Journal of Fundamentals of Renewable Energy and Applications

Croonenbroeck and Ambach, J Fundam Renewable Energy Appl 2015, 6:1
DOI: 10.4172/2090-4541.1000195

Research Article Open Access

An Implementation of the Mycielski Algorithm as a Predictor in R

Carsten Croonenbroeck[1*] **and Daniel Ambach**[2]

[1]*Department of Environmental Science, University of Rostock, Germany*
[2]*Department of Statistics, European University, Germany*

Abstract

Univariate time series analysis is usually performed by arbitrarily complex parametric modeling. At least for prediction, a simple non-parametric alternative is the Mycielski algorithm, a forecasting method based on pat- tern matching. The reproducible research presented here shows how to perform out of sample forecasts using the methodology of Mycielski. The algorithm provides well results in scenarios where usual univariate models such as ARIMA family models return limited accuracy. In this article we describe the idea of the Mycielski based prediction algorithm in general. We contribute a reference implementation in R and give a short example.

Keywords: Time series analysis; Predictor; Forecasting; Mycielski; R

Introduction

Univariate time series modeling has a history that lasts for decades. Modern modeling approaches are highly specialized in the applications for which they are designed. Although many models are rather complex, most of them are still based on autoregressive processes, e.g. ARMA or ARIMA [1] and/or ARCH and GARCH models [2,3]. The usefulness of these models is quite indisputable. They estimate ("learn") their parameters from historical data and thus, provide important information on the latent data generating process. ARMA family based models have been applied to manifold problems in practice. For example, Valipour et al. [4] show how to forecast dam inflow data based on ARMA and ARIMA models. In addition, Valipour et al. [5] compare the results to artificial neural network forecasts.

ARMA based models are widely accepted forecasting tools. The base models can be extended in many aspects. A usual extension to the model is the inclusion of seasonal effects (SARIMA, seasonal ARIMA), as applied to, e.g., hydrology research by Valipour [6]. However, the challenge is to find the proper model type, specification and parametrization for a given data set. Particularly in cases at which the model fit is not of interest per se but is only necessary to calculate out of sample forecasts of the data, incorrect modeling may lead to vastly false predictions. Furthermore, the model's accuracy depends on a sufficient number of historical observations. Valipour [7] determines the number of observations necessary for reliable results in a practical example. To overcome this hazard, an alternative idea is to use those historical observations as prediction values that are the most likely ones, dependent on the respective state of the time series data. The outcome of this idea is a predictor known as Mycielski algorithm. This forecasting method is based on pattern matching. Other pattern matching algorithms than the Mycielski approach has been researched. For example, Croonenbroeck [8] shows how to pick an optimal match by a local entropy criterion. Samy et al. [9] provide an overview of artifical neural network based pattern matching as well as CLONAL Propagation, which is designed to simulate a natural biology immune system.

The Mycielski algorithm is developed by Ehrenfeucht and Mycielski [10]. It is originally proposed as a pseudo random number generator. Jacquet et al. [11] describe a modification of this algorithm to impose a universal predictor. Besides, Hocaoglu et al. [12] use the method for wind speed prediction, and compare their results to the data measured at three different sites in Turkey.

Gan et al. [13] apply the method to another three more wind speed measurement stations. More recently, Croonenbroeck and Ambach [14] apply the algorithm to wind power data and compare this predictor to sophisticated parametric models.

The Mycielski algorithm is similar to Markov chain models. These models build up transition probabilities from one condition to another. Thus, they learn from the history of the whole data set. In contrast, the Mycielski algorithm directly picks values out of the history as predictors. It uses the successor of the longest matching pattern in the entire history of data that is identical to the chain in the most recent history. This straightforward approach comes to reliable results as long as there is a large number of historical observations available. Additionally, the results are quite well if there is some periodicity in the data, i.e. if patterns return after a certain while. While there are many ways at hand to model periodicity, e.g. by Fourier series, B-Splines or SARIMA models, all of these approaches require a proper parametrization. The Mycielski approach, however, does not, since it is a non-parametric procedure. If, in contrast, the data set is highly non-seasonal, it cannot be predicted well using the Mycielski approach.

This article contributes a reference implementation of the Mycielski algorithm in R. In addition, we provide an example where we use the Mycielski algorithm as predictor. The paper is structured as follows. Section 2 provides a short introduction of the methodology. Section 3 discusses the implementation in R. Section 4 presents a real world example and Section 5 concludes.

Mycielski Algorithm

Assume a time series $\{x_t\}_{t} \mathbb{N}$. Now, a forecast for time $t+k$ is desired, based on historical data for time $1, 2, \ldots, t$. The Mycielski

***Corresponding author:** Carsten Croonenbroeck, Department of Environmental Science, University of Rostock, Justus-von-Liebig-Weg 2, 18059 Rostock, Germany, Tel: +49/381/498-3267; Fax: +49/381/498-3262; E-mail: carsten.croonenbroeck@uni-rostock.de

Received October 13, 2015; **Accepted** December 02, 2015; **Published** December 04, 2015

Citation: Croonenbroeck C, Ambach D (2015) An Implementation of the Mycielski Algorithm as a Predictor in R. J Fundam Renewable Energy Appl 6: 195. doi:10.4172/20904541.1000195 algorithm picks a short pattern from the most recent history of the data and searches the entire his- tory for that pattern. Say, the pattern is denoted as $\mathbf{p}$ R^d and consists of d elements. In the simplest case, $d = 1$, so $\mathbf{p} = x_t$. Every successor of historical observations that are identical to x_t is then a possible predictor for time $t+1$. In the more general case, we observe that the kth successor of that observation would be the predictor. Whenever multiple candidates are found, d is increased by 1, so in the second step, $d = 2$ and $\mathbf{p} = (x_{t-1}, x_t)$. That pattern is searched again. The algorithm continues until the longest matching chain is found or a maximum chain length limit is reached. As soon as no longer chain can be found, the longest previously found chain is taken.

The default assumption of our function is that the respective successor of the longest matching chain is the most probable predictor. However, Jacquet et al. [11] argue that a fractional maximal suffix is used to build a universal predictor. Formally, Jacquet et al. [11] as well as Hocaoglu et al. [12] note the algorithm as follows. The forecast is calculated as

$$\hat{x}_{t+k} = f_{t+k}(x_1, \ldots, x_t), \tag{1}$$

Where $f(\cdot)$ performs the iterative search algorithm: Let μ be the index of the kth successor of the longest matching string, which itself ends at index o. Then μ is found by

$$\mu = \arg\max\left(x_t = x_o, x_{t-1} = x_{o-1}, \ldots, x_{t-L+1} = x_{o-L+1}\right) \tag{2}$$

Where L denotes chain length. Thus, it follows that $f_{t+k}(x_1, \ldots, x_t) = \hat{x}_{t+k} = x_\mu$.[1]

The data is usually denoted in decimal values. Therefore, it is unlikely to find long strings that exactly match the search pattern. To overcome this, our implementation does not search for exact matches in the history, but only for similar ones. This fuzzy search method can be applied by searching for chains that may not match exactly, but only lie within a certain tolerance interval. Our code allows for choosing a proper tolerance level when calling the prediction function. We denote the one-sided tolerance level as τ. Moreover, we assume the tolerance interval to be symmetric, so the tolerance interval width is 2τHowever, a generalization for non-symmetric intervals is straightforward.

Note: [1]If L extends to the entire set of observations, no prediction can be found. Thus, for real-world scenarios, one could incorporate an upper limit for L and if it is reached, the most recent value of observations could be taken as a forecast (martingale assumption).

Implementation in R

The main function expects the committed data object to be a simple numeric vector. Vectors of type "ts" as defined in package t-series can also be used. The code aims at generating pseudo out of sample (OOS) forecasts to compare those forecasts to the actual values. Hence, a modification for usage as a plain out of sample forecaster for real world problems would be straightforward, and be basically a simplification of the code.

The main function expects five mandatory arguments: The data as a numeric vector (Data), the index of the vector element for which a forecast is to be found (Desired), the forecasting horizon (Steps, basically k, as discussed in section 2), the level of tolerance (Tolerance, τ) and an upper chain length level (MaxLength).

The function, as provided in listing 1, begins by preparing the return object (of type list ()), isolates the in sample data set and then calls the actual search function MSearch (). The returned object from that function contains the forecast value and the length of the longest matching chain. Finally, three pieces of information are returned from the main function: The pseudo OOS

forecast, the respective actual value and the chain length (Listing 1).

The search function, as in listing 2, sets up a for loop that increases the pattern length d by one at each cycle. The matching pattern is generated and the in sample data is prepared. Subsequently, the low-level function IsIdentical () is called. This function performs the search and comparison run and simply returns the index μ of the historical value that succeeds the longest chain. As aforementioned and by default, the function searches for the longest chain length, but it is possible to set up a shorter matching pattern. The best forecast may depend on the pattern length and the tolerance, so therefore it is possible to adjust τ and to limit d.

Whenever Is Identical () is unable to find a matching chain, the returned object is of length 0 and function MSearch () deals with this accordingly (Listing 2).

Finally, the function Is Identical () begins by setting up the tolerance interval. Then, a cascaded for loop is launched that runs through the data (outer loop) and through the pattern (inner loop). To conserve computing time, the inner loop is aborted as soon as one of the observations does not fit the pattern. The returned object either contains the index μ or is an empty object, if no match could have been found. Listing 3 shows the code (Listing 3).

For convenience, we also provide a wrapper function GetMycielski () that expects the data set. The beginning and ending of the pseudo out of sample period (Begin and End) must be provided. Furthermore, the function allows for choosing the forecasting horizon, tolerance,

```
GetForecast = function(Data, Desired, Steps, Tolerance, MaxLength)
    {
    RetObj = list()

    Temp = Data[1:(Desired - Steps)]
    Obj = MSearch(Data = Temp, Tolerance = Tolerance, MaxLength =MaxLength)

    Forecast = Data[Obj$Results[1] + (Obj$Length - 1) + (Steps - 1)]
    Actual = Data[Desired]

    RetObj$Forecast = Forecast
    RetObj$Actual = Actual
    RetObj$Len = Obj$Length

    return(RetObj)
}
```

Listing 1: The main function, GetForecast().

```
MSearch = function(Data, Tolerance, MaxLength) {
    Findings = as.numeric()
    ReturnList = list()
    for (Len in 1:MaxLength)
    {
        MyPattern = Data[(length(Data) - Len + 1):length(Data)]
        Search = Data[1:(length(Data) - Len)]

        DidFind = IsIdentical(Search, MyPattern, Tolerance)

        if (length(DidFind) == 0)
        {
            ReturnList$Length = Len
            ReturnList$Results = Findings
            return(ReturnList)
        }else
        {
            Findings = DidFind
            ReturnList$Length = Len
            ReturnList$Results = Findings
        }
    }
    ReturnList$Length = Len
    ReturnList$Results = Findings
    return(ReturnList)
}
```

Listing 2: Function MSearch().

minimum chain length and maximum chain length (Steps, Tolerance, MinLength and MaxLength). The latter four parameters are optional and are set to default values, if not provided.

The function, see listing 4, performs the search. Moreover, it provides progress information during the computation and returns a list object that contains fore- casts, actual values and the lengths of the observed chains for each forecast. All entries for which no matching chain was found (this also includes the case that found chains are shorter than the provided parameter MinLength) are set to NA (Listing 4).

Out of Sample Predictions

As an example, we use a publicly accessible data set [15]. The data describe the monthly production of milk in pounds per cow in the United States, time frame January 1962 through December 1975 (168 observations). The data has originally been provided by Cryer [16] and is also available via R package fma by Makridakis et al. [17].

```
IsIdentical = function(Find, Pattern, Tolerance = 0) {
    Plus = Pattern + Tolerance
    Minus = Pattern - Tolerance

    n = length(Find)
    k = length(Pattern)

    Found = as.numeric()
    FoundCounter = 0
    FoundIndicator = FALSE
    for (i in 1:(n - k))
    {
        for (j in 1:k)
        {
            if (Find[i + j - 1] >= Minus[j] & Find[i + j - 1] <=Plus[j])
            {
                FoundIndicator = TRUE
            }else
            {
                FoundIndicator = FALSE
                break
            }
        }
        if (FoundIndicator)
        {
            FoundCounter = FoundCounter + 1
            Found[FoundCounter] = i
        }
    }
    return(Found)
}
```

Listing 3: Function IsIdentical().

Panel (a) in (Figure 1) presents the time series plot of the data. The structure suggests a strong yearly periodic pattern and an overall increasing trend. We use the first 129 observations as an in sample data set and seek to obtain out of sample forecasts for observations 130 through 168.

We choose the tolerance $\tau = 2$, and for the minimum chain length we set $d = 2$. The function auto.arima () from package forecast suggests an $ARIMA$ (1, 1, 3) model for the data. Thus, we compare our Mycielski forecasts to forecasts generated from an $ARIMA$ (1, 1, 3) model fit using functions arima () and predict t (). The results of the out of sample performance are shown in panel (b) of Figure 1.

While the sparse parametrization of the ARIMA model rather

```
GetMycielski = function(Data, Begin, End, Steps = 1, Tolerance = 0,
MinLength = 0, MaxLength = 10) {
    OldPercent = 0

    Forecasts = as.numeric()
    Actual = as.numeric()
    Len = as.numeric()

    n = End - Begin + 1
    j = 1

    MyText = "0%."
    cat(MyText)
    flush.console()

    for (i in Begin:End)
    {
        DoRun = GetForecast(Data = Data, Desired = i, Steps = Steps, Tolerance = Tolerance, MaxLength = MaxLength)
        if (length(DoRun$Forecast) == 0) DoRun$Forecast = 0
        Forecasts[j] = DoRun$Forecast
        Actual[j] = DoRun$Actual
        Len[j] = DoRun$Len
        if (Len[j] < MinLength) Forecasts[j] = NA

        Percent = round(j / n * 100)

        if (Percent != OldPercent)
        {
            b = nchar(MyText)
            c = rep("\b", b)
            cat(c, sep = "")
            flush.console()

            OldPercent = Percent

            MyText = paste(Percent, "%.", sep = "")
            cat(MyText)
            flush.console()
        }

        j = j + 1
    }
    ReturnObjAll = list()
    ReturnObjAll$Forecast = Forecasts
    ReturnObjAll$Actual = Actual
    ReturnObjAll$Len = Len

    cat("\n")
    flush.console()

    return(ReturnObjAll)
}
```

Listing 4: Wrapper function GetMycielski ().

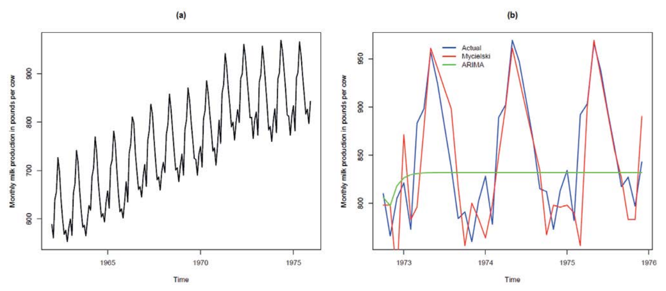

Figure 1: Milk production over time. The entire sample from January 1962 to December 1975 is shown on the left hand side (a). The actual values, Mycielski predictions and ARIMA predictions (out of sample) from October 1972 to December 1975 are shown on the right hand side (b).

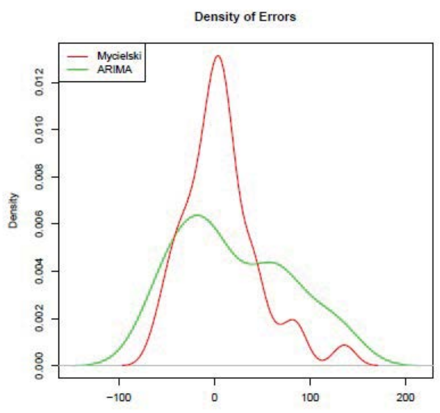

Figure 2: Densities of forecasting errors for the Mycielski approach and the ARIMA model.

early leads to the typical behavior of following the mean of the process, the Mycielski forecasts follow the actual data's periodic pattern. Root Mean Squared Error (RMSE) for Mycielski is $RMSE_{\text{Mycielski}} = 250.025$, and for the ARIMA model we obtain $RMSE_{\text{ARIMA}} = 386.803$. In addition, we obtain $MAE_{\text{Mycielski}} = 28.692$ and $MAE_{\text{ARIMA}} = 49.568$ for the respective Mean Absolute Errors and $MAPE_{\text{Mycielski}} = 3.428$ and $MAPE_{\text{ARIMA}} = 5.659$ for the respective mean Absolute Percentage Errors. To provide more insight into the distribution of the errors, (Figure 2) provides the kernel density estimates of both series of errors. Finally, listing 5 provides our analysis code (Listing 5).

Conclusion

The pattern matching approach, i.e. the Mycielski algorithm, provides a simple non-parametric approach for univariate data forecasting. No fitting, i.e. no parameter estimation is necessary, as the algorithm "learns" from the data on the fly. Our straightforward and reproducible code can serve as a reference implementation for methods comparison and improvements on the Mycielski idea.

```
library(zoo)

plot(Milk$Month, Milk$Milk, xlab = "Time", ylab = "Monthly milkproduction in pounds per cow", type = "l", lwd = 2)

##

Mycielski = GetMycielski(Milk$Milk, Begin = 130, End = 168, Steps
      = 1, Tolerance = 2, MinLength = 2)

Mycielski$Forecast = na.approx(Mycielski$Forecast)
Mycielski$Month = Milk$Month[130:168]

ARIMA.fit = arima(Milk$Milk[1:129], order = c(1, 1, 3))
ARIMA.forecast = predict(ARIMA.fit, n.ahead = 39)

plot(Mycielski$Month, Mycielski$Actual, xlab = "Time", ylab = "
      Monthly milk production in pounds per cow", type = "l", lwd =
      2, col = "blue")
lines(Mycielski$Month, Mycielski$Forecast, lwd = 2, col = "red")
lines(Mycielski$Month, ARIMA.forecast$pred, lwd = 2, col = "green"
      )

legend(x = 108356315, y = 970, c("Actual", "Mycielski", "ARIMA"),
      col = c("blue", "red", "green"), lty = 1, bty = "n", lwd = 2)

library(forecast)

accuracy(Mycielski$Forecast, Mycielski$Actual)
accuracy(ARIMA.forecast$pred, Mycielski$Actual)

Mycielski$Errors = Mycielski$Actual - Mycielski$Forecast
ARIMA.errors = Mycielski$Actual - ARIMA.forecast$pred

a = density(Mycielski$Errors)
MLimX = c(min(a$x), max(a$x))
MLimY = c(min(a$y), max(a$y))
a = density(ARIMA.errors)
ALimX = c(min(a$x), max(a$x))
ALimY = c(min(a$y), max(a$y))

MyLimX = c(min(MLimX[1], ALimX[1]), max(MLimX[2], ALimX[2]))
MyLimY = c(0, max(MLimY[2], ALimY[2]))

plot(density(ARIMA.errors), xli
      m = MyLimX, ylim = MyLimY, m
      ain = " Density of Errors", c
      ol = "green", lwd = 2, xlab =
      "")
lines(density(Mycielski$Errors), col = "red", lwd = 2)
legend("topleft", c("Mycielski", "ARIMA"), col = c("red", "green")
      , lty = c(1, 1), lwd = 2)
```

Listing 5: Data analysis code.

This papers shows the implementation of the Mycielski algorithm in R and provides an example on publicly available data for reproducibility. The exam- ple emphasizes the quality of the pattern matching approach as a forecasting method. It shows that the obtained forecasts are able to follow and exploit a periodic pattern of the data. Therefore, the algorithm is able to outperform simple models that are widely used in time series analysis. Still, models that are particularly designed for some underlying data set usually provide much better performance. Thus, the Mycielski algorithm mostly provides its strength in scenarios at which little or nothing is known about the data generating process "behind" the observed data.

References

1. Box G, Jenkins G (1970) Time Series Analysis: Forecasting and Control, San Francisco: Holden Day.
2. Engle RF (1982) Autoregressive Conditional Heteroskedasticity with Estimates of the Variance of United Kingdom Inflation. Econometrica 50: 987-1008.
3. Bollerslev T (1986) Generalized Autoregressive Conditional Heteroskedasticity, J Econometr 31: 307-327.
4. Valipour M, Banihabib ME, Behbahani SMR (2012) Parameters Estimate of Autoregressive Moving Average and Autoregressive Integrated Moving Average Models and Compare Their Ability for Inflow Forecasting. J Math Stat 8: 330-338.
5. Valipour M, Banihabib ME, Behbahani SMR (2013) Comparison of the ARMA, ARIMA, and the Autoregressive Artificial Neural Network Models in Forecasting the Monthly Inflow of Dez Dam Reservoir. J Hydrol 476: 433-441.
6. Valipour M (2015) Long-term Runoff Study Using SARIMA and ARIMA Models in the United States. Meteorological Applications 22: 592-598.
7. Valipour M (2012) Number of Required Observation Data for Rainfall Fore-casting According to the Climate Conditions. Am J Scientific Res 74: 79-86.
8. Croonenbroeck C (2013) Local Entropy Based Image Reconstruction. IJEME 3: 137-141.
9. Samy JA, Krishnan PS, Kiong TS (2012) Utilizing CLONALPropa- gation Algorithm for Pattern Matching with Training Data Sets. Am J Intel ligent Syst 2: 26-34.
10. Ehrenfeucht A, Mycielski J (1992) A Pseudorandom Sequence - How Random Is It? The American Mathematical Monthly 99: 373-375.
11. Jacquet P, Szpankowski W, Apostol L (2002) A Universal Predictor Based on Pattern Matching. IEEE Transactions on Information Theory 48: 1462-1472.
12. Hocaoglu FO, Fidan M, Gerek ON (2009) Mycielski Approach for Wind Speed Prediction. Ener Convers Manag 50: 1436-1443.
13. Gan M, Ding M, Huang Y, Dong X (2012) The Effect of Different State Sizes on Mycielski Approach for Wind Speed Prediction. J Wind Eng Indus Aerodyn 109: 89-93.
14. Croonenbroeck C, Ambach D (2015) A Selection of Time Series Models for Short- to Medium-Term Wind Power Forecasting. J Wind Engineer Indus Aerodyn 136: 201-210.
15. http://datamarket.com/data/set/22ox/monthly-milk-production-pounds-per-cow-jan-62-dec-75
16. Cryer JD (1986) Time Series Analysis. Duxbury Press 286.
17. Makridakis SG, Wheelwright SC, Hyndman RJ (1998) Forecasting: Methods and Applications. John Wiley & Sons: New York.