Towards Learning Spatio-Temporal Data Stream Relationships for Failure Detection in Avionics

Sida Chen, Shigeru Imai, Wennan Zhu, and Carlos A. Varela

Rensselaer Polytechnic Institute, Troy, NY 12180, USA
{chens15,imais,zhuw5}@.rpi.edu, cvarela@cs.rpi.edu

Abstract. Spatio-temporal data streams are often related in non-trivial ways, for example, while the airspeed that an aircraft attains in cruise phase depends on the weight it carries, it also depends on many other factors, some of them controllable such as engine inputs or the airframe's angle of attack, while others contextual, such as air density, or turbulence. It is therefore a challenge to develop failure models that can help recognize errors in the data, such as an incorrect fuel quantity, a malfunctioning pitot-static system, or other abnormal flight conditions. In this paper, we extend our PILOTS programming language [1] to support machine learning techniques that will help data scientists: (1) create parameterized failure models from data and (2) continuously train a statistical model as new evidence (data) arrives. The linear regression approach learns parameters of a linear model to minimize least squares error for given training data. The Bayesian approach classifies operating modes according to supervised offline training and can discover new statistically significant modes online. Using synthetic data, we compare the accuracy, response time, and adaptability of these machine learning techniques. We expect these fundamental techniques to be the key building blocks to tackle more complex dynamic data-driven failure models, which will in turn enable us to perform more accurate flight planning in emergency conditions.

Keywords: data streaming, spatio-temporal data, declarative programming, linear regression, Bayesian classification and learning.

1 Introduction

Detecting and recognizing patterns from stream data generated by multiple aircraft sensors has become an important research area for flight safety. In the Air France flight 447 accident in 2009, iced pitot tubes caused an error in air speed data, and the pilots failed to react correctly, leading to crash [2]. While there have been advances in information fusion [3] and virtual modeling [4] for avionics control and user warnings, there is still a need for further research in methods that allow for fault detection and recovery techniques to be easily realized and implemented with minimal risk of software errors. Using redundant information provided by different sensors, this tragedy could have been avoided by Dynamic Data-Driven Avionics Systems (DDDAS) based on the concept of

Dynamic Data-Driven Application Systems [5]. DDDAS can expand the flight safety envelope of automation and support pilots with real-time decision making.

In some situations, detecting and recovering from sensor data errors is non-trivial, even for human experts and flight assistant systems. In the Tuninter 1153 flight accident in 2005, the fuel quantity indicator of a different aircraft model was installed, causing the instrument to display an incorrect amount of fuel, which led to fuel exhaustion of the aircraft [6]. This accident could have been avoided if the weight error could be detected by checking the relationship between lift and weight during flight cruise phase. Lift depends on airspeed, air density, wing surface area, and coefficient of lift. The coefficient of lift itself depends on the angle of attack and this relationship will change with different aircraft types. Understanding such complex relationships from multiple sensor data streams is critical to accurately detecting sensor faults. In this paper, we propose using machine learning techniques to estimate parameterized models of aircraft sensor data relationships, and statistically determine aircraft operating modes.

Using offline training parameters and known relationships among redundant stream data, in prior research, we have been able to detect and correct for sensor data errors using actual flight accident data [7, 8]. However, aircraft models might change due to significant aircraft emergencies, e.g. loss of a wing or loss of engines. To get more accurate results, an online system should be able to incrementally update pre-calculated models, and detect new modes that may not be in the offline training data set. The naïve Bayes classifier is a suitable method for offline training and incremental learning, but needs to be extended to detect previously unknown modes.

We are developing the ProgrammIng Language for spatiO-Temporal data Streaming applications (PILOTS) and its run-time system for fault detection and correction in data streams, which is especially important for flight safety. PILOTS has evolved gradually to date. We first designed the PILOTS programming language and proposed the concept of error signatures [9]. Next, we implemented a compiler and a runtime system for PILOTS [1] and then added error detection and correction support to the compiler and the runtime [10]. Finally, we applied PILOTS to data streams obtained from actual accidents, Air France 447 [2] and Tuninter 1153 [6], and confirmed the effectiveness of its error detection and correction capabilities [7, 8].

In this paper, we extend PILOTS to support machine learning techniques including linear regression for linear models, and Bayesian classification and learning for dynamic models. We use synthetic data streams to verify and compare these approaches. Using the X-Plane flight simulator [11], we generate flight sensor data to train the relationship between angle of attack and coefficient of lift during cruise phase. With the training results and the relationship between lift and weight during cruise phase, PILOTS is able to detect and correct for weight data errors using error functions and error signatures. We also implemented a dynamic Bayes classifier for offline training and incremental online learning of

different modes, which also detects new modes as the stream data switches to an unknown pattern.

The rest of the paper is organized as follows. Section 2 describes error signature-based error detection and correction methods as well as the PILOTS programming language and the architecture of its runtime system. Section 3 discusses the design and implementation of the machine learning component in PILOTS. Section 4 describes an instantiation of the machine learning component to estimate parameters for a linear model using regression. Section 5 introduces the dynamic Bayesian classification and learning. Section 6 discusses the methods and results of the machine learning techniques using a case study of airplane weight error detection and correction. Section 7 describes related work. Finally we briefly describe future research directions and conclude the paper in Section 8.

2 Background

PILOTS[1] is a highly-declarative programming language that has been applied to both the Air France 447 [2] and the Tuninter 1153 [6] accidents data, showing that PILOTS was able to successfully detect the data errors in both cases, and correct the error in the case of Air France 447 [8].

2.1 Error Detection and Correction Methods

Error functions Error functions are used to detect possible faults among redundant input stream data. An error function should have the value zero if there is no error in the input data, when the whole system is working in the normal mode.

For example, in the cruise phases of a flight, the lift equals the weight of the airplane. The lift can also be calculated using other input data, including airframe's angle of attack, air density, temperature, pressure and air speed. In this case, an error function could simply be defined as:

$$e(lift, weight) = lift - weight \tag{1}$$

The lift in Equation 1 is calculated using other input data. In the normal cruise phase mode, the value of this equation should be zero. If there is an error in the weight indicator, and the input weight data is lower than the real weight, Equation 1 should be greater than zero. Similarly, if the input weight data is higher than the real weight, Equation 1 should be smaller than zero. Thus, the validity of the input data could be determined from the value of the error function.

The values of input data are assumed to be sampled periodically from corresponding spatio-temporal data streams. Thus, an error function e changes its value as time proceeds and can be represented as $e(t)$.

[1] All sample programs in this paper use v.0.3.2. PILOTS v.0.3.2 is available at http://wcl.cs.rpi.edu/pilots

Error signatures An *error signature* is a constrained mathematical function pattern that is used to capture the characteristics of an error function $e(t)$. Using a vector of constants $\bar{K} = \langle k_1, \ldots, k_m \rangle$, a function $f(t, \bar{K})$, and a set of constraint predicates $\bar{P} = \{p_1(\bar{K}), \ldots, p_l(\bar{K})\}$, the error signature $S(\bar{K}, f(t, \bar{K}), \bar{P}(\bar{K}))$ is defined as follows:

$$S(f(t, \bar{K}), \bar{P}(\bar{K})) \triangleq \{\ f \mid p_1(\bar{K}) \wedge \cdots \wedge p_l(\bar{K})\}. \tag{2}$$

Mode likelihood vectors Given a vector of error signatures $\langle S_0, \ldots, S_n \rangle$, we calculate $\delta_i(S_i, t)$, the distance between the measured error function $e(t)$ and each error signature S_i by:

$$\delta_i(S_i, t) = \min_{g(t) \in S_i} \int_{t-\omega}^{t} |e(t) - g(t)| dt. \tag{3}$$

where ω is the window size. Note that our convention is to capture "normal" conditions as signature S_0. The smaller the distance δ_i, the closer the raw data is to the theoretical signature S_i. We define the *mode likelihood vector* as $L(t) = \langle l_0(t), l_1(t), \ldots, l_n(t) \rangle$ where each $l_i(t)$ is:

$$l_i(t) = \begin{cases} 1, & \text{if } \delta_i(t) = 0 \\ \frac{\min\{\delta_0(t), \ldots, \delta_n(t)\}}{\delta_i(t)}, & \text{otherwise.} \end{cases} \tag{4}$$

Mode estimation Using the mode likelihood vector, the final mode output is estimated as follows. Observe that for each $l_i \in L, 0 < l_i \leqslant 1$ where l_i represents the ratio of the likelihood of signature S_i being matched with respect to the likelihood of the best signature.

Because of the way $L(t)$ is created, the largest element l_j will always be equal to 1. Given a threshold $\tau \in (0, 1)$, we check for one likely candidate l_j that is sufficiently more likely than its successor l_k by ensuring that $l_k \leqslant \tau$. Thus, we determine j to be the correct mode by choosing the most likely error signature S_j. If $j = 0$ then the system is in *normal mode*. If $l_k > \tau$, then regardless of the value of k, *unknown error* mode (-1) is assumed.

Error correction Whether or not a known error mode i is recoverable is problem dependent. If there is a mathematical relationship between an erroneous value and other independently measured values, the erroneous value can be replaced by a new value estimated from the other independently measured values.

2.2 Spatio-Temporal Data Stream Processing System

Figure 1 shows the architecture of the PILOTS runtime system, which implements the error detection and correction methods described in Section 2.1. It consists of three parts: the *Data Selection*, the *Error Analyzer*, and the *Application Model* modules. The Application Model obtains homogeneous data streams $(d'_1, d'_2, \ldots, d'_N)$ from the Data Selection module, and then it generates outputs $(o_1, o_2, \ldots, o_M)$ and data errors $(e_1, e_2, \ldots, e_L)$. The Data Selection module takes

heterogeneous incoming data streams $(d_1, d_2, \ldots, d_N)$ as inputs. Since this runtime is assumed to be working on moving objects, the Data Selection module is aware of the current location and time. Thus, it returns appropriate values to the Application Model by selecting or interpolating data in time and location, depending on the data selection method specified in the PILOTS program. The Error Analyzer collects the latest ω error values from the Application Model and keeps analyzing errors based on the error signatures. If it detects a recoverable error, then it replaces an erroneous input with the estimated one by applying a corresponding estimation equation. The Application Model computes the outputs based on the estimated inputs produced by the Error Analyzer.

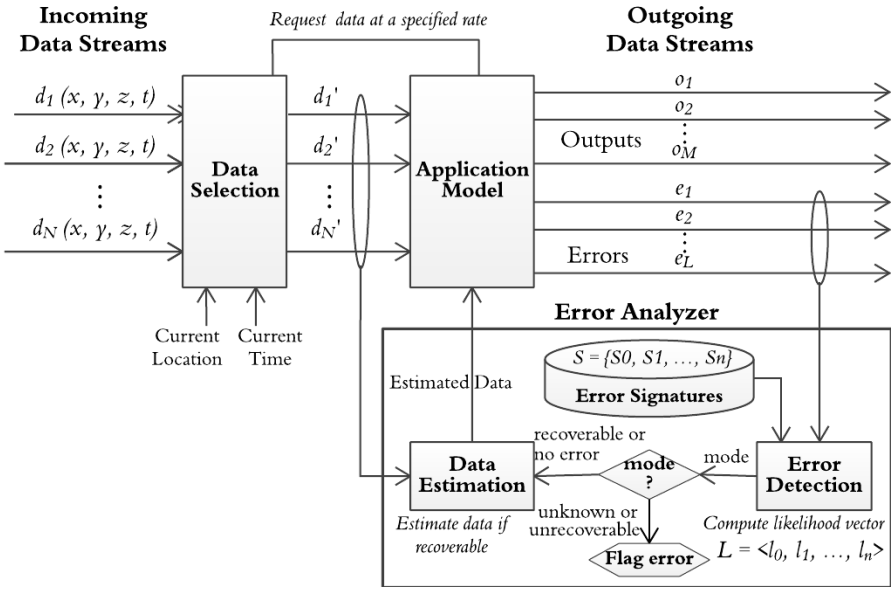


Fig. 1. Architecture of the PILOTS runtime system.

3 Design of Machine Learning Component

The PILOTS system can detect and correct for errors in data streams using models that define the relations between data streams, for example the relationship between wind speed, air speed, and ground speed. For non-trivial relations or relations with unknown parameters, we introduce prediction functionality in a new machine learning component for PILOTS.

3.1 Prediction in PILOTS Programming Language

To support prediction in the language, a new data selection method `predict` is defined in addition to `closest`, `euclidean`, and `interpolate`. Method `predict` (*model*, d'_{i_1},d'_{i_2},$\cdots$,d'_{i_n}) takes the identifier of the model as used for prediction as the first argument, *model*, and the data streams, $d'_{i_1}, d'_{i_2}, \cdots, d'_{i_n}$, used for input as the following arguments. `predict` method is implemented as an interface, accepting different machine learning models including online models, offline models, regressors, and classifiers. Figure 2 shows a simple example PILOTS program `PredictionTest`, where $a(t)$ and $b(t)$ are data streams retrieved by `closest` method meaning that the values of $a(t)$ and $b(t)$ with closest timestamp t to current time are chosen, and the predicted data stream $c(t)$ uses the prediction method with `linear_regression` as predictive model and $a(t)$, retrieved by `closest`, as input stream to the `linear_regression`. Assuming data streams $a(t)$ and $b(t)$ have a linear relationship, which is captured by the `linear_regression`, $c(t)$ is the prediction result of the `linear_regression` from $a(t)$. The `outputs` section compares $c(t)$ with $b(t)$ to produce pairwise difference between actual data (stream b) and output of the `linear_regression` (stream c).

```
program PredictionTest;
  inputs
    a,b (t) using closest (t);
    c (t) using predict(linear_regression, a);
  outputs
    difference: b - c at every 1 sec;
end
```

Fig. 2. A simple PILOTS program example outputting error.

3.2 Prediction in PILOTS Runtime

Figure 3 shows the updated PILOTS runtime system. To support the new prediction feature in PILOTS language syntax, Data Selection is altered to support communication to outside components through a socket. When Application Model requests p from Data Selection module, it first computes the input vector $\mathbf{x} = [d'_{i_1} d'_{i_2} \cdots d'_{i_n}]^T$ using data selection method defined for each d_i, and then sends *model* along with input vector $\mathbf{x}$ to the Learning Engine where the prediction is made by requested *model* using input vector $\mathbf{x}$. The online Learning Engine updates the dynamic online learning model for every prediction made and gives prediction result p back to the Data Selection component, which sends the prediction result to the Application Model as requested. The offline Learning Engine trains learning models using three major parts: (1) training definition including learning hypothesis, learning algorithm configuration, preprocessing methods and data file configuration; (2) learning algorithms such as

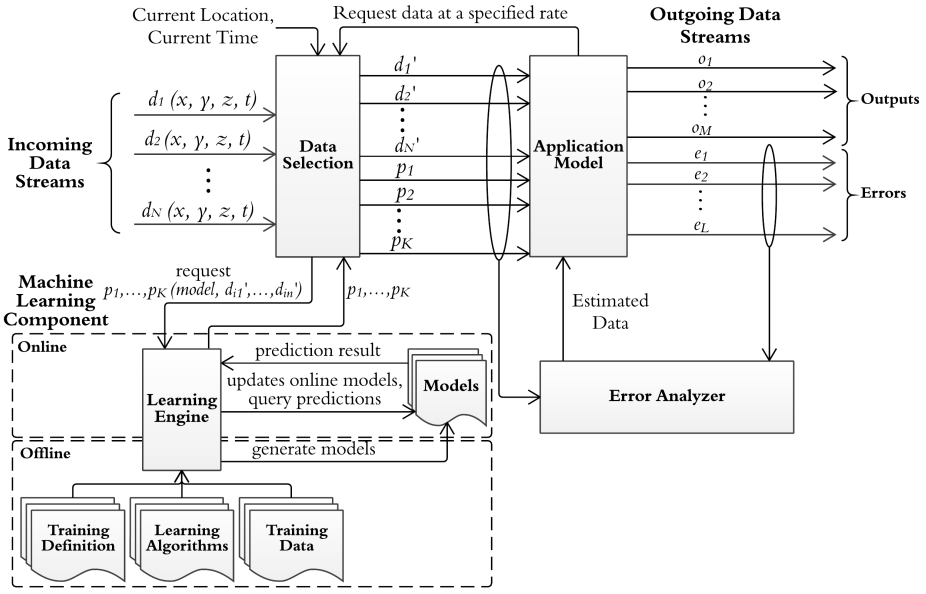


Fig. 3. The updated PILOTS runtime architecture with machine learning components.

least squares, Bayesian classifier or others; (3) training data, which refers to data stored in files. The other parts of the PILOTS runtime remain the same to maintain backward compatibility.

4 Data-driven Learning of Linear Models

Linear regression is a well-studied and powerful tool for estimating inter-variable relations in linear models. The equation for a linear regression model is

$$y = X\beta + \epsilon$$

where

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, X = \begin{pmatrix} x_{11} & \cdots & x_{1m} \\ x_{21} & \cdots & x_{2m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{pmatrix}, \beta = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{pmatrix}, \epsilon = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

y_i is dependent variable; x_{ij} is independent variable; β_i is regression coefficient; ϵ_i is error term.

4.1 Learning Algorithm

There are multiple methods to solve linear models. One of the learning algorithms implemented in our system is ordinary least squares, of which the target

is minimizing square of the Euclidean norm $||y - X\beta||^2$ by finding the best coefficient vector $\hat{\beta}$

$$\hat{\beta} = \arg\min_{\beta} ||y - X\beta||^2$$

Assuming the columns in X are linearly independent, we could retrieve $\hat{\beta}$ in closed form

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

4.2 Linear Model Accuracy

- **Coefficient of determination**: This metric is used to evaluate goodness of regression model fitting on training set $R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$ where y_i is measured/dependent variable, $\hat{y}_i$ is estimated variable, $\bar{y}$ is the average of all y_i.
- **Root Mean Squared Error**: This metric is used to evaluate the amount of error produced by prediction on average $RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{f}(x_i))^2}$ where $\hat{f}$ is an estimator, x_i is an independent variable vector $[x_{i1}\ x_{i2}\ \cdots\ x_{im}]^T$.

5 Statistical Learning of Dynamic Models

Naïve Bayes classifiers [12, 13] are commonly used in supervised training and classification. For continuous data, if the values of samples in each class are assumed to be normally distributed, the classifiers are called Gaussian naïve Bayes classifiers [14]. In the training phase, tagged samples of different classes are processed to train the parameters of the classifier. The parameters include the mean value, standard variance, and prior probability of each class. In the testing phase, the trained Bayes classifier decides the class of untagged input samples.

One limitation of the traditional naïve Bayes classifier is that the input samples in the testing phase will only be classified into classes that appeared in the training phase. If a sample of a previously unknown class appears, it will be classified into one of the known classes, even if the probability that it belongs to that class is very low. However, with dynamic stream data, new modes not in the training set could occur in some complex situations. For example, if a Bayes classifier is trained to recognize the "normal weight" and "underweight" modes of the weight indicator on an airplane during flights, and a previously unknown mode "overweight" appears in testing phase, the classifier will not be able to detect this new mode, but will classify the samples to "normal weight" or "underweight" based on the value and prior probability of the modes.

To tackle this limitation of the naïve Bayes classifier, we extend it into a dynamic Bayes classifier that has two phases: (1) *Offline*: Supervised learning, which is the same as Gaussian naïve Bayes classifiers. (2) *Online*: Unsupervised dynamic incremental learning, that classifies samples in known modes, updates

parameters of the model, and create new modes for samples in previously unknown modes. Because we focus on processing stream data during flights and deciding the normal or error operational modes of an airplane, the words "mode" and "class" are used interchangeably and have the same meaning in this paper.

5.1 Offline Supervised Learning

Gaussian Naïve Bayes Classifiers In a Gaussian naïve Bayes classifier [14], each input sample X is described by a feature vector $(x_1, \ldots, x_n)$, and each sample is classified into a target class $y \in \{y_1, \ldots, y_m\}$. In this paper, we consider samples of only one feature x, but the results can be generalized to n features. By Bayes' theorem, the conditional probability $P(y|x)$ is:

$$P(y|x) = \frac{P(y)P(x|y)}{P(x)} \tag{5}$$

As the samples in each feature are assumed to be normally distributed, $P(x|y)$ is calculated by:

$$P(x|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{-\frac{(x-\mu_y)^2}{2\sigma_y^2}} \tag{6}$$

Where μ_y is the mean of the values in x associated with class y, and σ_y is the standard deviation of the values in x associated with class y.

The corresponding classifier $\hat{y}$ is:

$$\hat{y} = \arg\max P(y|x) \tag{7}$$

Because $P(x)$ is the same for each class, $\hat{y}$ is:

$$\hat{y} = \arg\max P(y|x) = \arg\max \ P(y)P(x|y) \tag{8}$$

Offline Learning Phase In the offline supervised learning phase, input data tagged with mode labels are processed by a Gaussian naïve Bayes classifier. The mean value μ_y, standard deviation σ_y, and the prior probability $P(y)$ of each mode y, are calculated by the classifier, as in Figure 4.

5.2 Dynamic Online Unsupervised Learning

Major and Minor Modes To support dynamically changing modes during the online learning phase, the concepts of *major* and *minor* modes are introduced. The modes in the offline supervised learning phase are major modes. In the prediction and online learning phase, before a sample is processed by the Bayes classifier, the value is checked by a pre-processor to decide if it is in the range of each major mode. As 95% of the values in a normal distribution lie within $\mu \pm 2\sigma$, if the sample is not within that range of any major mode, a new minor mode

is created. As more data are processed, when the number of samples in a minor mode exceeds a threshold, it will be turned into a major mode. Minor modes are used to keep samples differentiated from known major modes. A threshold is used to diminish the impact of noise. Mode ID is automatically assigned when a new mode is detected.

Online Learning Phase The process of dynamic online unsupervised learning is shown in Figure 4. The parameters of initial major modes are from the training results of the offline training phase. As untagged samples are processed, if the value is within $\mu \pm 2\sigma$ of any major mode, the sample will be classified by naive Bayes classifier, and the parameters are incrementally updated. If the value is not within $\mu \pm 2\sigma$ of any major mode, but is within $\mu \pm 2\sigma$ of a minor mode, it will be classified into the closest minor mode, and the parameters of minor modes are updated accordingly. Finally, if the value of the sample is not within $\mu \pm 2\sigma$ of any major or minor mode, a new minor mode will be created for this sample. σ of the new minor mode is initially set as the average σ of the existing major modes. When the size of the minor mode is greater than a threshold, we start to calculate and use the real σ of the minor mode. The reason is that the σ might be bias if the number of samples is too small. Each time when the parameters of a minor mode are updated, if the number of samples exceeds a certain threshold, it will be upgraded into a major mode.

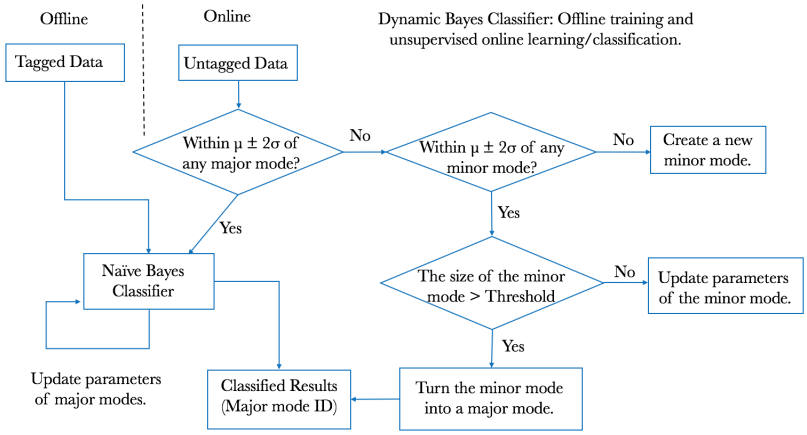


Fig. 4. Online classification and incremental learning using dynamic Bayes classifier.

6 Case Study: Airplane Weight Estimation

To help prevent accidents caused by fuel quantity indicator errors such as the Tuninter 1153 flight accident, we use the X-Plane flight simulator to generate flight sensors data and simulate airplane weight error scenarios. With the synthetic data, machine learning techniques are applied for inferring airplane model parameters from data, airplane weight error detection and actual weight estimation.

6.1 Experimental Settings

Data Generation X-Plane 9.7 is used to generate flying data of ATR72-500 in different altitudes, gross weights and power settings. The data is split by selecting 3 flights' 25 cruise phases, 1251 minutes in total, as training set, and 1 20-minutes flight with 4 cruise phases as testing set. The model is trained by 25 cruise phases in the training set and tested by 4 cruise phases in the testing set. To evaluate the PILOTS error detection accuracy, the whole testing set is modified to introduce artificial measurement errors as follows: weight data in the range from 1 to 100 and 750 to 800 seconds is multiplied by 0.9, from 1025 to 1099 seconds is multiplied by 1.1, from 390 to 490 seconds is multiplied by 1.05, from 570 to 648 seconds is multiplied by normal distribution of error with mean at 1 and standard deviation at 0.1, from 291 to 377 seconds are multiplied by uniform distribution of error ranging from 0.9 to 1.1. The cruise phases of testing set lie between 5 to 164, 230 to 395, 470 to 688 and 780 to 1108 seconds. We can visualize this data as "measured" in Figure10.

Implementation and Evaluation of Learning Algorithms For learning algorithms, the Sci-Kit package is used for the implementation of least squares algorithm and evaluation of the trained models.

6.2 Aerodynamic Model Parameter Estimation by Linear Regression

Synthetic data with simple relationships are used to verify the integration of machine learning approaches into PILOTS. In this example, simulated ATR-72 500 airplane data is used for PILOTS to detect and correct for weight error in the data streams. The relation between coefficient of lift and angle of attack is investigated and under certain assumptions about known variables, an estimation of weight from angle of attack, ambient temperature, ambient pressure and true air speed is made possible using linear regression by the PILOTS learning component.

Assumption To simulate and test linear regression implemented in PILOTS machine learning component, we assume certain known variables. It is required that the following variables are correctly measured and known: gross weight W,

ambient pressure p, true airspeed v, wing surface area S, special gas constant for dry air R', and ambient temperature T.

Linear Regression Model In cruise phase, when yaw, roll angles are close to zero and pitch is small, we assume $L = W$, in which L is total lift and W is gross weight. Based on the assumption, we can estimate W by the lift equation:

$$W = L = \frac{1}{2}v^2 S\rho C_l, \tag{9}$$

where ρ is air density and C_l is coefficient of lift. From ideal gas law, we know $\rho = \frac{p}{R'T}$ and replace ρ with $\frac{p}{R'T}$ in Equation 9 to get:

$$W = \frac{pv^2 SC_l}{2R'T} \tag{10}$$

and by transforming Equation 10, C_l, coefficient of lift could be represented by:

$$C_l = \frac{2WR'T}{pv^2 S}. \tag{11}$$

Generally C_l depends on the shape of airfoil and the shape of an aircraft. To roughly estimate C_l, the complex physical model is simplified using Thin-Airfoil theory , which predicts a linear relationship [15] between coefficient of lift, C_l, and the angle of attack, α, for low values of α, as shown in Figure 5 between dashed vertical lines. This relationship can be expressed as:

$$C_l = \beta_1 \alpha + \beta_2 + \epsilon \tag{12}$$

where ϵ is noise and α is known while β_1 and β_2 are distinct values for different aircrafts. A linear model could be formulated as the following:

$$y = X\beta + \epsilon \tag{13}$$

$$y = \begin{pmatrix} C_{l_1} \\ C_{l_2} \\ \vdots \\ C_{l_n} \end{pmatrix}, C_{l_i} = \frac{2W_i R'T_i}{p_i v_i^2 S}, X = \begin{pmatrix} \alpha_1 & 1 \\ \alpha_2 & 1 \\ \vdots & \\ \alpha_n & 1 \end{pmatrix}, \beta = \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix}$$

Because each column in X is independent, we could use the least squares method defined in Section 4.1 to retrieve $\hat{\beta}$, and predict $\hat{W}$ using the following equation:

$$\hat{W} = \frac{pv^2 S(\beta_1 \alpha + \beta_2)}{2R'T} \tag{14}$$

where we have substituted the linear estimation of C_l, in Equation 10.

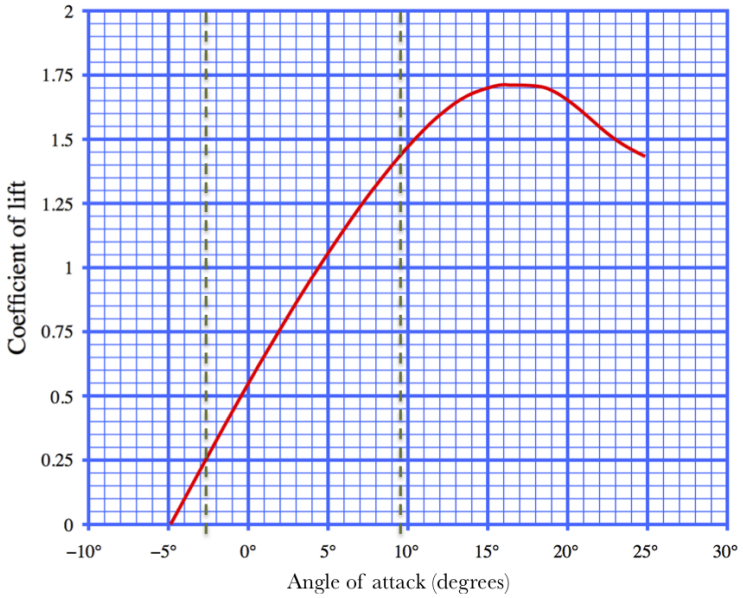


Fig. 5. Typical graph of section coefficient of lift versus angle of attack for a cambered airfoil, adapted from https://en.wikipedia.org/wiki/Lift_coefficient.

Mode	Error Signature	
	Function	Constraints
Normal	$e = k$	$-0.035 < k < 0.035$
Overweight	$e = k$	$k > 0.035$
Underweight	$e = k$	$k < -0.035$

Table 1. Vector of error signatures for weight correction.

6.3 Error Detection and Correction Using Error Signatures

PILOTS Program The linear regression model is trained with synthetic data using training parameters as shown in Figure 7. `data` defines the training file including file type and schema similar to Figure 8 as an example, and constants used in features and labels; `preprocessing` defines the preprocessing methods used on the training set; `model` contains functions for features, labels and training algorithms. The error function e is given by Equation 15 as the percentage of discrepancy between predicted weight $\hat{W}$ and measured weight W. The vector of error signatures uses a threshold of 3.5% because this number is more rigorous than the percentage of discrepancy between error weight and actual weight in Tuninter 1153 accident, which is about 10%. A PILOTS program named `WeightCorrection` implementing the vector of error signatures in Table 1 is shown in Figure 6 . If the error signature s_1 or s_2 is detected, the program estimates weight using Equation 14. The data selection module computes v', a',

p', te', w' using data points with the closest time stamp, and uses a' as an input matrix to predict cl' using model the with id `linear_regression`.

$$e = \frac{W - \hat{W}}{W} \tag{15}$$

```
program WeightCorrection;
  /* v = true air speed (m/s), a = angle of attack (Radian) */
  /* p = pressure (Pa), te = temperature (K), w = gross weight (N) */
  /* cl = coefficient of lift, R = 286.9 J/(kg K), S = 61(m^2)*/
  inputs
      v, a, p, te, w (t) using closest(t);
      cl (t) using predict(linear_regression, a);
  outputs
      corrected_weight: w at every 1 sec;
  errors
      e: (w - p*(v*v)*61*cl/(2*286.9*te))/w;
  signatures
      s0(K): e = K, -0.035 < K < 0.035          ''Normal'';
      s1(K): e = K, K > 0.035                  ''Overweight''
          estimate w = p*(v*v)*61*cl/(2*286.9*te);
      s2(K): e = K, K < - 0.035                ''Underweight''
          estimate w = p*(v*v)*61*cl/(2*286.9*te);
end
```

Fig. 6. A declarative specification of `WeightCorrection` PILOTS program using error signature.

Error Detection Evaluation Criteria: We evaluate the performance of error detection based on *accuracy* and *response time*, which are defined as follows:

- **Accuracy**: This metric is used to evaluate how accurately the algorithm determines the true mode. Assuming the true mode transition $m(t)$ is known for $t = 0, 1, 2, \ldots, T$, let $m'(t)$ for $t = 0, 1, 2, \ldots, T$ be the mode determined by the error detection algorithm. We define $accuracy(m, m') = \frac{1}{T}\sum_{t=0}^{T} p(t)$, where $p(t) = 1$ if $m(t) = m'(t)$ and $p(t) = 0$ otherwise.
- **Maximum/Minimum/Average Response Time**: This metric is used to evaluate how quickly the algorithm reacts to mode changes. Let a tuple (t_i, m_i) represent a mode change point, where the mode changes to m_i at time t_i. Let
$$M = \{(t_1, m_1), (t_2, m_2), \ldots, (t_N, m_N)\},$$
and
$$M' = \{(t'_1, m'_1), (t'_2, m'_2), \ldots, (t'_{N'}, m'_{N'})\},$$
where M and M' are the sets of true mode changes and detected mode changes respectively. For each $i = 1 \ldots N$, we can find the smallest t'_j such that $(t_i \leqslant t'_j) \wedge (m_i = m'_j)$; if not found, let t'_j be t_{i+1}. The response time r_i

```
{
  "data":{
    "file": ["training.csv"],
    "type": "csv",
    "header_type": "csvheader",
    "schema": "schema.json",
    "constants": {"S": 61.0, "R": 286.9}
  },
  "preprocessing":{
    "unit_transformation": {"v":"m/s", "p":"pascal","t":"kelvin","w":"newton","a
        ":"radian"}
  },
  "model":{
    "features": ["{a}"],
    "labels": ["2*{w}/({v}**2*({p}/{R}/{t})*{S})"],
    "algorithm":{
    "id": "linear_regression",
    "param": {},
    "save_file": "regression.estimator"
    }
  }
}
```

Fig. 7. Offline training parameters for the linear regression model.

```
{
  "names": ["v","p","t","w","a"],
  "units": ["knot","in_Hg","celsius","force_pound","degree"]
}
```

Fig. 8. Example data schema file.

for the true mode m_i is given by $t'_j - t_i$. We define the maximum, minimum, and average response time by $\max_{1 \leqslant i \leqslant N} r_i$, $\min_{1 \leqslant i \leqslant N} r_i$, and $\frac{1}{N} \sum_{i=1}^{N} r_i$ respectively.

Software Parameter Settings See Section 6.1 for data generation. PILOTS program `WeightCorrection` in Figure 6 is executed with different combinations of window sizes $\omega \in \{1, 2, 4, 8, 16\}$ and thresholds $\tau \in \{0.2, 0.4, 0.6, 0.8, 0.99\}$ to investigate the accuracy and average response time.

Results Figure 9 shows the training result of linear relationship between angle of attack and coefficient of lift, where the learned parameters are $\beta_1 = 6.3888$ and $\beta_2 = 0.3589$. The evaluation of the trained model gives $R^2 = 0.9949, RMSE = 0.00794$, showing a strong linear relationship with low in-sample error. Using Equation 9, we compute the training error between measured weight and estimated weight, resulting in $RMSE = 2687N$. Figure 10 shows the estimated weight and measured weight during the 18 minutes flight where $\omega = 1$ and

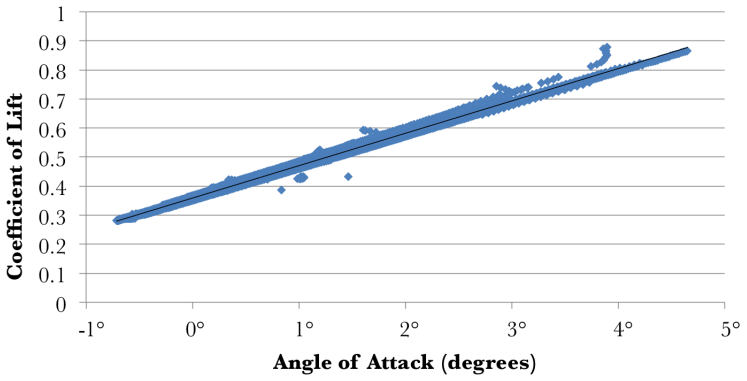


Fig. 9. The linear relation between angle of attack and coefficient of lift in cruise phase of training set.

$\tau = 0.99$, the best combination among all combinations in accuracy and response time. The PILOTS program can successfully detect and correct underweight and overweight conditions in cruise phases, with root mean squared error close to 1617N on average. The program performs the best in system failure simulation regions where the weight drifts by 10% or 5%, and performs well in random error simulation regions. The overall accuracy is 97.6% and the minimum response time is 0 seconds; maximum response time is 84 seconds and the average response time is 1.45 seconds. Outside cruise phase, the program does not estimate weight properly as the assumption $L = W$ does not hold.

6.4 Error Detection Using the Dynamic Bayes Classifier

PILOTS Program We use $\hat{W} - W$ as the feature for the dynamic Bayes classifier. Estimated weight is calculated by Equation 9 using the method described in Section 6.2. The dynamic Bayes classifier is trained with both "normal" and "underweight" tagged data in the offline learning phase. Figure 12 shows the parameters setting for the offline training phase. `data.file` is the input file for training. `data.constants` are parameters we used for features. `model.features` are features for Bayes classifier. In this example, the feature is the discrepancy between estimated weight $\hat{W}$ by Equation 14 and measured weight W. `model.algorithm.param` is the software parameters setting. A PILOTS program shown in Figure 11 named `WeightErrorMode` is used for the online learning and classification to detect different weight error modes.

Mode Prediction Evaluation We use the same evaluation criteria for major mode prediction: accuracy and response time as in Section 6.3.

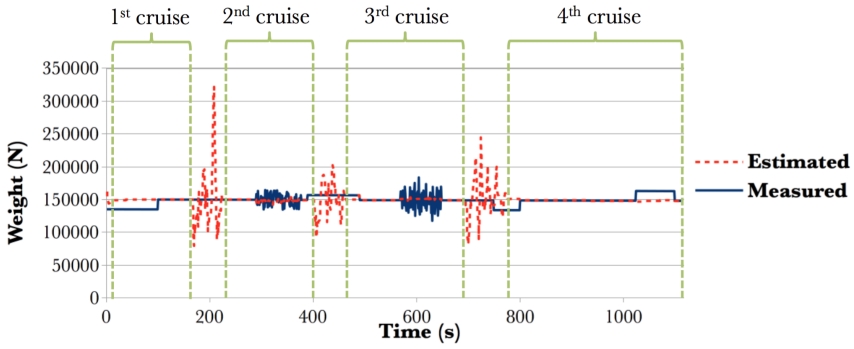


Fig. 10. Error detection and correction using $\omega = 1, \tau = 0.99$ for X-Plane simulated data.

```
program WeightErrorMode;
  /* v = true air speed (m/s), a = angle of attack (Radian) */
  /* p = pressure (Pa), te = temperature (K), w = gross weight (N) */
  inputs
    v, a, p, te, w (t) using closest(t);
    mode (t) using predict(bayes, v, a, p, te, w);
  outputs
    estimated_mode: mode at every 1 sec;
end
```

Fig. 11. A declarative specification of the `WeightErrorMode` PILOTS program using the dynamic Bayes classifier.

Experimental Settings See Section 6.1 for data generation. We use the same testing data, and 8000 seconds training data in cruise phase modified as follows: weight data in the range from 1526 to 3129 second are multiplied by 1.1 to simulate overweight mode. There are two major modes in the tagged training data: mode 0 for normal status and mode 1 for overweight status. For online learning, we set the threshold of the sample number to turn a minor mode into a major mode to 100. The sample number threshold for calculating σ of a new mode instead of using average σ is also set as 100. Figure 13 shows the feature and tagged mode of training data.

Results Figure 14 shows the results of weight error mode detection by dynamic Bayes classifier. Using the same testing data as in Figure 10, the dynamic Bayes classifier successfully detects three major modes in the cruise phases: mode 0 for normal status, mode 3 for underweight status, and mode 1 for overweight status. Mode 0 and mode 1 are major modes that appeared in the tagged training data, mode 3 is a new major mode detected by the classifier during the online incremental learning and prediction phase. There are also 22 minor modes generated

```
{
  "data":{
    "file": ["bayes_error_train.csv"],
    "type": "csv",
    "header_type": "csvheader",
    "schema":"bayes_schema.json",
    "constants": {"Beta_1": 6.38883559, "Beta_2": 0.35885757, "S": 61.0}
  },
  "preprocessing":{
    "unit_transformation": {"v":"m/s", "p":"pascal","t":"kelvin","w":"newton","a
        ":"radian"}
  },
  "model":{
    "features": ["({w}-0.5*({v}**2*({p}/286.9/{t})*{S})*({Beta_1}*{a}+{Beta_2}))
        "],
    "labels": ["{mode}"],
    "algorithm": {
      "id": "bayesonline",
      "param": {"sigma_scale": 2, "threshold": 100},
      "save_file": "bayes_online.estimator",
      "serialize_function": "to_json",
      "deserialize_function": "load_json"
    }
  }
}
```

Fig. 12. Offline training parameters for the dynamic Bayes classifier.

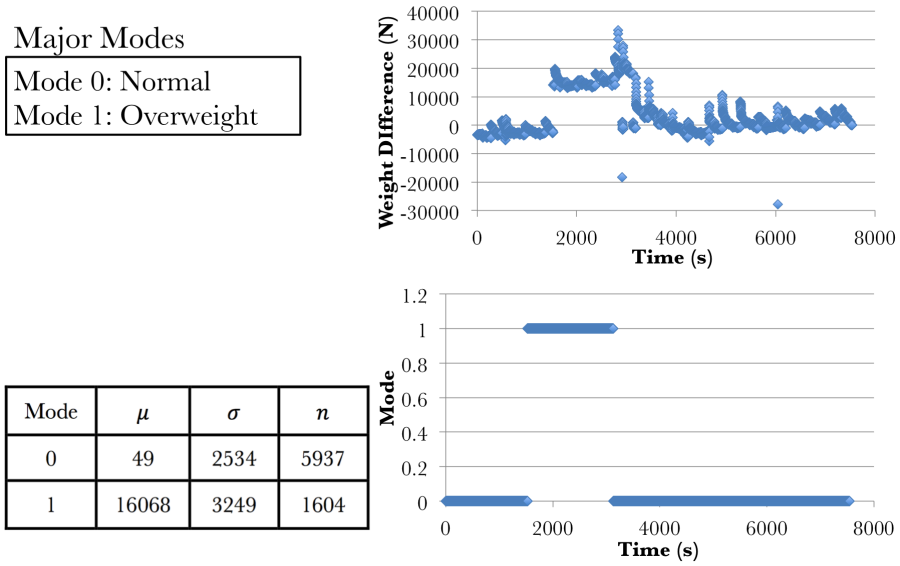
by the noise and non-cruise phase data in the testing set. The accuracy of major mode detection is 86.3% and the average response time is 3.43 seconds.

6.5 Comparison between Error Signatures and Dynamic Bayes Classifier

The average response time of the error signatures approach with 0.035 as threshold, $\omega = 1$, and $\tau = 0.99$, is 58% shorter than that of the dynamic Bayes classifier, and the error signatures approach is 11.3% more accurate than the dynamic Bayes classifier. However, the dynamic Bayes classifier discovers discrete error states dynamically and automatically while the error signatures approach is static, that is, every signature must be predefined manually.

7 Related Work

Stream data processing has been an important technique in flight safety systems. Fault detection, isolation, and reconfiguration (FDIR) has also been actively studied in the control community [16]. The FDIR systems evaluate a set of residuals (what we call error functions) to detect if a fault has occurred, then isolate the type of the fault, and reconfigure the system to recover from the fault. To alleviate the effect of noise on residuals, robust residual generation techniques, such as a Kalman Filter based approach [17], have been used. Error

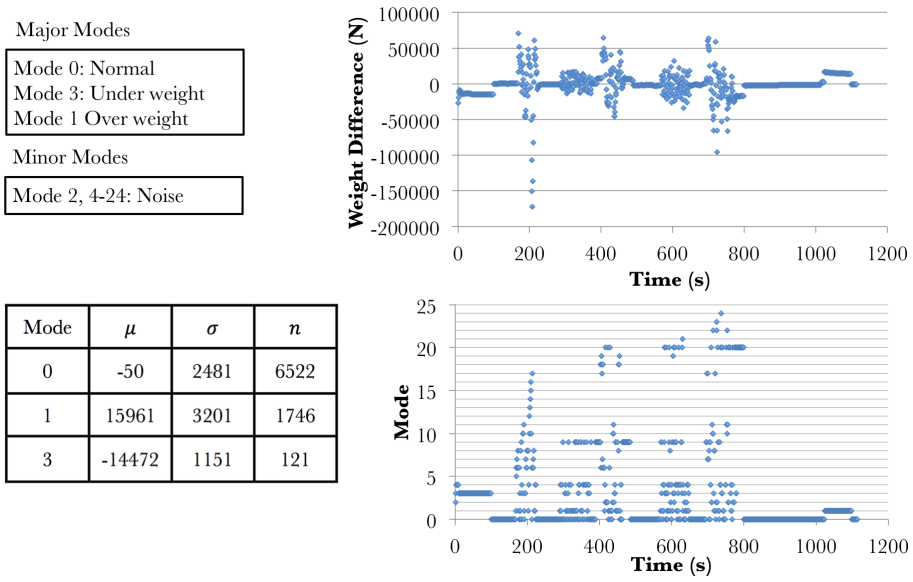


Mode	μ	σ	n
0	49	2534	5937
1	16068	3249	1604

Fig. 13. Weight error mode training data for the dynamic Bayes classifier.

residuals from different sources including ground speed, estimated wind speed, and propeller speed, are used to successfully detect and isolate airspeed sensor faults [18]. In some study, the residuals are not considered as binary value, but are assumed to have different distributions according to different modes [19]. The false positive and false negative detection rate of a FDIR method can be evaluated by several statistical models [20, 21]. The PILOTS language was designed for spatio-temporal data stream filtering, error detection and correction. PILOTS has been shown to detect and recover from sensor errors using actual flight data from commercial accidents [8]. The PILOTS framework enables users to implement fault detection and correction with tens of lines of code to describe error conditions.

There have been many systems that combine data stream processing and data base management, *i.e.*, Data Stream Management Systems (DSMS). PLACE [22] and Microsoft StreamInsight [23] are DSMS-based systems supporting spatio-temporal streams. Also, the concept of the moving object data base (MODB) which adds support for spatio-temporal data streaming to DSMS is discussed in [24]. Also, a DSMS-based traffic congestion estimation system has been proposed [25]. These DSMS-based spatio-temporal stream management systems support general continuous queries for multiple moving objects such as "Find all the cars running within a diameter of X from a point Y in the past Z time". Unlike these DSMS-based systems which handle multiple spatio-temporal objects, a PILOTS program is assumed to be moving and tries to extrapolate data that



Mode	μ	σ	n
0	-50	2481	6522
1	15961	3201	1746
3	-14472	1151	121

Fig. 14. Weight error mode detection using dynamic Bayes classifier.

is relevant to the current location and time. This approach narrows down the applicability of PILOTS; however, users can more easily design error signatures to estimate data on the fly thanks to the declarative programming approach.

In the context of big data processing, distributed, scalable, and fault-tolerant data streaming systems have been widely used. Such systems include MillWheel [26], Storm [27], and Spark Streaming [28]. Since these systems are expected to run over many computer nodes, they are designed to continue producing correct results with reasonably degraded performance even in the case of node failures. Unlike PILOTS, they are not aware of application-level data failures. On the other hand, PILOTS itself does not have any fault-tolerance mechanism to node failures.

Machine learning techniques have been widely used in stream data processing. There is a multi-dimensional regression method for time-series data streams [29], and a regression-based temporal pattern mining scheme for data streams [30]. Neural networks have been applied for supervised real-time learning and classification [31], and unsupervised active mining methods could be used to estimate the error of the model on new data streams [32]. In this paper, we extend PILOTS to support linear regression of stream data, and also combined offline supervised learning and dynamic online incremental learning as implemented by the dynamic Bayes classifier.

8 Discussion and Future Work

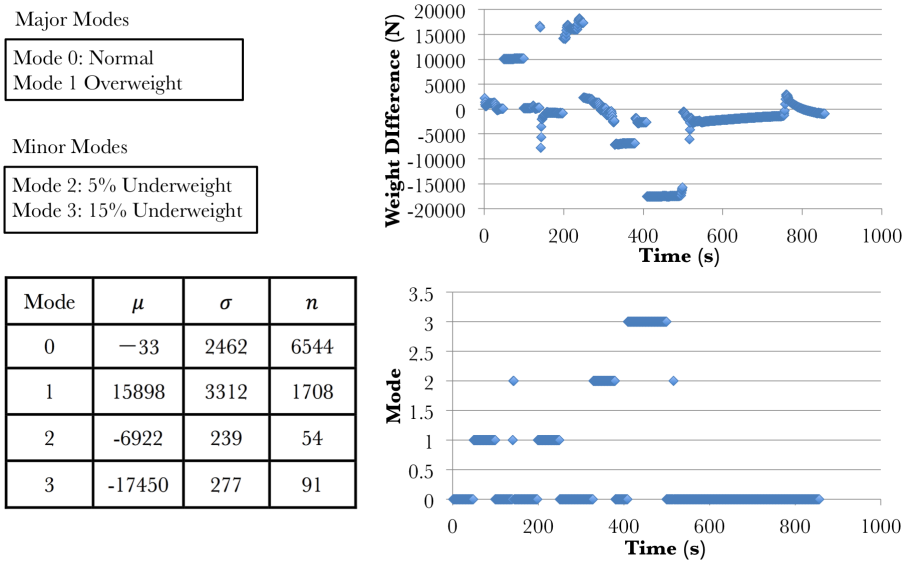
In this paper, we extend the PILOTS programming language to support machine learning techniques. A linear regression approach is applied to learn the relationship between coefficient of lift and the angle of attack during flights. With the training results, and Equation 9 to calculate airplane weight during cruise phase, the PILOTS program successfully detects and corrects underweight and overweight conditions in simulated flight data by using error signatures. In this case, we only consider possible weight errors, while other sensor data like airspeed needs additional signatures to ensure its correctness. Using dynamic Bayes classifier, when the system is trained by normal and underweight data, the PILOTS program is able to detect a new mode when an overweight situation occurs in the online learning phase. Error signatures and dynamic Bayes classifier both have their advantages and limitations. Error signatures detect and correct for data errors, while dynamic Bayes classifier only detects for data errors, but is not able to fix them. Dynamic Bayes classifier detects statistically significant new modes during the online learning phase, while error signatures can only detect pre-defined modes.

When using the dynamic Bayes classifier to detect weight error, we noticed that the system could not only detect "normal", "underweight", "overweight" modes, but also classifies "5% underweight" and "15% underweight" as two different modes, see Figure 15. This information is useful if different strategies need to be taken for different extent of weight errors, otherwise it would be unnecessarily misleading to classify them into different modes. Thus, the dynamic Bayes classifier should be adjusted to the requirements of various use cases. This would result in a semi-supervised online learning approach.

Future work includes exploring distributed computing for large scale data processing to get higher efficiency. For the dynamic Bayes classifier, it would be helpful to involve human feedback in the online learning phase, especially when a new mode is detected, to get more accurate classification parameters and decision making. Techniques are needed to add error correction to the dynamic Bayes classifier and learning. Take the weight error case for example, for any mode except the normal mode, simply use the estimated weight instead of detected weight as error correction. Machine learning techniques could also be used to learn parameters in error signatures from data. Another possible direction is to combine logic programming and probabilistic programming, as in ProbLog [33], to help analyze spatio-temporal stream data. Finally, uncertainty quantification [34] is an important future direction to associate confidence to data and error estimations in support of decision making.

9 Acknowledgement

This research is partially supported by the DDDAS program of the Air Force Office of Scientific Research, Grant No. FA9550-15-1-0214, NSF Grant No. 1462342, and a Yamada Corporation Fellowship.



Mode	μ	σ	n
0	−33	2462	6544
1	15898	3312	1708
2	-6922	239	54
3	-17450	277	91

Fig. 15. Another weight error mode detection using dynamic Bayes classifier.

References

1. S. Imai and C. A. Varela, "Programming spatio-temporal data streaming applications with high-level specifications," in *3rd ACM SIGSPATIAL International Workshop on Querying and Mining Uncertain Spatio-Temporal Data (QUeST) 2012*, (Redondo Beach, California, USA), November 2012.
2. Bureau d'Enquêtes et d'Analyses pour la Sécurité de l'Aviation Civile, "Final Report: On the accident on 1st June 2009 to the Airbus A330-203 registered F-GZCP operated by Air France flight AF 447 Rio de Janeiro - Paris." https://www.bea.aero/fileadmin/documents/docspa/2009/f-cp090601.en/pdf/f-cp090601.en.pdf. Accessed 09-15-2016.
3. E. P. Blasch, D. A. Lambert, P. Valin, M. M. Kokar, J. Llinas, S. Das, C. Chong, and E. Shahbazian, "High level information fusion (hlif): survey of models, issues, and grand challenges," *IEEE Aerospace and Electronic Systems Magazine*, vol. 27, no. 9, pp. 4–20, 2012.
4. J. T. Oden, E. E. Prudencio, and P. T. Bauman, "Virtual model validation of complex multiscale systems: Applications to nonlinear elastostatics," *Computer Methods in Applied Mechanics and Engineering*, vol. 266, pp. 162–184, 2013.
5. F. Darema, "Dynamic data driven applications systems: A new paradigm for application simulations and measurements," in *Computational Science-ICCS 2004*, pp. 662–669, Springer, 2004.
6. A. N. per la Sicurezza del Volo, "Final Report: Accident involving ATR 72 aircraft registration marks TS-LBB ditching off the coast of Capo Gallo (Palermo - Sicily), August 6th, 2005." Accessed 03-31-2015.

7. S. Imai, R. Klockowski, and C. A. Varela, "Self-healing spatio-temporal data streams using error signatures," in *2nd International Conference on Big Data Science and Engineering (BDSE 2013)*, (Sydney, Australia), December 2013.
8. S. Imai, A. Galli, and C. A. Varela, "Dynamic data-driven avionics systems: Inferring failure modes from data streams," in *Dynamic Data-Driven Application Systems (DDDAS 2015)*, (Reykjavik, Iceland), June 2015.
9. S. Imai and C. A. Varela, "A programming model for spatio-temporal data streaming applications," in *Dynamic Data-Driven Application Systems (DDDAS 2012)*, (Omaha, Nebraska), pp. 1139–1148, June 2012.
10. R. S. Klockowski, S. Imai, C. Rice, and C. A. Varela, "Autonomous data error detection and recovery in streaming applications," in *Proceedings of the International Conference on Computational Science (ICCS 2013). Dynamic Data-Driven Application Systems (DDDAS 2013) Workshop*, pp. 2036–2045, May 2013.
11. Laminar Research, "X-Plane." http://www.x-plane.com/. Accessed 09-15-2016.
12. I. Rish, "An empirical study of the naive Bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, pp. 41–46, IBM New York, 2001.
13. E. T. Jaynes, *Probability theory: The logic of science*. Cambridge university press, 2003.
14. G. H. John and P. Langley, "Estimating continuous distributions in Bayesian classifiers," in *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pp. 338–345, Morgan Kaufmann Publishers Inc., 1995.
15. J. D. Anderson Jr, *Fundamentals of aerodynamics*. Tata McGraw-Hill Education, 2010.
16. I. Hwang, S. Kim, Y. Kim, and C. E. Seah, "A survey of fault detection, isolation, and reconfiguration methods," *Control Systems Technology, IEEE Transactions on*, vol. 18, no. 3, pp. 636–653, 2010.
17. T. Menke and P. Maybeck, "Sensor/actuator failure detection in the Vista F-16 by multiple model adaptive estimation," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 31, pp. 1218–1229, oct 1995.
18. S. Hansen and M. Blanke, "Diagnosis of airspeed measurement faults for unmanned aerial vehicles," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 50, pp. 224–239, January 2014.
19. C. Svärd, M. Nyberg, E. Frisk, and M. Krysander, "Data-driven and adaptive statistical residual evaluation for fault detection with an automotive application," *Mechanical Systems and Signal Processing*, vol. 45, no. 1, pp. 170–192, 2014.
20. A. Zolghadri, "Advanced model-based FDIR techniques for aerospace systems: Today challenges and opportunities," *Progress in Aerospace Sciences*, vol. 53, pp. 18–29, August 2012.
21. J. Marzat, H. Piet-Lahanier, F. Damongeot, and E. Walter, "Model-based fault diagnosis for aerospace systems: a survey," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 226, pp. 1329–1360, January 2012.
22. M. F. Mokbel, X. Xiong, W. G. Aref, and M. a. Hammad, "Continuous query processing of spatio-temporal data streams in PLACE," *GeoInformatica*, vol. 9, pp. 343–365, 2005.
23. M. H. Ali, B. Chandramouli, B. S. Raman, and E. Katibah, "Spatio-temporal stream processing in Microsoft StreamInsight," *IEEE Data Eng. Bull.*, pp. 69–74, 2010.

24. K. An and J. Kim, "Moving objects management system supporting location data stream," in *Proceedings of the 4th WSEAS International Conference on Computational Intelligence, Man-Machine Systems and Cybernetics*, CIMMACS'05, (Stevens Point, Wisconsin, USA), pp. 99–104, World Scientific and Engineering Academy and Society (WSEAS), 2005.
25. S. Geisler, C. Quix, S. Schiffer, and M. Jarke, "An evaluation framework for traffic information systems based on data streams," *Transportation Research Part C: Emerging Technologies*, vol. 23, pp. 29–55, 2012.
26. T. Akidau, A. Balikov, K. Bekiroğlu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle, "Millwheel: fault-tolerant stream processing at internet scale," *Proceedings of the VLDB Endowment*, vol. 6, no. 11, pp. 1033–1044, 2013.
27. The Apache Software Foundation, "Apache Storm." http://storm.apache.org/, January 2015. Accessed 09-15-2016.
28. M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, "Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters," in *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Computing*, pp. 10–10, USENIX Association, 2012.
29. Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang, "Multi-dimensional regression analysis of time-series data streams," in *Proceedings of the 28th International Conference on Very Large Data Bases*, pp. 323–334, VLDB Endowment, 2002.
30. W.-G. Teng, M.-S. Chen, and P. S. Yu, "A regression-based temporal pattern mining scheme for data streams," in *Proceedings of the 29th International Conference on Very large data bases-Volume 29*, pp. 93–104, VLDB Endowment, 2003.
31. G. A. Carpenter, S. Grossberg, and J. H. Reynolds, "Artmap: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network," *Neural networks*, vol. 4, no. 5, pp. 565–588, 1991.
32. W. Fan, Y.-a. Huang, H. Wang, and S. Y. Philip, "Active mining of data streams.," in *Proceedings of the 2004 SIAM International Conference on Data Mining*, pp. 457–461, SIAM, 2004.
33. L. De Raedt, A. Kimmig, and H. Toivonen, "Problog: A probabilistic prolog and its application in link discovery.," in *International Joint Conference on Artificial Intelligence*, vol. 7, pp. 2462–2467, 2007.
34. D. Allaire, D. Kordonowy, M. Lecerf, L. Mainini, and K. Willcox, "Multifidelity DDDAS methods with application to a self-aware aerospace vehicle," in *DDDAS 2014 Workshop at ICCS'14*, pp. 1182–1192, June 2014.