Online Prediction of Time Series with Assumed Behavior

ARIEL ROSENFELD, Bar-Ilan University, Israel
MOSHE COHEN, Bar-Ilan University, Israel
SARIT KRAUS, Bar-Ilan University, Israel
JOSEPH KESHET, Bar-Ilan University, Israel

The prediction of future time series values is essential for many fields and applications. In some settings, the time series behavior is expected to follow distinct patterns which in turn may change over time due to a change in the user's preferences/behavior or a change in the environment itself. In this article, we propose to leverage the assumed time series behavior by developing specialized novel online machine learning algorithms. To demonstrate the potential benefits of our approach compared to existing practices we focus on two commonly assumed time series behaviors: exponential decay and sigmoidal. We present two innovative online learning algorithms, *Exponentron* for the prediction of exponential decay time series and *Sigmoidtron* for the prediction of sigmoidal time series. We provide an extensive evaluation of both algorithms both theoretically and empirically using synthetic and real-world data. Our results show that the proposed algorithms compare favorably with the classic time series prediction methods commonly deployed today by providing a substantial improvement in prediction accuracy. Furthermore, we demonstrate the potential applicative benefit of our approach for the design of a novel automated agent for the improvement of the communication process between a driver and its automotive climate control system. Through an extensive human study with 24 drivers we show that our agent improves the communication process and increases drivers' satisfaction.

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**; *Learning paradigms*; • **Applied computing** → Forecasting;

Additional Key Words and Phrases: Time Series, Online learning, Human-Agent Interaction

ACM Reference Format:
Ariel Rosenfeld, Moshe Cohen, Sarit Kraus, and Joseph Keshet. 2010. Online Prediction of Time Series with Assumed Behavior. *ACM Trans. Intell. Syst. Technol.* 9, 4, Article 39 (March 2010), 24 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

Time series are a popular modeling technique for real-world phenomena and processes. These include natural phenomena (e.g., weather), industrial processes (e.g., crop yield), economical indicators (e.g., stock market indices) and many others. Generally, time series differ from other popular data modeling techniques in the temporal ordering of observations or values.

A major benefit of using time series is the use of the learned time series to predict the series' future values which, in turn, can be translated into better decisions from stakeholders. Specifically, intelligent systems often use time series prediction to improve their decision-making; A few examples include the repositioning of bikes in bike-sharing

Authors' addresses: Ariel Rosenfeld, Bar-Ilan University, Israel, ariel.rosenfeld@biu.ac.il; Moshe Cohen, Bar-Ilan University, Israel; Sarit Kraus, Bar-Ilan University, Israel; Joseph Keshet, Bar-Ilan University, Israel.

systems based on the prediction of bike usage [O'Mahony and Shmoys 2015], maintenance scheduling based on the prediction of ongoing performance [Shvartzon et al. 2016] and the prediction of passenger demand for better taxi routing [Moreira-Matias et al. 2013]. A common theme among these systems is the use of classical, *general purpose* time series prediction methods such as the Bayesian [Bernardo and Smith 2001], regressive [Draper et al. 1966] and autoregressive [Box et al. 1994] methods as the basis for their *domain-specific* proposed approach and design.

Existing time series prediction algorithms introduce two major limitations: First, most algorithms are *batch* in nature and need to get the whole training set in advance. As such, *batch* algorithms cannot adapt to changes over time in the modeled phenomenon which may occur after the initial training phase, for example when modeling human preferences over time. Second, these algorithms are *general purpose*, and do not exploit the assumed time series behavior (if such an assumption exists). It turns out that, in many domains, distinct time series behaviors are a-priori known or assumed. These time series behaviors often include two interesting behaviours: (i) the *exponential decay behavior*, which is evident, for example, in the decrease in radioactivity levels of radioactive substances, the cooling of an object in a cold environment [Newton 1701] and human decline of memory retention over time [Ebbinghaus 1913]; and (ii) the *sigmoidal behavior*, which can be used to model, for example, the growth of a population with limited resources [Verhulst 1838], the response of crop yields to soil salinity [Van Genuchten and Gupta 1993] and the progress in which new skills are learned by people over time [Murre 2014]. The focus of this study is to propose online algorithms to predict time series which are assumed to follow the above behaviors.

A voluminous body of work has been devoted to address the time series prediction using online learning. See, for example, [Anava et al. 2013, 2015; Kozdoba et al. 2018; Liu et al. 2016; Richard et al. 2009; Shalev-Shwartz 2011; Yang et al. 2019, 2018] and the many references therein. These works assume a generic hypotheses set, which is suitable to learn any time series. In this work we were rather focused on a restricted hypotheses set, namely the decaying exponent and the sigmoid, which is often found in many tasks. Given that the time series function is known in advance, such a restriction can end up with a higher accuracy, given the very small amount of training examples. The contributions of this paper are as follows:

(1) We develop two innovative online learning algorithms, *Exponentron* – for the prediction of exponential decay time series – and *Sigmoidtron* – for the prediction of sigmoidal time series. The algorithms are the first of their kind as they are both online and tailored to unique classes of time series. Both algorithms are aimed at optimizing the classic square loss function, and like other online learning algorithms, they do not require any training data before deployment. We provide a thorough analysis and empirical evaluation of the algorithms using both synthetic and real-world data.

(2) To exemplify the applicative benefits of our approach, we use the *Exponentron* algorithm for the design of a novel intelligent agent, named the *NICE* agent. The agent is intended to assist a human driver in better managing and changing the parameters of her automotive climate control system (abbreviated CCS, sometimes referred to as an automotive air conditioner). We show that our agent reduces the driver's need for interaction with the CCS, and increases the driver's subjective satisfaction thanks to the specialized and online nature of the *Exponentron* algorithm.

Several works have tried to address online-learning prediction of time series using deep neural networks [Guo et al. 2016; Liang et al. 2006; Zhang et al. 2018]. Most of these works build a mathematical wrapper over the stochastic gradient descent (SGD) optimization algorithm in order to deal with new instances. Naturally, deep networks require a large amount of labeled examples in order to converge, which often cannot be obtained in many online applications such as modeling a user's preference over time. This is also the drawback of our proposed method: we will only be able to efficiently and effectively learn time series which behave similar to the presumed hypotheses set.

2 TIME SERIES PRELIMINARIES

A time series $\boldsymbol{s} = (s_0, s_1, \ldots, s_T)$ is an ordered sequence of values measured in equally spaced time intervals, where $s_t \in \mathbb{R}$ denotes an element at time t, $0 \leq t \leq T$.

Assuming $(s_0, s_1, \ldots, s_{t-1})$ is the beginning of a series, the prediction of the next element is denoted $\hat{s}_t$. Among the most commonly applied techniques for predicting the continuation of a time series given its beginning are the autoregressive (AR) model, the autoregressive-moving average (ARMA) model (see [Box et al. 1994] for a review) and the exponential smoothing (ES) forecasting method (see [Gardner 1985] for a review) which we review next.

The prediction of the autoregressive (AR) model is generated using the following equation:

$$\hat{s}_t = c + \sum_{i=1}^{p} \varphi_i s_{t-i} \tag{1}$$

where c, p and φ_i are parameters of the model.

An *AR* model is in fact a linear regression of the current value of the time series against one or more prior values of the time series. The value of p is called the order of the AR model. The most commonly applied AR method, which is also used in this study, uses $p = 1$. This model is sometimes denoted $AR(1)$.

A popular extension of the AR model uses a moving average (MA), resulting in the autoregressive-moving average (ARMA) model [Box et al. 1994]. ARMA is also known as the Box-Jenkins Approach. The ARMA model generates its prediction using the following equation:

$$\hat{s}_t = c + \sum_{i=1}^{p} \varphi_i s_{t-i} + \sum_{i=1}^{q} \theta_i e_{t-i}. \tag{2}$$

where $e_j = s_j - \hat{s}_j$ and $c, p, q, \varphi_i, \theta_i$ are parameters of the model.

An *ARMA* model is in fact a linear regression of the current value of the time series against one or more prior values of the time series and one or more prior noise terms. The value of p is the order of the AR part of the model and q is the order of the MA part of the model. The most commonly applied ARMA method, which is also used in this study, uses $p = q = 1$. This model is sometimes denoted $ARMA(1, 1)$.

Another prediction method is the exponential smoothing (ES) scheme (also known as the exponentially weighted moving average), which weighs past elements of the time series using exponentially decreasing weights. The most suitable exponential smoothing method, which is also used in this study, is the *double* exponential smoothing method, denoted *ES*. The continuation of a time series is forecast by *ES* using the following equations:

$$\begin{aligned} \hat{s}_t &= \alpha s_{t-1} + (1-\alpha)(\hat{s}_{t-1} + b_t) \\ b_t &= \gamma(\hat{s}_t - \hat{s}_{t-1}) + (1-\gamma) b_{t-1} \end{aligned} \tag{3}$$

where $0 < \alpha \leq 1$ and $0 < \gamma \leq 1$.

There are several methods for choosing $\hat{s_0}$ and b_0. The most common one, which is also used in this study, is $\hat{s_0} = s_0$ and $b_0 = 0$.

Note that *single* exponential smoothing does not fit time series which present trends, and therefore is unsuitable for this study. *Triple* exponential smoothing, also known as the Holt-Winters exponential smoothing technique [Goodwin

2010], is popular in forecasting *seasonal* time series. In our examined settings, we assume no seasonality. The smoothing parameters, α and γ, used by the *ES* method are usually found using grid search.

Note that the above three methods are both *general purpose* (that is, they can fit a large variety of time series) and work *batch* (the models' parameters do not change during the prediction phase).

In this work we provide a solution to the task of *online* predicting the continuation of an assumed *exponential decay* or *sigmoidal* time series. To the best of our knowledge, no intelligent system or machine learning algorithm has addressed these challenges to date.

The above three models (*AR*, *ARMA*, *ES*) are evaluated as baseline models of this study. Recall that a deep neural network approach such as Long-Short-Term-Memory (LSTM) [Hochreiter and Schmidhuber 1997] may require a large amount of labeled examples and does not allow for a "drifting hypothesis" (i.e., changing the hypothesis over time). As a result, LSTM and similar deep learning-based models for time series prediction cannot be compared with our approach.

In online learning settings the learning takes place in rounds. In round t, the algorithm observes the series $(s_0, s_1, \ldots, s_{t-1})$ and is required to make a prediction for the next element in the series, $\hat{s}_t$. The algorithm maintains a set of parameters that are updated every round. After making its prediction, $\hat{s}_t$, the correct value, s_t, is revealed and an instantaneous loss $\ell(\hat{s}_t, s_t)$ is encountered. The round ends with an update of the parameters $\boldsymbol{\theta}$ according to the encountered loss. In this work we use the squared loss function, namely

$$\ell(\hat{s}_t(\boldsymbol{\theta}), s_t) = (\hat{s}_t(\boldsymbol{\theta}) - s_t)^2 \tag{4}$$

In this article, we provide a solution to the task of *online* predicting the continuation of an assumed *exponential decay* and *sigmoidal* time series. To the best of our knowledge, no intelligent system or machine learning algorithm has addressed this challenged to date.

3 EXPONENTIAL DECAY TIMES SERIES

Exponential decay time series are popular in modeling real-world phenomena, especially physical processes (e.g., [Leike 2001]).

In this section, it is assumed that a given time series was created by an exponential decay process, possibly with noise. We first present an online learning algorithm, which we call *Exponentron*, for the prediction of exponential decay time series. The Exponentron algorithm focuses on a special hypothesis family capturing the assumed exponential decay behavior of time series. It is aimed at optimizing the square loss function and, like other online learning algorithms, it does not require any training data before deployment and is able to adapt to changing exponential decay behavior over time. Then, we state a regret bound for the Exponentron algorithm. Regret bounds are common in the analysis of online learning algorithms. A regret bound measures the performance of an online algorithm relative to the performance of the best competing hypothesis, which can be chosen in hindsight from a class of hypotheses, after observing the entire time series. Last, we empirically evaluate the *Exponentron* against the baseline algorithms discussed in Section 2, using both synthetic and real-world datasets.

3.1 The Exponentron Algorithm

We assume the following set of hypotheses:

$$\hat{s}_t(\boldsymbol{\theta}) = a + b\, e^{-c\,(t-t_0)}, \tag{5}$$

where $\boldsymbol{\theta} = (a, b, c)$ is the set of 3 parameters that should be estimated, $\boldsymbol{\theta} \in \mathbb{R}^3_+$, and $t_0 \in \mathbb{R}_+$ is a time offset parameter.

Our algorithm, which is called *Exponentron*, is given in Algorithm 2. The algorithm is aimed at minimizing the cumulative loss.

The algorithm starts with a set of feasible parameters $\boldsymbol{\theta}_0 \in \mathbb{R}^3_+$, that is, $\boldsymbol{\theta}_0 = (a_0, b_0, c_0)$ which satisfies the constraints on the parameters. We initialize t_0 by setting the first prediction to be correct, namely, $\hat{s}_t = s_t$ for $t = 0$, and get

$$t_0 = \log((s_0 - a)/b)/c. \tag{6}$$

Now the set of parameters needs to satisfy the constraints $a \leq s_0$, $b \geq 0$ and $c \geq 0$.

The following proposition states that the hypothesis function is a convex function.

Proposition 1. The hypothesis function

$$\hat{s}_t(\boldsymbol{\theta}) = a + b\, e^{-c\,(t-t_0)} \tag{7}$$

is a convex function in the set of parameters $\boldsymbol{\theta} = (a, b, c)$.

Proof. Since $b > 0$, we can rewrite it as $b = e^{\tilde{b}}$, and the hypothesis becomes $\hat{s}_t(\boldsymbol{\theta}) = a + e^{\tilde{b}-c\,(t-t_0)}$. Now it is easy to verify that

$$\hat{s}_t(\alpha\boldsymbol{\theta}_1 + (1-\alpha)\boldsymbol{\theta}_2) \leq \alpha\hat{s}_t(\boldsymbol{\theta}_1) + (1-\alpha)\hat{s}_t(\boldsymbol{\theta}_2),$$

for the sets $\boldsymbol{\theta}_1 = (a_1, \tilde{b}_1, c_1)$ and $\boldsymbol{\theta}_2 = (a_2, \tilde{b}_2, c_2)$ which satisfy the constraints, and $\alpha \in (0, 1)$. □

Since our loss function in Eq. (4) is also a convex function, it turns out that the loss is convex.

Our algorithm is based on *gradient projected methods* [Bertsekas 1999, pp. 228]. The algorithm starts with a set of feasible parameters $\boldsymbol{\theta}_0 \in \mathbb{R}^3_+$, that is, $\boldsymbol{\theta}_0$ satisfies the constraints. At the t-th round the algorithm predicts the next element in the time series based on the parameters $\boldsymbol{\theta}_{t-1}$. Then, if the encountered loss, $\ell(\hat{s}_t(\boldsymbol{\theta}_{t-1}), s_t)$, is greater than zero, the parameters are updated by a gradient step: $\boldsymbol{\theta}' = \boldsymbol{\theta}_{t-1} + \eta_t \nabla_{\boldsymbol{\theta}} \ell$, where the gradient of the loss is the following vector:

$$\nabla_t = 2(\hat{s}_t(\boldsymbol{\theta}) - s_t)[1, e^{-c\,(t-t_0)}, -b(t-t_0)e^{-c\,(t-t_0)}]. \tag{8}$$

The parameter η_t is the learning rate. Specifically in our case we set $\eta_t = \eta_0/\sqrt{t}$, where η_0 is chosen when the algorithm starts (as in [Bottou 2012; Zinkevich 2003]).

At the end of each round the algorithm projects $\boldsymbol{\theta}'$ on the set of constraints in order for $\boldsymbol{\theta}_t$ to form a feasible vector.

Algorithm 1 The Exponentron Algorithm

```
Require: initialize $\boldsymbol{\theta}_0$, learning parameter $\eta$.
1: observe $s_0$ and set $t_0$ according to Eq. (6)
2: for $t = 1, 2, \ldots, T$ do
3:     predict $\hat{s}_t = a_{t-1} + b_{t-1}\, e^{-c_{t-1}\,(t-t_0)}$
4:     observe true value $s_t$
5:     encounter loss $\ell(\hat{s}_t, s_t)$
6:     update parameters and project
7:         $a_t = \min\{s_0, a_{t-1} - 2\eta_t(\hat{s}_t - s_t)\}$
8:         $b_t = \max\{0, b_{t-1} - 2\eta_t(\hat{s}_t - s_t)e^{-c_{t-1}\,(t-t_0)}\}$
9:         $c_t = \max\{0, c_{t-1} + 2\eta_t(\hat{s}_t - s_t)b_{t-1}(t-t_0)e^{-c_{t-1}\,(t-t_0)}\}$
```

Note that the Exponentron algorithm does not require any training before deployment.

Often the performance of an online algorithm is measured by how *competitive* it is with the hypothesis of the best *fixed* parameters $\boldsymbol{\theta}^*$. This is captured by the notion of the algorithm's *regret*, which is defined as the excess loss for not consistently predicting with the parameters $\boldsymbol{\theta}^*$,

$$\text{regret}(\boldsymbol{\theta}^*, T) \triangleq \sum_{t=1}^{T} \ell(\hat{s}_{t-1}(\boldsymbol{\theta}), s_t) - \sum_{t=1}^{T} \ell(\hat{s}_t(\boldsymbol{\theta}^*), s_t). \tag{9}$$

The following theorem states that the regret of the Exponentron algorithm is bounded.

Theorem 2. *The Exponentron algorithm has the following regret bound for every $\boldsymbol{\theta}^*$ in $\mathbb{R}^3_+$,*

$$\text{regret}(\boldsymbol{\theta}^*, T) \leq \frac{\sqrt{T}}{2}\|\boldsymbol{\theta}\|^2 + \frac{1}{2\sqrt{T}}\sum_{t=1}^{T}\|\nabla_t\|^2. \tag{10}$$

Proof. The analysis is based on the stochastic gradient descent with projection analysis [Bertsekas 1999]. Denote by $\boldsymbol{\theta}_{t-1}$ the set of parameters before the update, by $\boldsymbol{\theta}_{t-1/2}$ the set of parameters after the gradient step, and by $\boldsymbol{\theta}_t$ the set of parameters after the projection step. We have

$$\begin{aligned}
&\|\boldsymbol{\theta}_t - \boldsymbol{\theta}^*\|^2 - \|\boldsymbol{\theta}_{t-1} - \boldsymbol{\theta}^*\|^2 \\
&\quad = \|\boldsymbol{\theta}_t - \boldsymbol{\theta}^*\|^2 - \|\boldsymbol{\theta}_{t-1/2} - \boldsymbol{\theta}^*\|^2 \\
&\qquad\qquad + \|\boldsymbol{\theta}_{t-1/2} - \boldsymbol{\theta}^*\|^2 - \|\boldsymbol{\theta}_{t-1} - \boldsymbol{\theta}^*\|^2 \\
&\quad \leq \|\boldsymbol{\theta}_{t-1/2} - \boldsymbol{\theta}^*\|^2 - \|\boldsymbol{\theta}_{t-1} - \boldsymbol{\theta}^*\|^2 \\
&\quad = \|\boldsymbol{\theta}_{t-1} - \eta\nabla_t - \boldsymbol{\theta}^*\|^2 - \|\boldsymbol{\theta}_{t-1} - \boldsymbol{\theta}^*\|^2 \\
&\quad = -2\eta\,(\boldsymbol{\theta}_{t-1} - \boldsymbol{\theta}^*) \cdot \nabla_t + \eta^2\|\nabla_t\|^2 \\
&\quad \leq -2\eta\Big(\ell(\hat{s}_t(\boldsymbol{\theta}_{t-1}), s_t) - \ell(\hat{s}_t(\boldsymbol{\theta}^*), s_t)\Big) + \eta^2\|\nabla_t\|^2
\end{aligned}$$

From the second line to the third line we used the property of projections, $\|\boldsymbol{\theta}_t - \boldsymbol{\theta}^*\|^2 \leq \|\boldsymbol{\theta}_{t-1/2} - \boldsymbol{\theta}^*\|^2$. We now sum over all t, and get

$$\begin{aligned}
&\|\boldsymbol{\theta}_T - \boldsymbol{\theta}^*\|^2 - \|\boldsymbol{\theta}_0 - \boldsymbol{\theta}^*\|^2 \\
&\leq -2\eta\sum_{t=1}^{T}\Big(\ell(\hat{s}_t(\boldsymbol{\theta}_{t-1}), s_t) - \ell(\hat{s}_t(\boldsymbol{\theta}^*), s_t)\Big) + \frac{1}{2}\eta\sum_{t=1}^{T}\|\nabla_t\|^2
\end{aligned}$$

By rearranging terms we get the desired result. □

3.2 Exponentron Evaluation

We first evaluate the Exponentron algorithm using synthetic exponential decay time series. Then, the Exponentron algorithm is examined using two real-world prediction tasks of significant importance: First, we evaluate the Exponentron in predicting drivers' desired interior cabin temperature during a drive. Specifically, we focus on the cooling condition, where a driver wishes to cool the interior cabin temperature of a car in order to achieve a comfortable state. Second, we wish to predict the number of arriving calls at a call center. Specifically, we consider a call center in which human service agents can handle both inbound calls and other back-office tasks, making the prediction of arriving calls an important factor in real-time work schedule adjustment and managerial decision-making [Gans et al. 2003].

We compare Exponentron against 3 well-known batch time series prediction methods, *AR*, *ARMA* and *ES*, which are described in Section 2.

3.2.1 *Synthetic Data Prediction.*

To gain intuition about the relative strengths of the Exponentron algorithm, we evaluate the Exponentron in a synthetic exponential decay time series prediction task.

To this end, we synthetically generated 1,000 exponential decay time series, which all start with a value of 100 at $t = 0$, denoted $s_0 = 100$. Each time series is represented as a tuple $\boldsymbol{\theta} = (a, b, c)$ and is generated according to Equations 5 and 6; $s_t(\boldsymbol{\theta}) = a + b\, e^{-c\,(t-t_0)}$ where $t_0 = \log((s_0 - a)/b)/c$. We synthetically generated the time series by sampling $a \in U[10, 80]$, $b \in U[20, 90]$ and $c \in U[0.1, 0.5]$.[1] We then randomly assigned 900 time series to a training set and the remaining 100 time series were assigned to the test set. The training set time series are used to set the parameters of the Exponentron algorithm, alongside the three baseline models described in Section 2. The coefficients used by the *AR* and *ARMA* methods were learned using a simple linear regression over the training set time series. Similarly, the smoothing parameters used by the *ES* method were found using a grid search. The Exponentron's initial parameters, $\boldsymbol{\theta}_0$, were set to the least squares regression parameters calculated based on the training data as described in [Smyth 2002].

The four models were tested over the test set time series. Overall, Exponentron significantly outperforms each of the tested baseline models using univariate ANOVA with the prediction method as the independent factor and the prediction mean absolute error as the dependent variable followed by pairwise t-tests, $p < 0.05$. Table 1 summarizes the results. An illustration of the predicted values made by the Exponentron and the *ARMA* model is presented in Figure 1.

Method	Mean Absolute Error
AR(1)	0.15
ARMA(1, 1)	0.12
ES	0.2
Exponentron	**0.03**

Table 1. Prediction mean absolute error made by each prediction method for the synthetic data.

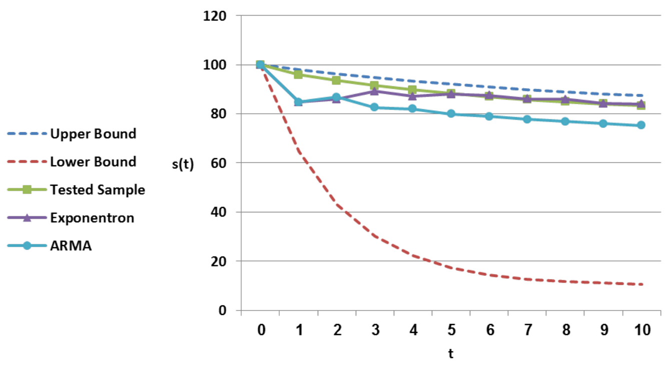

Fig. 1. All time series were sampled from the confined space between the upper and lower bounds. The green line is one of the tested time series (represented by $a = 79$, $b = 20$, $c = 0.15$) and the purple and light blue lines are the predictions made by the Exponentron and ARMA, respectively.

[1] a, b and c were chosen as such to allow a significant range of possible hypotheses and yet restrict the range to allow a reasonable first estimation of a, b and c by each of the tested algorithms.

We further evaluate the four models on the same set of time series while adding noise to the data. Specifically, to every value in each of the test set time series a Gaussian noise was added with a mean of 0 and a standard deviation of 1, 2 or 3. Interestingly, for a low noise level (1) , the Exponentron algorithm significantly outperforms each of the tested baseline models using univariate ANOVA with the prediction method as the independent factor and the prediction mean absolute error as the dependent variable followed by pairwise t-tests, $p < 0.05$. However, for higher noise levels (2 and 3), the Exponentron performs significantly worse than the *ARMA* baseline. Table 2 summarizes the results. Note that by inducing noise over the test set time series, the mean absolute error of every model is significantly greater. It is also important to note that since the Exponentron is the only online learning algorithm of the four, it is more vulnerable to noise compared to static models.

Method	1	2	3
$AR(1)$	1.19	1.45	2.16
$ARMA(1,1)$	1.09	1.19	**2.04**
ES	1.22	1.32	3.15
$Exponentron$	**0.85**	2.42	4.02

Table 2. Prediction mean absolute error made by each prediction method for the *noisy* synthetic data.

This synthetic experiment demonstrates the advantage of the Exponentron in exponential decay time series prediction. However, it also demonstrates that the Exponentron is sensitive to high noise levels, in which case it may be counter-productive. In order to better understand the potential benefits and limitations of the Exponentron, we turn to investigate the Exponentron using two real-world data sets.

3.2.2 *Predicting Desired Climate Changes in a Car's Interior.*

Data Collection. The cabin temperature time series is $(s_0, s_1, \ldots)$ where s_0 is the initial cabin temperature when a driver turns the CCS on, and s_j is the cabin temperature k seconds after s_{j-1}. In our setting, k was set to 15 seconds.[2] A driver's *preferred* cabin temperature time series is $(s^*_0, s^*_1, \ldots)$ where $s^*_0 = s_0$ and s^*_i is the desired cabin temperature at time frame i. A cabin temperature is said to be **steady** if in a period of 1 minute the driver does not change the CCS features and the cabin temperature does not change more than ϵ. In our setting, ϵ was set to 0.1°C.[3]. We focus on the task of predicting s^*_i given $(s^*_0, \ldots, s^*_{i-1})$ until a steady cabin temperature has been reached.

We recruited 28 drivers, ranging in age from 25 to 57 (average of 35), 22 males and 6 females. Each subject was asked to enter a parked car in order to experience the environmental conditions – temperatures ranging from 21°C to 31°C, averaging 27°C. Each subject was presented with a newly designed graphical interface presented on a tablet which we call the *natural interface*. The natural interface presents natural terms such as "Too cold", "Too hot" and "Noisy", which are unavailable in most CCSs. Each subject was instructed to interact with the system, such that after any button was pressed the subject had to *manually* change the features of the CCS using the car's standard interface with the help of our research assistant. Namely, the natural interface did not have any functionality behind it at this point. The session stopped once the cabin temperature of the car was *steady*. While in the car, the subject was given a cell phone with a

[2]A time interval of 15 seconds was chosen to allow sufficient time for our thermometer to adapt to the changing temperature inside the car. The thermometer specification states that it takes up to 15 seconds for the thermometer to adapt to its environment.

[3]The inaccuracy interval of our thermometer.

driving simulator “Bus Simulator 3D"[4] to be played while the experiment was conducted. The motivation was to set the conditions similar to regular driving conditions and give the subjects something to do. Unfortunately, due to insurance reasons, we could not conduct the study while subjects were actually driving.[5] The subject was then asked to exit the car for a period of 10 minutes while the car's doors were left open in order to recreate the initial conditions. Note that the subject could choose to click on any button at any given moment, thereby changing the CCS (manually).

During the session, the internal cabin temperature of the car was recorded using a state-of-the-art thermometer that we placed between the driver's and the front passenger's seats. The temperature was measured once every 15 seconds (again, to allow sufficient time for our thermometer to adapt).

Overall, 56 time series were collected. The shortest time series consisted of 6 data points whereas the longest consisted of 26 data points (mean of 13).

Analysis. The Exponentron algorithm, alongside the three baseline models described in Section 2, was evaluated based on the collected data.

The baseline models were trained and evaluated using a one-left-out methodology. Namely, we took one series at a time out of the data set and used the remaining series as training data. All four models were trained as described in Section 3.2.1.

Note that the Exponentron algorithm does not necessitate any training prior to deployment, but instead requires an initialization of the parameters (θ_0 in Algorithm 1). Nevertheless, we chose to set the initial parameters to the least squares regression parameters calculated based on the training data using a one-left-out methodology as described in [Smyth 2002]. We also examined the initialization of the Exponentron's parameters using only a subset of the training data. Surprisingly, using *any* single time series from the training data to determine the Exponentron's initial parameters resulted in a less than 10% decrease in the Exponentron's accuracy compared to using all of the training data.

The Exponentron's mean absolute error was found to be 0.13°C per value in the time series. The Exponentron's predictions are 35% more accurate as compared to the best tested baseline model, *ARMA*, which yields a 0.2°C mean absolute error. Table 3 provides a summary of the tested models' prediction errors.

Overall, Exponentron significantly outperforms each of the tested baseline models using univariate ANOVA with the prediction method as the independent factor and the prediction mean absolute error as the dependent variable followed by pairwise t-tests, $p < 0.05$.

Method	Mean Absolute Error (per 15 seconds)
$AR(1)$	0.21
$ARMA(1,1)$	0.2
$ES(1,0)$	0.24
Exponentron	0.13

Table 3. Prediction of the desired interior temperature of the car. Numbers indicate the mean absolute error made by each prediction method per 15 second frame.

3.2.3 *Inbound Calls in a Real-World Call Center.*

[4]Available free at Google Play store.
[5]“Bus Simulator 3D" was also used in previous human-CCS interaction studies in order to simulate driving conditions [Azaria et al. 2015].

Real-World Call Center – Secondary Data. We use data that was collected and analyzed in [Mandelbaum et al. 2001]. The data accounts for all inbound calls arriving at the small call center of a bank in 1999 [Guedj and Mandelbaum 2000]. On weekdays the center is open from 7am to midnight (17 hours) and provides service to over 2,000 callers (on average). We focus on the 16:00-24:00 (8 hours) time frame, in which an average of approximately 750 calls arrive at the call center in an assumed exponential decay manner.

Following the original analysis procedure, we processed the data such that all national holidays were removed and each of the daily recordings was translated into a time series. For this evaluation we used a time series of the form $(c_{17}, c_{18}, \ldots, c_{24})$ where c_i is the number of arriving calls during the $(i-1)^{\text{th}}$ hour of the day. For example, all calls arriving between 17:00 and 18:00 will count as c_{18}.

Overall, 222 time series were constructed. Each time series consists of 8 data points.

Analysis. The Exponentron algorithm, alongside baseline models described in Section 2, was evaluated using the same procedure as described in Section 3.2.2. Again, we noticed that using *any* single time series from the training data to determine the Exponentron's initial parameters resulted in a less than 15% decrease in the Exponentron's accuracy compared to using the entire training data.

At each time frame of one hour, the Exponentron's mean absolute error was found to be 5.8 calls. The Exponentron's predictions are 41% more accurate than the best tested baseline model, *ARMA*, which yields a mean absolute error of 9.8 calls. Table 4 provides a summary of the tested models' prediction errors.

Method	Mean Absolute Error (hourly)
$AR(1)$	10.2
$ARMA(1,1)$	9.8
$ES(0.9,1)$	13.7
Exponentron	5.8

Table 4. Prediction of call arrivals in a real-world call center. Numbers indicate the mean absolute error made by each prediction method.

Figure 2 demonstrates the predictions provided by the Exponentron and $ARMA(1,1)$ for the number of arriving calls per hour during the randomly selected evening of February 8th, 1999.

Overall, Exponentron significantly outperforms each of the tested baselines using univariate ANOVA with the prediction method as the independent factor and the prediction mean absolute error as the dependent variable followed by pairwise t-tests, $p < 0.05$.

Note that both data sets examined above contain noise. The favorable results of the Exponentron suggest that the Exponentron is indeed capable of handling reasonable levels of noise.

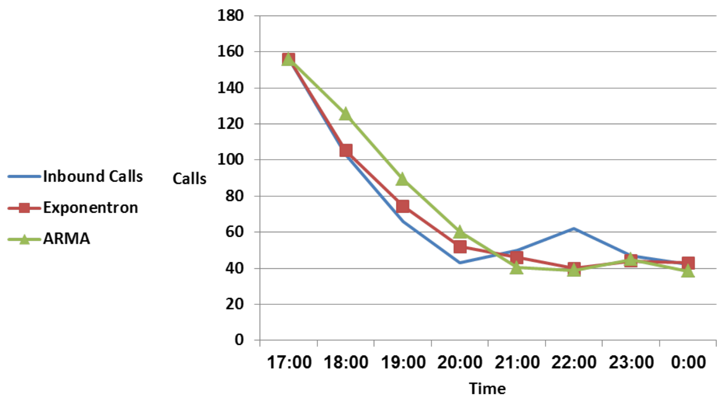

Fig. 2. The prediction of inbound calls made by the *ARMA* and Exponentron models for the evening of 8/2/1999.

4 SIGMOIDAL TIME SERIES

Sigmoid functions are popular in modeling real-world phenomena, especially in biology [Vandermeer 2010].

Similar to Section 3, in this section it is assumed that a given time series was created by a sigmoidal process, possibly subject to noise. We present an online learning algorithm, which we call *Sigmoidtron*, for the prediction of a sigmoidal time series. The *Sigmoidtron* algorithm focuses on a special hypothesis family capturing the assumed sigmoidal behavior of a time series. It is aimed at optimizing the square loss function, and, like before, it does not require any training data before deployment and is able to adapt to changing sigmoidal behavior over time. Empirically, we evaluate the *Sigmoidtron* against the baseline algorithms using both synthetic and real-world data sets.

4.1 The Sigmoidtron Algorithm

We assume the following set of hypotheses:

$$\hat{s}_t(\boldsymbol{\theta}) = a + \frac{b}{c + e^{d\,(t+t_o)}} \tag{11}$$

where $\boldsymbol{\theta} = (a, b, c, d)$ is a set of four parameters that should be estimated, such that $a, b, c \in \mathbb{R}_+$ and $d \in \mathbb{R}_-$.

Following the same rationale as in Section 3, we focus on the online settings, where learning takes place in rounds. In round t, the algorithm observes the series $(s_0, s_1, \ldots, s_{t-1})$ and is required to make a prediction for the next element in the series, $\hat{s}_t$. The algorithm maintains a set of parameters that are updated every round. After making its prediction, $\hat{s}_t$, the correct value, s_t, is revealed and an instantaneous loss $\ell(\hat{s}_t, s_t)$ is encountered. The round ends with an update of the parameters $\boldsymbol{\theta}$ according to the encountered loss. We use the squared loss function as before.

Our algorithm, which is called *Sigmoidtron*, is given in Algorithm 2. The algorithm is aimed at minimizing the cumulative loss.

The algorithm starts with a set of feasible parameters $a, b, c \in \mathbb{R}_+$ and $d \in \mathbb{R}_-$, that is, $\boldsymbol{\theta}_0 = (a_0, b_0, c_0, d_0)$ satisfies the constraints on the parameters. We set t_0 such that the first prediction will be correct, namely $\hat{s}_t = s_t$ for $t = 0$, and get

$$t_0 = \frac{\ln(\frac{b}{s_0 - a} - c)}{d}. \tag{12}$$

Now the set of parameters needs to satisfy the constraints $a < s_0$, $c < \frac{b}{s_0 - a}$.

It can be shown that the sigmoid function is not convex, hence we cannot prove a regret bound. Nevertheless, it can be shown that the *Stochastic Gradient Decent* method converges to a *local* minima as in deep learning training [Bertsekas 1999, pp.239].

The algorithm starts with a set of feasible parameters $\boldsymbol{\theta}_0$, that is, $\boldsymbol{\theta}_0$ satisfies the constraints. At the t-th round the algorithm predicts the next element in the time series based on the parameters $\boldsymbol{\theta}_{t-1}$. Then, if the encountered loss, $\ell(\hat{s}_t(\boldsymbol{\theta}_{t-1}), s_t)$, is greater than zero, the parameters are updated by a gradient step: $\boldsymbol{\theta}' = \boldsymbol{\theta}_{t-1} + \eta_t \nabla_{\boldsymbol{\theta}} \ell$ where the gradient of the loss is the following vector:

$$\nabla_t = 2(\hat{s}_t(\boldsymbol{\theta}) - s_t)[-1, -\frac{1}{c + e^{d\,(t+f)}}, \frac{b}{(c + e^{d\,(t+f)})^2}, \frac{b\,(t+f)\,e^{d\,(t+f)}}{(c + e^{d\,(t+f)})^2}, \frac{b\,d\,e^{d\,(t+f)}}{(c + e^{d\,(t+f)})^2}]. \tag{13}$$

The parameter η_t is the learning rate. As before, we set $\eta_t = \eta_0/\ln(t+1)$, where η_0 is chosen when the algorithm starts.

At the end of each round the algorithm projects $\boldsymbol{\theta}'$ on a set of constraints in order to get $\boldsymbol{\theta}_t$, a feasible vector.

Algorithm 2 The Sigmoidtron Algorithm

```
Require: initialize $\boldsymbol{\theta}_0$, learning parameter $\eta$.
1:  for $t = 1, 2, \ldots, T$ do
2:      predict $\hat{s}_t = a_{t-1} + \frac{b_{t-1}}{c_{t-1} + e^{d_{t-1}(t+f_{t-1})}}$
3:      observe true value $s_t$
4:      encounter loss $\ell(\hat{s}_t, s_t)$
5:      $\boldsymbol{\theta}' = \boldsymbol{\theta}_{t-1} - \eta_t \nabla_t \ell$ using Eq.13
6:      project $\boldsymbol{\theta}'$ on the set of constraints
7:              $a_t = \max\{\epsilon, a_t\}$
8:              $b_t = \max\{\epsilon, b_t\}$
9:              $c_t = \max\{\epsilon, c_t\}$
10:             $d_t = \min\{d_t, -\epsilon\}$
```

Note that the Sigmoidtron algorithm does not require any training before deployment.

Next, we empirically show the significant benefits of the Sigmoidtron using both synthetic and real-world datasets.

4.2 Sigmoidtron Evaluation

We first evaluate the Sigmoidtron algorithm using a synthetic sigmoidal time series. Then, we evaluate the Sigmoidtron algorithm in predicting academic researchers' future H-Index values [Hirsch 2005]. The H-Index is an important statistic for academic institutions that consider hiring or promoting a researcher, since they would want to estimate the researcher's future contributions based on her past contributions.

We compare the Sigmoidtron to the three baseline prediction methods described in Section 2, in three experimental settings: First, we got intuition about the relative strength of the Sigmoidtron algorithm by testing it on a synthetic dataset. We then evaluated the Sigmoidtron performance using a large real-world H-Index dataset. Lastly, we re-examined the real-world H-Index dataset, but instead of predicting future values one year at a time, we consider the task of predicting the H-Index expected trajectory for the next *10 years*.

4.2.1 Synthetic Data Prediction. To gain intuition about the relative strengths of the Sigmoidtron algorithm, we evaluate the Sigmoidtron in a synthetic sigmoidal time series prediction task.

To this end, we synthetically generated 1,000 sigmoidal time series, starting at a randomly selected value between 1 and 10 at $t = 0$, denoted s_0. Each time series is represented as a tuple $\boldsymbol{\theta} = (a, b, c, d)$ and is generated according to Equations 11 and 12; $s_t(\boldsymbol{\theta}) = a + \frac{b}{c+e^{d(t+f)}}$ where $t_0 = \frac{\ln(\frac{b}{s_0-a}-c)}{d}$. We synthetically generated the time series by sampling $a \in U[1,3]$, $b \in U[3,6]$, $c \in U[1e^{-10}, 1e^{8}]$ and $d \in U[-1, -1e^{-10}]$.[6] We then randomly assigned 900 time series to a training set and the remaining 100 time series were assigned to the test set. The training set time series are used to train the Sigmoidtron algorithm, alongside the three baseline models. The coefficients used by the AR and ARMA methods were learned using a simple linear regression over the training set time series. Similarly, the smoothing parameters used by the ES method were found using a grid search. The Sigmoidtron's initial parameters, $\boldsymbol{\theta}_0$, were calculated based on the first observation of each series, as described above.

Overall, the Sigmoidtron significantly outperforms each of the tested baseline models using univariate ANOVA with the prediction method as the independent factor and the prediction mean absolute error as the dependent variable followed by pairwise t-tests, $p < 0.05$. An illustration of the predicted values made by the Sigmoidtron and the *AR* model is presented in Figure 3. Table 5 provides a summary of the tested models' prediction errors.

Method	Mean Absolute Error
$AR(1)$	0.28
$ARMA(1,1)$	1.12
$ES(0.9,1)$	0.17
$Sigmoidtron$	**0.12**

Table 5. Prediction mean absolute error made by each prediction method for the synthetic data.

We further evaluate the four models on the same set of time series while adding noise to the data. Specifically, to every value in each of the test set time series a Gaussian noise was added with a mean of 0 and a standard deviation of 0.05, 0.07 or 0.1. Interestingly, for a low noise level (0.05) , the Sigmoidtron algorithm significantly outperforms each of the tested baseline models using univariate ANOVA with the prediction method as the independent factor and the prediction mean absolute error as the dependent variable followed by pairwise t-tests, $p < 0.05$. However, for higher noise levels (0.07 and 0.1), the Sigmoidtron performs worse than the *ES* baseline, yet the difference is only significant for a noise level of 0.1. Table 6 summarizes the results.

Method	0.05	0.07	0.1
$AR(1)$	0.36	0.37	0.39
$ARMA(1,1)$	1.05	1.24	1.26
$ES(0.9,1)$	0.3	0.33	**0.35**
$Sigmoidtron$	**0.26**	0.35	0.4

Table 6. Prediction mean absolute error made by each prediction method with varying levels of noise over the synthetic data.

[6]a, b, c and d were chosen as such to allow a significant range of possible hypotheses and yet restrict the range to allow a reasonable first estimation of a, b, c and d by each of the tested algorithms.

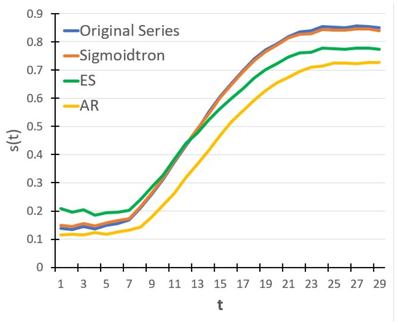

Fig. 3. Sample predicted series (with 0.05 noise) of the best performing methods. The blue line is a sampled time series (with noise); the orange, green and yellow lines are the fitted curves by the Sigmoidtron, ES and AR algorithms, respectively. To allow comparison by average, all s(t) values are normalized to the same range – [0, 1].

This synthetic experiment demonstrates the advantage of the Sigmoidtron in sigmoidal time series prediction. However, it also demonstrates that the Sigmoidtron is sensitive to high noise levels, in which case it may be counterproductive. We now turn to investigate the Sigmoidtron's advantages using a real-world data set.

4.2.2 *Academic H-Indices.*

Data Collection. We used the DBLP Citation Network (v10) [Tang et al. 2008] which is freely available online[7] and is updated frequently. The database contains information for over 3 million publications, and over 25 million citations in total. The database consists of publication records, where each record contains the paper's abstract, authors, year, venue, title and a list of referenced papers. We processed the raw database and produced a clean dataset that contains the names of all of the authors that are mentioned in any publication in the raw database, and for each one we calculated her H-Index value for each year starting with her first publication until the present (2018). In total, we extracted information for over 30,000 authors. Authors whose academic seniority was too short (less than 5 years) and ones whose H-Index value did not change at all (i.e., non-active authors) were filtered out. We were left with about 10,000 authors' H-Index time series. We denote each time series as $(s_0, s_1, \ldots, s_T)$ where s_0 is the H-Index value of the researcher after her first academic activity year, and s_k is her H-Index value after $k+1$ years of academic activity.

The shortest time series consisted of 25 data points, whereas the longest consisted of 75 data points (mean of 31).

Analysis. The Sigmoidtron algorithm, alongside the three baseline models described in Section 2, was evaluated based on the collected data.

The baseline models were trained and evaluated as before, by processing the entire data using one series at a time, calculating the optimal parameters and then fitting each series according to the calculated parameters. All three models were trained as described in Section 4.2.1.

Note that the Sigmoidtron algorithm does not necessitate any training prior to deployment, but instead requires an initialization of the parameters (θ_0 in Algorithm 2).

The Sigmoidtron's mean absolute error was found to be 0.6 per value in the time series. The Sigmoidtron's predictions are 26.6% more accurate compared to the best baseline model, *ES*, which yields a 0.75 mean absolute error. Table 7 provides a summary of the tested models' prediction errors.

[7]https://aminer.org/citation

Method	Mean Absolute Error
$AR(1)$	0.8
$ARMA(1, 1)$	1.08
ES	0.76
Sigmoidtron	**0.6**

Table 7. Prediction of authors' H-Index. Numbers indicate the mean absolute error made by each prediction method per year.

Method	Mean Squared Error
$AR(1)$	8.21
$ARMA(1, 1)$	15.13
ES	7.7
Sigmoidtron	**5.87**

Table 8. Prediction of the H-Index trajectory 10 years ahead. Numbers indicate mean absolute error made by each prediction method per year.

Overall, Sigmoidtron significantly outperforms each of the tested baseline models using univariate ANOVA with the prediction method as the independent factor and the prediction mean absolute error as the dependent variable followed by pairwise comparisons, $p < 0.05$.

4.2.3 *Academic H-Indices: The Applicable Setting.*

In the above analysis, we focused on the prediction of the next year's H-Index value in the time series. However, in real-world decision-making, e.g., when an institution is considering hiring or promoting a researcher, one may be more interested in predicting a researcher's future H-Index *trajectory* for more than a single year. Therefore, in this analysis, we focus on the task of predicting the H-Index trajectory 10 years ahead.

Analysis. Here we focus only on researchers who have at least 15 academic years seniority, and try to predict for each of them the 10 year trajectory starting at year 5. Unlike previous settings, here the Sigmoidtron *does not update it's parameters after the initial learning phase.* The baseline models were trained and evaluated as before.

Naturally, the prediction of a 10 year trajectory (without updating the parameters) is far more complex than before. However, the Sigmoidtron is still found to be superior over the baseline models: Sigmoidtron's mean absolute error was found to be 5.87% per value in the time series, which is 41% more accurate compared to the best baseline model, *ES*, which yields a 7.7 mean absolute error. *AR* yields a 8.21 mean absolute error and *ARMA* yields a 15.13 mean absolute error. Table 8 provides a summary of the tested models' prediction errors.

Again, the Sigmoidtron significantly outperforms each of the tested baseline models using univariate ANOVA with the prediction method as the independent factor and the prediction mean absolute error as the dependent variable followed by pairwise t-tests, $p < 0.05$.

5 APPLICATIVE VALUE

It is claimed that a prediction method is only as good as its resulting agent's performance [Rosenfeld and Kraus 2018]. Therefore, in order to demonstrate the *applicative benefit* of our specialized time series prediction approach, in this section we develop and evaluate a novel intelligent agent which is based on our Exponentron algorithm (Section 3.1). We choose to use the Exponentron for this evaluation since it provides both empirical benefits as well as theoretical performance guarantees.

We choose to focus on an agent-human interaction challenge in Climate Control Systems (CCSs). Specifically, we aim to automatically adjust the CCS features (e.g, fan speed, temperature) in order to provide comfortable settings for the user. We address a situation in which a driver enters a hot vehicle and the agent's goal is to automatically set and adjust the car's CCS features throughout the ride to the driver's satisfaction.

In an interview-based preliminary experiment that we conducted with 18 drivers, we noticed that the drivers' satisfaction from their CCS is affected not only by the target cabin setting of the car but, and even more importantly, by the cabin setting endured during the adjustment process. Furthermore, we observed that people have different preferences for both situations. However, their preferences during the adjustment process present similar *exponential decay tendencies*. For example, Alice's target interior cabin temperature is 21° C and she wishes to reach it as fast as possible (she does not mind enduring extreme CCS settings in the process). Bob, on the other hand, wants to reach 19° C but refrains from settings in which the fan speed is higher than 3 and thus prefers milder adjustments. However, both prefer that the interior temperature decrease exponentially.

5.1 Background

Recent evidence suggests that drivers' current user experience often does not meet drivers' wishes, making many drivers desire more natural car interfaces [Li et al. 2012; Ramm et al. 2014]. For that purpose, some intelligent systems use drivers' observed behavior to automatically elicit the drivers' state or goals [Damiani et al. 2009]. For example, in [Rosenfeld et al. 2015b], the authors have shown that learning drivers' behavior can improve the performance of the adaptive cruise control system to the drivers' satisfaction. Others offer more expressive interfaces that are more natural for the driver to use and understand [Hwang et al. 2011; Politis et al. 2015]. The most relevant works within the context of CCS are [Azaria et al. 2016, 2015; Rosenfeld et al. 2015a], in which the authors try to elicit drivers' climate control preferences in order to provide advice to the driver that will help her reduce the climate system's energy consumption. The authors did not account for the possibility of the agent automatically changing the CCS settings nor did they allow for natural input from the driver.

The thermal comfort of human subjects has been exhaustively investigated over the last four decades, resulting in the ISO 7730 standard[8] [ASHRAE 2013]. The standard, which was also found to be applicable in car cabins, is aimed at predicting the degree of thermal comfort of an *average person* exposed to a certain *steady* environment (see [Croitoru et al. 2015] for a recent survey). Unfortunately, the standard does not provide answers on how a system should bring about a comfortable state.

Furthermore, the standard relies on the assumption that user-specific parameters are available, such as thermal sensitivity, clothing and activity level. Despite recent attempts to personalize thermal comfort models [Auffenberg et al. 2015], state-of-the-art thermal comfort models do not provide personalized or adaptive thermal comfort predictions.

Using the Exponentron algorithm, we will next describe a competing approach which does not necessitate the identification of user-specific characteristics prior to its deployment.

5.2 The NICE Agent Design

We present an intelligent agent for Natural Interaction with humans in CCS using the Exponentron algorithm, which we named the *NICE* agent for short. The agent's goal is to minimize the number of interactions needed by a driver to reach her desired comfort state and maximize the driver's satisfaction from the interaction process.

[8] Also known as Fanger's Predicted Mean Vote (PMV) criteria.

The agent implements the Exponentron algorithm (Algorithm 1) in order to predict the driver's desired climate changes during the ride and thereby change the CCS setting.

During the process, the driver may provide feedback to the agent using natural comments, such as "Too cold" or "Too hot", using the natural interface described in Section 3.2.2. These comments, in turn, are used to adapt the Exponentron's predictions, as we will soon describe.

A CCS setting is a tuple $\omega =< temp, f, d >$, where *temp* is the set temperature (an integer between 16 and 35 degrees C), f is the fan strength (an integer between 1 and 8) and d is the air delivery (1=face only, 2=face and feet, 3=feet). Two additional parameters are e, which is the external temperature (the temperature outside the car), and i, which is the internal cabin temperature. At time t, we denoted the CCS setting as ω_t, the external temperature as e_t and the internal cabin temperature as i_t.

The NICE agent uses 3 models; a *CCS model*, a *human driver model* and an *Exponentron prediction model*. The construction of these models is described later in this section. The NICE agent uses the three models in the following manner: At time t, the NICE agent predicts the driver's desired cabin temperature for the next time frame, $\hat{i}_{t+1}$, using the Exponentron prediction model. Given $\hat{i}_{t+1}$, the agent calls the CCS model and receives and implements a CCS setting ω_t which is predicted to bring about $\hat{i}_{t+1}$. Given that no comment is presented by the driver during the next 15 seconds, the agent assumes that the Exponentron's prediction, $\hat{i}_{t+1}$, and the CCS setting ω_t suit the driver's preferences and the process is repeated. The *com* signal takes the value of 0 since no comment was given by the driver. The driver can interrupt the above process (which otherwise will continue throughout the entire ride) by providing feedback. If a comment (c) is given within 15 seconds of implementing ω_t, then the agent uses the human driver model to predict the driver's desired CCS setting, $\hat{\omega}_t$, and implements it instantaneously in the CCS. The system then maintains the new CCS setting until 15 seconds pass in which no further feedback is provided by the driver. Namely, if the driver provides another comment within 15 seconds of his last comment, the human driver's model is called on once again and the 15-second timer is re-set. Once 15 seconds pass without further comments, the resulting cabin temperature is used to update the Exponentron's parameters. To that end, the *com* signal is set to 1. That is, the Exponentron's parameters can only be adjusted when the driver interacts with the agent. Figure 4 illustrates the agent's algorithmic scheme.

5.2.1 *The CCS Model.*

Recall that the CCS model is used to determine which CCS setting ω_t will bring about the desired change in the internal cabin temperature over a course of 15 seconds. For that purpose, the model receives i_t, ω_{t-1} and $\hat{i}_{t+1}$.

In order to train the CCS model, thirty distinct CCS settings were selected such that their set temperature, *temp*, was lower than the initial cabin temperature, i_0, at the time of the experiment. This property is required to enforce a cooling condition, which we examine in this study. We counter-balanced the selected CCS settings to account for the different possible ωs; namely, different *temp*, f and d values. Each CCS setting was manually configured to the CCS at the beginning of the trial. The cabin temperature, i, and the external temperature, e, were recorded every 15 seconds until the car's cabin temperature reached a steady state. Between every 2 consecutive experiments the car was turned off and the car's doors were left open for 10 minutes so as to recreate the initial conditions.

From the 30 trials we conducted over the course of 3 days, we recorded 657 measurements. Each of the measurements corresponds to a change in the car's cabin temperature, that is $- i_{j+1} - i_j$, given e_j, and the CCS setting ω used in the trial. We fit the data using a simple linear regression model which yields the best fit out of the tested models[9]. Namely,

[9]We also examined other, more sophisticated modeling, for example using SVM with kernels. These models did not provide a significant improvement in prediction accuracy.

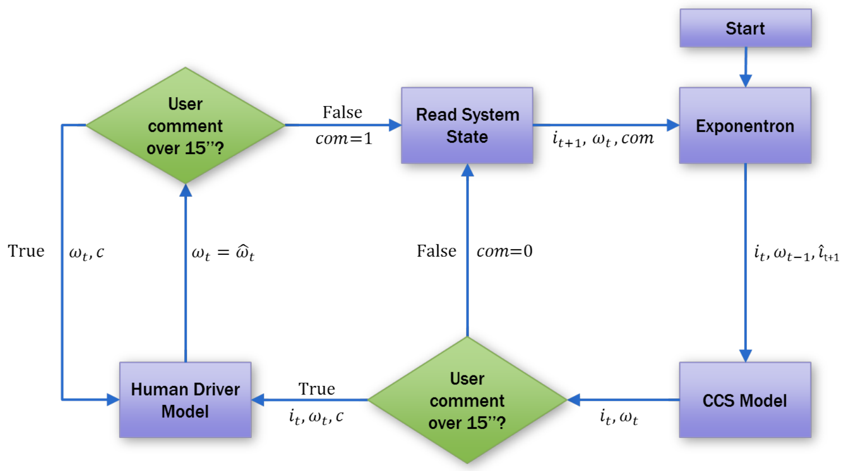

Fig. 4. The NICE agent's algorithmic scheme.

we constructed a model which, given i_t and ω, predicts i_{t+1}. To find a ω which is most likely to bring about the desired change, we iterate through all possible ωs. In the case of a tie, where more than a single CCS setting is expected to change the cabin temperature in the desired manner, the model outputs one of the CCS settings which is most similar to the previous CCS setting, ω_{t-1}. Recall that a CCS setting is a vector $< temp, f, d >$, therefore similarity is easily defined. In this work we used the cosine similarity.

Using cross-validation, the learned model yields a mean absolute error of 0.15°C and a strong correlation coefficient of 0.9. In comparison, using the last cabin temperature change, $\Delta_j = i_j - i_{j-1}$, as an estimation for the next cabin temperature, $\hat{i}_{j+1} = i_j + \Delta_j$, yields a mean absolute error of 0.51 and a correlation coefficient of 0.3.

5.2.2 The Human Driver Model. Given a driver's comment, the human driver model is used to predict the driver's desired CCS settings. The model is based on multi-dimensional regression: At time t when a comment (denoted as c) is given (i.e, a button is pressed), the model predicts the desired CCS setting $\hat{\omega}_t$, given ω_t, e_t, i_t, c_0 and the last 2 previously provided comments, denoted c_1, c_2.

In order to train the human driver model, we used the data collected in Section 3.2.2. Recall that in our experiment drivers were asked to interact with the newly designed natural interface while changing the CCS settings *manually*. The experiment recordings were translated into more than 100 vectors of the form $< \omega_t, e_t, i_t, c_0, c_1, c_2 >$ as described above, with the drivers' manually set CCS setting, ω', as their label. Each session resulted in a different number of vectors, depending on the session's length.

The multi-dimensional regression consists of 3 linear regression models, each predicting a different component of the desired CCS setting $\omega' =< temp, f, d >$. Using cross-validation, the prediction model yields a mean absolute error of 0.9 in predicting the next fan speed, f, and a mean absolute error of 1.02°C for the next set temperature, t. A high 97% accuracy in predicting the desired air delivery, d, was also recorded. Note that the NICE agent's algorithmic scheme (Figure 4) is designed in such a way that the human driver model can be replaced with another. Note, however, that deploying the baseline models (i.e., AR, ARMA or ES) seems inappropriate since they cannot adapt their predictions to each individual driver, which is central to this application setting.

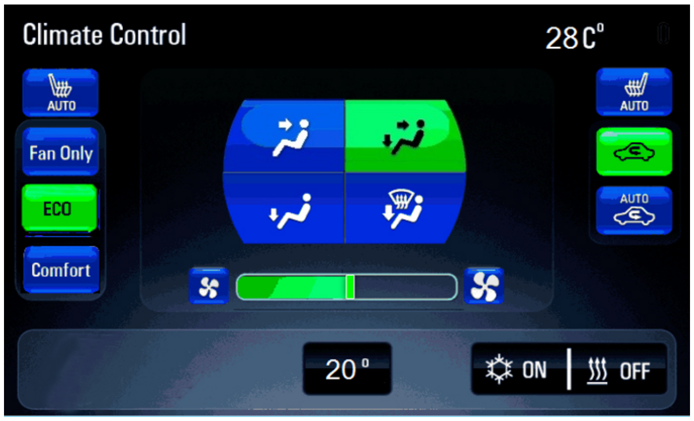

Fig. 5. The technical CCS used in our experiments.

5.2.3 The Exponentron Prediction Model. The Exponentron prediction model implements the Exponentron algorithm. The Exponentron is trained with the same procedure used in Section 3.2.2. The model receives an additional input bit *com*, signaling whether the driver provided a comment in the last time frame. If the *com* bit is 1, then an adaptation of the model's parameters is executed using the current cabin temperature i_t.

The θ learned parameters represent the driver's preferences. Specifically, the parameter a represents the driver's intended steady state cabin temperature and the parameters b and c represent the way in which the driver wishes to bring about the desired cabin temperature.

5.3 Evaluation

5.3.1 Experimental Methodology. We recruited 24 drivers who did not participate in the data collection phase described in Section 3.2.2, with an equal number of males and females, ranging in age from 25 to 60 (average of 34). In a similar protocol to that described in Section 3.2.2, each subject was asked to enter a car that was parked in a garage, recreating the environmental conditions - with temperatures ranging from 32°C to 37°C, averaging 35°C. Each subject participated in two consecutive trials. In each trial the subject was equipped with either the *Technical CCS* or the *NICE agent*. The technical CCS (see Figure 5) presented buttons similar to those available in the common CCS, with which the driver can explicitly select her desired CCS setting. Namely, we focus on two scales: one for the fan speed and the other for the temperature. The driver could change the setting by selecting her preferred fan speed and temperature on the designated scales. Namely, in a single interaction, the driver could change the CCS setting completely. Note that no intelligent agent was implemented to support it. The NICE agent uses the natural interface which is the same interface as described in Section 3.2.2. In both conditions, the graphical user interface was presented on a tablet covering the car's central stack in order to avoid biasing the results. Each subject was instructed to interact with the system as she saw fit by using the buttons available in the presented interface. While in the car the subject was given a cell phone with a driving simulator "Bus Simulator 3D" to be played while the experiment was conducted.

Once the cabin temperature, i, reached a steady state, the session came to an end. Each session lasted 2-6 minutes (mean of 4 minutes). After the session ended, the driver was asked to exit the car for a period of 10 minutes while the car's doors were left open in order to recreate initial conditions. The process was repeated once more under the condition that was not examined in the first session. Subjects were counter-balanced as to which condition they experienced first in order to maintain the scientific integrity of the results.

During each session we recorded the number of interactions needed by the driver in order to reach her desired steady state. At the end of the experiment, drivers were asked to fill out a post-experiment questionnaire aimed at evaluating their satisfaction from the examined interfaces.

5.3.2 Results and Analysis. We first analyze the number of interactions needed by drivers to reach their desired steady states under the examined conditions. Then we summarize the subjects' answers in the post-experiment questionnaire. Note that the technical CCS is the current state-of-the-art CCS and acts as the benchmark in the following analysis.

The NICE agent required a significantly lower number of interactions from the driver compared to the technical CCS using a t-test ($p < 0.05$). The NICE agent averaged 5.35 interactions carried out by the driver until a steady state was reached, while the technical CCS averaged 6.54 interactions. Out of the 24 subjects, only 8 subjects required more interactions while equipped with the NICE agent compared to their benchmark score. See Figure 6 for a summary.

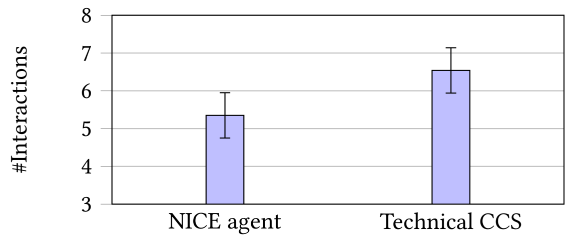

Fig. 6. Average number of interactions per interface; (the lower the better). Error bars indicate standard errors.

Recall that the NICE agent's goal is to automatically set and adjust the car's CCS setting throughout the ride to the driver's satisfaction. In order to assess the driver's satisfaction from the interaction, at the end of the experiment we asked each driver which, if any, of the tested conditions she would want to see available in her car. Out of the 24 subjects, 13 subjects stated that they want to use the NICE agent, while 10 subjects stated their preference for the technical CCS. We also asked subjects to state their satisfaction level from the tested conditions. Subjects reported an average score of 6.2 out of 10 when asked for their satisfaction from the technical CCS. This result is significantly lower, using t-test ($p < 0.05$), compared to the NICE agent which recorded an average score of 7.2 out of 10.

To summarize, the results indicate that the NICE agent is able to reduce the average number of interactions needed by drivers in order to achieve their desired comfort states (6.54 vs. 5.35, $p < 0.05$) to the drivers' satisfaction, as portrayed in the increase of the subjects' subjective satisfaction (7.2 vs. 6.2, $p < 0.05$) and the subjects' preferred interaction mode (13 vs. 10).

It is also important to discuss the limitations of our experiment. First, due to insurance reasons, participants did not actually drive a car during the experiment. As a result, the experiment only emulates real driving conditions. In addition, the technical CCS interface offers more options than the natural interface, a fact which may influence drivers' behavior. Finally, since the natural interface is new and exciting for most participants, it may also be the case that participants "experimented" with it more than they did with the standard technical CCS interface.

6 LIMITATIONS

When presenting a new approach, in our case a new approach for the online prediction of time series, it is worthwhile to discuss its limitations. We can identify three major limitations:

First and foremost, our approach is built upon an assumed time series behavior that need not exist for every domain. For example, for the prediction of stock market indeces it is unlikely to assume any a-priori behavioral pattern of the time series [Firth 1991]. As shown in Sections 3.2.1 and 4.2.1, very noisy time series, which poorly follow a known behavioral pattern, are unsatisfactorily predicted by the proposed Exponentron or Sigmoidtron algorithms.

Second, it is reasonable to expect that if the wrong behavior pattern is assumed, then a specialized algorithm may not provide any advantage over classical approaches or may even prove to be countereffective. To examine this hypothesis we conduct the following experiment: we evaluated the performance of the Exponentron algorithm on the academic H-Index dataset (Section 4.2.2) and the Sigmoidtron algorithm on the inbound calls in a call center dataset (Section 3.2.3). The results of this experiment clearly support our hypothesis – the Exponentron is significantly outperformed by the Sigmoidtron on the academic H-Index dataset, $p < 0.05$, and is statistically indistinguishable from the benchmark models, scoring an average error of 0.82 compared to 0.76 achieved by the *best* performing benchmark model (*ES*) and 1.08 achieved by the *worst* performing benchmark model (*ARMA*). Similarly, the Sigmoidtron is significantly outperformed by the Exponentron for the inbound calls dataset, $p < 0.05$, and is statistically indistinguishable from the benchmark models, scoring an average error of 13 compared to 9.8 achieved by the *best* performing benchmark model (*ARMA*) and 13.7 achieved by the *worst* performing benchmark model (*ES*). See Table 9 for the results.

Method	Mean Absolute Error (H-Index)	Mean Absolute Error (Inbound Calls)
$AR(1)$	0.8	10.2
$ARMA(1,1)$	1.08	9.8
$ES(0.9,1)$	0.76	13.7
Exponentron	*0.82*	**5.8**
Sigmoidtron	**0.6**	*13*

Table 9. Prediction of H-Index values and call arrivals in a real-world call center. Numbers indicate the mean absolute error made by each prediction method.

Finally, as was the case for the sigmoid function, a regret bound cannot be derived for non-convex time series behaviors. This fact limits our ability to guarantee a worst case performance for some time series behaviors.

7 CONCLUSIONS

In this article, we propose the development of "tailored" online time series prediction algorithms to assumed time series behaviors. Our approach was demonstrated by the presentation of two online algorithms: the Exponentron which is directed at the online prediction of assumed exponential decay time series; and the Sigmoidtron which is directed at the online prediction of assumed sigmoidal time series.

Through an extensive analysis and evaluation, we demonstrate our algorithms' potential benefits and limitations compared to the classic time series prediction algorithms commonly applied today. Specifically, for the Exponentron algorithm, we demonstrate its potential benefits both theoretically using regret bounds, empirically using synthetic and two real-world datasets and applicatively using the novel NICE agent which was tested in the field with human participants. For the Sigmoidtron algorithm, its potential benefits are demonstrated through an extensive empirical study using both synthetic and real-world datasets. In addition, we show how both our algorithms are sensitive to high noise levels.

Future work will include the investigation of other time-dependent phenomena that are likely to adhere to an a-priori assumed functional behavior. For example, time series which include cyclic components, commonly used to capture the notion of seasonality, are popular in modeling economic phenomena such as unemployment [Ghysels 1996]. The developed models would be tested as a component in a decision support system that assists different stakeholders such as investors or employers. Another possible avenue for further investigation is the automatic identification of the expected behavior of a given time series based on its beginning or other, similar, time series.

ACKNOWLEDGMENTS

This article extends a previous report from the 22nd European Conference on Artificial Intelligence (ECAI) [Rosenfeld et al. 2016] which only addressed the exponential decay time series condition.

In this article we provide an investigation of the sigmoidal time series condition, develop the Sigmoidtron algorithm and evaluate it using synthetic and real-world datasets. These additions allow us to demonstrate the benefits of our proposed approach to more than a single time series behavior and enhance the credibility and validity of our approach for real-world deployment.

REFERENCES

Oren Anava, Elad Hazan, Shie Mannor, and Ohad Shamir. 2013. Online learning for time series prediction. In *Conference on learning theory*. 172–184.

Oren Anava, Elad Hazan, and Assaf Zeevi. 2015. Online time series prediction with missing data. In *International Conference on Machine Learning*. 2191–2199.

ASHRAE. 2013. Standard 55-2013. Thermal environmental conditions for human occupancy. ASHRAE. *American Society of Heating, Refrigerating and Air-Conditioning Engineering* (2013).

Frederik Auffenberg, Sebastian Stein, and Alex Rogers. 2015. A personalised thermal comfort model using a Bayesian network. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*.

Amos Azaria, YaâĂŹakov Gal, Sarit Kraus, and Claudia V Goldman. 2016. Strategic advice provision in repeated human-agent interactions. *Autonomous Agents and Multi-Agent Systems* 30, 1 (2016), 4–29.

Amos Azaria, Ariel Rosenfeld, Sarit Kraus, Claudia V Goldman, and Omer Tsimhoni. 2015. Advice Provision for Energy Saving in Automobile Climate-Control System. *AI Magazine* 36, 3 (2015), 61–72.

José M Bernardo and Adrian FM Smith. 2001. Bayesian theory. (2001).

Dimitri P Bertsekas. 1999. Nonlinear programming. (1999).

Léon Bottou. 2012. Stochastic gradient tricks. *Neural Networks, Tricks of the Trade, Reloaded* (2012), 430–445.

George Box, Gwilym M. Jenkins, and Gregory C. Reinsel. 1994. *Time Series Analysis: Forecasting and Control*. Prentice-Hall.

Cristiana Croitoru, Ilinca Nastase, Florin Bode, Amina Meslem, and Angel Dogeanu. 2015. Thermal comfort models for indoor spaces and vehiclesâĂŤCurrent capabilities and future perspectives. *Renewable and Sustainable Energy Reviews* 44 (2015), 304–318.

Sergio Damiani, Enrica Deregibus, and Luisa Andreone. 2009. Driver-vehicle interfaces and interaction: where are they going? *European transport research review* 1, 2 (2009), 87–96.

Norman Richard Draper, Harry Smith, and Elizabeth Pownell. 1966. *Applied regression analysis*. Vol. 3. Wiley New York.

Hermann Ebbinghaus. 1913. *Memory: A contribution to experimental psychology*. Number 3. University Microfilms.

WJ Firth. 1991. Chaos–predicting the unpredictable. *BMJ: British Medical Journal* 303, 6817 (1991), 1565.

Noah Gans, Ger Koole, and Avishai Mandelbaum. 2003. Telephone call centers: Tutorial, review, and research prospects. *Manufacturing & Service Operations Management* 5, 2 (2003), 79–141.

Everette S Gardner. 1985. Exponential smoothing: The state of the art. *Journal of forecasting* 4, 1 (1985), 1–28.

Eric Ghysels. 1996. On the economics and econometrics of seasonality. In *Advances in Econometrics. Sixth World Congress*, Vol. 1. 257–322.

Paul Goodwin. 2010. The Holt-Winters Approach to Exponential Smoothing: 50 Years Old and Going Strong. *Foresight: The International Journal of Applied Forecasting* 19 (2010), 30–33.

I. Guedj and A. Mandelbaum. 2000. *Call Center Data*. Technical Report. Technion, Israel Institute of Technology. http://iew3.technion.ac.il/serveng/callcenterdata/index.htmll

Tian Guo, Zhao Xu, Xin Yao, Haifeng Chen, Karl Aberer, and Koichi Funaya. 2016. Robust online time series prediction with recurrent neural networks. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. Ieee, 816–825.

Jorge E Hirsch. 2005. An index to quantify an individual's scientific research output. *Proceedings of the National academy of Sciences* 102, 46 (2005), 16569–16572.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

Jee Yeon Hwang, Kent Larson, Ryan Chin, and Henry Holtzman. 2011. Expressive driver-vehicle interface design. In *Proceedings of the 2011 Conference on Designing Pleasurable Products and Interfaces*. ACM, 19.

Mark Kozdoba, Jakub Marecek, Tigran Tchrakian, and Shie Mannor. 2018. On-Line Learning of Linear Dynamical Systems: Exponential Forgetting in Kalman Filters. *arXiv preprint arXiv:1809.05870* (2018).

Arnd Leike. 2001. Demonstration of the exponential decay law using beer froth. *European Journal of Physics* 23, 1 (2001), 21.

Li Li, Ding Wen, Nan-Ning Zheng, and Lin-Cheng Shen. 2012. Cognitive cars: A new frontier for ADAS research. *Intelligent Transportation Systems, IEEE Transactions on* 13, 1 (2012), 395–407.

Nan-Ying Liang, Guang-Bin Huang, Paramasivan Saratchandran, and Narasimhan Sundararajan. 2006. A fast and accurate online sequential learning algorithm for feedforward networks. *IEEE Transactions on neural networks* 17, 6 (2006), 1411–1423.

Chenghao Liu, Steven CH Hoi, Peilin Zhao, and Jianling Sun. 2016. Online arima algorithms for time series prediction. In *Thirtieth AAAI conference on artificial intelligence*.

Avishay Mandelbaum, Anat Sakov, and Sergey Zeltyn. 2001. *Empirical analysis of a telephone call center*. Technical Report. Technion, Israel Institute of Technology.

Luis Moreira-Matias, Joao Gama, Michel Ferreira, João Mendes-Moreira, and Luis Damas. 2013. Predicting Taxi–Passenger Demand Using Streaming Data. *Intelligent Transportation Systems, IEEE Transactions on* 14, 3 (2013), 1393–1402.

Jaap MJ Murre. 2014. S-shaped learning curves. *Psychonomic bulletin & review* 21, 2 (2014), 344–356.

Isaac Newton. 1701. *Scala graduum caloris: calorum descriptiones & signa*. Royal Society of London.

Eoin O'Mahony and David B Shmoys. 2015. Data Analysis and Optimization for (Citi) Bike Sharing. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Ioannis Politis, Stephen Brewster, and Frank Pollick. 2015. To Beep or Not to Beep?: Comparing Abstract versus Language-Based Multimodal Driver Displays. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 3971–3980.

Simon Ramm, Joseph Giacomin, Duncan Robertson, and Alessio Malizia. 2014. A First Approach to Understanding and Measuring Naturalness in Driver-Car Interaction. In *Proceedings of the 6th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*. ACM, 1–10.

Cédric Richard, José Carlos M Bermudez, and Paul Honeine. 2009. Online prediction of time series data with kernels. *IEEE Transactions on Signal Processing* 57, 3 (2009), 1058–1067.

Ariel Rosenfeld, Amos Azaria, Sarit Kraus, Claudia V Goldman, and Omer Tsimhoni. 2015a. Adaptive advice in automobile climate control systems. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 543–551.

Avi Rosenfeld, Zevi Bareket, Claudia V Goldman, David J LeBlanc, and Omer Tsimhoni. 2015b. Learning DriversâĂŹ Behavior to Improve Adaptive Cruise Control. *Journal of Intelligent Transportation Systems* 19, 1 (2015), 18–31.

Ariel Rosenfeld, Joseph Keshet, and Sarit Goldman, Claudia V.and Kraus. 2016. Online Prediction of Exponential Decay Time Series with Human-Agent Application. In *Proceedings of the 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands*. 595–603.

Ariel Rosenfeld and Sarit Kraus. 2018. Predicting Human Decision-Making: From Prediction to Action. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 12, 1 (2018), 1–150.

Shai Shalev-Shwartz. 2011. Online learning and online convex optimization. *Foundations and Trends in Machine Learning* 4, 2 (2011), 107–194.

Avraham Shvartzon, Amos Azaria, Sarit Kraus, Claudia V Goldman, Joachim Meyer, and Omer Tsimhoni. 2016. Personalized Alert Agent for Optimal User Performance. In *Proceedings of the 30th International Conference on Artificial Intelligence (AAAI)*. AAAI.

Gordon K Smyth. 2002. Nonlinear regression. *Encyclopedia of environmetrics* (2002).

Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 990–998.

M Th Van Genuchten and S Gupta. 1993. A reassessment of the crop tolerance response function. *Journal of the Indian Society of Soil Science* 41 (1993), 730–737.

J Vandermeer. 2010. How populations grow: the exponential and logistic equations. *Nature Education Knowledge* 3, 10 (2010), 15.

Pierre-François Verhulst. 1838. Notice sur la loi que la population suit dans son accroissement. *Corresp. Math. Phys.* 10 (1838), 113–126.

Haimin Yang, Zhisong Pan, and Qing Tao. 2019. Online Learning for Time Series Prediction of AR Model with Missing Data. *Neural Processing Letters* (2019), 1–17.

Haimin Yang, Zhisong Pan, Qing Tao, and Junyang Qiu. 2018. Online learning for vector autoregressive moving-average time series prediction. *Neurocomputing* 315 (2018), 9–17.

Yongzhi Zhang, Rui Xiong, Hongwen He, and Michael G Pecht. 2018. Long short-term memory recurrent neural network for remaining useful life prediction of lithium-ion batteries. *IEEE Transactions on Vehicular Technology* 67, 7 (2018), 5695–5705.

Martin Zinkevich. 2003. Online convex programming and generalized infinitesimal gradient ascent. (2003).