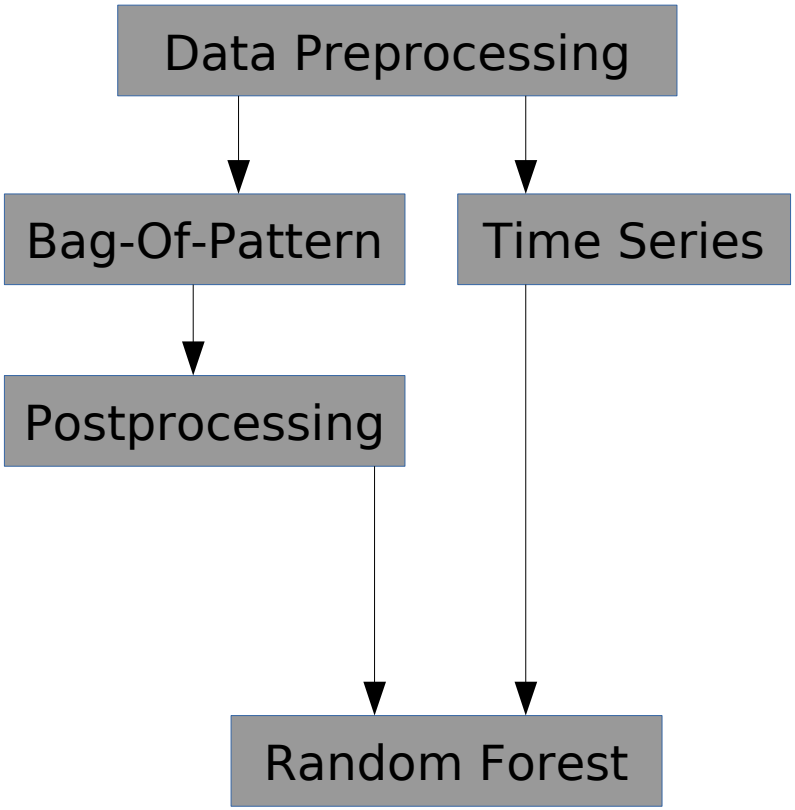
Distributed Multivariate Bag-of-Pattern Classification

Arik Ermshaus
Markus Richter

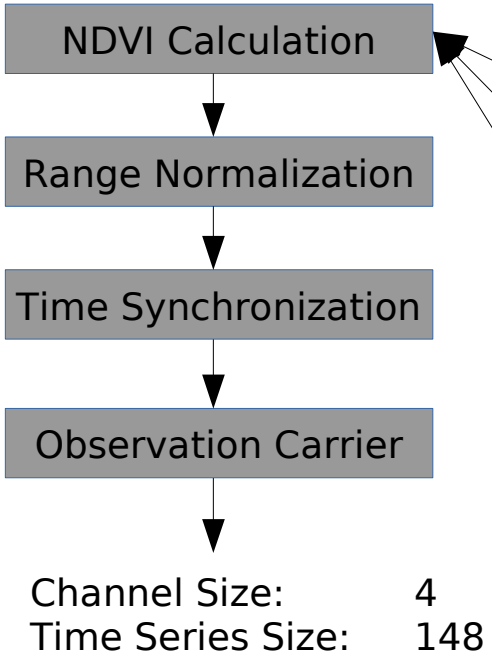
Time Series Classification Pipeline Overview



- Implementation
  - numpy
  - sklearn
  - saxpy
- Scalability
  - 1M training samples
  - 30 processes
- Test Accuracy
  - F1-score: 0.7376
  - ~1 hour runtime

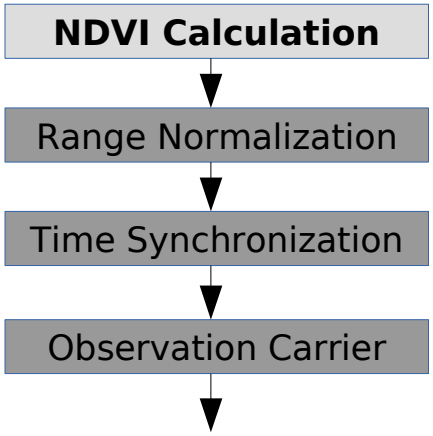
Data Preprocessing Pipeline

9	190	91	65	181	85	63	194	81	61	?	?	?	?	?	?	176	43	38	351	69	75
18	?	?	?	293	78	78	315	65	70	?	?	?	370	62	69	253	31	32	427	74	84
18	344	83	74	329	75	71	381	56	60	?	?	?	456	50	57	300	27	28	440	71	79



9	190	181	194	?	?	176	351
	91	85	81	?	?	43	69
	65	63	61	?	?	38	75
18	?	293	315	?	370	253	427
	?	78	65	?	62	31	74
	?	78	70	?	69	32	84
18	344	329	381	?	456	300	440
	83	75	56	?	50	27	71
	74	71	60	?	57	28	79

Normalized Difference Vegetation Index (NDVI)



Channel Size: 4
Time Series Size: 148

$$NDVI(\mathrm{x}) = \frac{(\mathrm{x.nir} - \mathrm{x.red})}{(\mathrm{x.nir} + \mathrm{x.red})}$$

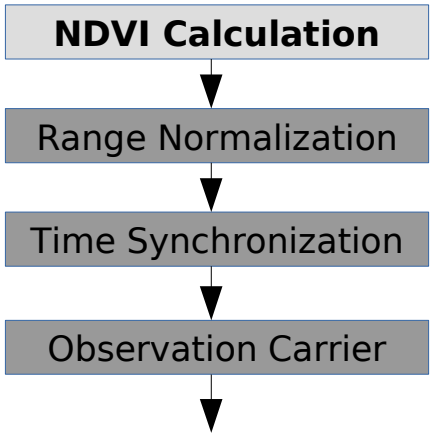
Computed index in range [-1,1]

[-1,0) indicates a water body

[0,1] indicates extent of vegetation

9	190	181	194	?	?	176	351
	91	85	81	?	?	43	69
	65	63	61	?	?	38	75
18	?	293	315	?	370	253	427
	?	78	65	?	62	31	74
	?	78	70	?	69	32	84
18	344	329	381	?	456	300	440
	83	75	56	?	50	27	71
	74	71	60	?	57	28	79

Normalized Difference Vegetation Index (NDVI)

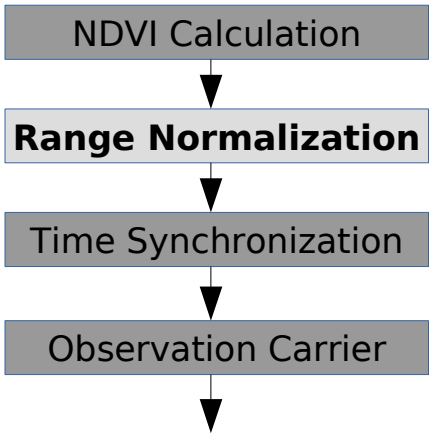


Channel Size: 4
Time Series Size: 148

$$NDVI(\mathrm{x}) = \frac{(\mathrm{x.nir} - \mathrm{x.red})}{(\mathrm{x.nir} + \mathrm{x.red})}$$

9	190	181	194	?	?	176	351
	91	85	81	?	?	43	69
	65	63	61	?	?	38	75
	0.352	0.361	0.411	?	?	0.607	0.671
18	?	293	315	?	370	253	427
	?	78	65	?	62	31	74
	?	78	70	?	69	32	84
	?	0.580	0.658	?	0.713	0.782	0.705
18	344	329	381	?	456	300	440
	83	75	56	?	50	27	71
	74	71	60	?	57	28	79
	0.611	0.629	0.744	?	0.802	0.835	0.722

Range Normalization

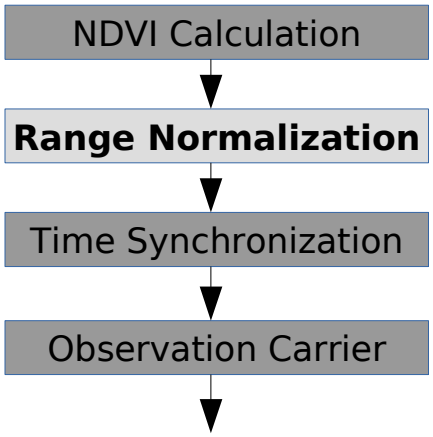


	min	max
nir	176	456
red	27	91
green	28	84
ndvi	0.352	0.835

- Determine min and max values
- Linearly scale the data to [0,1]

9	190	181	194	?	?	176	351
	91	85	81	?	?	43	69
	65	63	61	?	?	38	75
	0.352	0.361	0.411	?	?	0.607	0.671
18	?	293	315	?	370	253	427
	?	78	65	?	62	31	74
	?	78	70	?	69	32	84
	?	0.580	0.658	?	0.713	0.782	0.705
18	344	329	381	?	456	300	440
	83	75	56	?	50	27	71
	74	71	60	?	57	28	79
	0.611	0.629	0.744	?	0.802	0.835	0.722

Range Normalization

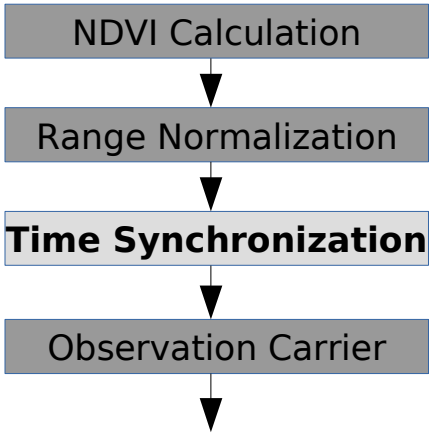


- Determine min and max values
- Linearly scale the data to [0,1]

	min	max
nir	176	456
red	27	91
green	28	84
ndvi	0.352	0.835

9	0.050	0.018	0.064	?	?	0.000	0.625
	1.000	0.906	0.844	?	?	0.250	0.656
	0.661	0.625	0.589	?	?	0.179	0.839
	0.000	0.018	0.121	?	?	0.528	0.661
18	?	0.418	0.496	?	0.693	0.275	0.896
	?	0.797	0.594	?	0.547	0.063	0.734
	?	0.893	0.750	?	0.732	0.071	1.000
	?	0.471	0.633	?	0.747	0.890	0.730
18	0.600	0.546	0.732	?	1.000	0.443	0.943
	0.875	0.750	0.453	?	0.359	0.000	0.688
	0.821	0.768	0.571	?	0.518	0.000	0.911
	0.537	0.573	0.811	?	0.933	1.000	0.766

Time Synchronization



20060206
20060217
20060314
20060320
20060326
20060329
20060502

0
11
36
42
48
51
85

- Normalize dates
- Interpolate to 2D intervals

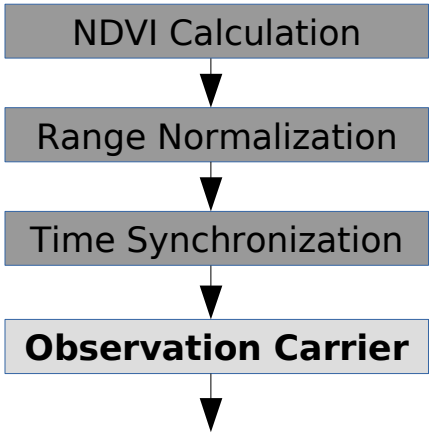
	0	11	36	42	48	51	85
9	0.050	0.018	0.064	?	?	0.000	0.625
	1.000	0.906	0.844	?	?	0.250	0.656
	0.661	0.625	0.589	?	?	0.179	0.839
	0.000	0.018	0.121	?	?	0.528	0.661
18	?	0.418	0.496	?	0.693	0.275	0.896
	?	0.797	0.594	?	0.547	0.063	0.734
	?	0.893	0.750	?	0.732	0.071	1.000
	?	0.471	0.633	?	0.747	0.890	0.730
18	0.600	0.546	0.732	?	1.000	0.443	0.943
	0.875	0.750	0.453	?	0.359	0.000	0.688
	0.821	0.768	0.571	?	0.518	0.000	0.911
	0.537	0.573	0.811	?	0.933	1.000	0.766

Time Synchronization

	0	11	...
9	0.050	0.018	...
	1.000	0.906	...
	0.661	0.625	...
	0.000	0.018	...
18	?	0.418	...
	?	0.797	...
	?	0.893	...
	?	0.471	...
18	0.600	0.546	...
	0.875	0.750	...
	0.821	0.768	...
	0.537	0.573	...

	0	2	4	6	8	10	12	...
9	0.050	0.044	0.038	0.033	0.027	0.021	0.020	...
	1.000	0.983	0.966	0.949	0.932	0.915	0.904	...
	0.661	0.654	0.648	0.641	0.635	0.628	0.624	...
	0.000	0.003	0.007	0.010	0.013	0.016	0.022	...
18	?	?	?	?	?	?	0.421	...
	?	?	?	?	?	?	0.789	...
	?	?	?	?	?	?	0.887	...
	?	?	?	?	?	?	0.477	...
18	0.600	0.590	0.580	0.571	0.561	0.551	0.553	...
	0.875	0.852	0.830	0.807	0.784	0.761	0.738	...
	0.821	0.811	0.802	0.792	0.782	0.773	0.760	...
	0.537	0.544	0.550	0.557	0.563	0.570	0.583	...

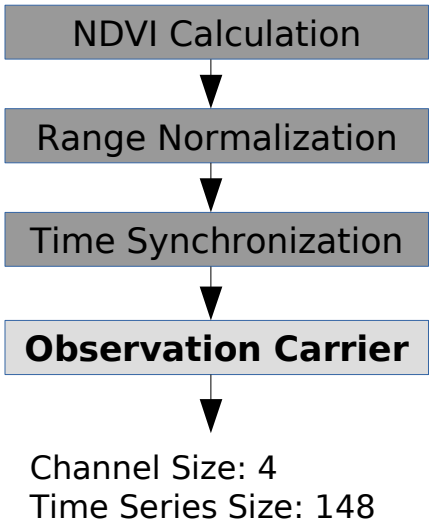
Observation Carrier



- Fill NaN values with neighbours
- Use predecessor/successor values

	0	2	4	6	8	10	12	...
9	0.050	0.044	0.038	0.033	0.027	0.021	0.020	...
	1.000	0.983	0.966	0.949	0.932	0.915	0.904	...
	0.661	0.654	0.648	0.641	0.635	0.628	0.624	...
	0.000	0.003	0.007	0.010	0.013	0.016	0.022	...
18	?	?	?	?	?	?	0.421	...
	?	?	?	?	?	?	0.789	...
	?	?	?	?	?	?	0.887	...
	?	?	?	?	?	?	0.477	...
18	0.600	0.590	0.580	0.571	0.561	0.551	0.553	...
	0.875	0.852	0.830	0.807	0.784	0.761	0.738	...
	0.821	0.811	0.802	0.792	0.782	0.773	0.760	...
	0.537	0.544	0.550	0.557	0.563	0.570	0.583	...

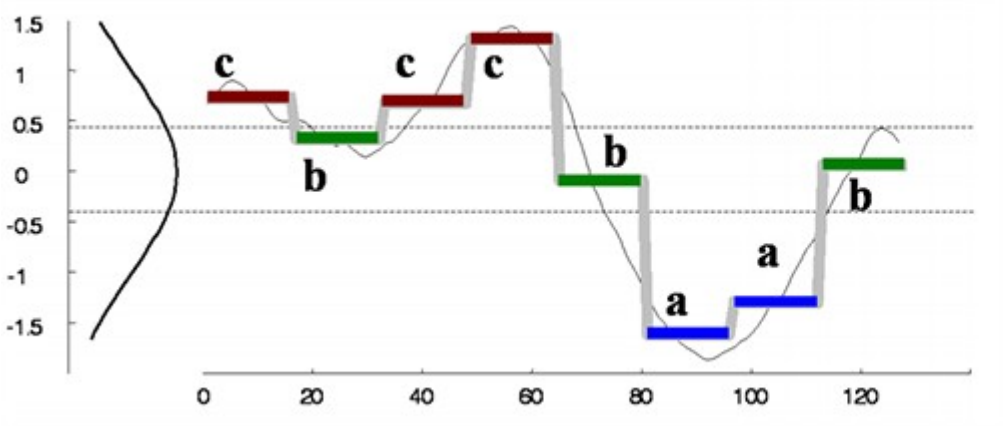
Observation Carrier

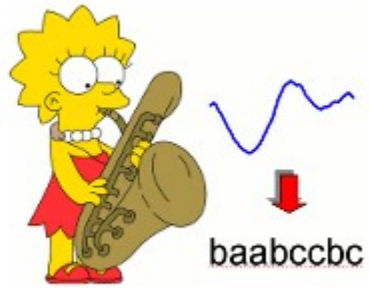


- Fill NaN values with neighbours
- Use predecessor/successor values

	0	2	4	6	8	10	12	...
9	0.050	0.044	0.038	0.033	0.027	0.021	0.020	...
	1.000	0.983	0.966	0.949	0.932	0.915	0.904	...
	0.661	0.654	0.648	0.641	0.635	0.628	0.624	...
	0.000	0.003	0.007	0.010	0.013	0.016	0.022	...
18	0.421	0.421	0.421	0.421	0.421	0.421	0.421	...
	0.789	0.789	0.789	0.789	0.789	0.789	0.789	...
	0.887	0.887	0.887	0.887	0.887	0.887	0.887	...
	0.477	0.477	0.477	0.477	0.477	0.477	0.477	...
18	0.600	0.590	0.580	0.571	0.561	0.551	0.553	...
	0.875	0.852	0.830	0.807	0.784	0.761	0.738	...
	0.821	0.811	0.802	0.792	0.782	0.773	0.760	...
	0.537	0.544	0.550	0.557	0.563	0.570	0.583	...

SAX Transformation





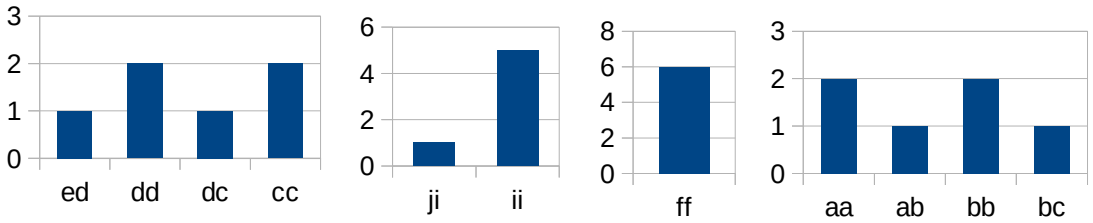
Source: https://github.com/seninp/saxpy

- Transform time series to sequence of letters
- Use fixed alphabet a = {a,b,c,...}, size |a|
- Use mean of disjoint/overlapping intervals

Multivariate Bag-of-Pattern Feature Engineering

	0	2	4	6	8	10	12	...
9	e	d	d	d	c	c	c	...
	j	i	i	i	i	i	i	...
	f	f	f	f	f	f	f	...
	a	a	a	b	b	b	c	...

Multivariate
SAX-transformed time series

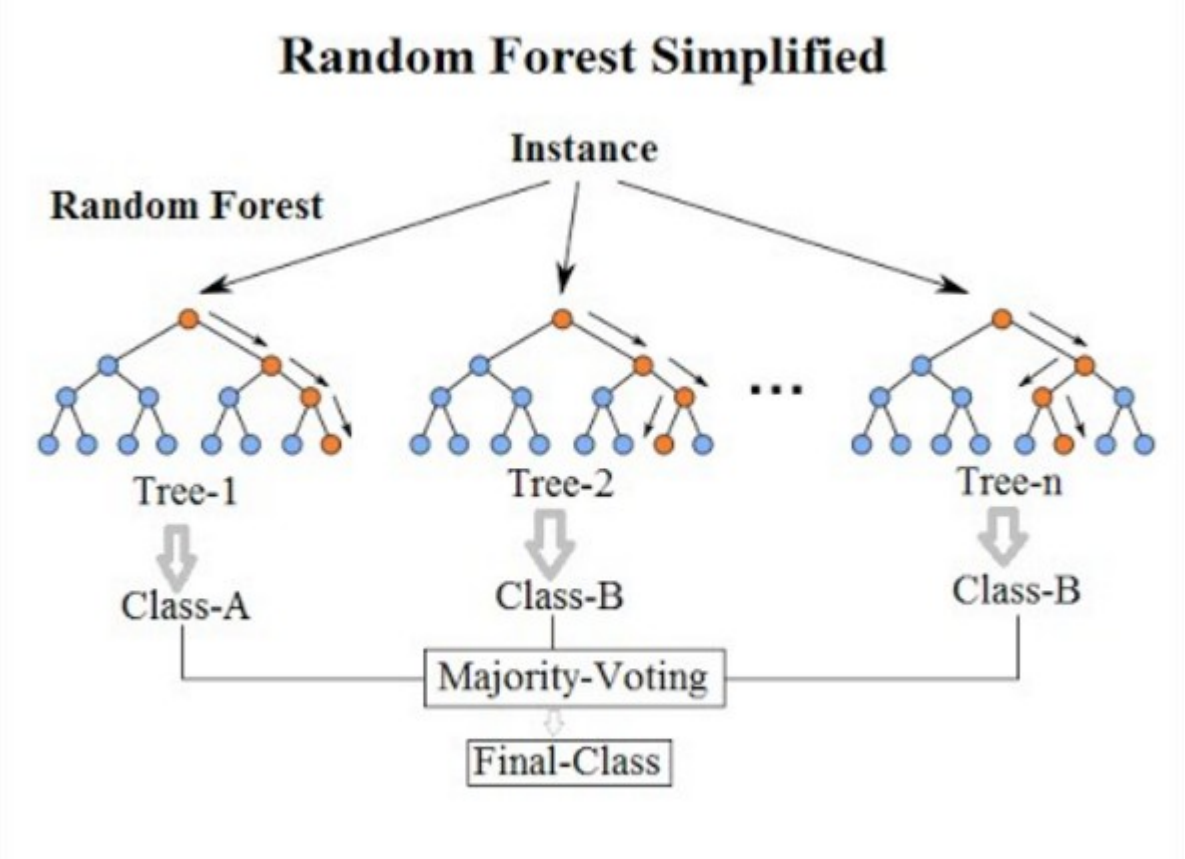


Multivariate
Histogram, window_n = 2

	ed	ji	ff	aa	8	dd	ii	...
9	1	0	0	0	0	2	0	...
	0	1	0	0	0	0	5	...
	0	0	6	0	0	0	0	...
	0	0	0	2	0	0	0	...

Multivariate
Feature Vector

Random Forest Classification



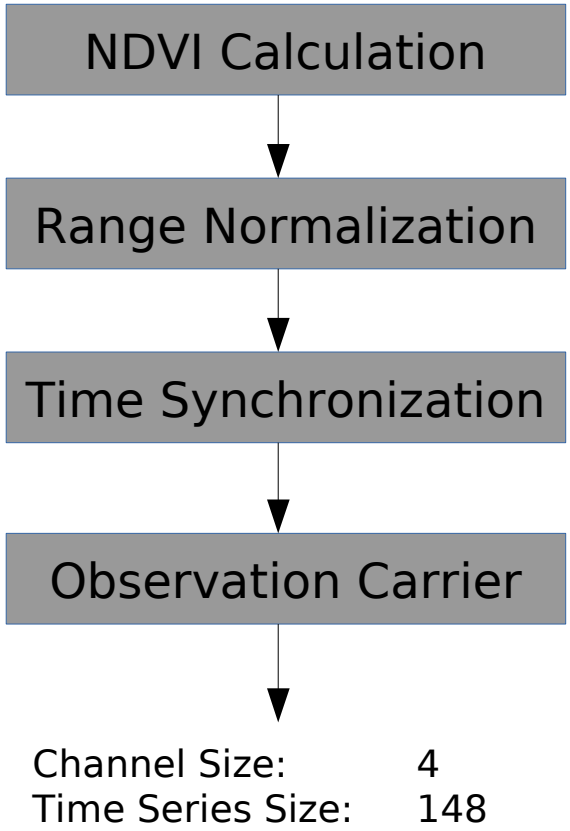
Source: medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d

Things we tried, that didn’t work ...

- 2-6M data samples
- Missing Data
- Resampling
- Cubic Splines
- Moving Averages
- Data Binning
- Fourier Transform
- ...

- tsfresh
- Statistics
- KNN, ExtraTrees
- Logistic Regression
- AdaBoost, SVM
- Gradient Boosting
- Model Stacking
- ...

Scaling and Distributed Computation



Local Computation

n workers

- NDVI
- Time Synchronization
- Observation Carrier

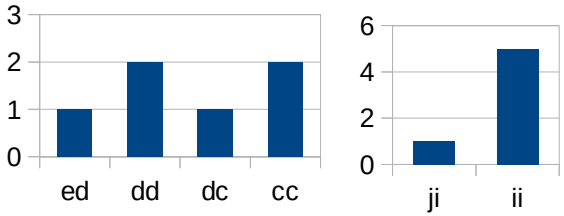
Global Computation

1 worker

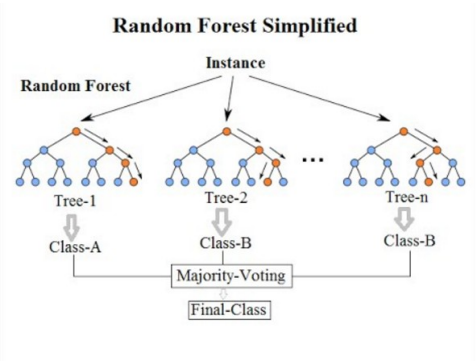
- Range Normalization

Scaling and Distributed Computation

For time series 1..m



For decision tree 1..n



Local Computation

n workers

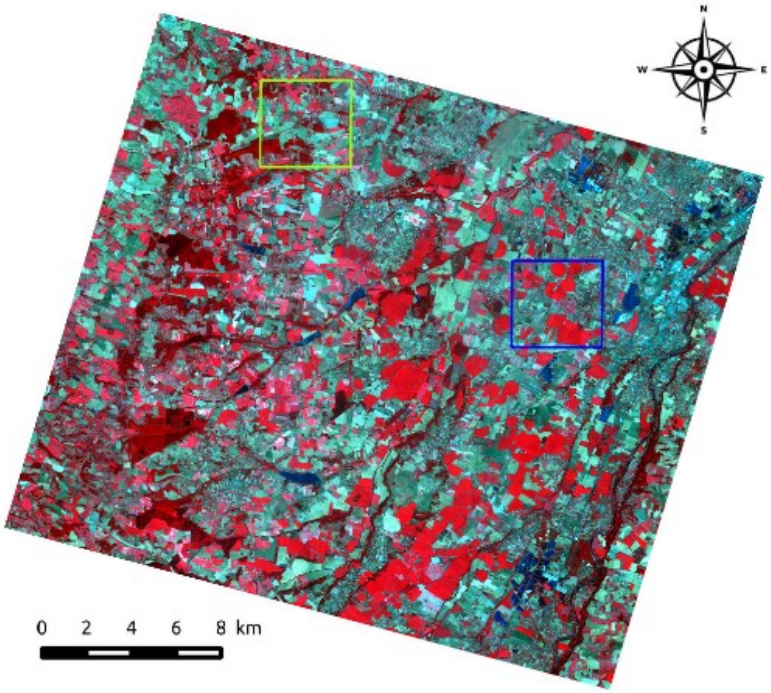
- Bag-of-Pattern
- Random Forest

Global Computation

1 worker

- Feature Selection

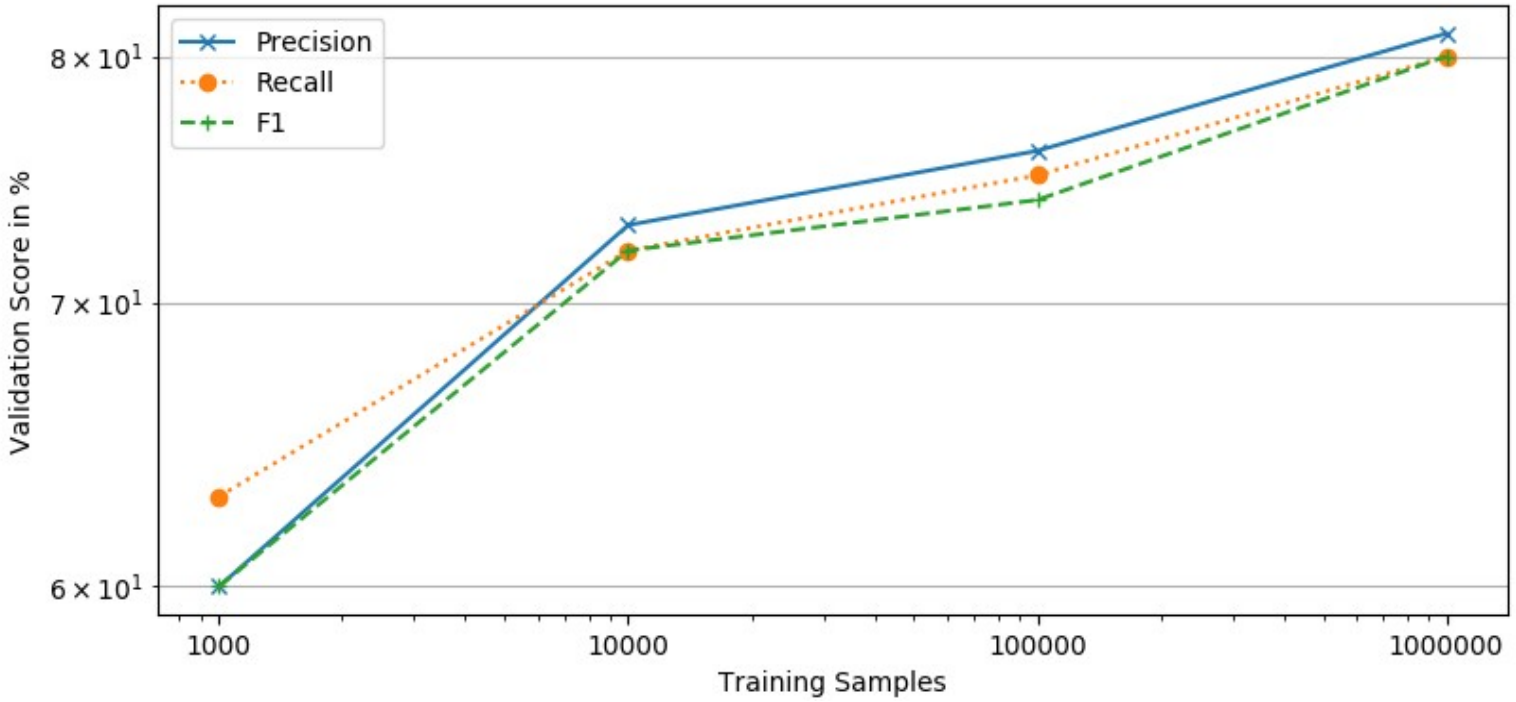
Dataset Overview



Source: https://arxiv.org/pdf/1811.10166.pdf

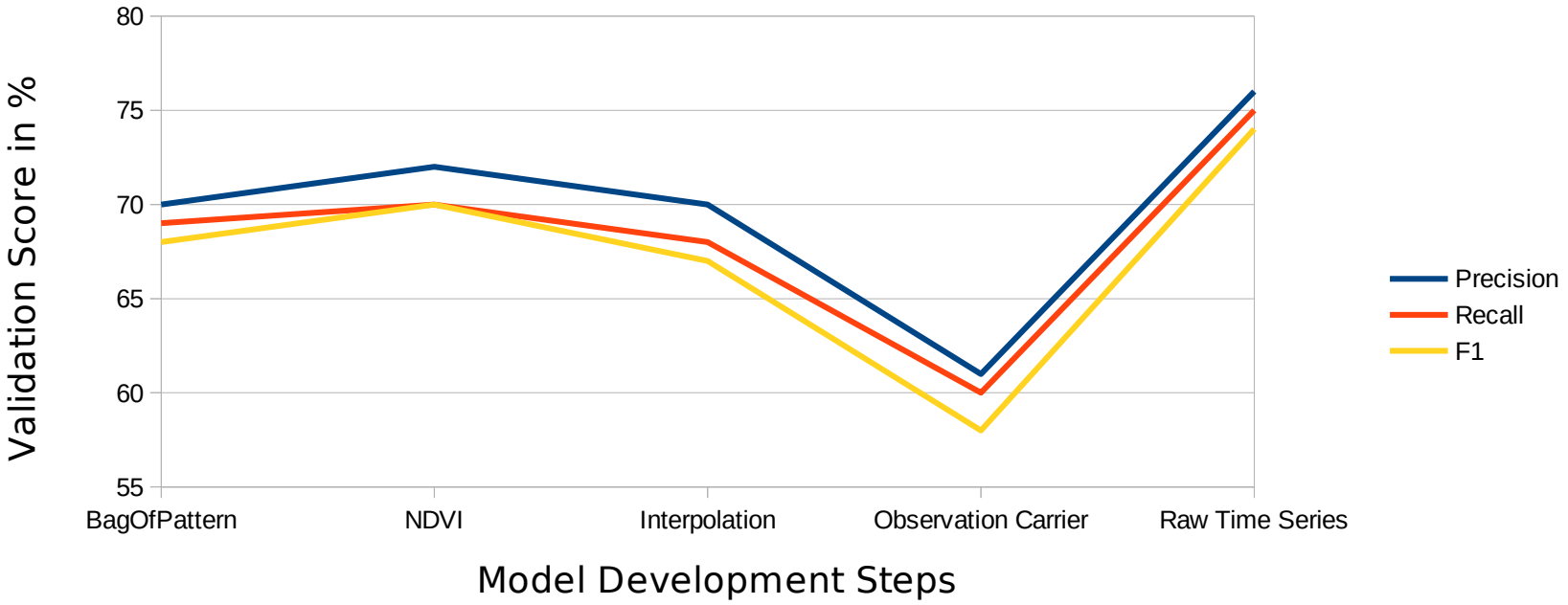
- FORMOSAT-2 images
- South West of France (24 km×24 km area)
- 2,4 GB time series data
- 3 surface reflectances (NIR, Red, Green)
- 46 time stamps
- 24 land cover classes
  - cultivation
  - surroundings
  - water

Model Performance



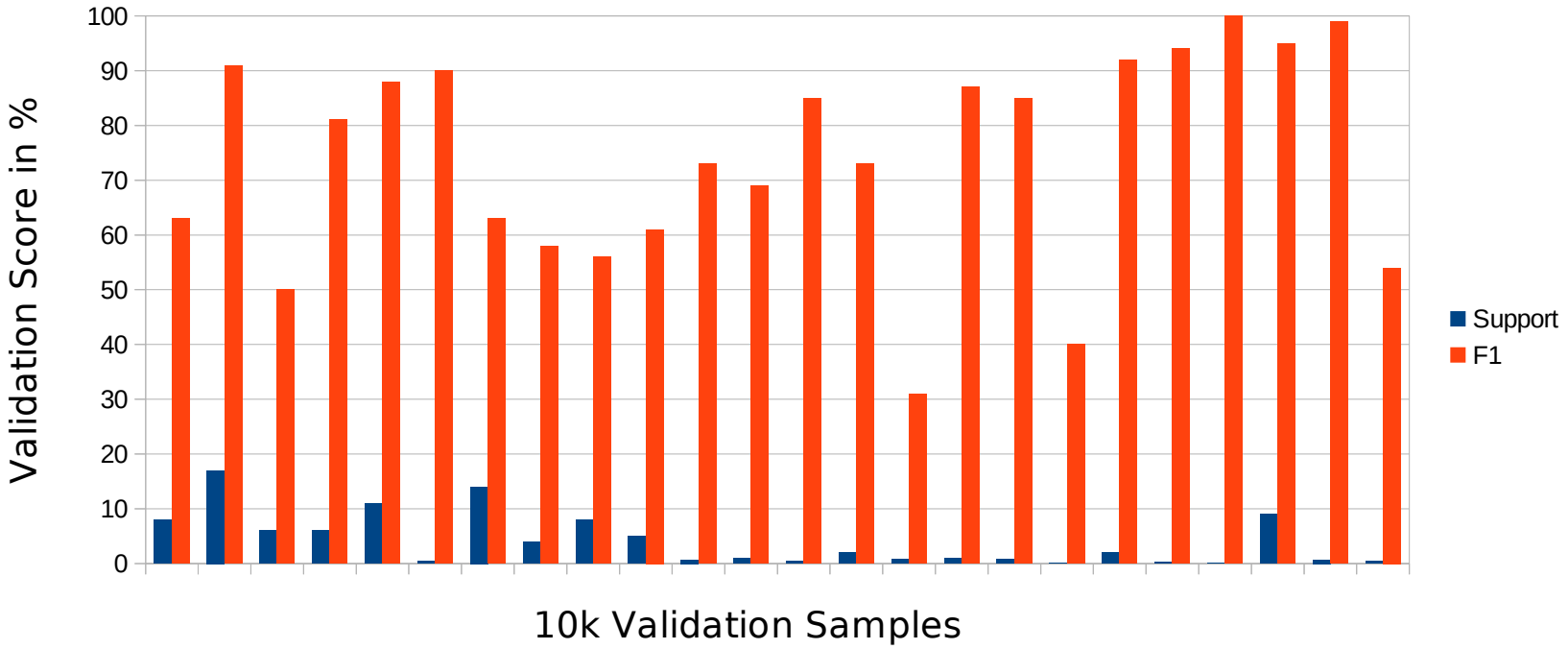
Scaling increases performance

Model Performance Development



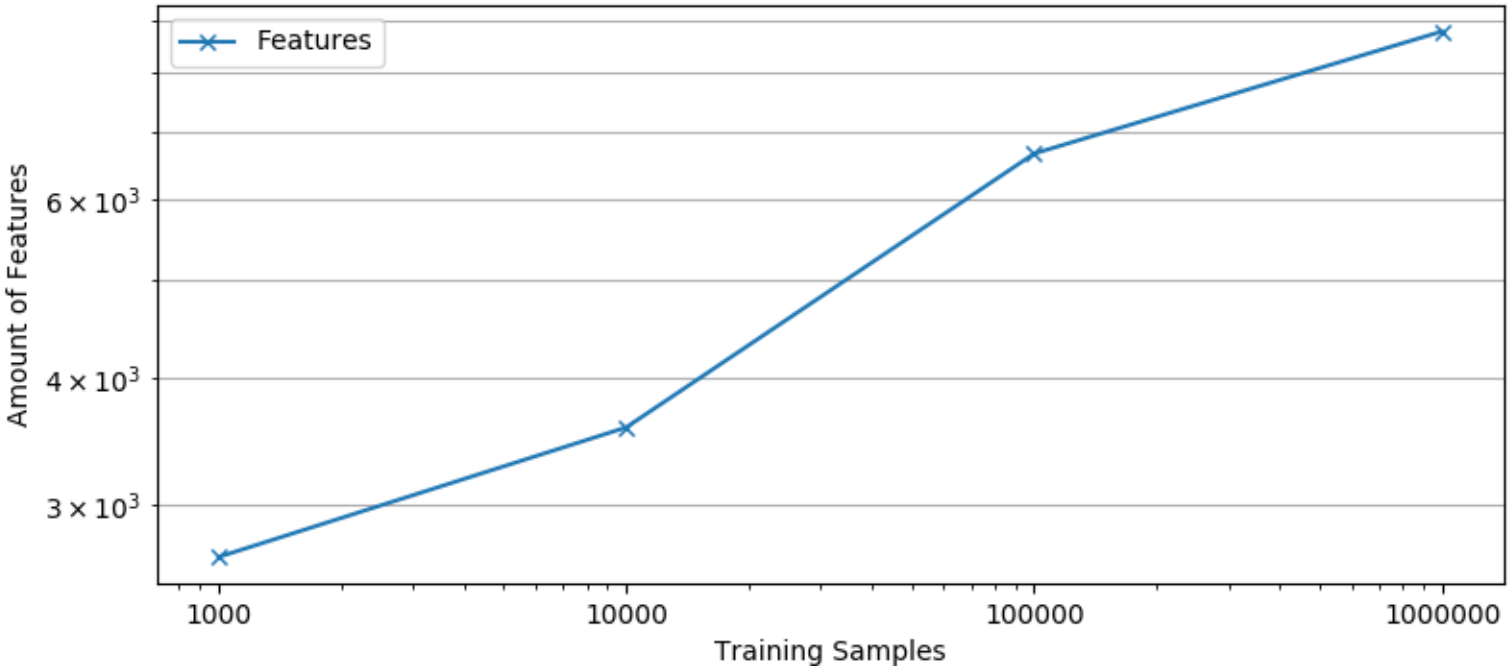
Intermediate models must not necessarily increase performance

Model Performance by Class



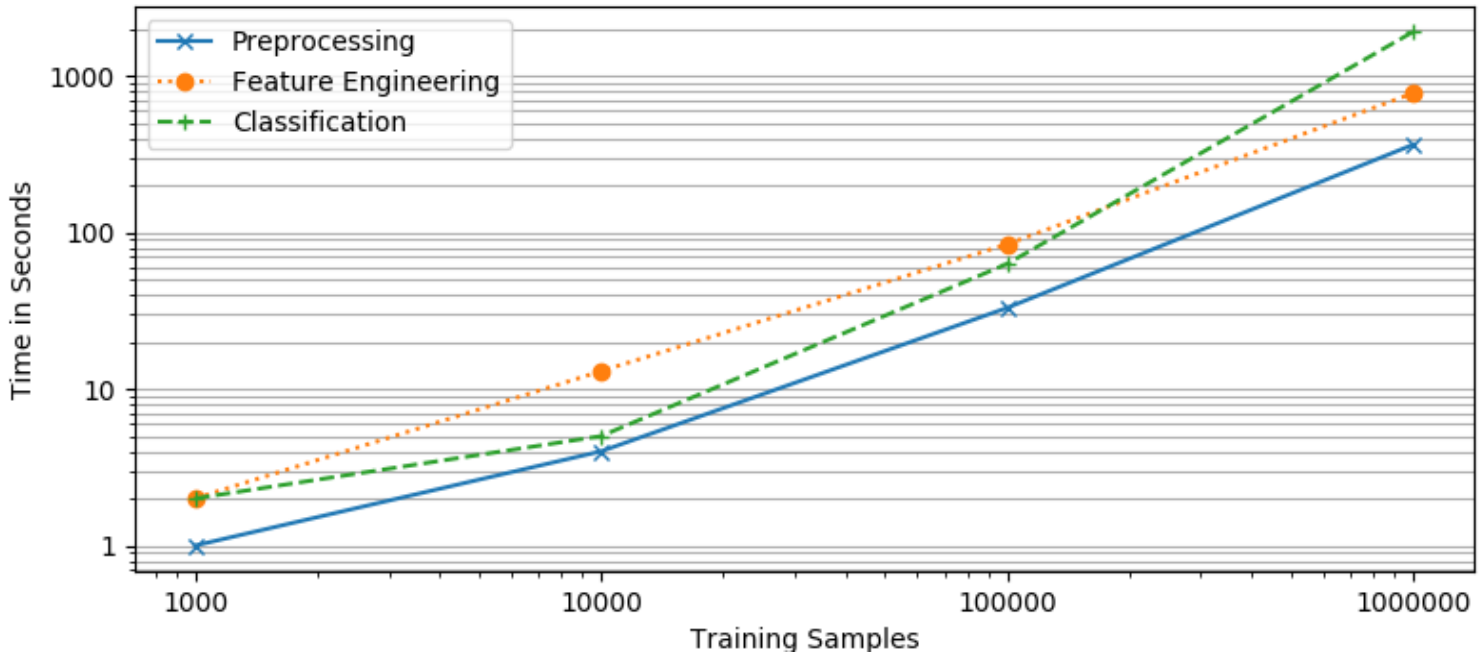
Class performance is independent of class distribution

Amount of Relevant Features



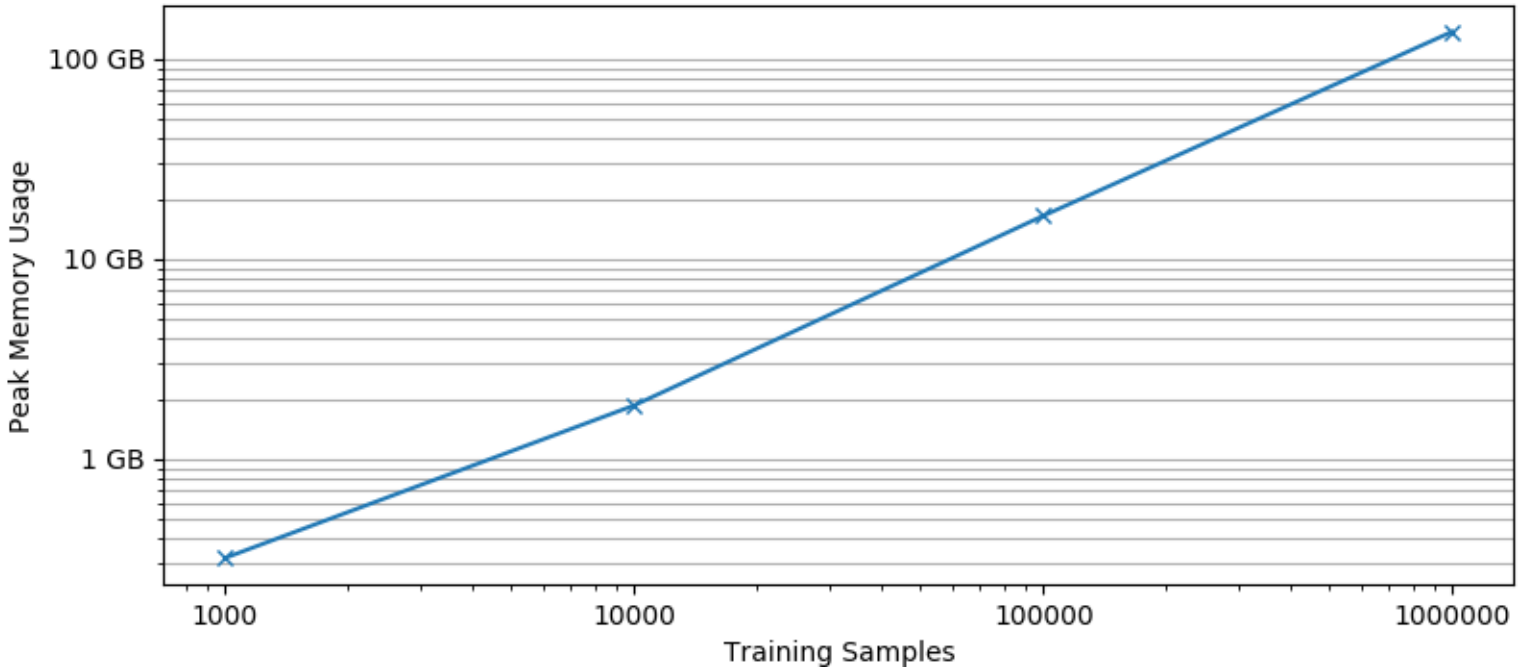
Amount of relevant features grows slowly

Model Runtime



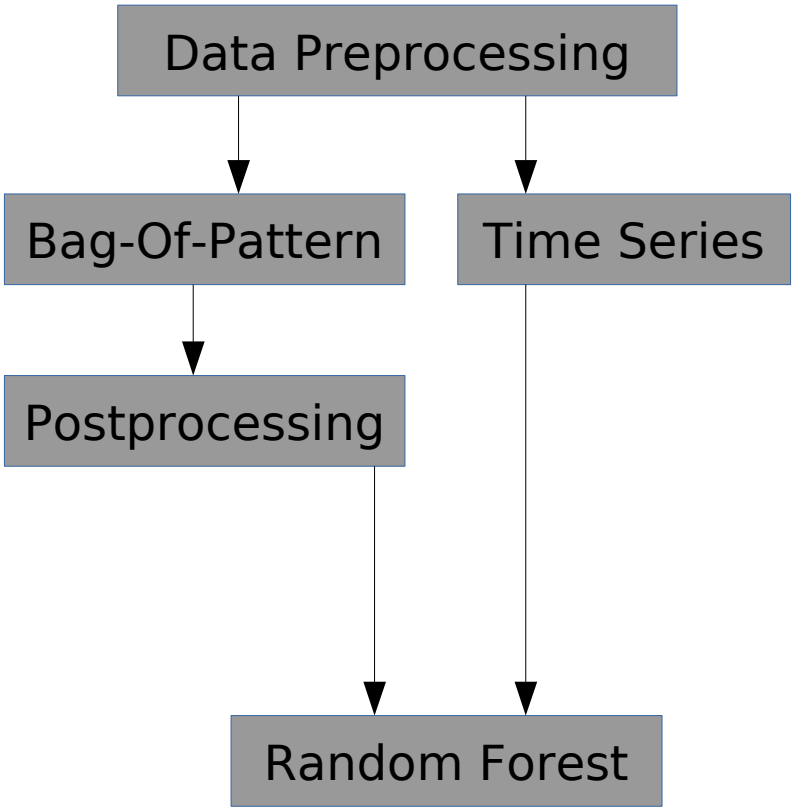
Runtime explodes with
>100k samples

Memory Peak Usage



Memory explodes with
>100k samples

Time Series Classification Pipeline Overview



- Implementation
  - numpy
  - sklearn
  - saxpy
- Scalability
  - 1M training samples
  - 30 processes
- Test Accuracy
  - F1-score: 0.7376
  - ~1 hour runtime