Forecasting Intermittent Demand with the Croston Model

Forecasting products with intermittent demand is complicated. I present here Croston's model that was specifically designed to forecast those time series.

Croston model

Initial Idea

In 1972, J.D. Croston published "Forecasting and Stock Control for Intermittent Demands," an article introducing a new technique to forecast products with intermittent demand. His idea could be summarized in three simple steps:
- Evaluate the average demand level when there is a demand occurrence.
- Evaluate the average time between two demand occurrences.
- Forecast the demand as the demand level (when there is an occurrence) multiplied by the probability of having an occurrence.

Let's do this step-by-step with an example:

1. If we estimate that we have a demand occurrence every four periods on average (i.e., a 25% chance to have an occurrence),
2. and we expect that —when there is a demand occurrence—the average demand level is five units,
3. then, we forecast the demand to be 5 x 0.25 = 1.25 per period going forward.

The Croston model answers the question ***How much demand will we have on average per period?***

Model

Now that we understand the basic idea of the Croston model, we need to answer two questions:

1. How do we estimate the probability of having a demand occurrence?

2. How do we estimate the demand level when there is an occurrence?

Croston used a technique close to the one used by Holt & Winters to answer these questions. For the different exponential smoothing models, we will be looking at both the **previous estimation** of each variable (i.e., level & periodicity) and its **most recent observation**.

Demand level
Let's note our level estimate **a** (like for the exponential models) and note the **actual** demand observations **d**. We will update our level estimate **a** only when we have an actual observation, so that

$$\text{if } d_t > 0, \text{ then } a_{t+1} = \alpha d_t + (1-\alpha) a_t$$

$$\text{if } d_t = 0, \text{ then } a_{t+1} = a_t$$

As in the different exponential smoothing models, we use a learning parameter alpha (0 < alpha < 1) to allocate more or less importance to the most recent observations or the historical ones.

Periodicity
Let's note the estimation of the time between two demand occurrences **p** (for **p**eriodicity), and the time elapsed since the previous demand occurrence **q**. We will only update **p** when we have a demand occurrence

$$\text{if } d_t > 0, \text{ then } p_{t+1} = \alpha q + (1-\alpha) p_t$$

$$\text{if } d_t = 0, \text{ then } p_{t+1} = p_t$$

Note that we again use alpha as the learning parameter. This is the same parameter we used to estimate the demand level.

Forecast

The forecast is straightforward: it is the demand level (**a**) divided by the periodicity (**p**).

$$f_{t+1} = \frac{a_t}{p_t}$$

Summary

We can then summarize the whole model.

$$\text{if } d_t > 0 \begin{cases} a_{t+1} = \alpha d_t + (1-\alpha) a_t \\ p_{t+1} = \alpha q + (1-\alpha) p_t \\ f_{t+1} = \frac{a_t}{p_t} \end{cases}$$

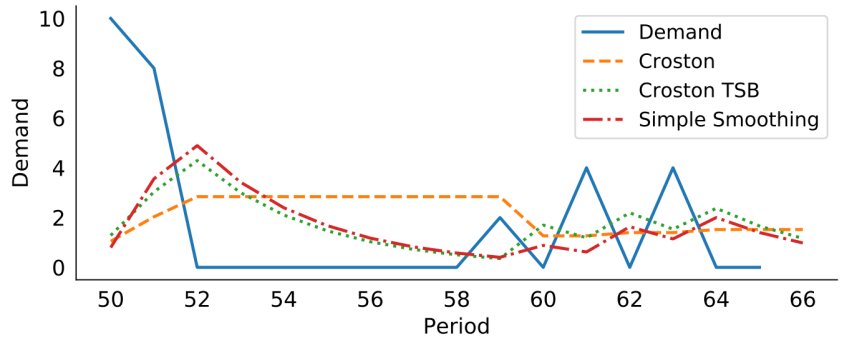
$$\text{if } d_t = 0 \begin{cases} a_{t+1} = a_t \\ p_{t+1} = p_t \\ f_{t+1} = f_t \end{cases}$$

As you can see, when there is no demand observation, none of the parameters are updated. This will be one of the limitations of this model.

Insights

Croston vs. Simple Exponential Smoothing

The most significant addition of the Croston method compared to the simple exponential smoothing is its ability to estimate the time between two demand occurrences. Let's see in the figure below if this helps us to forecast intermittent products.



Unfortunately, MAE is around the same value for both models (with a slight advantage in favor of the simple smoothing method): 139% for Croston and 130% for Simple Smoothing.

Limitations

As you can see in the graph above, when there is no demand occurrence, the Croston forecast does not change. This is weird for two reasons.

1. An extended period without demand occurrences should be an incentive to reduce our forecast.

2. When we have the first demand occurrence after a long period without one (look at periods 52–60), we decrease the forecast.

This is counter-intuitive. Imagine we are in period 51: you just sold 8 units, so you update your forecast to 3 pieces/period. From period 52 to 58, you won't sell a single piece, but you will **not** update your forecast once. Actually, you will have to wait for the 2-piece sale in period 59 to wake up and decrease your forecast to 2 pieces.
This does not make any sense. Why couldn't we update our forecast during the periods without sales to reflect the idea that we expect our demand to be lower?

Improving Croston

In 2011, in their paper "Intermittent demand: Linking forecasting to inventory obsolescence," Ruud Teunter, Aris Syntetos, and Zied Babai proposed an improvement to the Croston model. Their idea was to allow the model to update (*decrease*) its periodicity estimate even if there is no demand observation.
That's perfect as this limitation was the main one we had with the vanilla Croston model.

TSB Model
Level—The TSB model (for **T**eunter, **S**yntetos & **B**abai) does not change how the level is estimated compared to the regular Croston model.
Periodicity—The periodicity **p** will now be expressed as the probability of having a demand occurrence. Actually, **p** will now denote a frequency (i.e., a probability) ranging from 0 (demand never occurs) to 1 (demand occurs at every period). We will update this periodicity at every period——— even if there is no demand occurrence.
The periodicity will
- **decrease** if there is no demand occurrence (as you expect a reduction in the probability for a demand to occur). This decrease will be exponential (like for all the exponential smoothing models);
- **increase** if there is a demand occurrence.

We then have

$$\text{if } d_t > 0 \text{ then } p_{t+1} = 1 \times \beta + (1-\beta)p_t$$
$$\text{if } d_t = 0 \text{ then } p_{t+1} = 0 \times \beta + (1-\beta)p_t$$

which can be simplified into

$$\text{if } d_t > 0 \text{ then } p_{t+1} = \beta + (1-\beta)p_t$$
$$\text{if } d_t = 0 \text{ then } p_{t+1} = (1-\beta)p_t$$

Remember that in the vanilla Croston model, **p** was the expected number of periods between two demand occurrences.

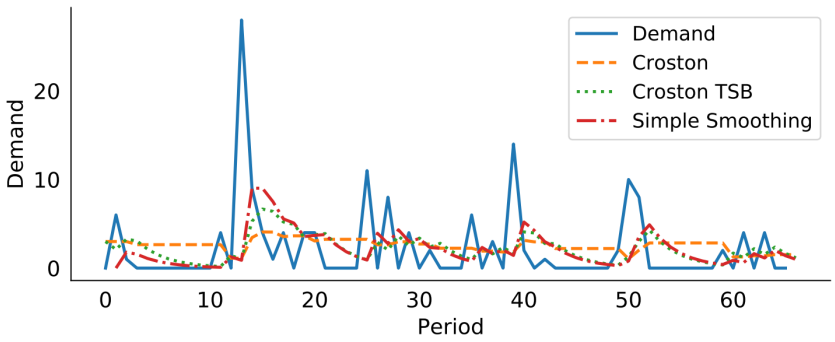
Summary We now obtain this set of equations for the TSB model.

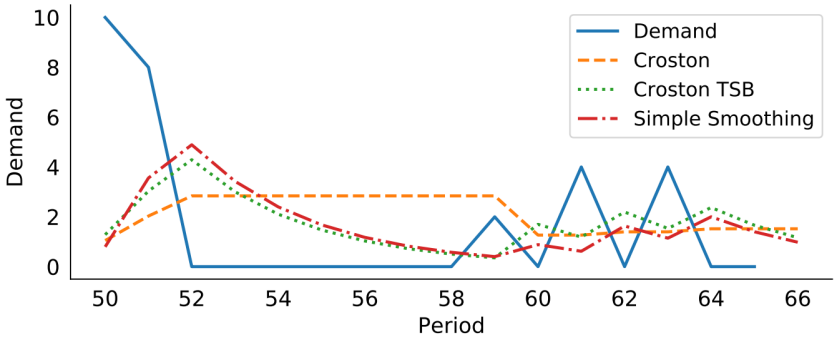
$$\text{if } d_t > 0 \begin{cases} a_{t+1} = \alpha d_t + (1-\alpha)a_t \\ p_{t+1} = \beta + (1-\beta)p_t \\ f_{t+1} = p_{t+1}a_{t+1} \end{cases}$$

$$\text{if } d_t = 0 \begin{cases} a_{t+1} = a_t \\ p_{t+1} = (1-\beta)p_t \\ f_{t+1} = p_{t+1}a_{t+1} \end{cases}$$

Pay attention:
- the forecast **f** is defined as the periodicity **p multiplied** by the level **a** (and not divided by it, as in the original model);
- the forecast for **t+1** is defined based on the level and periodicity estimates of **t+1** (and not **t**).

Insights
Let's compare our new TSB model against the original Croston model and a simple smoothing. As you can see in the two graphs below, TSB and the simple model are very similar. Actually, the difference lies in the fact that the TSB model keeps track of both the demand level and the demand probability. In contrast, the simple smoothing model only keeps track of the level (which includes the idea of the likelihood for a demand to occur).





We now obtain a mean absolute error of 134% thanks to TSB. The simple smoothing model seems to be still slightly better than the two others for this product. Of course, this is just an example, and Croston/TSB might work better on other products.

Do It Yourself

You can find below an implementation in python of both Croston & TSB models.

```
def Croston(ts,extra_periods=1,alpha=0.4):
```

```
d = np.array(ts) # Transform the input into a numpy array
    cols = len(d) # Historical period length
    d = np.append(d,[np.nan]*extra_periods) # Append np.nan
into the demand array to cover future periods

    #level (a), periodicity(p) and forecast (f)
    a,p,f = np.full((3,cols+extra_periods),np.nan)
    q = 1 #periods since last demand observation

    # Initialization
    first_occurence = np.argmax(d[:cols]>0)
    a[0] = d[first_occurence]
    p[0] = 1 + first_occurence
    f[0] = a[0]/p[0]
```

```
# Create all the t+1 forecasts
    for t in range(0,cols):
        if d[t] > 0:
            a[t+1] = alpha*d[t] + (1-alpha)*a[t]
            p[t+1] = alpha*q + (1-alpha)*p[t]
            f[t+1] = a[t+1]/p[t+1]
            q = 1
        else:
            a[t+1] = a[t]
            p[t+1] = p[t]
            f[t+1] = f[t]
            q += 1

    # Future Forecast
    a[cols+1:cols+extra_periods] = a[cols]
    p[cols+1:cols+extra_periods] = p[cols]
    f[cols+1:cols+extra_periods] = f[cols]

    df =
pd.DataFrame.from_dict({"Demand":d,"Forecast":f,"Period":p,"
Level":a,"Error":d-f})
```

```
return df
```

```
def Croston_TSB(ts,extra_periods=1,alpha=0.4,beta=0.4):
```

```
d = np.array(ts) # Transform the input into a numpy array
    cols = len(d) # Historical period length
    d = np.append(d,[np.nan]*extra_periods) # Append np.nan
into the demand array to cover future periods

    #level (a), probability(p) and forecast (f)
    a,p,f = np.full((3,cols+extra_periods),np.nan)


# Initialization
    first_occurence = np.argmax(d[:cols]>0)
    a[0] = d[first_occurence]
    p[0] = 1/(1 + first_occurence)
    f[0] = p[0]*a[0]

    # Create all the t+1 forecasts
    for t in range(0,cols):
        if d[t] > 0:
            a[t+1] = alpha*d[t] + (1-alpha)*a[t]
            p[t+1] = beta*(1) + (1-beta)*p[t]
        else:
            a[t+1] = a[t]
            p[t+1] = (1-beta)*p[t]
        f[t+1] = p[t+1]*a[t+1]

    # Future Forecast
    a[cols+1:cols+extra_periods] = a[cols]
    p[cols+1:cols+extra_periods] = p[cols]
    f[cols+1:cols+extra_periods] = f[cols]

    df =
pd.DataFrame.from_dict({"Demand":d,"Forecast":f,"Period":p,"
Level":a,"Error":d-f})
```

```
return df
```

```
def Croston(ts,extra_periods=1,alpha=0.4):
```

```
d = np.array(ts) # Transform the input into a numpy array
    cols = len(d) # Historical period length
    d = np.append(d,[np.nan]*extra_periods) # Append np.nan
into the demand array to cover future periods

    #level (a), periodicity(p) and forecast (f)
    a,p,f = np.full((3,cols+extra_periods),np.nan)
    q = 1 #periods since last demand observation

    # Initialization
    first_occurence = np.argmax(d[:cols]>0)
    a[0] = d[first_occurence]
    p[0] = 1 + first_occurence
    f[0] = a[0]/p[0]
```

```
# Create all the t+1 forecasts
    for t in range(0,cols):
        if d[t] > 0:
            a[t+1] = alpha*d[t] + (1-alpha)*a[t]
            p[t+1] = alpha*q + (1-alpha)*p[t]
            f[t+1] = a[t+1]/p[t+1]
            q = 1
        else:
            a[t+1] = a[t]
            p[t+1] = p[t]
            f[t+1] = f[t]
            q += 1

    # Future Forecast
    a[cols+1:cols+extra_periods] = a[cols]
    p[cols+1:cols+extra_periods] = p[cols]
    f[cols+1:cols+extra_periods] = f[cols]

    df =
pd.DataFrame.from_dict({"Demand":d,"Forecast":f,"Period":p,"
Level":a,"Error":d-f})

return df
```

About the author

👉 Let's connect on LinkedIn!

Nicolas Vandeput — Consultant, Founder — SupChains | LinkedIn

View Nicolas Vandeput's profile on LinkedIn, the world's largest professional community. Nicolas ...

www.linkedin.com

Nicolas Vandeput is a supply chain data scientist specialized in demand forecasting and inventory optimization. He founded his consultancy company SupChains in 2016 and co-founded SKU Science —a fast, simple, and affordable demand forecasting platform—in 2018. Passionate about education, Nicolas is both an avid learner and enjoys teaching at universities: he has taught forecasting and inventory optimization to master students since 2014 in Brussels, Belgium. Since 2020, he is also teaching both subjects at CentraleSupelec, Paris, France. He published *Data Science for Supply Chain Forecasting* in 2018 (2nd edition in 2021) and *Inventory Optimization: Models and Simulations* in 2020.