# 8th place with Kalman filters

posted in Web Traffic Time Series Forecasting 3 months ago

26

I also did quite well without using any neural networks, XGBoost or other modern methods. Here's an overview of my not-so-elegant but fully "classical" approach.

It's based on the observation that most of the time series are low-traffic, noisy and seemingly very unpredictable (figure 1) while some of them behave quite nicely (figure 2). My main idea was to use Kalman filters to predict well-behaved time series while falling back to a more robust median-of-medians for the bulk of the data.

I was heavily aided by the public kernels: my fall-back method was almost the same as the best public kernel: the Fibonacci median of medians, grouped by weekend vs weekdays, rounded to the nearest integer (this one https://www.kaggle.com/rshally/web-traffic-cross-valid-round-and-wk-lb-44-5).

In more detail, the solution worked as follows

1. For spiders, always use the rounded Fibonacci median of medians but *without weekly seasonality*. This was because the vast majority of the spider data was very noisy and I assumed that bots do not care about weekends. Leaving out weekly seasonality did not have much effect in my final CV benchmarks but I decided to use the simpler method since I thought it would be more robust to possible abnormalities caused by this competition.
2. For non-spider data, try three different smoothing parameters *s*, starting from least smoothing:
    - run a Kalman smoother *K(s)* on the log1p-transformed data to get the smoothed mean *y*
    - compute deltas: *dy* as `np.diff(y)`
    - compute the yearly seasonality error as `R = 0.5 * |dy1 - dy2| / (|dy1| + |dy2|)` , where *dy1* is the last year of *dy* and *dy2* the second last year of *dy* (i.e., compare the data to itself, shifted by one year, very similar to what Nathaniel Maddux seems to have done in his solution)
    - if *R* is less than a threshold value (0.95), the *s*-smoothed data is seasonal: predict the new data with the Kalman filter *K(s)* adding yearly seasonality from `np.cumsum(dy2)` . Finally apply `exp(x)-1` and round to the nearest integer
    - otherwise try the next smoothing level *s*
3. If the data is not seasonal with any smoothing level, fall back to rounded Fibonacci median of medians by weekend

I used a 8-state Kalman filter representing a local level (no trend) and weekly seasonality, parametrized by a smoothing parameter *s* which determined the covariance matrices (process and measurement noise). Adding the yearly seasonality directly to the Kalman filter would have exploded the number of states or required special tricks so I handled that separately as described above.

I wrote my own SIMD-style vectorized implementation of the Kalman filters which allowed running them relatively fast in Python (Numpy). EDIT: These custom Kalman filter codes can be found here https://github.com/oseiskar/simdkalman

The total execution time of the final model was about 25 minutes on a 4-core i5, which was nice since that was about the time I had available for dealing with the final data. My two submissions were the same model executed on the 8/30 and 9/10 data. The former scored only 39.2.

In the final cross-validation / back-testing I used 5 different truncations of the data and ensured that the submitted model worked well for *all* of these cases (i.e., max error instead of mean error). It had quickly became apparent to me that optimizing the public LB was the same as over-fitting for predicting January, which was quite different from predicting October or December.

Figures (red: predicted, dashed: actual)

📎 figure-1-noisy-low-traffic-page.png (169.05 KB)
📎 figure-2-nice-spanish-page-with-yearly-seasonality.png (124.23 KB)

**Oscar Takesh...** • (67th in this Competition) • 3 months ago • Options • Reply 1

It is nice to hear how well established approaches like Kalman filter worked. Congrats. What would be the running time without SIMD?

**os** • (8th in this Competition) • 3 months ago • Options • Reply 2

Good question. I ran an simplified benchmark with a small subset of the data and the SIMD version was about 7x faster. EDIT: published the Kalman filter code in Github: https://github.com/oseiskar/simdkalman

```
import kalman

# define model
level_noise = 0.2
season_noise = 1e-3
measurement_noise = 1
n_seasons = 7

state_transition = np.zeros((n_seasons+1, n_seasons+1))
state_transition[0,0] = 1 # steady level

# season cycle
state_transition[1,1:-1] = [-1.0] * (n_seasons-1)
state_transition[2:,1:-1] = np.eye(n_seasons-1)

# Initial state
initial_covariance = np.eye(n_seasons+1)
initial_state = np.zeros((n_seasons+1,1))

kf = kalman.Smoother(
    state_transition,
    process_noise = np.diag([level_noise, season_noise] + [0]*(n_seasons-1))**2,
    measurement_model = np.array([[1,1] + [0]*(n_seasons-1)]),
    measurement_noise = np.array([[measurement_noise**2]]))

def run_smoother(data):
    r = kf.compute_matrix(
        data,
        n_test=0,
        initial_value = initial_state,
        initial_covariance = initial_covariance,
        store_covariances = False)

    # smoothed data (expected observations) and smoothed level
    return (r.smoothed_observations, r.smoothed_means[:,:,0])
```

```
import timing_utils

@timing_utils.timed("simd")
def run_simd(data):
    return run_smoother(data)

@timing_utils.timed("non-simd")
def run_not_simd(data):
    obs = np.zeros(data.shape)
    level = np.zeros(data.shape)
    for j in range(data.shape[0]):
        obs[j,:], level[j,:] = run_smoother(data[j,:][np.newaxis,:])
    return obs, level
```

```
training_data = np.log(data.as_matrix()[100000:100100,:]+1)
training_data.shape
```

```
(100, 803)
```

```
obs, level = run_simd(training_data)
```

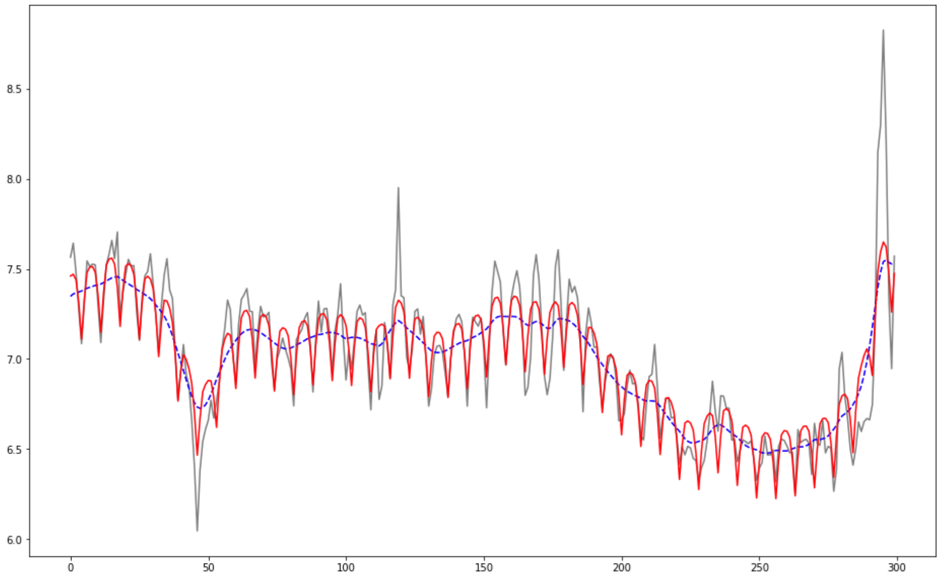
```
simd 1653.09 ms
```

```
obs, level = run_not_simd(training_data)
```

```
non-simd 11902.81 ms
```

## Here's an example of how the Kalman-smoothed data might look like

```
idx = 55
n_days = 300
plt.plot(training_data[idx,-n_days:],'gray')
plt.plot(level[idx,-n_days:],'b--')
plt.plot(obs[idx,-n_days:], 'r-')
plt.gcf().set_size_inches(16, 10)
```



Ben Ogorek • (269th in this Competition) • 2 months ago • Options • Reply 1

Very interested in your solution. I'm going through your simdkalman library now. Though your description above is very well written, would you be willing to add your code from the contest as well?

Ben Ogorek • (269th in this Competition) • 2 months ago • Options • Reply 1

Also, if getting the code ready is too much work, could you maybe share just the pieces that

1. Set up the 8-dimensional state vector
2. sent smoothing parameter s to the two covariance matrices?

os • (8th in this Competition) • 2 months ago • Options • Reply 1

Thanks! I appreciate your interest. The full solution code is a bit too messy to be published as is, but a wrote a Github gist demonstrating the usage of the new `simdkalman` package for the competition data and defining the matrices in the Kalman filter.