Pattern and Knowledge Extraction using Process Data Analytics: A Tutorial

Yiting Tsai [*] **Qiugang Lu** [*] **Lee Rippon** [*] **Siang Lim** [*]
Aditya Tulsyan [**] **Bhushan Gopaluni** [*]

[*] *Department of Chemical and Biological Engineering, University of British Columbia, Vancouver, BC V6T 1Z3, Canada (yttsai@chbe.ubc.ca; qglu@chbe.ubc.ca; leeripp@chbe.ubc.ca; siang.lim@ubc.ca; bhushan.gopaluni@ubc.ca)*
[**] *Department of Chemical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, USA (e-mail: tulsyan@mit.edu)*

Abstract: Traditional techniques employed by control engineers require a significant update in order to handle the increasing complexity of modern processes. Conveniently, advances in statistical machine learning and distributed computation have led to an abundance of techniques suitable for advanced analysis. In this tutorial we introduce data analytics techniques and discuss their theory and application to chemical processes. Although the focus is more on theory, the applications will be explored more widely in a follow-up journal paper. The ultimate goal is to familiarize control engineers with how these techniques are used to extract valuable knowledge from raw data, which can then be utilized to make smarter process control decisions.

Keywords: Machine learning, Classification, Regression, Neural networks, Gaussian processes

1. INTRODUCTION

Process control in the chemical industry is undergoing a dramatic revolution, as the ability to extract knowledge from data is becoming a reality. Immense amounts of historical data are available from advanced sensing and actuation devices. Due to computational and theoretical limitations, past control strategies have been limited by the use of small and cleanly-structured data. By doing so, a significant amount of potentially valuable information is dismissed. However, the recent development of powerful computing tools have allowed machine learning (ML) algorithms to be fully utilized on *big data*. Engineers are thus equipped with the ability to extract knowledge from data and use it to improve control actions.

To optimally leverage both historical data and advanced sensor data, control engineers can employ a new set of intuitive and user-friendly tools. In this tutorial we introduce basic background knowledge behind ML and model validation, then explore the theory behind important classification and regression algorithms. Finally, the paper concludes after presenting techniques for dimensional reduction, advanced learning and handling of dynamic data.

2. BACKGROUND AND BASIC TERMINOLOGY

The nomenclature found in standard ML texts such as Bishop (2006) and Murphy (2012) represent the input data matrix as X and output data matrix as Y. Moreover, X contains N rows of samples, and d_x columns of variables. Input and output data are divided into **training**, **validation** and **testing** sections. For datasets with a few hundred or thousand samples, the training/validation/testing ratio ranges between 50/25/25 and 90/5/5. However, for datasets with more than a million entries, a ratio between 90/10/10 and 98/1/1 is preferred according to Murphy (2012) and Goodfellow et al. (2016). The purpose of each data division is as follows:

- **Training:** Mathematical mappings between the input and output data (known as *models*) are formed.
- **Validation:** For each model formed during training, the one (with the combination of parameters and hyperparameters) which results in the lowest error is selected, using k-fold cross-validation.
- **Testing:** The capability of the selected model to generalize to new samples is measured by evaluating its prediction accuracy.

Note that the test set cannot influence the choice of model structure, parameters or hyperparameters (Murphy, 2012). The training, validation and testing errors are evaluated as either

$$\text{Error Fraction} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}(\hat{y}^i \neq y^i) \text{ (Classification)}, \quad (1)$$

$$\text{Error Rate} = \frac{1}{n}\sum_{i=1}^{n} f(\hat{y}^i - y^i) \text{ (Regression)}, \quad (2)$$

where $\mathbb{1}$ represents the indicator function and $\hat{y}^i$ the estimated output for the i^{th} sample using the selected model. The error is calculated as a fraction of mismatched samples in the case of classification, or as an average sum of errors in the case of regression, where f is the case-

[⋆] This work was supported in part by the Institute for Computing, Information and Cognitive Systems (ICICS) at UBC.

specific error function (e.g., the mean squared error for least-squares).

The *bias-variance tradeoff* is important to consider when trying to train a high-quality predictive model. The goal of prediction is to provide consistent estimates that are close to the true values. A general model which *"under-fits"* the training data will estimate testing data with large bias but a small variance. Conversely, a complex model that *"over-fits"* the training data will capture excessive noise, produce estimates with small bias but high variance, and thus generalize poorly.

3. CLASSIFICATION ALGORITHMS

Classification aims to separate input data X into corresponding discrete classes $y^i \in [1, C]$ where C is the total number of classes. Some common classification algorithms are explored in what follows.

3.1 k-Nearest Neighbours (kNN)

The goal of kNN is to group similar data samples into clusters. The model is formed by loading input samples x^i that correspond to specific classes $y^i \in [1, C]$. The class prediction for a new input sample is the mode of classes of its nearest k-neighbours. The value of hyperparameter k is determined by cross-validation. The distance between samples x^i and x^j can be defined in several forms as shown in Table 3.1.

Table 1. Typical distance formulas used

Name	$D(x^i, x^j)$
Euclidean	$\|\|x^i - x^j\|\|_2$
Manhattan	$\|\|x^i - x^j\|\|_1$
Cosine Similarity	$\frac{x^{iT} x^j}{\|\|x^i\|\|_2 \cdot \|\|x^j\|\|_2}$
Jaccard Similarity	$\frac{\mathbf{1}(x^i=c \,\cap\, x^j=c)}{\mathbf{1}(x^i=c \,\cup\, x^j=c)}, c \in [1, \cdots, C]$

The test error of kNN converges to a minimum, irreducible error, as the total number of samples N and k both grow to infinity such that $\frac{k}{N} = 0$. This is a direct result of "Stone's Theorem" (Vapnik and Vapnik, 1998).

3.2 k-Means

The goal of k-Means is to group existing data into clusters. However, it does not predict the clusters of new samples, and hence it is an *unsupervised* algorithm. The hyperparameter k represents the number of mean nodes. The clusters learned in k-Means are labelled arbitrarily; this is known as the *label-switching* phenomenon (Celeux, 1998). The three key steps of k-Means are as follows:

(1) Initialize k random mean nodes spread across the span of the dataset.
(2) Assign each sample x^i to the mean closest to it, with respect to a distance metric.
(3) Update each mean node by averaging the coordinates of its samples.

This iterative procedure can be visually interpreted in Fig. 1. Usually, k-Means is performed using many different initializations to obtain a consistent set of clusters.

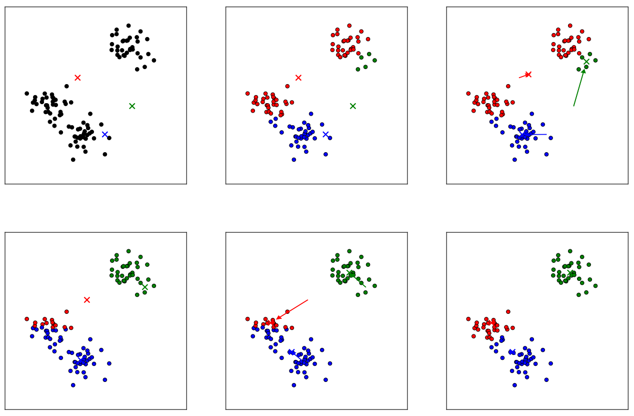

Fig. 1. Visualization of k-means, adapted from (Jain, 2010).

3.3 Support Vector Machines

The goal of SVM is use training data to form a model between samples and their corresponding classes, then predict new samples based on the model. The *softmax* function is used for multi-class data, as it calculates the probabilities of a sample belonging to each individual class. The softmax coefficients corresponding to each specific class are expressed as a vector w. If x^i belongs to class c or y_i, then its coefficients are denoted as w_c or w_{y^i}, respectively. The softmax probability is expressed as:

$$p(y^i|w, x^i) = \frac{exp(w_{y^i}^T x^i)}{\sum_{c=1}^{C} exp(w_c^T x^i)}. \tag{3}$$

The separating hyperplanes between the classes are defined as those that maximize their distance from the closest datapoints of their respective classes. Their exact mathematical expressions can be found in Cortes and Vapnik (1995).

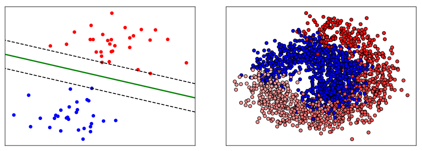

Fig. 2. A linearly-separable dataset (left), versus a non-linearly-separable dataset (right), adapted from (Lemm et al., 2011).

The optimal SVM model coefficients w are found by minimizing the following objective function using gradient descent:

$$f(w) = \sum_{i=1}^{n} \left[-w_{y^i}^T + ln\left(\sum_{c=1}^{C} exp\left(w_{y^i=c}^T x^i\right) \right) \right]. \tag{4}$$

To reduce overfitting, common regularizers such as L_1 $(\lambda||w||_1)$ or L_2 $(\frac{\lambda}{2}||w||_2^2)$ can be used.

4. REGRESSION ALGORITHMS

In regression, due to the continuous nature of outputs Y, a loss function L of differences between Y and $\hat{Y}$ is minimized instead. The specific loss function used depends on the type of regression.

4.1 Ordinary Least Squares

The name "least-squares" comes from the loss function being the absolute sum of squared differences between Y and $\hat{Y}$ as follows:

$$L = \sum_{i=1}^{N}(y^i - \hat{y}^i)^2 = \sum_{i=1}^{N}(y^i - w^T x^i)^2 = ||Y - XW||_2^2. \tag{5}$$

A column of ones is usually appended to the left of the X matrix to account for linear offset (from origin). The optimal weights W^* which minimizes the loss function can be computed by gradient descent or analytically as follows:

$$W^* = \underset{W}{argmin}||Y - XW||_2^2 = (X^TX)^{-1}X^TY. \tag{6}$$

L_1 or L_2 regularizers ($||W||_1$ or $||W||_2$, respectively) are used to both overfitting and also to deal with singular X^TX matrices. For example, in the case of L_2 regularization, the optimal weights become:

$$\underset{W}{argmin}\left[||Y - XW||_2^2 + \frac{\lambda}{2}||W||_2^2\right] = (X^TX + \lambda I_{d_x})^{-1}(X^TY). \tag{7}$$

4.2 Kernel Regression

Ordinary least squares is often performed on an extremely large number of non-linear features. A shortcut known as the *kernel* trick can be used to re-write the dot-product of the non-linear feature space $\Phi(X)$ as a function of the dot-product of original linear features in X:

$$\kappa(x^i, x^j) = \Phi^T(x^i)\Phi(x^j). \tag{8}$$

Table 2 shows several examples of feature-spaces $\Phi(X)$ and their equivalent kernel representations.

Table 2. Examples of Kernels

$\Phi(x^i)$	$\Phi^T(x^i)\Phi(x^j) = \kappa(x^i, x^j)$
$\left[(x_1^i)^2 \ \sqrt{2}x_1^i x_2^i \ \cdots \ (x_{d_x}^i)^2\right]^T$	$(x^{iT}x^j)^2$
$\left[1 \ 2x_1^i \ 2x_2^i \ (x_1^i)^2 \ \sqrt{2}x_1^i x_2^i \ \cdots \ (x_{d_x}^i)^2\right]^T$	$1 + 2x^{iT}x^j + (x^{iT}x^j)^2$
$\prod_{j=1}^{d_x} exp(-\frac{1}{2\sigma^2}\lvert\lvert x_j^i\rvert\rvert_2^2)\left[1 \ \frac{1}{\sqrt{2!}}(x_1^i)^2 \ \cdots \ 1 \ \frac{1}{\sqrt{2!}}(x_{d_x}^i)^2 \ \cdots\right]^T$	$exp(-\frac{\lvert\lvert x^i - x^j\rvert\rvert_2^2}{2\sigma^2})$

The Gaussian Radial Basis Function (RBF) kernel is an example of an infinite-dimensional kernel. The "validity" of a kernel can be confirmed by using Mercer's Theorem (Murphy, 2012). Given an unknown dataset with no prior knowledge, the traditional approach is to try typical kernels and use cross-validation to select the best one. A kernel representation of a high-dimensional, non-linear feature space can be visualized in Figure 3.

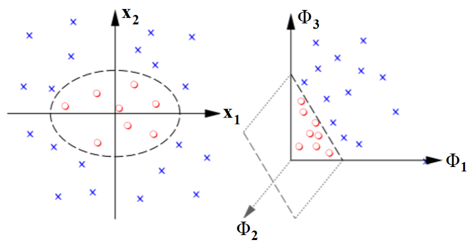

Fig. 3. For data X that is not linearly-separable (right), the use of a suitable kernel-space Φ will make the data linearly separable. Adapted from (Jakkula, 2006).

Once a suitable kernel has been selected, the following steps are required for the prediction of new samples:

(1) Form the gram matrix $K = \Phi\Phi^T$, with elements $\kappa(x^i, x^j)$ in the i^{th} row and j^{th} column.
(2) Form the prediction gram matrix $\hat{K} = \hat{\Phi}\Phi^T$, with elements $\kappa(\hat{x}^i, x^j)$ where $\hat{x}^i$ are samples extracted from the test set.
(3) Take training outputs Y, then calculate $\hat{Y} = \hat{K}(K + \lambda I_N)^{-1}Y$ (in case of L_2 regularization) to predict outputs for the test set.

5. DIMENSIONALITY REDUCTION ALGORITHMS

In many datasets the original features x_j may be both correlated and autocorrelated. Dimensionality reduction algorithms compress the original d_x features into a set of k independent "latent features" which can be used for knowledge extraction.

5.1 Principal Component Analysis (PCA)

Data X of d_x dimensions can be compressed into data Z_x of k dimensions using the following transformation:

$$Z_x = XW_x + E_{Z_x}, \tag{9}$$

where Z_x is a n-by-k matrix containing the k-dimensional latent variables. These are also known as *scores* which represent the orthogonal projections from the d_x-dimensional space into the lower k-dimensional space. The linear weights are known as *loadings* which are expressed as a d_x-by-k matrix W_x. The columns of W_x represent the principal components (PCs), the directions of maximum variance in the original data X. The *lower-dimensional reconstruction* of X is XW_x, and therefore E_{Z_x} represents the *reconstruction errors* between the true scores Z_x and reconstructions XW_x.

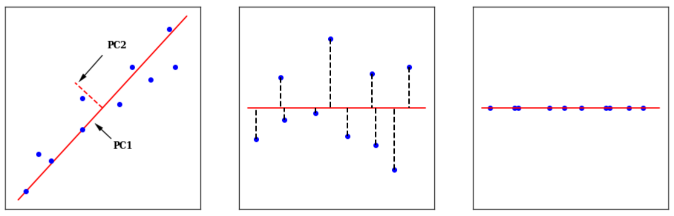

Fig. 4. A data-set with $d_x = 2$ (left) is compressed into data with $k = 1$ (center) using PCA. $PC_1 = W_1$ and $PC_2 = W_2$ represent the directions of maximal variance in the original 2D space. Z_1 (right) is the score or latent variable as a result of orthogonal projection onto W_1.

Parameters Z_x and W_x are determined by the step-wise minimization of the following objective function:

$$(Z_x, W_x) = \underset{Z_x, W_x}{argmin}||X - Z_xW_x||_F, \tag{10}$$

where F denotes the *Frobenius* norm. Initializations of Z_x and W_x are iterated in (10). One of Z_x or W_x is held constant, and the minimization is performed with respect to the other variable. The constant variable is then switched, and the process repeats until convergence. In many PCA algorithms, orthogonality is enforced on the loadings W_x to ensure that the latent variables Z_x are *independent*. Finally, PCA can also be performed analytically by singular value decomposition (SVD), which is outlined in detail in Murphy (2012).

5.2 *Partial Least Squares (PLS)*

A specific application of PCA is known as PLS, where it is desired to find a set of latent variables that:

(1) Explain the trends observed in inputs X.
(2) Explain the behaviour observed in outputs Y.

During PLS, PCA is performed simultaneously on both inputs X and outputs Y. Equivalent, k-dimensional weights W_x, W_y and scores Z_x, Z_y are found to satisfy the following equations:

$$X = Z_x W_x^T + E_x, \tag{11}$$

$$Y = Z_y W_y^T + E_y. \tag{12}$$

The covariance between scores Z_x and Z_y are maximized, as opposed to just the variance of scores Z_x. The detailed optimization procedure can be found in literature by Gerlach et al. (1979) and Wold (1985) who originally developed PLS. Geladi (1989), Nomikos and MacGregor (1994), and Qin and McAvoy (1996) improved PLS by adapting it to both batch and continuous processes, as well as account for process dynamics (Li et al., 2011).

From a least-squares perspective, PLS essentially finds the following model:

$$Z_y = \beta Z_x + E_z, \tag{13}$$

which explains its name: only a *partial* set of independent latent variables in X are responsible for explaining the relevant *partial* set of latent variables in Y.

5.3 *Isometric Mapping (ISOMAP)*

Unlike PCA, ISOMAP (developed by Tenenbaum et al. (2000)) preserves pair-wise *geodesic distances* between data during compression. The *geodesic distance* is defined as the shortest distance between two points while always staying on the manifold spanned by the data. By constrast, the *Euclidean distance* is the shortest distance in space between two points. If *Euclidean distance* is used instead of *geodesic*, then two datapoints of high dissimilarity may be mapped as similar points during compression, if the manifold is non-linear. A visualization of this pitfall is illustrated in Fig. 5.

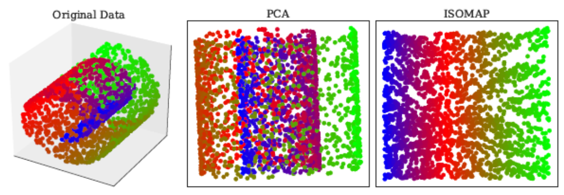

Fig. 5. The original Swiss roll in $d_x = 3$ dimensions, to be compressed to $k = 2$ dimensions (left). PCA fails to retain the neighbourhood information between highly-dissimilar blue and green points (center). ISOMAP perfectly retains neighborhood information due to its use of geodesic distances (right).

The detailed procedure of ISOMAP can be summarized into the following steps:

(1) For each sample x^i, find its k_N-nearest neighbours within a pre-defined radius r.
(2) Construct a graph between x^i and its neighbours by treating the points as nodes and inter-point distances as edges.
(3) Assign a weight to each edge using a selected distance metric.
(4) Determine the shortest path between all points with respect to the weighted edges.
(5) Perform Multi-Dimensional Scaling.

5.4 *Locally Linear Embedding (LLE)*

Data of d_x dimensions down can also be compressed to k dimensions using LLE. For non-linear manifolds, however, PCA may prove ineffective since it assumes all data lie on a linear manifold. On the other hand, LLE assumes data lie on *locally-linear manifolds*, formed between each point and its k_N nearest-neighbours. Numerically, LLE optimizes a constrained least-squares objective, then solves an eigenvalue problem described by Roweis and Saul (2000) in detail. Therefore one of the main drawbacks of LLE is its high computational burden. The following example shows data compression by LLE.In the following case-study, an S-shaped roll is compressed from $d_x = 3$ down to $k = 2$. Notice that the sequence of colours from red to blue, or the "neighbourhood information," is perfectly preserved during compression by LLE. In unsupervised LLE, the hyperparameter k_N is specified arbitrarily. In supervised LLE (SLLE) as developed by De Ridder et al. (2003), k_N is determined by cross-validation.

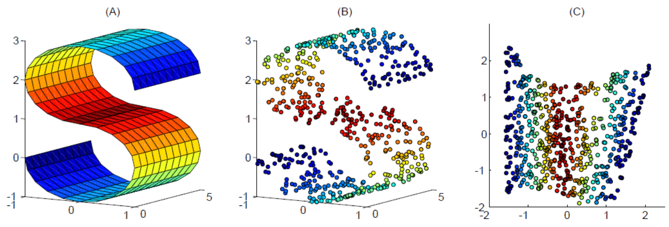

Fig. 6. A 3-D S-shaped roll (left) is compressed to 2-D, using nearest-neighbour seeking (center) and data reconstruction (right).

5.5 *Canonical Correlation Analysis (CCA)*

Dimensionality reduction can also be accomplished by CCA (Larimore, 1996). This algorithm seeks the maximum correlation between linear combinations of two sets of random variables, known as *canonical variables*. These variables are denoted as vectors $X \in \mathbb{R}^{d_x}$ and $Y \in \mathbb{R}^{d_y}$. Their correlations are known as *canonical correlations*. The means and covariances are:

$$\mathbb{E}\left(\begin{bmatrix} X \\ Y \end{bmatrix}\right) = \begin{bmatrix} \mu_X \\ \mu_Y \end{bmatrix}, cov\left(\begin{bmatrix} X \\ Y \end{bmatrix}\right) = \begin{bmatrix} \Sigma_{XX} & \Sigma_{XY} \\ \Sigma_{YX} & \Sigma_{YY} \end{bmatrix},$$

where Σ_{XX} and Σ_{YY} are both assumed to be non-singular. CCA seeks a set of linear projections respectively for X and Y such that the canonical correlations are maximized. If the pairs of linear projection are defined as $u_i \in \mathbb{R}^m$ and $v_i \in \mathbb{R}^n$, $i \in [1, d]$, respectively, then the i^{th} canonical correlation is expressed as:

$$corr(u_i^T X, v_i^T Y) = \frac{u_i^T \Sigma_{XY} v_i}{\sqrt{u_i^T \Sigma_{XX} u_i}\sqrt{v_i}^T \Sigma_{YY} v_i}. \tag{14}$$

The objective of CCA is:

$$\max_{u_i,v_i} \quad corr(u_i^T X, v_i^T Y), \tag{15}$$

$$\text{s.t.} \quad u_i^T \Sigma_{XX} u_i = 1, v_i^T \Sigma_{YY} v_i = 1, \tag{16}$$

$$u_i^T \Sigma_{XX} u_j = 0, v_i^T \Sigma_{YY} v_j = 0, j = 1, \dots, i-1. \tag{17}$$

In (14), any scaling of u_1 and v_1 does not affect the correlation. However, with the presence of constraint (16), a unique pair of projection vectors can be found. Moreover, constraint (17) ensures that each canonical variate is uncorrelated with preceding ones. Just like PLS and PCA, (15)-(17) can be solved using generalized eigenvalue decomposition:

$$\begin{bmatrix} 0 & \Sigma_{XY} \\ \Sigma_{YX} & 0 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \lambda \begin{bmatrix} \Sigma_{XX} & 0 \\ 0 & \Sigma_{YY} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}. \tag{18}$$

The optimal projection vectors are the eigenvectors corresponding to the first d largest eigenvalues in (18). A more efficient and stable solution to (15)-(17) involves the singular value decomposition (SVD) of the form

$$\Sigma_{XX}^{-1/2} \Sigma_{XY} \Sigma_{YY}^{-1/2} = U \Sigma V^T, \tag{19}$$

and the optimal projections are respectively $U_d^T \Sigma_{XX}^{-1/2}$ and $V_d^T \Sigma_{YY}^{-1/2}$, where U_d and V_d are the first d columns of U and V in (19).

6. ADVANCED LEARNING ALGORITHMS

6.1 Artificial Neural Networks (ANNs)

The manipulation of data dimensions can be enhanced by two simple but major improvements:

(1) Dimensionality expansion (finding sets of latent variables with higher dimensionality), as well as reduction.
(2) Performing dimensionality reduction or expansion more than once throughout the model.

ANNs were spearheaded by Pearlmutter and Hinton (1986) and Waibel et al. (1989), and made popular by recent advancements in parallel and quantum computing. The standard ANN notation is as follows:

Table 3. Standard Neural Network notation

Notation	Definition
$Z^{[l]}$	Matrix of all latent variables in all samples in l^{th} layer
$z_j^{[l]}$	Vector of all samples of j^{th} neuron (latent variable) in l^{th} layer
$z^{[l](i)}$	Vector of all latent variables in l^{th} layer in i^{th} sample

Each x^i enters as the *input layer*, and y^i exits from the *output layer*. Hidden layers are formed in between, which contain latent variables Z of various dimensions. Each hidden layer consists of several *neurons*. An ANN has a total of L layers and for any layer within the network, each neuron consists of two parts:

(1) An affine term $Z^{[l]} = W^{[l]} A^{[l-1]} + B^{[l]}$ which is a weighted sum of activations from the previous layers, plus a bias term.
(2) An activation function $A^{[l]} = g^{[l]}(Z^{[l]})$ where g is a non-linear function.

Some typical activation functions include sigmoid, hyperbolic tangent, and rectified linear unit. A linear activation is almost never used because it reduces the neural network to a trivial linear regression between X and Y. Similarly, if an ANN has only input and output layers but no hidden layers, then it reduces to many of the algorithms outlined in Sections 3 and 4. As more hidden layers are introduced, the non-linear relationship between X and Y becomes more complex, and the prediction power of the ANN increases. Once the activation functions are decided, the values of A, Z, W, and B are calculated using back-propagation (Rumelhart et al., 1988). The final output $\hat{Y}$ can be either continuous (regression) or categorical (classification). Finally, the term *deep learning (DL)* refers to ANNs that use many hidden layers $L > 15$ (Goodfellow et al., 2016).

6.2 Gaussian Processes

Gaussian Processes (GPs) as developed by Rasmussen and Williams (2006) form *non-parametric, probabilistic* models between inputs X and outputs Y. The training set contains both outputs and inputs $\mathcal{D} \equiv \{\mathbf{Y}, \mathbf{X}\}$, which are related as follows:

$$y^i = f(\mathbf{x^i}) + \epsilon^i, \tag{20}$$

where f is the underlying process model and $\epsilon^i \sim \mathcal{N}(0, \sigma^2)$ is a zero-mean Gaussian noise with variance σ^2. A GP is completely defined by a mean function and a covariance function, such that

$$\mathbf{f}(\mathbf{X}) \sim \mathcal{GP}(\boldsymbol{\mu}(\mathbf{X}), \boldsymbol{\Sigma}(\mathbf{X}, \mathbf{X})). \tag{21}$$

The mean and covariance matrices are defined as $\boldsymbol{\mu}_\theta(\mathbf{X})$ and $\boldsymbol{\Sigma}_\theta(\mathbf{X}, \mathbf{X})$, where $\theta \in \mathbb{R}^{n_\theta}$ represents hyperparameters. A common choice for Σ is the Gaussian-RBF kernel, which assigns a higher correlation if the Euclidean distance between inputs in the set $\{\mathbf{x}^i, \mathbf{x}^j\}$ are small:

$$\Sigma_\theta(\mathbf{x}^i, \mathbf{x}^j) = \beta \exp\left(-\frac{||x^i - x^j||_2^2}{2\alpha^2}\right). \tag{22}$$

In (22), the hyperparameters θ include β (length scale) and α (signal variance). These affect the *smoothness* of the function f. Given $\mathcal{D}$, the model learns both hyperparameters and any other unknown parameters denoted by the set Γ. Learning is accomplished by maximizing the following likelihood function (Tulsyan et al., 2018):

$$p(\mathbf{Y}|\mathbf{X}) = \mathcal{N}(\mathbf{0}_N, \boldsymbol{\Sigma}_\theta(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}_N). \tag{23}$$

Unknown parameters $\gamma \equiv \{\theta, \sigma^2\} \in \Gamma$ can be estimated by minimizing the following:

$$\gamma^* \in \arg\max_{\gamma \in \Gamma} \log p(\mathbf{Y}|\mathbf{X}), \tag{24}$$

where $\gamma^* \in \Gamma$ is an optimal estimate, and

$$\log p(\mathbf{Y}|\mathbf{X}) = -\frac{1}{2} \mathbf{Y}^{\mathrm{T}} \boldsymbol{\Sigma}_\gamma^{-1} \mathbf{Y} - \frac{1}{2} \log |\boldsymbol{\Sigma}_\gamma| - \frac{N}{2} \log 2\pi, \tag{25}$$

where $\boldsymbol{\Sigma}_\gamma \equiv \boldsymbol{\Sigma}_\theta + \sigma^2 \mathbf{I}_N$. The partial derivatives with respect to γ are

$$\frac{\partial}{\partial \gamma} \log p(\mathbf{Y}|\mathbf{X}) = \frac{1}{2} \operatorname{Tr}\big((\boldsymbol{\alpha}\boldsymbol{\alpha}^{\mathrm{T}} - {}_{\gamma}^{-1} \frac{\partial \boldsymbol{\Sigma}_\gamma}{\partial \gamma})\big), \tag{26a}$$

where $\boldsymbol{\alpha} = \boldsymbol{\Sigma}_\gamma^{-1} \mathbf{Y}$, a non-convex function with multiple local optima. Gradient descent can be used but it must be initialized carefully. Once a GP is trained, $\hat{Y}$ is predicted by first constructing a joint density between $\mathbf{Y}$ and test $f(\hat{\mathbf{X}})$ conditioned on both training $\mathbf{X}$ and test $\hat{\mathbf{X}}$. This joint density is given by

$$p(\mathbf{Y}, f(\hat{\mathbf{X}})|\mathbf{X}, \hat{\mathbf{X}}) = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{\Sigma}_\gamma(\mathbf{X},\mathbf{X}) & \mathbf{\Sigma}_\theta(\mathbf{X},\hat{\mathbf{X}}) \\ \mathbf{\Sigma}_\theta(\hat{\mathbf{X}},\mathbf{X}) & \Sigma_\theta(\hat{\mathbf{X}},\hat{\mathbf{X}}) \end{bmatrix}\right), \tag{27}$$

where $\mathbf{\Sigma}_\gamma \equiv \mathbf{\Sigma}_\theta(\mathbf{X},\mathbf{X}) + \sigma^2\mathbf{I}_N$. Then a posterior distribution over $f(\hat{\mathbf{X}})$ can be computed as:

$$p(f(\hat{\mathbf{X}})|\mathcal{D}, \hat{\mathbf{X}}) = \mathcal{N}(\hat{\mu}_\theta, \hat{\Sigma}_\theta), \tag{28}$$

and the mean and covariance matrices are computed as:

$$\hat{\mu}_\theta = \mathbf{\Sigma}_\theta(\hat{\mathbf{X}},\mathbf{X})[\mathbf{\Sigma}_\gamma(\mathbf{X},\mathbf{X})]^{-1}\mathbf{Y}, \tag{29}$$

$$\hat{\Sigma}_\theta = \Sigma_\theta(\hat{\mathbf{X}},\hat{\mathbf{X}}) - \mathbf{\Sigma}_\theta(\hat{\mathbf{X}},\mathbf{X})[\mathbf{\Sigma}_\gamma(\mathbf{X},\mathbf{X})]^{-1}\mathbf{\Sigma}_\theta(\mathbf{X},\hat{\mathbf{X}}). \tag{30}$$

Finally, the posterior for $\hat{Y}$ can be predicted as follows:

$$p(\hat{Y}|\mathcal{D}, \hat{\mathbf{X}}) = \mathcal{N}(\hat{\mu}_\theta, \hat{\Sigma}_\theta + \sigma^2). \tag{31}$$

For a single test input $\hat{\mathbf{X}}$, the GP prediction gives a distribution of outputs. However, for process control and monitoring, the point-estimate value $\widehat{Y} = \hat{\mu}$ is preferred over the distribution.

7. DEALING WITH DYNAMIC DATA

The typical assumption of inputs $x^1, \cdots, x^N$ being *identically and independently distributed (IID)* often does not apply. Process data can contain temporal samples which are auto-correlated. One simple solution is to augment the input matrix X to include all significant *lagged variables*, as determined by time-series models such as Box-Jenkins (Ljung, 1998). The enormous amount of features resulting from these temporal samples compressed using algorithms in Section 5 or Section 6.1. Finally, *imputation* approaches, such as particle filters (Gopaluni et al., 2009; Tulsyan et al., 2016), Bayesian methods (Tulsyan et al., 2013), or the expectation maximization algorithm (Zhang et al., 2015) can be used to fill in missing data.

8. CONCLUSIONS

This tutorial paper summarizes several commonly-used classification, regression, dimensionality-reduction, and advanced learning methods that can be applied to historical process data to enhance control decisions. The current paper will extend into two more sophisticated works:

(1) An interactive workshop introducing the use of ML techniques for fault identification and diagnosis, as well as data treatment software such as Python and Jupyter notebooks.
(2) An in-depth journal paper exploring the suitability of each ML technique for fault identification and diagnosis, both theoretically and using simulation results on real-time plant data.

REFERENCES

Bishop, C.M. (2006). Pattern Recognition and Machine Learning, volume 1. Springer.

Celeux, G. (1998). Bayesian inference for mixture: The label switching problem. In Compstat, 227–232. Springer.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. Machine learning, 20(3), 273–297.

De Ridder, D., Kouropteva, O., Okun, O., Pietikäinen, M., and Duin, R.P. (2003). Supervised locally linear embedding. In ICANN, 333–341. Springer.

Geladi, P. (1989). Analysis of multi-way (multi-mode) data. Chemometrics and Intelligent Laboratory Systems, 7, 11–30.

Gerlach, R.W., Kowalski, B.R., and Wold, H.O. (1979). Partial least-squares path modelling with latent variables. Analytica Chimica Acta, 112(4), 417–421.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep learning. MIT press.

Gopaluni, R.B., Schön, T.B., and Wills, A.G. (2009). Particle filter approach to nonlinear system identification under missing observations with a real application. IFAC Proceedings Volumes, 42(10), 810–815.

Jain, A.K. (2010). Data clustering: 50 years beyond k-means. Pattern recognition letters, 31(8), 651–666.

Jakkula, V. (2006). Tutorial on support vector machine (svm). School of EECS, Washington State University, 37.

Larimore, W.E. (1996). Statistical optimality and canonical variate analysis system identification. Signal Processing, 52(2), 131–144.

Lemm, S., Blankertz, B., Dickhaus, T., and Müller, K.R. (2011). Introduction to machine learning for brain imaging. Neuroimage, 56(2), 387–399.

Li, G., Liu, B., Qin, S.J., and Zhou, D. (2011). Quality relevant data-driven modeling and monitoring of multivariate dynamic processes: The dynamic t-pls approach. IEEE transactions on neural networks, 22(12), 2262–2271.

Ljung, L. (1998). System identification. In Signal analysis and prediction, 163–173. Springer.

Murphy, K.P. (2012). Machine Learning: A Probabilistic Perspective, volume 1. MIT Press.

Nomikos, P. and MacGregor, J.F. (1994). Monitoring batch processes using multiway principal component analysis. American Institute of Chemical Engineers, 40(8), 1361–1375.

Pearlmutter, B.A. and Hinton, G.E. (1986). G-maximization: an unsupervised learning procedure for discovering regularities. In AIP conference proceedings, volume 151, 333–338. AIP.

Qin, S.J. and McAvoy, T.J. (1996). Nonlinear fir modelling via a neural net pls approach. Computers Chemical Engineering, 20(2), 147–159.

Rasmussen, C.E. and Williams, C.K. (2006). Gaussian processes for machine learning, volume 1. MIT press Cambridge.

Roweis, S.T. and Saul, L.K. (2000). Nonlinear dimensionality reduction by locally linear embedding. science, 290(5500), 2323–2326.

Rumelhart, D.E., Hinton, G.E., Williams, R.J., et al. (1988). Learning representations by back-propagating errors. Cognitive modeling, 5(3), 1.

Tenenbaum, J.B., De Silva, V., and Langford, J.C. (2000). A global geometric framework for nonlinear dimensionality reduction. science, 290(5500), 2319–2323.

Tulsyan, A., Garvin, C., and Undey, C. (2018). Machine-learning for biopharmaceutical batch process monitoring with limited data. In 10th IFAC Symposium on Advanced Control of Chemical Processes. Shenyang, China.

Tulsyan, A., Gopaluni, R.B., and Khare, S.R. (2016). Particle filtering without tears: A primer for beginners. Computers & Chemical Engineering, 95, 130–145.

Tulsyan, A., Huang, B., Gopaluni, R.B., and Forbes, J.F. (2013). On simultaneous on-line state and parameter estimation in non-linear state-space models. Journal of Process Control, 23(4), 516–526.

Vapnik, V.N. and Vapnik, V. (1998). Statistical learning theory, volume 1. Wiley New York.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K.J. (1989). Phoneme recognition using time-delay neural networks. IEEE transactions on acoustics, speech, and signal processing, 37(3), 328–339.

Wold, H. (1985). Partial least squares. Encyclopedia of statistical sciences.

Zhang, K., Gonzalez, R., Huang, B., and Ji, G. (2015). Expectation–maximization approach to fault diagnosis with missing data. IEEE Transactions on Industrial Electronics, 62(2), 1231–1240.