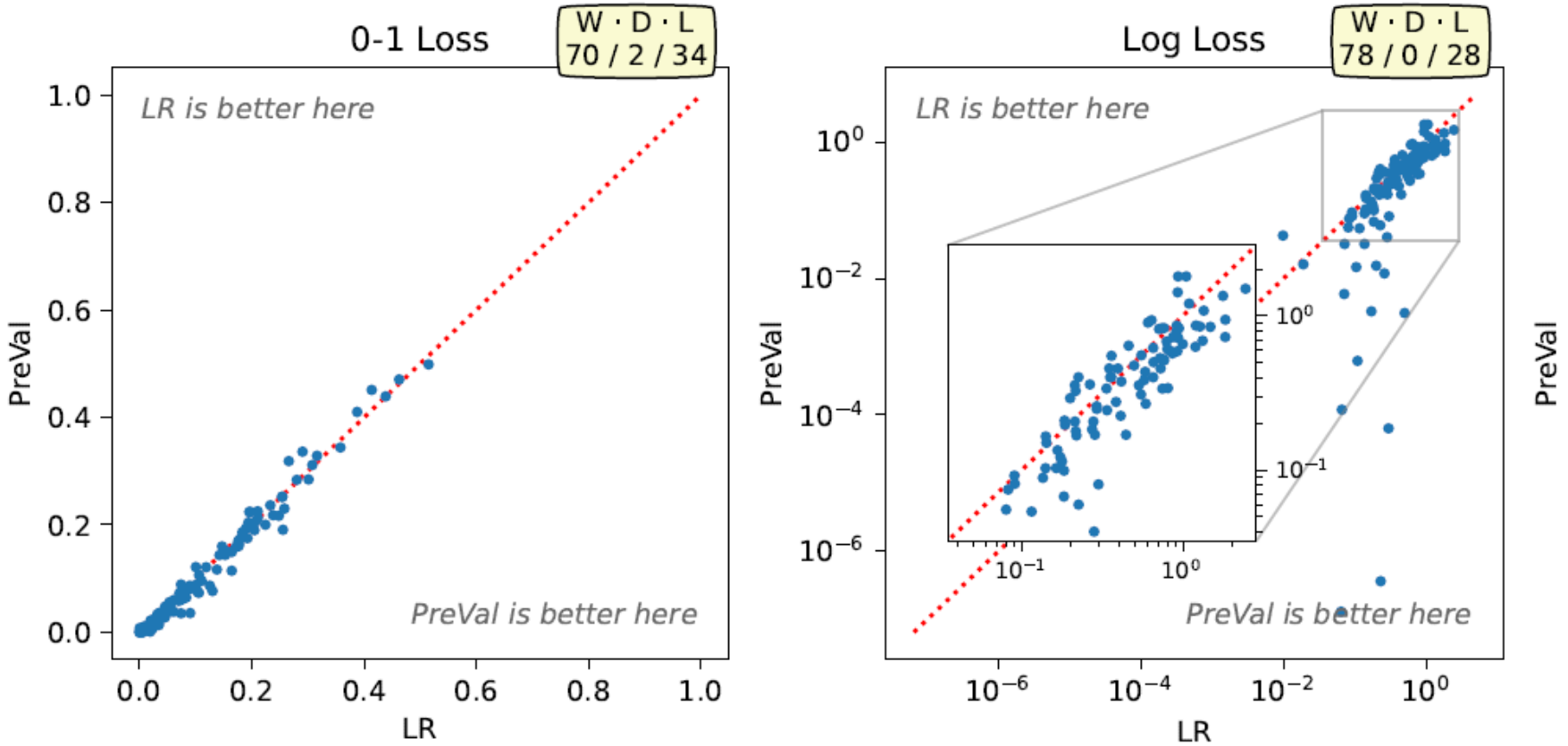
- Why should this work?
  - The matrix $\tilde{\mathbf{H}}_\lambda$ is a perturbed version of the predictions $\hat{\mathbf{H}}_\lambda$; it no longer provide perfect fits to the training data
  - The regularisation parameter is now chosen with respect to overall log-loss, rather than the squared-error of each of the regressions
- Why is it fast?
  - Let $m = \max(n, p)$ and $r = \min(n, p)$
  - We note that each linear predictor uses the same feature matrix $\mathbf{X}$
  - If we perform an SVD on this matrix (time complexity $O(mr^2)$, we perform dimensionality reduction and orthogonalisation
  - The ridge solutions for a λ can be computed in $O(r)$ time given an orthogonal design
  - Using the LOOCV "shortcut", all the LOOCV predictions can be computed in $O(r^2)$ time

$\implies$ Overall time complexity not greater than a single ridge fit

Results (1)

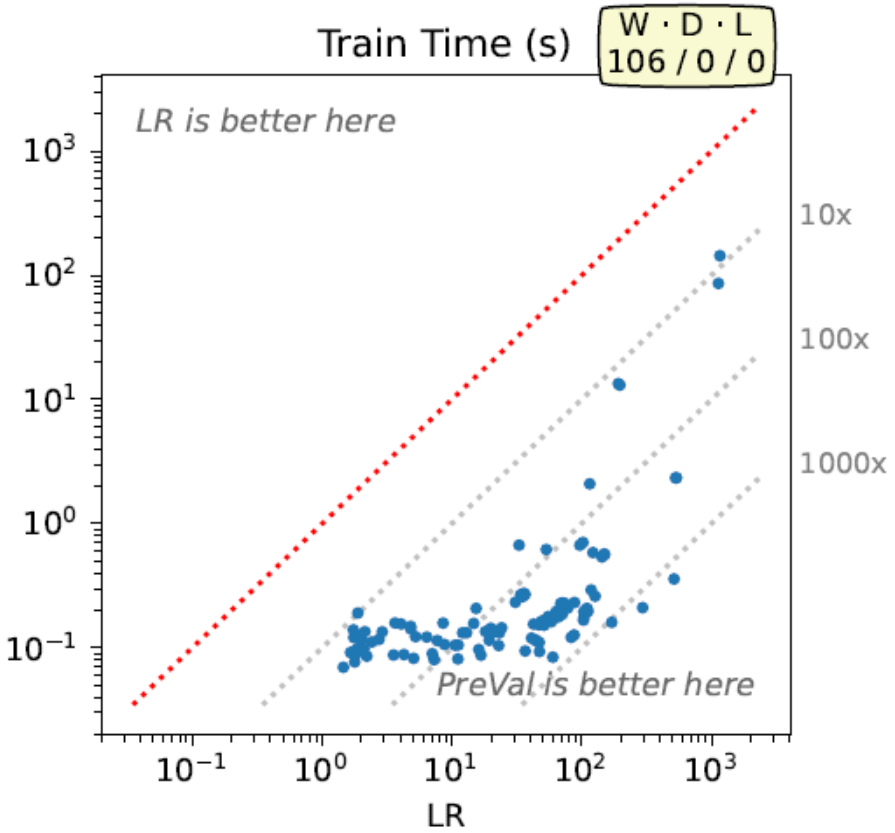
- We tested our method ("PreVal") on the time series classification setting
- We generated MiniRocket features for all 112 datasets in the UCR
- Then fitted linear classifiers using
  - calibrated pre-validated ridge regression predictions ("PreVal")
  - ℓ_2-regularised maximum likelihood with 5-fold CV
- For the latter we used the scikit-learn implementation
- For the former, we used our `preval` package
- Evaluated on 0-1 loss, log-loss and wall-clock time

Results (2)



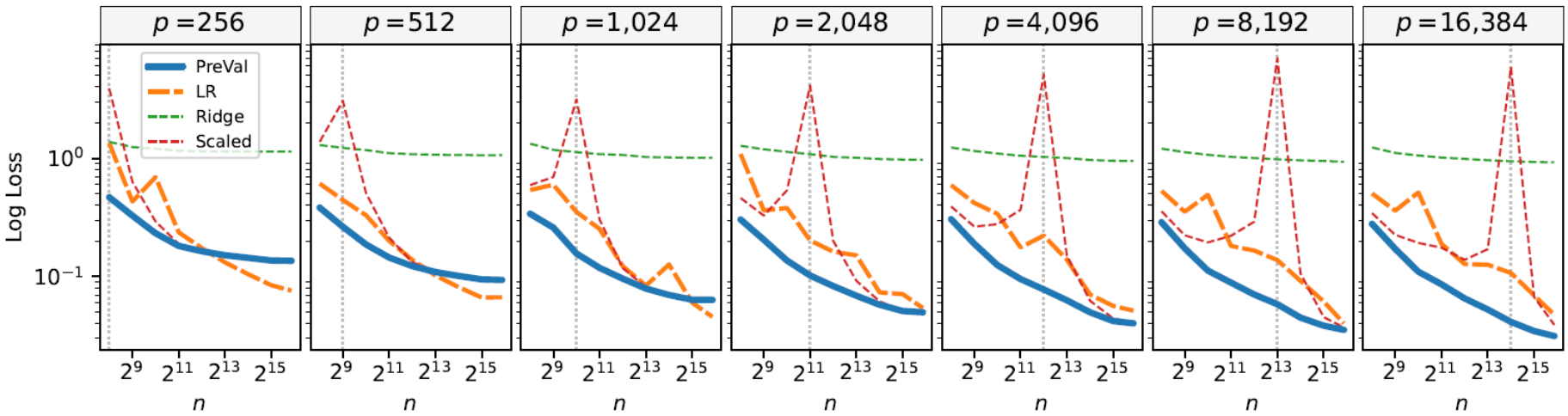
Pairwise 0-1 and log-loss for "PreVal" vs logistic regression (UCR data)
Logistic regression was trained using scikit-learn (5-fold CV)

Results (3)



Timing for Preval vs logistic regresion (UCR data)
Logistic regression was trained using scikit-learn (5-fold CV)

Results (4)



Learning Curves for Preval, logistic regression, ridge regression (unscaled) and ridge regression (scaled)

Note that direct scaling of ridge regression exhibits double-descent around $n \approx p$, while scaling pre-validated predictions removes this phenomena
For large p pre-validated is highly competitive; for large n one would expect logistic regression via maximum likelihood will outperform PreVal

Thank you!

- Thank you for your attention!
- Code is available at
  - `https://github.com/angus924/minirocket` (MiniRocket)
  - `https://github.com/angus924/preval` (PreVal)
- Relevant publications with more details
  - A. Dempster, F.Petitjean, G.I.Webb, "*Rocket: Exceptionally fast and accurate time series classification using random convolutional kernels*", Data Mining and Knowledge Discovery, 2019
  - A. Dempster, D.F.Schmidt and G.I.Webb, "*MiniRocket: A Very Fast (Almost) Deterministic Transform for Time Series Classification*", Proc. 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2021
  - A. Dempster, G.I.Webb, D.F.Schmidt, "*Prevalidated ridge regression is a highly-efficient drop-in replacement for logistic regression for high-dimensional data*", arXiv:2401.15610v1