Generalized *Dynamical* Machine Learning

PG Madhavan
15 September 2016

We introduce a new Machine Learning (ML) solution for Dynamical, Non-linear, In-Stream Analytics. Clearly, such a solution will accommodate Static, Linear and Offline (or any combination thereof) Machine Learning tasks. Beyond traditional ML use cases, major applications of Dynamical ML include IoT, Finance forecasting and other challenging real-life problems.

As the Data Models we use get more sophisticated, the associated estimation methods lead to more and more powerful Machine Learning (ML) paradigms. This note details concepts, theory, algorithms and applications of a new and powerful general solution. SYSTEMS Analytics book (PG Madhavan, 2016) is the definitive companion to the material in this note since all the basics leading up the sophisticated topics in this note are developed in the book.

Major topics addressed include –

1. Machine Learning (ML) that require static or dynamical, dynamical or time-varying dynamical, linear or non-linear mapping – meaning ANY Machine Learning problem!
2. Need for DYNAMICAL Machine Learning – real-life business solution requirements.
3. A novel ontology of Machine Learning.
4. State space, Bayesian Conditional Expectation, Kalman Filter, Kernel methods, Recurrence.
5. Applications to Classification and multi-step Forecasting.

Dynamical ML solution presented here is a unique composition of a few well-known methods with deep theoretical background and history.

a) Kalman Filtering: The depth and breadth of Kalman's theory and applications are unmatched. ***In this year of Rudolf Kalman's demise, this article is dedicated to his memory.***
b) Time-varying estimation: Many decades of contributions of Peter Young (2011) to the theory and applications of non-stationary stochastic process estimation form the basis of time-varying "Kalman Filtering" in this note.
c) Kernel-projection: Recent work of Huang and others in what they call "Extreme Learning Machines" form the basis of our Cover theorem implementation.
d) Recurrence in neural networks provides a valuable memory feature in the current solution. Some early work on the theory is available here.

An outstanding overall reference for related material is Neural Networks & Learning Machines (3rd edition, 2006), written by my beloved teacher, Simon Haykin.

Why Dynamical?

Machine Learning involves, given a set of inputs and outputs, finding a map between the two during supervised "Training" and using this map for business purposes during "Operation" (which is called "Testing" during pre-operation evaluation). By using the map from Training stage in Operation stage, we have implicitly assumed that the map is "static" – did not change between the two stages. In real life, *static* is hardly the case . . .

If you contribute to the view that *true learning is "generalization from past experience AND the results of new action*" and therefore ML business solutions ought to be like flu shots (adjust the mix and apply on a regular basis), then ***every ML application is a case of Dynamical Machine Learning***. Learning does not stop (hence not static) at Training but continues based on the results of actions during Operation – hence the ML map has to be ***dynamically*** updated!

Data Models

Data model is the starting point for any ML solution. It captures the static or dynamical nature of the ML solution.

Static Model: Regression models used in ML are usually static, defined as follows.
Multiple Linear Regression Model: $y = a_0 + a_1 x_1 + a_2 x_2 + \ldots + a_M x_M + w$

We know that it is static because there are NO time variables in the equation. In the classic Fisher Iris case, x's are the flower attributes and 'y' is the type of Iris. Let us make the lack of time dependence explicit.
$y[n] = a_0 + a_1 x_1[n] + a_2 x_2[n] + \ldots + a_M x_M[n] + w[n]$

It is the same time index, n, on both sides of the equation and hence there is no dependence on time.

Dynamical Model: Box & Jenkins time series model is a familiar dynamical model.
$y[n] = - a_1 y[n-1] - \ldots - a_D y[n-D] + b_1 x_1[n] + \ldots + b_M x_M[n-M+1] + e[n]$

This is the classic ARMA model. There are delayed time indices on the right-hand side (but note that the coefficients, a & b, are constants). This provides "memory" to the model and hence the output, y, evolves over time. Therefore, they are called Dynamical models.

Time-Varying Dynamical Model: In the model above, if the coefficients, a & b, were not constant but indexed by time, n, that is an example of a time-varying dynamical model. We choose to use a flexible model called State Space model.
State-space Model:

$$\mathbf{s}[n] = \mathbf{A}\,\mathbf{s}[n-1] + \mathbf{D}\,\mathbf{q}[n-1]$$
$$y[n] = \mathbf{H}[n]\,\mathbf{s}[n] + r[n]$$

Detailed discussion of such models is available in the book, "SYSTEMS Analytics", but we will note here that output, y, is a function of 's' (so-called "States") and the first equation shows that these States evolve according to a Markov process. This is akin to allowing the equation coefficients in the ARMA model to evolve over time, thus accommodating time-variability of the dynamics of the data.

The only other possibility that these Data Models do not accommodate is non-linearity. By choice, we will not generalize our data model any further since non-linearity is best accommodated in a different fashion which will be made clear later. With that caveat, ***we have the most flexible data model we will ever encounter in the State Space model.***

Machine Learning Ontology

In plain English**,** Machine Learning answers the question, "What is the likely Class that the measured Attributes belong to?"

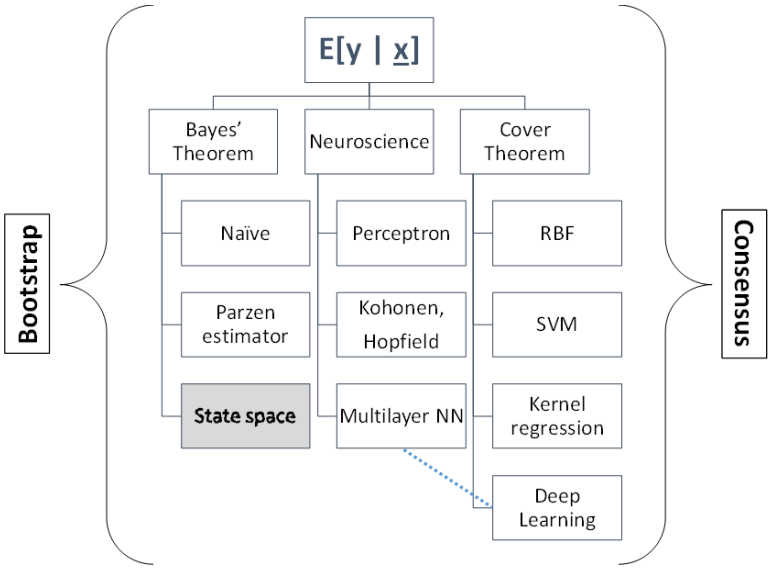
In Probability speak*: "**What is the ***Conditional Expectation of Class (y) given Attributes (x) or **E[y | x]?**"**

My orientation is Bayesian; so I have put together an ontology below that organizes all we know about ML from answering the Conditional Expectation question. The highlight of this ontology is the collection of the vast material under just 3 topics: Bayes Theorem, Cover Theorem and Neuroscience & ad hoc methods. In ML practice, these ML methods are "wrapped" by "bootstrap" and "consensus" methods.

- Cover Theorem states (from Haykin, 2006): *"A complex pattern classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated".*

- Estimating the Conditional Expectation, E[y | x], where 'y' is the output and 'x' is the input, from the conditional probability density function of the output is the stuff of hard core probabilistic approach. Here are the basics.
  E[y | $\underline{\mathbf{x}}$] = $\int_{-\infty}^{\infty} y\, p_{y|x}\,(y \mid \mathbf{x})\, dy$
  Conditional density is obtained as -
  $p_{y|x}(y|\mathbf{x}) = \frac{p_{x,y}(\mathbf{x},y)}{p_x(\mathbf{x})}$
  From the vast amount of Training Set data, one estimates the JOINT density function, $p_{x,y}(\mathbf{x}, y)$, from which the marginal density, $p_x(\mathbf{x})$ can be obtained by integrating out 'y'. Working from bottom to top, one gets the answer to the question, "What is the ***Conditional Expectation*** of Class (y) given Attributes (x) or E[y | $\underline{\mathbf{x}}$]?"

- The middle branch is a "catch all" one dominated by Neuroscience starting with Perceptrons. Deep Learning is the newest incarnation of this approach (combined with high-dimensionality) in this ontology.



Statistical Design of Experiments provides the "wrapper" that is all important for successful practical applications of ML.
Input side - Bootstrap methods: The objective is to maximize Training Set information use.

Output side - Consensus methods: Solve the problem using independent ML methods and combine the results.

Practical usage of ML maps requires a rigorous framework of Design of Experiments (Box, et al., an old classic). When you surround solid ML maps with strong statistical experimentation discipline, we get robust, repeatable and practically useful results.

Within this ontology, ***the State space block (shaded) is the only explicitly time-varying dynamical mapping approach*** (even though some of the other blocks can be made dynamical with varying amount of difficulty).

Generalized Solution Approach

We seek to finding a mapping that can be static or dynamical, dynamical or time-varying dynamical, linear or non-linear. This is a tall order!

Basic solution components are developed in a principled and mathematically rigorous fashion. Then they are assembled in an ad hoc fashion such that we achieve a "working" solution to the complex mapping problem above. We validate the solution by applying it to hard ML problems. This is a typical "engineering" approach to pragmatic problem solving which we readily embrace!

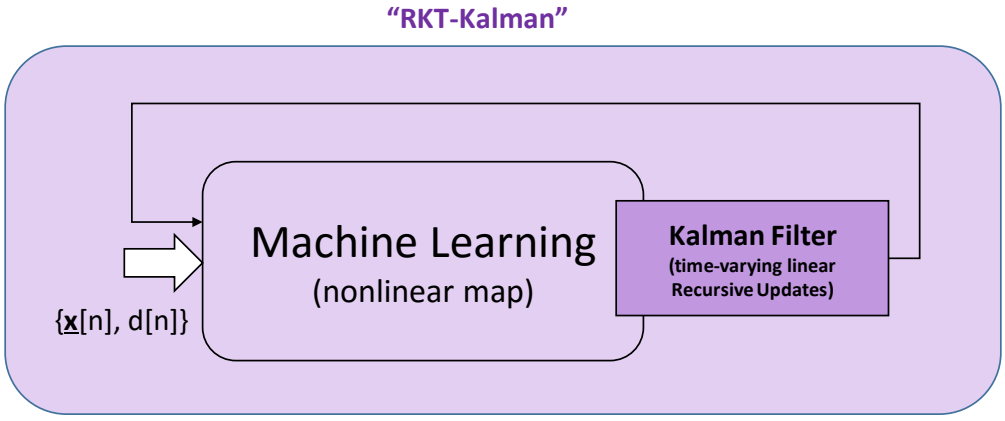
We have THREE major requirements for our Generalized Dynamical ML solution:

1. Data model must be time-varying dynamical – satisfies our "generalized learning" definition.
2. Mapping must be non-linear – satisfies Cover Theorem.
3. Map must have long and short-term memory – typically requires a system that has feedback from output to input or "recurrence" as it is called in artificial neural network theory.

We meet the three requirements using state-space "Recurrent Kernel-projection time-varying Kalman" method.

State-space "Recurrent Kernel-projection time-varying Kalman" Method

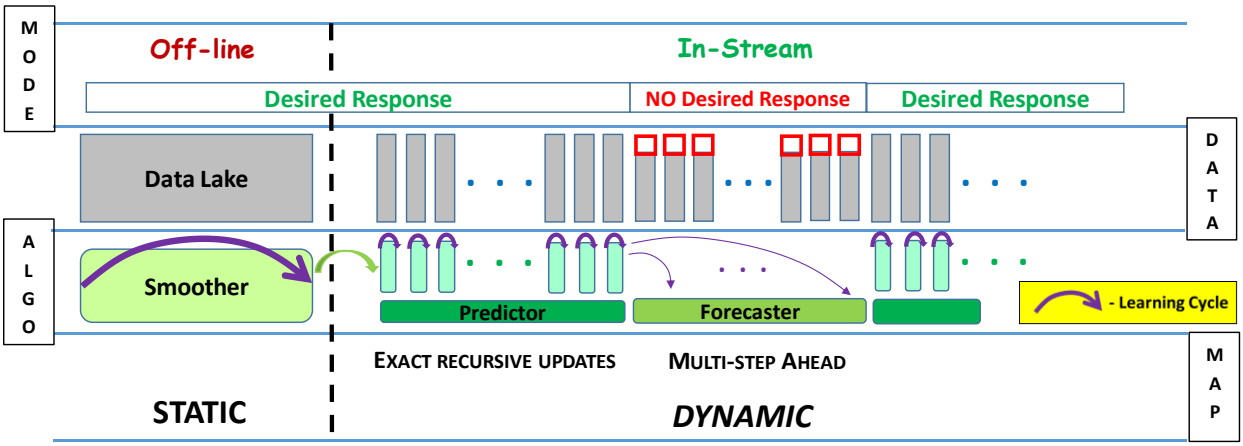
State-space Recurrent Kernel-projection Time-varying Kalman or **"RKT-Kalman"** method is a novel assemblage of well-known Kalman Filter, Kernel method and Recurrent architecture with some key integration "glue". Here is the solution at a high-level.



Attributes, $\underline{\mathbf{x}}$, are non-linearly transformed and projected to a high dimensional space (per Cover Theorem). Non-linearly transformed Attributes determine certain matrices of the State-space model. Kalman Predictor/Filter using the class labels, d, as the "desired response", recursively updates the States so as to obtain the Conditional Expectations required. Kalman output is fed back as an input to the system – architecturally, the "recirculation" so achieved preserves long term memory of weighted past Attributes and desired responses.

RKT-Kalman Usage Details

We will take a classification example to elucidate the usage of RKT-Kalman solution. There are many nuanced aspects to the practical application of RFT-Kalman. Once usage is well-understood, we will address the theory in detail.



Processing MODE: (1) Offline or (2) In-Stream

- During Offline condition, all the Training Set data, $\{\underline{\mathbf{x}}_i[n], d[n]\}$ for n=0 to N, are available.
- During In-Stream stage, $\{\underline{\mathbf{x}}_i[n], d[n]\}$ PAIRS are available in real-time. In real-life applications, it is possible that in some intervals, d[n]'s are not available and we have to make do with just $\{\underline{\mathbf{x}}_i[n]\}$.
- During Offline, algorithms can operate on all the data, i.e., $\{\underline{\mathbf{x}}_i[n], d[n]\}$ for n=0 to N, to produce one ML map – "Static" machine learning. During In-Stream processing, machine learning has to be "dynamic".

Kalman Filter: FOUR modes of operation to develop and update Machine Learning map.

1. Smoother: Estimate Conditional Expectation of States, $E[\underline{\mathbf{s}}[n] \mid \underline{\mathbf{x}}_i[n], d[n]$ for n=0 to N].
2. Predictor: Estimate 1-step ahead *predicted* States, $E[\underline{\mathbf{s}}[n] \mid \underline{\mathbf{x}}_i[n], d[n-1]]$.
3. Filter: Estimate filtered States, $E[\underline{\mathbf{s}}[n] \mid \underline{\mathbf{x}}_i[n], d[n]]$.
4. Forecaster: Estimate K-step (K > 1) ahead predicted States, $E[\underline{\mathbf{s}}[n+K] \mid \underline{\mathbf{x}}_i[n], d[n]]$.

NOTE: Time indexes of $\{\underline{\mathbf{x}}_i[.], d[.]\}$ in each case! Also, we reserve the term, "Prediction" for 1-step ahead forecast. At time instant = n, $\underline{\mathbf{x}}_i[n]$ is available at the START of instant, n, when Prediction is performed. THEN, d[n] becomes available and Filtering is done.

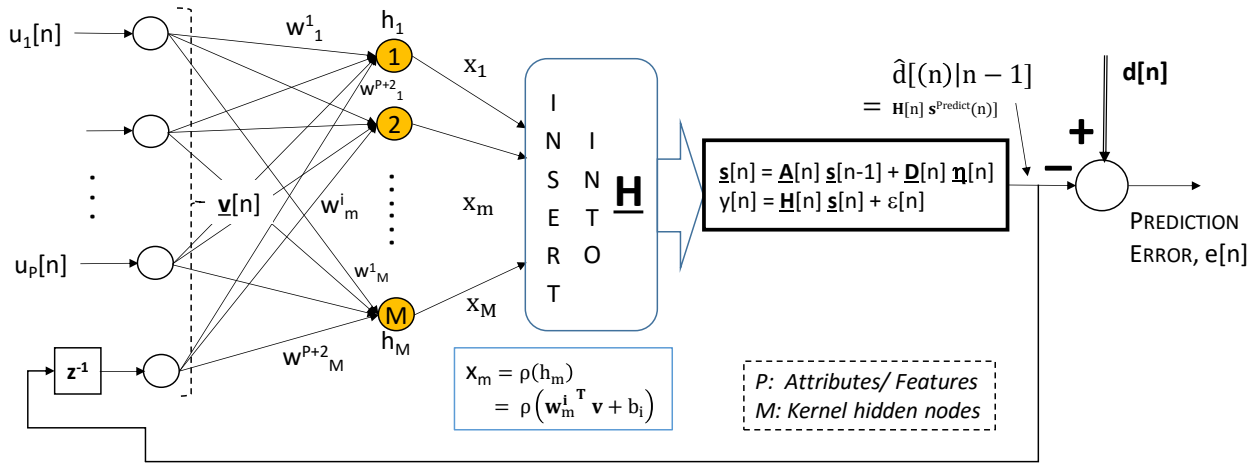
Description of RKT-Kalman Operation:

1. In the preparatory "Offline" stage, we develop a ML map using the huge amounts of data in the Data Lake.
2. Kalman SMOOTHER is the algorithm which utilizes all the data in its operation.
3. Kalman Smoother performance is assessed on a unseen Test Set and we assure that the classifier performance meets our requirement.
4. In-Stream Processing:
    a) Kalman FILTER is initialized using the Smoother we have estimated in the previous Offline stage.
    b) As each new $\underline{\mathbf{x}}_i[n]$ arrives, Kalman PREDICTOR updates the States and makes a prediction of the Class label.
    c) When the corresponding d[n] which is the True Class Label, is available, we can compute the error made by the predictor and using this error, Kalman Predictor and Filter States are updated for instant, n.
5. Forecasting:
    a) During In-stream operation, if a d[n] associated with a $\underline{\mathbf{x}}_i[n]$ is not available, we cannot compute the Class Label error in Step 4,c above.
    b) Using the last Kalman Predictor that was updated, we predict further out into the future. In the absence of true desired response, future desired responses are approximated as Kalman Filter output which enables error computation for updates. Clearly, the quality of forecasts will deteriorate rapidly as the "K" in K-steps ahead forecast increases. More details are discussed in the Theory section below.

NOTE on Recursive Estimation: Compared to Block method that needs all data at one time, Recursive estimation uses the current data to update the past estimate "*in-situ*". Recursive estimation permits DYNAMICAL estimation since updated estimates can "track" the time-variability of the ML map. *Recursion and Recurrence (an architectural feature) are unrelated!*

RKT-Kalman Dynamical ML Classifier

Here is the full architecture of the RKT-Kalman for a classification task.



Input, $\underline{\mathbf{v}}[n]$, consists of Attributes, $\underline{\mathbf{u}}[n]$ and 1-step delayed sample of the classifier output (z^{-1} indicates the 1-step delay). $\underline{\mathbf{v}}[n]$ is projected nonlinearly to a high-dimensional space of dimension M as follows.

Kernel-Projection:

- Choose M and function $\rho(.)$
- Assign random numbers to all $\underline{\mathbf{w}}^{i}_{m}$ and b_m
- At each 'n', $s_m[n] = \underline{\mathbf{w}}^{i}_{m}{}^{T}\ \underline{\mathbf{v}}[n] + b_m$
- Output, $\underline{\mathbf{x}}_m[n] = [\rho(s_1[n]) \ldots \rho(s_m[n]) \ldots \rho(s_M[n])]^T$

NOTE: More details about the choice of $\rho(.)$, M, etc. are fully discussed in High-Performance Extreme Learning Machines (2015).

Time-Varying Kalman Filtering:

Our State-space data model:

State Space Model:

$\underline{\mathbf{s}}[n] = \underline{\mathbf{A}}\ \underline{\mathbf{s}}[n-1] + \underline{\mathbf{D}}\ \underline{\mathbf{q}}[n-1]$

$d[n] = \underline{\mathbf{H}}[n]\ \underline{\mathbf{s}}[n] + r[n]$ where $\underline{\mathbf{q}}[n-1] = N[\underline{\mathbf{0}}, \underline{\mathbf{Q}}(n-1)]$ and $r[n] = N[0, R(n)]$ . . . (A)

The key notion of time-varying dynamics for States is to endow the State equation in equation (A) with more "degrees of freedom" to evolve. This is accomplished by AUGMENTING the State vector. Each State is defined as a 2-component vector - the State's value AND its slope (which is the change in value over a sampling interval).

I.e., $\underline{\mathbf{s}}_i[n] = \begin{pmatrix} s_i[n] \\ \nabla s_i[n] \end{pmatrix}$ which doubles the size of the State vector. What we get in return is the evolution of each State impacted by its slope also. Each State, 'i', evolves as follows:

$s_i[n] = \alpha\ s_i[n-1] + \beta\ \nabla s_i[n-1] + \delta\ q_i[n]$ and $\nabla s_i[n] = \gamma\ \nabla s_i[n-1] + \varepsilon\ q_i[n]$ . . . (B)

Evolution in equation (B) is accomplished by structuring the State Transition matrix, **A**, and Noise matrix, **D**, in special ways as shown below.

$\underline{\mathbf{A}} = \begin{bmatrix} \alpha & \beta \\ 0 & \gamma \end{bmatrix}$ and $\underline{\mathbf{D}} = \begin{bmatrix} \delta & 0 \\ 0 & \epsilon \end{bmatrix}$. The elements of **A** and **D** are called "hyper-parameters". They can be pre-selected or estimated optimally. We pre-select them and hence **A** & **D** are not time-varying matrices themselves but they allow the States, **s**, to evolve in desirable ways.

With this definition of the augmented States, the Observation matix, **H**[n] is chosen as follows.
$\underline{\mathbf{H}}[n] = [x_1[n]\ 0 \ldots x_m[n]\ 0 \ldots x_M[n]\ 0\ \hat{d}[n-1]\ 0]$ with a zero between each element of input and delayed output.

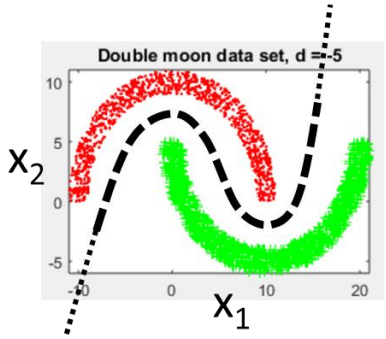
Kalman Predictor, Filter and Smoother algorithms specified in SYSTEMS Analytics book use H(n) and d(n) to perform Offline and In-Stream classification tasks. Since Kalman algorithms were derived and explained in detail in the book, we will not repeat the steps here.

In essence, the Multi-Input Single-Output (MISO) Classifier we have implemented is –

$\mathbf{d_{target}[n]} = \mathbf{a[n]}\ \mathbf{F^1\{d_{target}[n-1]\}} + \mathbf{c^T[n]}\ \mathbf{F^2\{u[n-1] + \gamma[n]\}}$ where $\mathbf{F^1\{.\}}$ and $\mathbf{F^2\{.\}}$ are non-linear functions and $\underline{\mathbf{u}}[n] = [u_1[n]\ u_2[n] \ldots u_P[n]]^T$. . . . (C)

Test Results:
In this section, we test all aspects of RTK-Kalman solution by simulating non-linearity and time-variability so that we can assess the performance.

We make use of the prototypical nonlinear "Double Moon" classification problem (Haykin, 2006) with the following choices.



Double Moon dataset:

- RED is Class 1 and GREEN is Class 2.
- As the BLACK curve indicates, the separating surface will have to be highly nonlinear.
- Inputs are $\{x_1, x_2\}$, the horizontal and vertical coordinates of each of the points on the Half Moons. Desired response, {d}, are the corresponding Class memberships, {RED, GREEN}. There are 2000 data points in this plot.
- The dataset is presented to the RKT-Kalman filter in a random order. Here, we drive the randomization based on a simple Markov Chain which adds temporal dynamics. There is one probability for the next sample to stay on the same Half Moon and another probability for the jump to the other Half Moon.

Challenge: ***Highly non-linear class separating surface and data stream that is time-varying!***

Offline:

- At counter = n, both $\{x_i[n], d[n]\}$ are available.
- Output of interest is the *smoothed* Kalman output, $y_S[n]$.
- We will also inspect the *smoothed* States, $\underline{\mathbf{s}}_S[n]$, which are the Conditional Expectations, $E[\underline{\mathbf{s}} \mid x, d]$.
- $y_S[n]$ is obtained from the Conditional Expectation of States via $y_S[n] = \underline{\mathbf{H}}[n]\, \underline{\mathbf{s}}_S[n]$ where $\underline{\mathbf{H}}[n]$ are known and hence non-random quantities.
  $\therefore$ $y_S = E[y \mid x]$, the Conditional Expectation.

In-Stream:

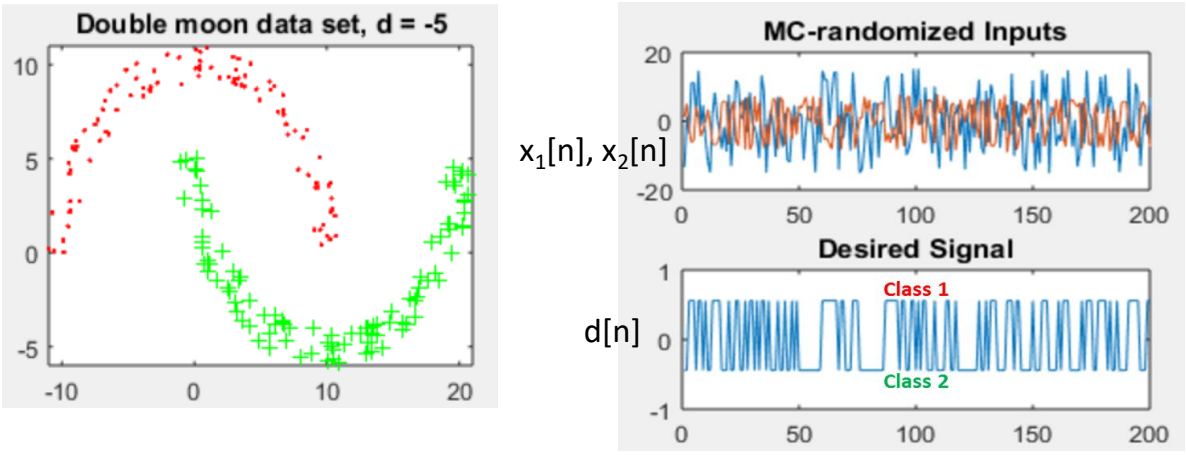
- At the start of counter = n, $\{x_i[n], d[n-1]\}$ are available.
- Output of interest is the *predicted* Kalman output, $y_P[n]$.
- The *predicted* States, $\underline{\mathbf{s}}_P[n]$, are the Conditional Expectations, $E[\underline{\mathbf{s}} \mid x, d]$.
- $y_P[n] = \underline{\mathbf{H}}[n]\, \underline{\mathbf{s}}_P[n]$.
- Once d[n] arrives, Kalman Filter updates $y_F[n]$ for the next recursion.

In the following experiments, M=40, $\rho(.) = \tanh(.)$ & hyper-parameters =

	α	β	δ	γ	ε
SRW Model:	0.3	1	0	1	1

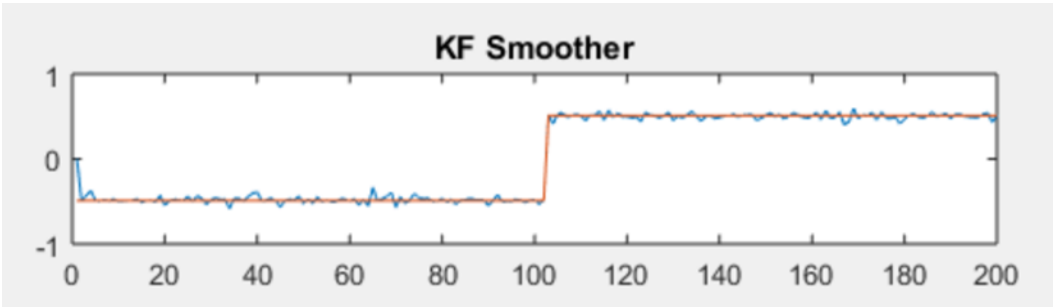
Offline Experiment Results:

- $\{x_i, d_i\}$; i=1 to 200 for this experiment.
- Supervised Learning is performed using Kalman Smoother.

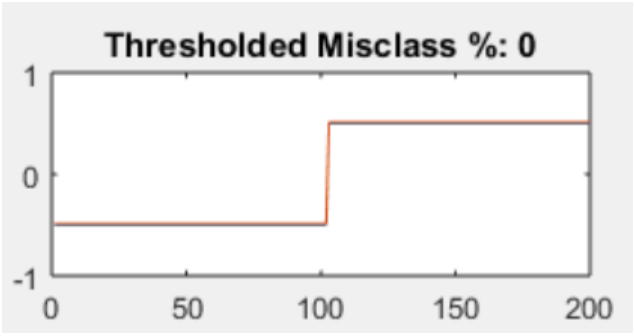


Double Moon data is randomized using Markov Chain. On the right, $\{x_i, d_i\}$, i=1 to 200, are shown on top and bottom, respectively.

Output below is the Smoother output, $y_S[n]$.



NOTE that the output of RKT-Kalman is generated in random order which has been re-ordered in the figure above. For classification purposes, threshold is selected as "0" and the continuous output is discretized into Class 1 and Class 2 (-0.5 & +0.5, respectively) in the figure below.



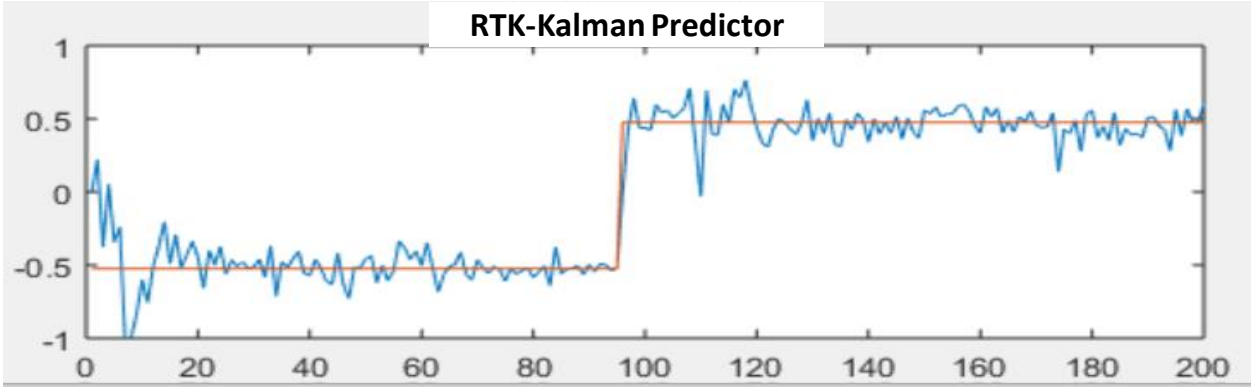
RTK-Kalman Smoother output above is of remarkably low variance! This is demonstrated by the ***ZERO misclassification Error.***

Similar results were seen on multiple reruns. The results are more stable when a run is longer than just 200 data points.

In-Stream Experiment Results:

- At Counter=n, assume that Inputs, $\{x_i[n]\}$ have arrived but d(n) is yet to arrive.
- Past d(n)'s are available.

During the In-Stream phase, use the predictor output, $y_P[n]$.

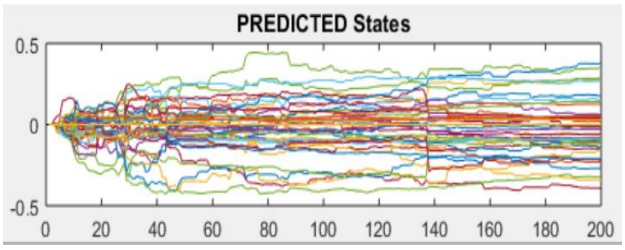


As one would expect, predictor shows convergence behavior noticeable near 0. In general, we expect the In-Stream Predictor solution to perform worse than Offline Smoother solution since the Smoother uses more data (past and future).

As expected, misclassification is larger (2%) now as we are working with predicted quantities. ***BUT 2% error for In-Stream processing for such a highly non-linear classification problem is remarkable!***

The plot of the State vector below shows the States are relatively stable.

In certain applications, the nature of the State trajectories can be used to corroborate the classification results and thus reduce False Positives and Negatives.

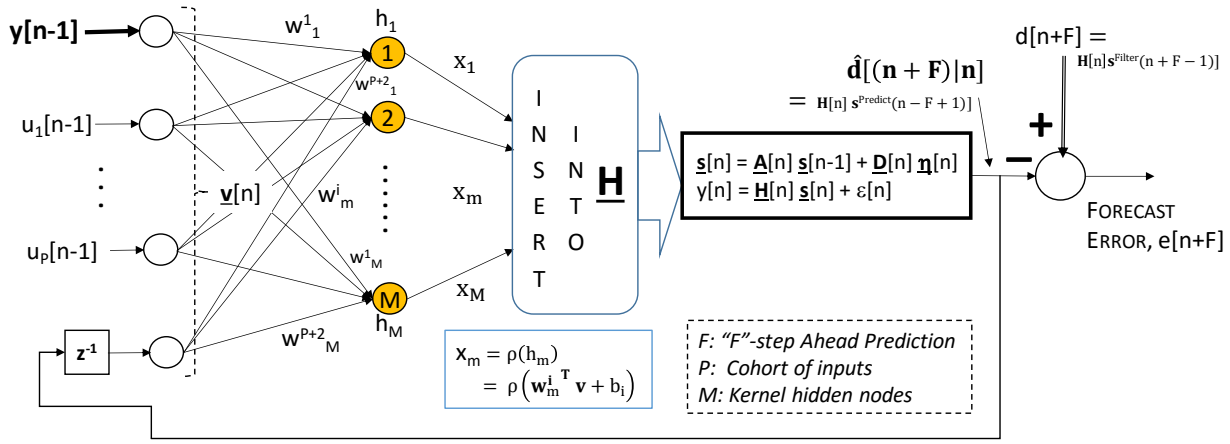


In this experiment, we did not consider the case where d[n]'s were not available and hence had to resort to K-step Ahead forecasting. This experiment will be performed in the next section.

As we see, RTK-Kalman is a complete solution for In-Stream processing: we start off with stored data and Smoothed estimates of the states. As each new data point arrives, Kalman Filter is updated and classification is PREDICTED. Recursively, the weights are updated at the end of the period, n. Classification accuracy is excellent.

RKT-Kalman Dynamical ML Forecasting

K-step Ahead forecasting of time-varying dynamical (or non-stationary) data is a hard practical problem. We take on such a real-data problem using RKT-Kalman in this section.



The structure of RKT-Kalman filter is very similar to the one for classification in the last section. Input, **v**[n], is a vector time series. y[n] is our "target" times series that we want to forecast F-steps Ahead; **u**[n] is a collection of "cohort" series that are related to our target, y[n].

A concrete example: Target time series is GDP and cohorts are housing starts, money supply, debt and other financial data, all measured at the same time intervals. Another example is a specific stock price returns as the target and other related stocks, stock index closing values, etc., as the cohorts.

The details of the Kernel-projection and Kalman filter are identical to the last section. As in the last section, the overall **Multi-Input Single-Output (MISO) model we have implemented is –**

$$\mathbf{d_{target}[n]} = \mathbf{a[n]\,F^1\{d_{target}[n-1]\}} + \mathbf{c^T[n]\,F^2\{u[n-1]} + \boldsymbol{\gamma}\mathbf{[n]\}}$$

Forecasting multi-step ahead is particularly daunting since Forecast Error, e[n+f] is NOT available at 'n'! Therefore, we proceed as follows.

Forecasting at instant = n:

- After d[n] is in hand at 'n', obtain Kalman "**FILTER**" States, $s^{Filter}[n]$.
- ***Approximate*** d[n+f], f=1 to F, as follows:
  Future desired response,
  $$d[(n+f)\,|\,n] = \mathbf{H}[n]\,\mathbf{s}^{\mathbf{Filter}}(n+f-1)] \text{ for } f = 1 \text{ to } F.$$
  Kalman "Predictor" output or ***"f-Step Ahead Forecast",***
  $$\hat{d}[(n+f)\,|\,n] = \mathbf{H}[n]\,\mathbf{s}^{\mathbf{P}}\,(n-f+1)] \text{ for } f = 1 \text{ to } F.$$
  ∴ Forecast Error,
  $$e[n+f] = \mathbf{H}[n]\,\mathbf{s}^{\mathbf{Filter}}(n-f+1)] - \mathbf{H}[n]\,\mathbf{s}^{\mathbf{P}}\,(n-f+1)]$$

➔ At f = 1, 2, . . . , F, use e[n+f] to update Kalman Predictor and Filter.

Real Data Experiment:

We consider the volume of products that a store sells per SKU per week and try to forecast 3-weeks ahead how much of that SKU will be sold. This information is very valuable to the store owner who can then adjust the inventory orders to just meet the customer demand, thus minimizing excess inventory while eliminating "out-of-stock" problem which shoppers dislike.

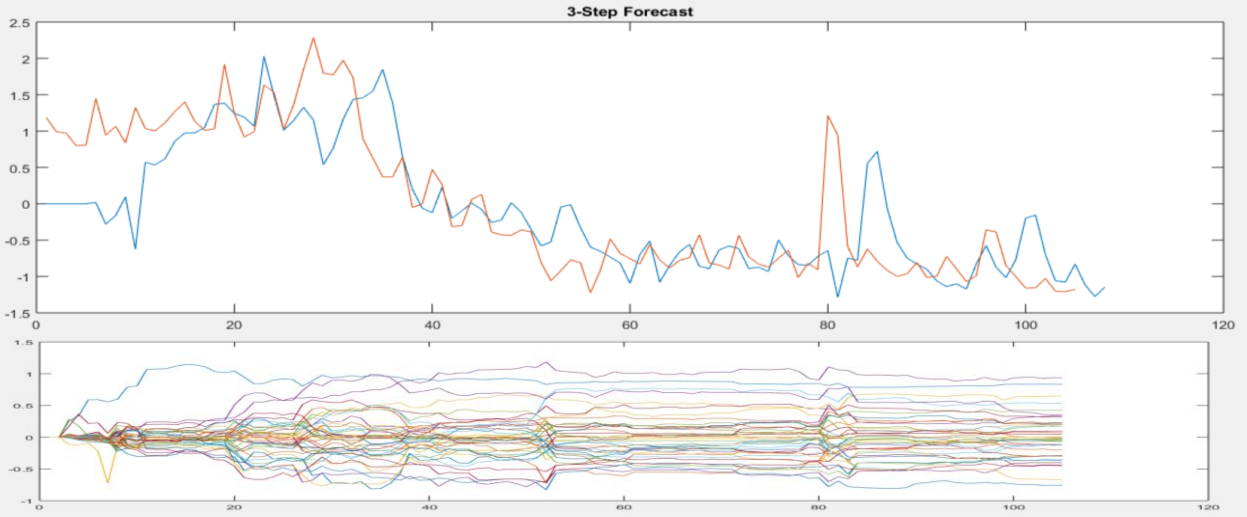
Similar to the classification experiment, we used the following RTK-Kalman parameters.
M=40, $\rho(.)$ = tanh(.) & hyper-parameters =

	α	β	δ	γ	ε
SRW Model:	0.3	1	0	1	1

No additional effort was spent in optimizing the parameters choices (which can yield significant improvements in the forecast performance).

Data for 15 SKUs for about 3 years were available. Of the 15 SKUs, one was picked as the target SKU (y[n]) and the others formed the cohort (hence, P=14).

In the figure below, RED is the actual target SKU demand per week and BLUE is the 3-week Ahead forecast.



- There is significant time-variability in the initial 50 or so weeks. Forecasts (BLUE) does an adequate job of following the true demand (RED) except in the 30 to 40-week range.
- After the 60th week, the series looks quite stationary and the forecasts look good except at Week 80. It is likely that the true demand (RED) at Week 80 is an outlier (or an error) since the amplitude jumps significantly; one can also notice a "disruption" in the trajectory of States in the bottom panel at Week-80.
- In repeated trials, RTK-Kalman 3-week Ahead Forecast is 50% to 60% accurate, i.e., the difference between the two curves, or forecast error, was about 40% of the signal energy.

Multi-step Ahead forecasting of short-length real data series with high amounts of time-variability is a hard problem as mentioned. From my past experience, I consider RTK-Kalman forecast result as extremely encouraging and will be valuable in the store product business case!

Conclusion

We set out to develop a solution for Dynamical, Non-linear, In-Stream Analytics and Machine Learning. The value of such a solution is significant because the same method can be used for classification and regression (including forecasting), offline and real-time applications and simple and hard ML problems.

We have achieved our objective in the form of State-space Recurrent Kernel-projection Time-varying Kalman or **"RKT-Kalman"** method. RKT-Kalman is a pragmatic combination of some well-known and less well-known rigorous theoretical solutions and some unique insights into how they can be combined effectively to solve practical problems.

RKT-Kalman utilizes non-linear projection and Cover Theorem, Kernel method, State-space data model, Bayesian Conditional Expectation estimation via Kalman Smoothing, Prediction, Filtering & Forecasting with State augmentation for time-varying estimation.

With a hard nonlinear classification problem and a hard real-life non-stationary data problem, we demonstrated the impressive performance of RTK-Kalman solutions. Since the tests performed address some of the extreme cases, it is fair to state that most other typical Machine Learning problems will be effectively solved using **Recurrent Kernel-projection Time-varying Kalman method**.

As we have shown, the practicality of RTK-Kalman Method in Offline and In-Stream mode means that learning can happen from *"past experience AND the results of new action*". Therefore, ML business solutions need not be "one and done" but can be deployed like flu shots (adjust the mix based on new learning and apply on a repeated basis), thus achieving the full promise of ***Dynamical Machine Learning*** for Data Science business applications.

PG Madhavan, Ph.D. - **“Data Science Player+Coach with deep & balanced** track record in Machine Learning **algorithms, products & business”**

https://www.linkedin.com/in/pgmad

END