Sequence Input in KDE

Sequence Input in KDE

Can we take sequence input into account in KDE?

Straightforward approach: use multivariate KDE

- Treat each sequence as a vector variable
- Learn an estimator as usual

Individual sequences in the new dataset are treated as independent:

- This is due to the basic assumptions behind KDE
- In practice, by choosing a (sliding) window length...
- ...We make an assumption about how many observations need to be considered
- ...Before dependencies become negligible

Many ML approaches make the same assumptions

Sequences via Multivariate KDE

First, we separate the training set as usual:

In [3]:
```
wdata_tr = wdata[wdata.index < train_end]
```

Then we choose the bandwidth:

In [4]:
```
gs_kde = GridSearchCV(KernelDensity(kernel='gaussian'),
                      {'bandwidth': np.linspace(0.01, 0.1, 20)}, cv = 5)
gs_kde.fit(wdata_tr)
gs_kde.best_params_
```

Out[4]:
```
{'bandwidth': 0.02421052631578947}
```

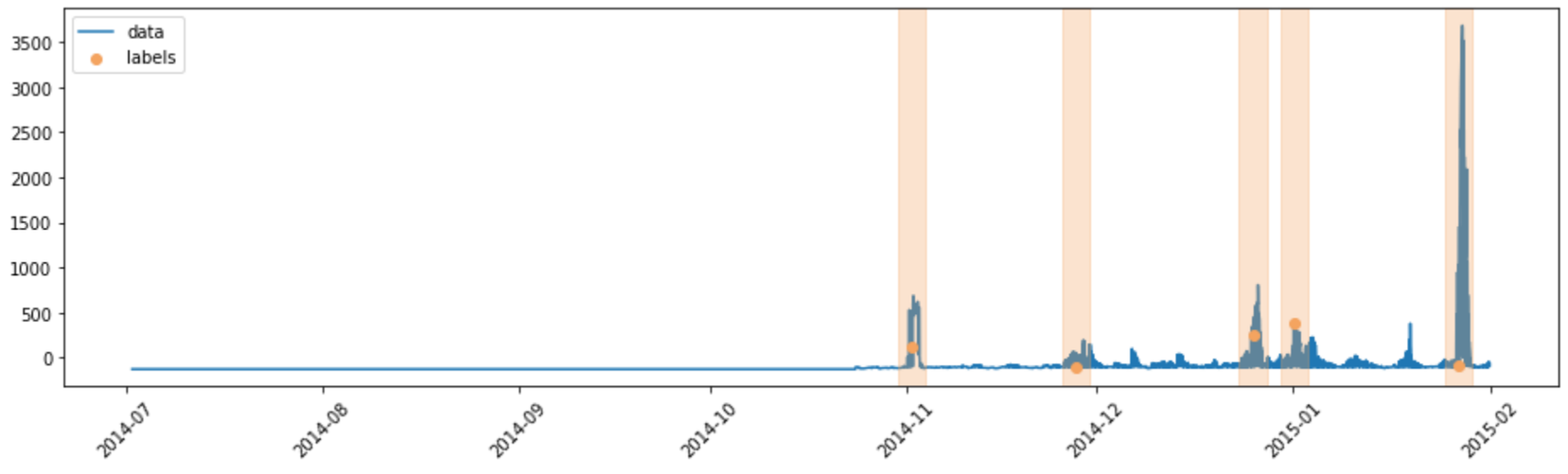
- This is an expensive operation
- In general, adding dimensions makes KDE slower

Sequences via Multivariate KDE

Now we can train a multivariate estimator and generate the alarm signal

In [5]:
```
h = gs_kde.best_params_['bandwidth']
kde = KernelDensity(kernel='gaussian', bandwidth=h)
kde.fit(wdata_tr); # ";" suppresses output
```

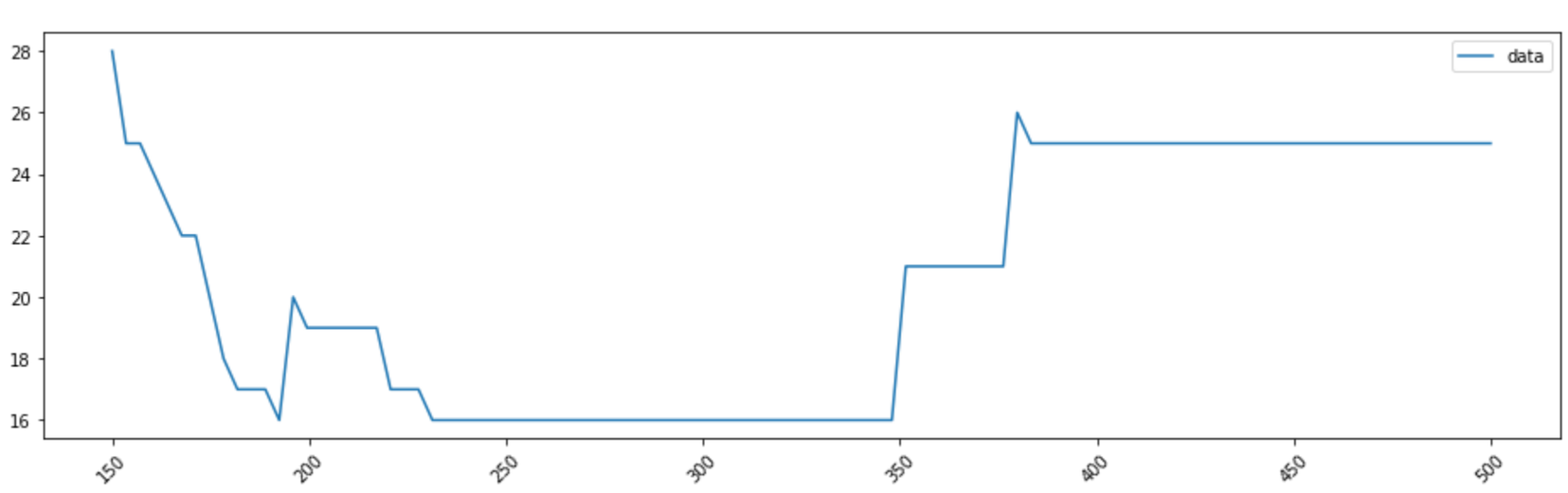
In [6]:
```
ldens = kde.score_samples(wdata)
signal = pd.Series(index=wdata.index, data=-ldens)
nab.plot_series(signal, labels, windows, figsize=figsize)
```



Effect of the Threshold

We can now look at the response surface (for a varying threshold)

In [7]:
```
thr_range = np.linspace(150, 500, 100)
cost_range = [cmodel.cost(signal, labels, windows, thr)
              for thr in thr_range]
cost_range = pd.Series(index=thr_range, data=cost_range)
nab.plot_series(cost_range, figsize=figsize)
```



Threshold Optimization

Finally, we can do threshold optimization as usual

In [8]:
```
signal_opt = signal[signal.index < val_end]
labels_opt = labels[labels < val_end]
windows_opt = windows[windows['end'] < val_end]

best_thr, best_cost = nab.opt_thr(signal_opt, labels_opt,
                                  windows_opt,  cmodel, thr_range)
print(f'Best threshold: {best_thr}, corresponding cost: {best_cost}')
```

```
Best threshold: 150.0, corresponding cost: 5
```

Cost on the whole dataset

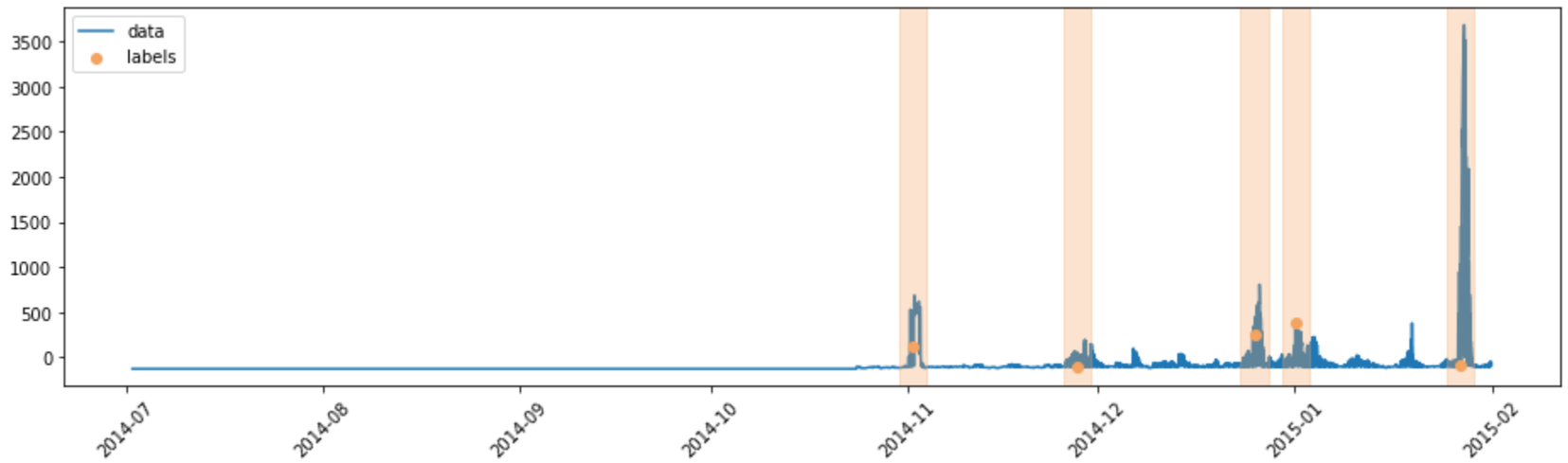
In [9]:
```
ctst = cmodel.cost(signal, labels, windows, best_thr)
print(f'Cost on the whole dataset {ctst}')
```

```
Cost on the whole dataset 28
```

Spotting Trouble

Let us look again at our alarm signal

In [10]:
```
nab.plot_series(signal, labels, windows, figsize=figsize)
```



- It is mostly, nice, but also very unstable
- It has frequent and wide oscillations

We will need to have a better look at that!

Considerations

Some considerations and take-home messages

Always prefer simpler approaches:

- They are typically easy to implement
- If they work, you win!
- If they do not work, you have a baseline
- ...And solid motivation for using a more powerful method

Always ask yourself:

Did I take advantage of all available information?