Evaluation of a PMML-Based GPR Scoring Engine on a Cloud Platform and Microcomputer Board for Smart Manufacturing

Max Ferguson and Kincho H. Law
Civil and Environmental Engineering
Stanford University
Stanford, CA, USA
{maxferg, law}@stanford.edu

Raunak Bhinge and David Dornfeld
Mechanical Engineering
University of California, Berkeley
Berkeley, CA, USA
raunakbh@berkeley.edu

Jinkyoo Park
Industrial and Systems Engineering
Korea Advanced Institute of Science and Technology
Daejeon, South Korea
jinkyoo.park@kaist.ac.kr

Yung-Tsun Tina Lee
Systems Integration Division
National Institute of Standards and Technology (NIST)
Gaithersburg, MD, 20899, USA
yung-tsun.lee@nist.gov

Abstract— **The use of data-driven predictive models is becoming increasingly popular in engineering and manufacturing sectors. This paper discusses the deployment of Gaussian Process Regression (GPR) predictive models for smart manufacturing. A scoring engine is developed based on the Predictive Model Markup Language (PMML) standard to illustrate the portability of predictive models among different statistical tools and different platforms. Specifically, we evaluate the tradeoffs between embedding GPR-based predictive models on a physical device and executing the predictive models on a managed cloud platform like the Google Compute Engine. We compare the performance of the two deployment strategies with two predictive models, namely an energy consumption model and a milling tool condition model, that are built with data from a Mori Seiki CNC milling machine. We describe how the response time of the two deployment strategies is related to the network latency and computational speed of the scoring machine hardware. It is shown that the time required to calculate model predictions is a significant factor in the overall response time of the embedded scoring engine. We demonstrate that the scoring engine on the cloud platform can achieve a lower response time and higher prediction rate than the microcomputer, due to the superior computational performance of the cloud-based hardware.**

Keywords— scoring engine; predictive model markup language; energy prediction model; milling machine.

I. INTRODUCTION

The rise of the Industrial Internet will inevitably be accompanied by an increased usage of machine learning algorithms for the monitoring and control of manufacturing machines [1]. Modern factories will utilize predictive models for a range of tasks, from identifying manufacturing defects to optimizing chemical processes [2]. The predictive models may be evaluated on embedded circuits located within the manufacturing machines. However, modern predictive models are increasingly demanding, both in terms of computational and storage requirements, to train and to evaluate [3, 4]. While using dedicated hardware to evaluate the predictive models is desirable for certain applications, an alternative approach is to evaluate the predictive models on a managed cloud platform. In this scenario, manufacturing machines send operating data to dedicated servers in the cloud, which evaluate the data according to the predictive models, and send back the results.

With the development and adoption of the Predictive Model Markup Language (PMML), it is now much easier to train and evaluate predictive models on separate computers. PMML is an XML-based language that enables the definition and sharing of predictive models between applications [5]. It provides a clean and standardized interface between the software tools that produce predictive models, such as statistical or data mining systems, and the consumers of models, such as applications that depend upon embedded analytics [6]. With PMML it is easy to train a model with a statistical package such as R, and save the model in a standardized format for use in a real engineering application.

For deployment, predictive models are normally evaluated by a *scoring engine*. A scoring engine is a piece of software specifically designed to load a model in a standardized format, and use it to evaluate new observations or data points. Scoring engines are responsible for executing the mathematical operations that transform model inputs into model outputs. To promote interoperability, scoring engines are normally written in languages such as Python or Java [7], which are supported by most embedded systems and computing environments. Therefore, it is feasible to run the same scoring engine on a development computer, a cloud server, or an embedded device, without modifying the scoring engine code. The development of standards-compliant scoring engines allows PMML models to be reliably evaluated on a range of devices.

In a smart manufacturing setting, the throughput and response time of the scoring engine can be critical to the performance of the manufacturing process. One way to meet the computational requirements of modern predictive models is to install the scoring engine on a managed cloud platform

978-1-4673-9005-7/16/$31.00 ©2016 IEEE

like the Google Compute Engine [8]. In this scenario, the scoring engine is transferred to the cloud system along with the relevant PMML model files. A network connection is established between the manufacturing machine and the cloud server. Data from the manufacturing machine is streamed to the cloud server, and passed to the scoring engine. The scoring engine evaluates the data and sends the relevant information back to the manufacturing machine. The advantage of this method is that cloud computing services provide scalable memory and compute power, which facilitate real-time evaluation of complex machine learning models. However, the network connection between the manufacturing machine and the cloud server adds latency to the prediction process.

An alternative solution is to install the scoring engine on an embedded device located near or within the manufacturing machine. With this solution the scoring engine is installed on the embedded device and the model is transferred using the PMML format. Data from the manufacturing process is sent to the scoring engine across a local area network (LAN) connection. The scoring engine evaluates the data and sends the relevant information back to the manufacturing machine. The advantage of this solution is that the network latency between the manufacturing machine and the scoring engine is minimized. However, the effectiveness of this approach relies heavily on the capability of the embedded system. There is uncertainty whether an embedded system can provide sufficient memory and computational power to evaluate a predictive model in real-time.

In this work we examine the tradeoffs of deploying a GPR predictive model on a microcomputer board and on a cloud platform. We develop a PMML-based scoring engine for the evaluation of Gaussian Process Regression (GPR) models. We then install the scoring engine on a Raspberry Pi microcomputer and a virtual server on the Google Compute Engine. The performance of the scoring engine is evaluated using two examples, namely an energy consumption model and tool condition model, that are drawn from automated manufacturing. The paper is concluded with a brief summary and discussion for future work.

II. GAUSSIAN PROCESS REGRESSION

Recently, a range of predictive modelling techniques have been proposed for use within automated manufacturing. We choose to focus on Gaussian Process Regression (GPR), as we have demonstrated how this technique can be applied to manufacturing problems [9, 10]. In this section, we briefly introduce the basic procedure for constructing a GPR model. We then discuss how a trained Gaussian Process (GP) model can be used to evaluate (score) new unseen data points.

A. Gaussian Process Regression (GPR) Modeling

GPR is a supervised machine-learning method that performs particularly well with noisy data [11]. The aim is to approximate an unknown target function $y = f(\boldsymbol{x})$ in a probabilistic manner. In general, we denote the inputs as $\boldsymbol{x} \in R^n$ and the target value as $y \in R$. The function values $\boldsymbol{f}^{1:n} = (f^1, \dots, f^n)$ corresponding to the input $\boldsymbol{x}^{1:n} = (\boldsymbol{x}^1, \dots, \boldsymbol{x}^n)$ are treated as random variables, where $f^i \coloneqq f(\boldsymbol{x}^i)$.

GPR uses GP as a prior to describe the distribution on the target function $f(\boldsymbol{x})$. A GP is a generalization of the Gaussian probability distribution for which any finite linear combination of samples has a joint Gaussian distribution. As the GP is a multivariate Gaussian distribution over the function $f(\cdot)$, it can be fully specified by its mean function $m(\cdot)$ and covariance function $k(\cdot,\cdot)$.

$$p(f^{1:n}) = GP(m(\cdot), k(\cdot,\cdot)) \tag{1}$$

In a GP, $m(\cdot)$ and $k(\cdot,\cdot)$ are not constant parameters but functions incorporating prior knowledge about the target function [11]. The mean function $m(\cdot)$ captures the overall trend in the target function value. The kernel function $k(\cdot,\cdot)$, is used to approximate the covariance between the two function values f^i and f^j.

In GPR, the type of kernel function chosen can strongly affect the representability of the GPR model, and influence the accuracy of the predictions. One widely used kernel function is the Automatic Relevance Determination (ARD) squared exponential covariance function:

$$k(\boldsymbol{x}^i, \boldsymbol{x}^j) = \gamma \exp\left(-\frac{1}{2}(\boldsymbol{x}^i - \boldsymbol{x}^j)^T \operatorname{diag}(\boldsymbol{\lambda})^{-2}(\boldsymbol{x}^i - \boldsymbol{x}^j)\right) \tag{2}$$

An ARD kernel provides the flexibility to adjust the relevance (weight) of each parameter in the feature vector. In the ARD squared exponential kernel, the parameter vector $\boldsymbol{\lambda} = (\lambda_1, \dots \lambda_i, \dots, \lambda_m)$ quantifies the relevancy of the input features in $\boldsymbol{x^i} = \left(x_1^i, \dots x_k^i, \dots, x_m^i\right)$, where m denotes the number of input variables, for predicting the response y. The parameter γ is generally referred to as the signal variance, and quantifies the overall magnitude of the covariance value.

It is assumed that the observed value y^i is measured with some random Gaussian noise ϵ^i, such that $y^i = f(\boldsymbol{x}^i) + \epsilon^i$. It is common to assume that the noise term ϵ^i is independent and identically distributed Gaussian $\epsilon^i \sim N(0, \sigma_\epsilon^2)$, in which case the noise variance σ_ϵ^2, quantifies the amount of noise that is assumed to exist in the response. In short, the GPR model can be fully parameterized by the hyperparameters $\boldsymbol{\theta} = (\sigma_\epsilon, \gamma, \boldsymbol{\lambda})$ for the case that the ARD squared exponential kernel is employed.

B. Training Procedure

The GPR training procedure involves selecting a set of hyperparameters so as to maximize the marginal likelihood of the training data. The marginal distribution of the observations can be expressed as:

$$p(\boldsymbol{y}^{1:n}|\boldsymbol{\theta}) = \int p(\boldsymbol{y}^{1:n}|\boldsymbol{f}^{1:n},\boldsymbol{\theta})\, p(\boldsymbol{f}^{1:n}|\boldsymbol{\theta})\, d\boldsymbol{f}^{1:n} \tag{3}$$

The unknown function $\boldsymbol{f}$ can be marginalized out of (3) to obtain the marginal likelihood of the training observations. An optimization equation is then formed to maximize the marginal likelihood, and obtain the optimum hyperparameters $\boldsymbol{\theta}^*$ [11]:

$$\boldsymbol{\theta}^* = \arg\max_{\theta} \log p\,(\mathbf{y}^{1:n}|\boldsymbol{\theta}\,) \tag{4}$$

$$\begin{aligned} = \arg\max_{\theta} \Big(& -\tfrac{1}{2}(\mathbf{y}^{1:n})^{\mathrm{T}}(\mathbf{K}+\sigma_\epsilon^2\mathbf{I})^{-1}\mathbf{y}^{1:n} \\ & -\tfrac{1}{2}\log|\mathbf{K}+\sigma_\epsilon^2\mathbf{I}| - \tfrac{n}{2}\log 2\pi \Big) \end{aligned} \tag{5}$$

where $\mathbf{K}$ is the covariance matrix whose (i,j)th entry is given by $\mathbf{K}_{ij} = k(\boldsymbol{x}^i, \boldsymbol{x}^j)$.

The process for obtaining the optimum hyperparameters is well documented in the literature [11]. In this work, the MATLAB GPML library [12] is used to optimize the hyperparameters.

C. Scoring a Gaussian Process Regression Model

Suppose a new data point, denoted as $\boldsymbol{x}^{new}$, is observed. The task at hand is to predict the posterior distribution on the response f^{new} for the newly observed input $\boldsymbol{x}^{new}$. In the case where the mean function is zero, the (hidden) response value f^{new} and the observed outputs $\boldsymbol{y}^{1:n} = \{y^1, \dots, y^n\}$ are given as:

$$\begin{bmatrix} \boldsymbol{y}^{1:n} \\ f^{new} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} (\mathbf{K}+\sigma_\epsilon^2\mathbf{I}) & \boldsymbol{k} \\ \boldsymbol{k}^T & k(\boldsymbol{x}^{new}, \boldsymbol{x}^{new}) \end{bmatrix}\right) \tag{6}$$

where $\boldsymbol{k}^T = \left(k(\boldsymbol{x}^1, \boldsymbol{x}^{new}), \dots, k(\boldsymbol{x}^n, \boldsymbol{x}^{new})\right)$.

Suppose the training dataset is denoted as $\boldsymbol{D}^n = \{(\boldsymbol{x}^i, y^i) | i = 1, \dots, n\}$. The posterior distribution on the response f^{new} for the newly observed input $\boldsymbol{x}^{new}$ given the historical data can then be expressed as a Gaussian distribution:

$$f^{new} \sim \mathcal{N}\left(\mu(\boldsymbol{x}^{new}|\boldsymbol{D}^n), \sigma^2(\boldsymbol{x}^{new}|\boldsymbol{D}^n)\right) \tag{7}$$

As the posterior distribution is 1D Gaussian, the posterior mean $\mu(\boldsymbol{x}^{new}|\boldsymbol{D}^n)$, and the associated variance (or standard deviation) $\sigma^2(\boldsymbol{x}^{new}|\boldsymbol{D}^n)$, are sufficient to fully describe the posterior distribution. That is, the posterior distribution can be calculated directly [11]:

$$\mu(\boldsymbol{x}^{new}|\boldsymbol{D}^n) = \boldsymbol{k}^T(\mathbf{K}+\sigma_\epsilon^2\mathbf{I})^{-1}\boldsymbol{y}^{1:n} \tag{8}$$

$$\sigma^2(\boldsymbol{x}^{new}|\boldsymbol{D}^n) = k(\boldsymbol{x}^{new}, \boldsymbol{x}^{new}) - \boldsymbol{k}^T(\mathbf{K}+\sigma_\epsilon^2\mathbf{I})^{-1}\boldsymbol{k} \tag{9}$$

The scoring procedure for the GPR model thus involves the operations as depicted in (8) and (9).

III. Test Cases for GPR Models

In this section we describe the two cases that were used to test the use of GPR for the development of predictive models. For both cases, the models are trained using the data recorded on a milling machine. We first introduce an energy consumption model built to predict the energy usage of a Mori Seiki NVD1500DCG milling machine, based on features extracted from the NC machining operations and power data. We then describe the tool condition model built to predict the condition of the milling machine tool, based on acceleration and acoustic features.

A. GPR Energy Consumption Model

Over 22% of energy and 30% of electricity in the United States is consumed by the industrial sector. While the efficiency of the US manufacturing sector continues to increase due to improved manufacturing tools and processes, the industrial sector remains a major energy consumer in the US [13]. Advances in sensor technology and data processing now allow continuous measurements of the operating parameters and energy consumption of manufacturing machines. With modern machine learning techniques, it is possible to further optimize manufacturing techniques to reduce energy consumption.

1) Data Collection and Processing

The first step when constructing an accurate predictive model for a manufacturing device is to collect and process the data from that device. For the milling machine, a systematic approach has been developed to extract the features necessary for constructing the energy prediction model [14]. The energy consumption of the machine was measured under a range of different operating conditions. The machining data, such as the process parameters and the tool position, was recorded from the FANUC controller. The power time series data was collected using a High Speed Power Meter (HSPM). The machine parameters and power consumption times series were synchronized using a MTConnect agent [14].

2) Feature identification and energy consumption model

A GPR model is developed to predict the energy consumption of the milling machine based on the machine parameters. The raw MTConnect data is parsed to extract useful parameters for the model. In particular, the average feed rate, average spindle speed, cut strategy and direction, and cut length and depth are extracted from the MTConnect data. The energy consumption $E \in \mathbb{R}$ is obtained by numerically integrating the power time series over the duration of each cut.

In this work we restrict the model to face-milling operations, but the same technique has been shown to be applicable to other milling operations [9, 10]. The input features for the model are defined as follows:

- Feed rate, $x_1 \in \mathbb{R}$: the average velocity at which the tool is fed material
- Spindle speed, $x_2 \in \mathbb{R}$: rotational speed of the tool
- Depth of cut, $x_3 \in \mathbb{R}$: depth of material that the tool is removing
- Active cutting direction, $x_4 \in \{1, 2, 3, 4\}$: 1 is for x-axis, 2 for y-axis, 3 for z-axis, and 4 for x-y axes
- Cutting strategy, $x_5 \in \{1, 2, 3\}$: The method for removing material, which controls the relationship between the rotation direction and the feed direction. 1 is for conventional, 2 for climb, and 3 for both conventional and climb milling

For the predictive model, we choose to use all of the measured input features and denote the input feature vector, $\boldsymbol{x} = \{x_1, \dots, x_5\}$. We define a new parameter y as the energy consumption per unit length, such that:

$$y^i = E^i / l^i \tag{10}$$

where E^i is the energy consumption for cut i with length l^i.

To construct the GPR model, we assume that the output $y = f(\boldsymbol{x}) + \epsilon$ is measured with noise $\epsilon \sim N(0, \sigma_\epsilon^2)$. We choose the ARD squared exponential function for the covariance kernel function and assume the mean function to be a zero function.

Each new prediction $\hat{y}^i$ is represented by a mean energy density function $\mu(\boldsymbol{x}^i | \boldsymbol{D}^{train})$ and associated standard deviation function $\sigma(\boldsymbol{x}^i | \boldsymbol{D}^{train})$. The estimated energy consumption $\hat{E}^i$ and the corresponding standard deviation S^i can then be calculated as:

$$\hat{E}^i = \mu(\boldsymbol{x}^i | \boldsymbol{D}^{train}) \times l^i \tag{11}$$

$$S^i = \sigma(\boldsymbol{x}^i | \boldsymbol{D}^{train}) \times l^i \tag{12}$$

The GPR model is trained using the procedure described in Section II. A number of experiments were carried out to obtain enough data to develop a comprehensive energy prediction model [9, 10]. Test parts were manufactured with the milling machine, using a range of different operating parameters. In total, 18 parts were machined to provide operational data for 196 face-milling operations. Figure 1 shows the predicted energy consumption as a function of the feed rate, along with some recorded observations. It can be seen that the trained GPR model predicts quite accurately the energy consumption for the face milling operations.

B. Recursive GPR Tool Condition Model

Reliable tool-condition monitoring can provide a number of benefits for the manufacturing industry, such as improved product quality and the prevention of tool breakage. Researchers have previously demonstrated that the condition of the machine tool can be inferred from features of the vibration and audio time series [15, 16]. A number of researchers have attempted to use the skew and kurtosis coefficients of the audio and acceleration time-series to predict the condition of the tool, with mixed results [17, 18]. In this case study, we develop a GPR model to predict the condition of a CNC milling machine tool.

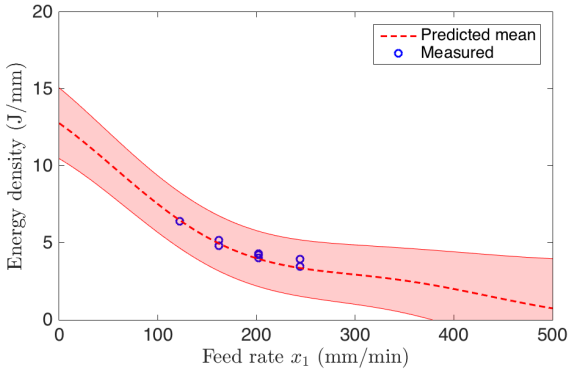

Figure 1. Predicted energy consumption density for generic test parts machined using face milling with y-direction cut, 2,400 RPM spindle speed, conventional cutting strategy, and 1mm cut depth. The shaded area shows one-standard deviation bounds for the prediction.

1) Data Collection and Processing

The Mori Seiki milling machine was programmed to produce a number of simple parts by removing material from a solid steel block. Each part consisted of 18 separate cutting actions performed by the milling machine. The machine was instructed to produce parts until the cutting tool became severely damaged.

For the experiments, the operating parameters of the machine were adjusted to artificially increase the rate of tool-wear. In a normal manufacturing environment, cutting tools generally last several days. To expedite the data collection process in the experiments, the operating lifetime of the cutting tool was reduced to about 30 minutes by increasing the feed rate and reducing the rotational speed.

A sensor from Infinite Uptime was used to measure the audio and acceleration signals inside the milling machine while the machine was operational. The acceleration (vibration) signals were recorded for each cut for all three (x-, y- and z-) directions. The acceleration signal was recorded at 1000 Hz while the audio signal was recorded at 8000 Hz. Welch's method [19] is used with a Hann window to estimate the periodogram for each time series signal. Welches method provides an estimation of the periodogram at N equally spaced points f_k, in the frequency domain.

The condition of the milling machine, denoted by $y_c \in [0,1]$, is defined based on the remaining lifetime of the tool, as estimated after manually examining the tool with a microscope. The scale is defined such that 100% indicates a new tool in perfect condition, and 50% indicates the condition at which the tool would be replaced in a commercial manufacturing setting. Figure 2 illustrates four different states of the tool.

Hereafter, we use the symbols $\hat{S}_a^i(f)$ and $\hat{S}_{v_j}^i(f)$ to denote, respectively, the acoustic periodogram and the

acceleration periodogram measured during cut i. The subscript $j \in \{0,1,2\}$ indicates the x-, y- and z-, directions. Figure 3 and Figure 4 show how the acceleration and acoustic periodograms are influenced by the condition of the tool.

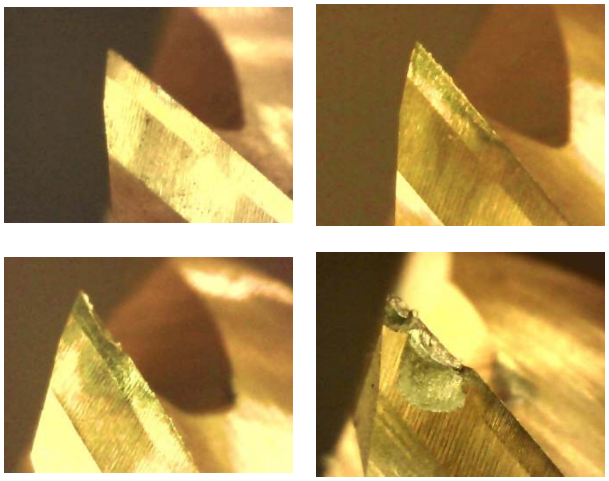

Figure 2. Tool flute in different states of condition. The top-left image shows a new tool with $y_c = 1$, the top-right image shows the tool at $y_c = 0.7$, the bottom-left image shows the tool at $y_c = 0.5$ and bottom-right image shows the tool at $y_c = 0.3$.

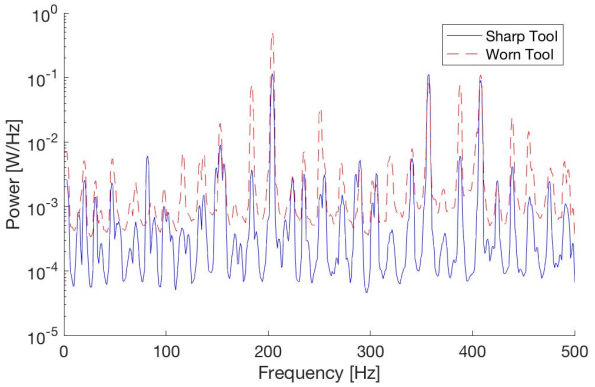

Figure 3. Periodogram computed using the recorded acceleration signal from a sharp tool, and that from a worn tool, for a single cutting operation.

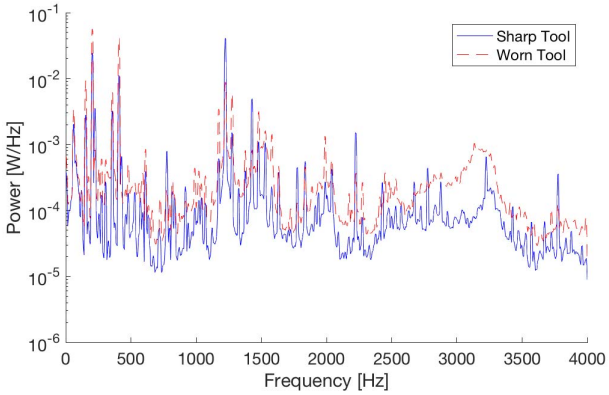

Figure 4. Periodogram computed using the recorded acoustic signal from a sharp tool, and that from a worn tool, for a single cutting operation.

2) Featurization and tool condition monitoring model

One important step for constructing a predictive model is to identify the relevant features that, in this case, are correlated with the condition of the milling machine tool. In this study, we identify a set of 5 features, $\boldsymbol{x} = \{x_1, x_2 \dots x_5\}$ for the tool condition monitoring model. The first four features, x_1, x_2, x_3 and x_4, are derived from the acceleration and acoustic periodograms, while the fifth feature x_5 represents the condition of the tool at the previous time step.

In each experiment, a periodogram is calculated for the first three cuts, and subsequently referred to as the *reference periodogram*. The reference periodogram represents the acceleration and acoustic frequency content produced by a sharp tool. We use the symbol $\hat{S}_a^0(f)$ to represent the reference acoustic periodogram and $\hat{S}_{v_j}^0(f)$ to represent the reference acceleration periodogram. The subscript $j \in \{0,1,2\}$ is used to indicate the direction of the measured acceleration signal.

For each cut i, two input features are defined as the increase in signal power with respect to the reference cut:

$$x_1^i = \sum_{k=0}^{N-1} \hat{S}_a^i(f_k) - \hat{S}_a^0(f_k) \tag{13}$$

$$x_2^i = \sum_{j=0}^{2} \sum_{k=0}^{N-1} \hat{S}_{v_j}^i(f_k) - \hat{S}_{v_j}^0(f_k) \tag{14}$$

where x_1 is a feature describing the increase in acoustic signal power and x_2 is a feature describing the increase in acceleration signal power.

Two additional features are defined as the maximum distance between the measured and the reference periodograms:

$$x_3^i = \max_{k=[0,N-1]} \left| \hat{S}_a^i(f_k) - \hat{S}_a^0(f_k) \right| \tag{15}$$

$$x_4^i = \max_{j,\ k=[0,N-1]} \left| \hat{S}_{vj}^i(f_k) - \hat{S}_{vj}^0(f_k) \right| \tag{16}$$

where x_3 is a feature describing the maximum increase in acoustic signal power and x_4 is a feature describing the maximum increase in vibration signal power.

There is a strong correlation between the tool condition of two sequential cuts. An additional feature x_5, is defined as the tool condition during the previous cut. During the training process the feature x_5, is defined as follows:

$$x_{5\,(training)}^i = \begin{cases} 1 & for\ i = 0 \\ y_c^{i-1} & otherwise \end{cases} \tag{17}$$

In a real manufacturing setting, the previous tool condition will not be available when making real-time predictions. Thus, for the testing process, the previous

prediction $\hat{y}^{i-1}$, is used in place of the previous tool condition y^{i-1}:

$$x^i_{5\,(testing)} = \begin{cases} 1 & for\ i = 0 \\ \hat{y}^{i-1}_c & otherwise \end{cases} \quad (18)$$

3) *Training and testing*

The milling machine was used to produce a total of 52 parts using 14 tools. Each tool was used to manufacture parts until it became heavily worn or broken. The tool condition was manually labeled after each part was produced. Linear interpolation was used to estimate the tool condition y^i for each cut. The data set was divided into a training set with data from 11 tools and a testing set with data from 3 tools. The training set contained 738 individual cuts and the testing set contained 216 individual cuts.

To construct the GPR model, we assume that the output $y_c = f(\boldsymbol{x}) + \epsilon$ is measured with noise $\epsilon \sim N(0, \sigma_\epsilon^2)$. We choose the ARD squared exponential function for the covariance kernel function and assume the mean function to be a zero function. The MATLAB GPML library is used to optimize the model hyperparameters, as described in Section II.

The model is used to predict the condition of the tool for entire testing data set. Predictions are made in the order that the testing set data was recorded. The predicted tool condition for the testing set is shown in Figure 5. The tool condition prediction is plotted against manually labelled observations. The shaded area shows one-standard deviation bounds for the prediction. It can be observed that the trained model predicts the tool conditions comparable to the human labelled results, particularly for tool conditions above the 50%. Under normal circumstances, a tool would have been replaced with less than 30% wearing (i.e. around 70% tool condition).

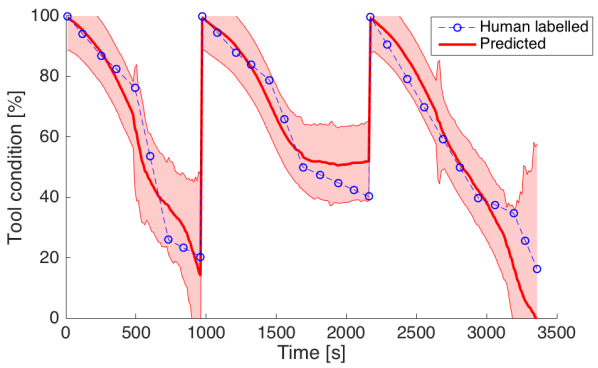

Figure 5. Tool condition prediction against manually labelled observations for the testing set, where the tool was replaced twice. The shaded area shows one-standard deviation bounds for the prediction.

IV. PMML Scoring Engine For GPR Models

To use a predictive model in a real manufacturing setting, a computer program must be designed to evaluate (score) new data points as they are observed. This type of software is commonly referred to as a scoring engine. A scoring engine is primarily responsible for executing the mathematical operations required to transform a new observation to a prediction. However, it must also be able to communicate with other devices in order to receive new observations, and send the corresponding results. A possible network architecture for a smart factory with a single scoring engine is shown in Figure 6. It can be seen that the scoring engine plays a critical role in the operation of the smart factory.

The scoring engine must either contain the optimized model parameters, or be capable of loading the model parameters from another source. The scoring engine described in this work was designed to load the model

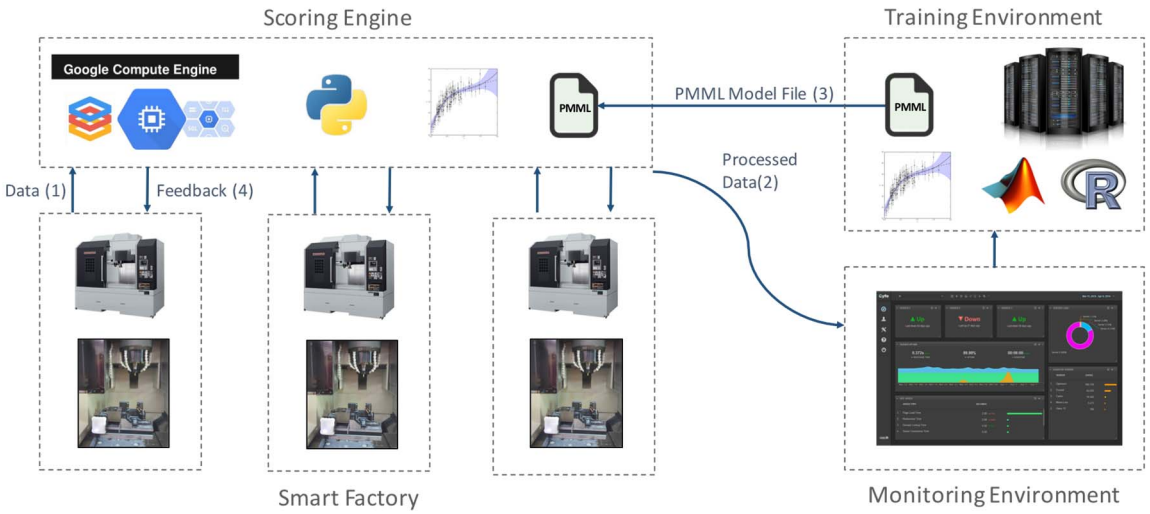

Figure 6. Network diagram showing the role of a PMML scoring machine in a smart factory. The arrows represent the flow of data over a network connection (either a local network or the Internet). While the training and monitoring process is iterative, the numbers indicate the timeline of data flow.

parameters and training data from a PMML file. Specifically, we employ a PMML standard representation of the GPR model [20] to transfer the trained model to the scoring engine. The full architecture of the scoring engine is shown in Figure 7. The standardized nature of the PMML format ensures that the scoring engine can be used to load any valid GPR PMML model, without any changes in the scoring engine code.

It can be seen that the scoring engine consists of three main components, a PMML parser to extract the GPR model, a GPR scoring algorithm to generate predictions and a webserver to communicate with other devices.

In this section, we describe how each component of the scoring engine was built. We start by describing how a parser was written to load the GPR model parameters from a PMML file. We then describe the GPR scoring algorithm that was used to compute the posterior distribution for each new observation. Lastly, we describe how the scoring engine connected to other devices using a simple Python webserver.

A. PMML Parser

Before performing any mathematical operations, the scoring engine must load the GPR model parameters from a PMML file. A PMML file is a text file that describes a machine learning model, using a standardized XML-based language. As the PMML format is based on XML, any compliant XML parser can be used to extract information from the PMML file.

The XML parser from the Python lxml library [21] was used as the PMML parser for the scoring engine. Based on the standard PMML representation of the GPR model [20], the parser was used to extract the relevant parameters and training data from the PMML file and store them in the computer memory.

B. GPR Scoring Algorithm

The GPR scoring algorithm is designed to calculate the posterior distribution on the response f^{new} corresponding to the new observation $\boldsymbol{x}^{new}$. As shown in (7), the posterior distribution can be fully described by the posterior mean $\mu(\boldsymbol{x}^{new}|\boldsymbol{D}^n)$, and the variance $\sigma^2(\boldsymbol{x}^{new}|\boldsymbol{D}^n)$. The scoring algorithm thus involves computing the posterior mean and variance given in (8) and (9). Therefore, the scoring procedure requires computing the inverse of the $n \times n$ matrix $\mathbf{K} + \sigma_\epsilon^2\mathbf{I}$, where n is the number of training data points. For models with a large number of training points, computing the inverse of $\mathbf{K} + \sigma_\epsilon^2\mathbf{I}$ can be computationally demanding.

As the kernel matrix $\mathbf{K}$ is real and symmetric positive semidefinite, Cholesky factorization can be used in lieu of computing the inverse. In particular, there exists a matrix $\mathbf{L}$ such that:

$$\mathbf{L}\mathbf{L}^{\mathrm{T}} = (\mathbf{K} + \sigma_\epsilon^2\mathbf{I}) \tag{19}$$

We define two new vectors $\boldsymbol{\alpha}$ and $\boldsymbol{v}$ such that:

$$\boldsymbol{\alpha} = \mathbf{L}^{-1}\boldsymbol{y}^{1:n} \tag{20}$$

$$\text{and } \boldsymbol{v} = \mathbf{L}^{-1}\boldsymbol{k} \tag{21}$$

The posterior mean and variance can then be calculated, respectively, as:

$$\mu(\boldsymbol{x}^{new}|\boldsymbol{D}^n) = \boldsymbol{v}^{\mathrm{T}}\boldsymbol{\alpha} \tag{22}$$

$$\sigma^2(\boldsymbol{x}^{new}|\boldsymbol{D}^n) = k(\boldsymbol{x}^{new}, \boldsymbol{x}^{new}) - \boldsymbol{v}^T\boldsymbol{v} \tag{23}$$

The kernel matrix $\mathbf{K}$ depends only on the training data; it does not change between predictions. Thus, the Cholesky decomposition of $(\mathbf{K} + \sigma_\epsilon^2\mathbf{I})$ only needs to be performed once. To reduce computational demand, the Cholesky decomposition is performed immediately after the training data is loaded from the PMML model file.

The GPR scoring algorithm is written using the Python numerical computing library NumPy [22], and is similar to the algorithm used by the machine learning library, Scikit-learn [23]. The NumPy library uses LAPACK [24] to perform the linear algebra calculations efficiently.

C. Network Webserver

In a manufacturing environment, the scoring engine must be able to communicate with the manufacturing devices. A network connection is established between the scoring engine and each manufacturing device, allowing the device to send

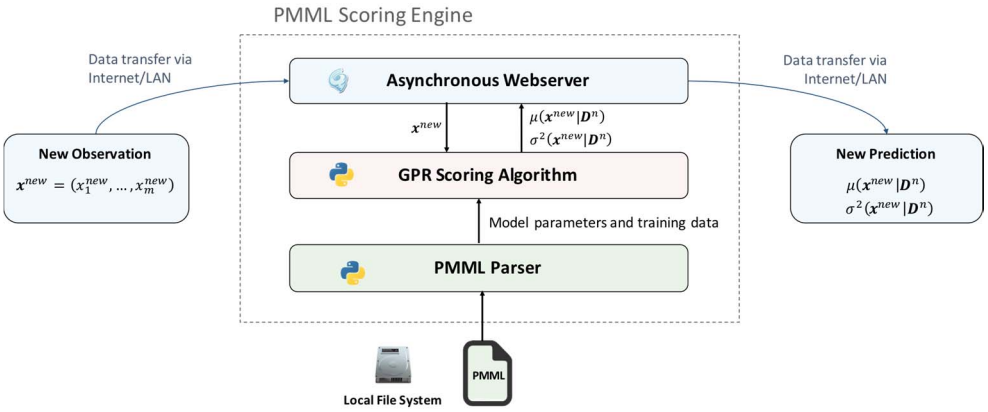

Figure 7. Architecture of the PMML scoring engine developed as part of this work.

new data observations to the scoring engine. This can be facilitated by either a Local Area Network (LAN) or an Internet connection.

The Python webserver Tornado [25], is used to connect the scoring engine to the local network. New data observations $\boldsymbol{x}^{new}$ are sent to the scoring engine in the JSON format via the HTTP protocol. The scoring engine computes the relevant scores and returns them with the HTTP response. Figure 8 provides an example of the JSON-encoded data that is transferred over the network.

V. Scoring Engine Performance

The performance of the scoring engine is critical to its value in a real manufacturing application. We choose to quantify the performance in terms of the scoring engine response time and request throughput. The scoring engine must be able to make fast predictions (response time) at a high rate (throughput), while maintaining very high accuracy.

In this section, we describe how the performance of the scoring engine is evaluated. We start by describing how the scoring engine is installed on a Raspberry Pi microcomputer, which represents the type of computing environment that could be embedded in a manufacturing device. We then install the scoring engine on a virtual server on the Google Compute Engine for comparison. Lastly, we comment on the performance of the two scoring engines.

A. PMML Scoring Engine on Raspberry Pi

Numerous authors have discussed how predictive models can be evaluated on an embedded system or programmable microcontroller. In this scenario, the scoring engine is installed on computing hardware which is embedded within, or attached to the manufacturing device.

Due to space and cost restrictions the compute power and memory capacity of the microcontroller is often restricted. To simulate this scenario we installed the PMML scoring engine on a Raspberry Pi 3 Microcomputer [26] running the Ubuntu 16.04 operating system. The Raspberry Pi microcomputer has a 1.2GHz quad core central processing unit (CPU) and 1GB random access memory (RAM). A desktop computer is connected to the microcomputer board via a Wireless Local Area Network (WLAN) connection, to simulate the data stream produced by a manufacturing machine. An additional heat sink was fitted to the CPU to prevent it from overheating, as shown in Figure 9.

```
{
    "xnew": [202.891712,
             2998.128423,
             1,
             2,
             1]
}
```

```
{
    "result": {
        "mu": 12.3939534,
        "sd": 1.43534623
    }
}
```

Figure 8. New observation encoded in the JSON format (left) and resulting prediction also encoded in the JSON format (right). Both the observation and the prediction are encoded and sent over the network in this format.

B. PMML Scoring Engine on Google Compute Engine

Cloud infrastructure services, such as the Google Compute Engine, provide significant amount of compute power for a relatively low cost. These services provide *virtual machines* for a fixed hourly cost. Virtual machines allow multiple users to share the computing resources within a data center, while ensuring that resources are shared securely.

The PMML scoring engine was installed on a Google Compute Engine virtual machine. The virtual machine was assigned two virtual processors (vCPUs) and 1.8 GB of RAM. A desktop computer was connected to the virtual machine over the Internet, and used to simulate the data stream produced by a manufacturing machine.

C. Scoring Engine Accuracy and Performance Tests

The performance of the scoring engine is evaluated by load testing it with real manufacturing data. In a load test, new observations are sent to the scoring engine at a predefined rate. The new observations are sent from a desktop computer referred to as the *client*. The client represents an operational manufacturing device, in a real manufacturing setting.

The rate of new observations is referred to as the *throughput*. The amount of time required for the scoring engine to compute a response is measured, averaged, and subsequently referred to as the *response time*. We define the response time as the total time taken to generate a prediction, from the point in time when the client starts sending the observation to the point when the prediction response is fully returned to the client.

The throughput is gradually increased at regular intervals, until the scoring engine is unable to respond in a reasonable amount of time. In this work, the throughput is increased at 60 second intervals. The initial throughput is chosen based on the complexity of the model.

The performance of scoring engine was measured on both the microcomputer and cloud-based platforms. Figure 10 demonstrates the prediction response time obtained from the energy consumption prediction model. Figure 11 demonstrates the response time for predictions made with the tool condition model.

Figure 9. Raspberry Pi 3 Microcomputer running Python GPR scoring engine on Ubuntu 16.04. An additional heat sink (blue) was fitted to the CPU to allow it to operate at maximum capacity for a long duration of time.

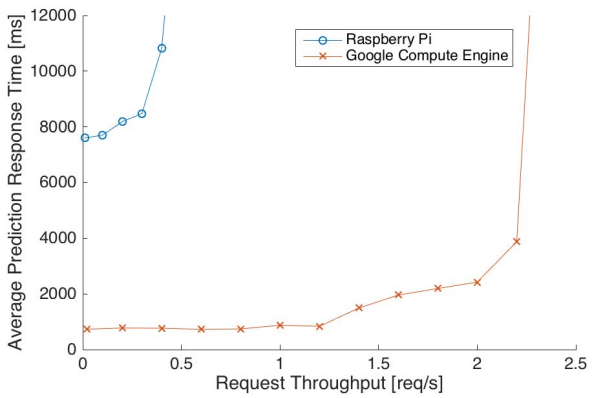

Figure 10. Response time of both scoring engines with the energy consumption model, including network latency.

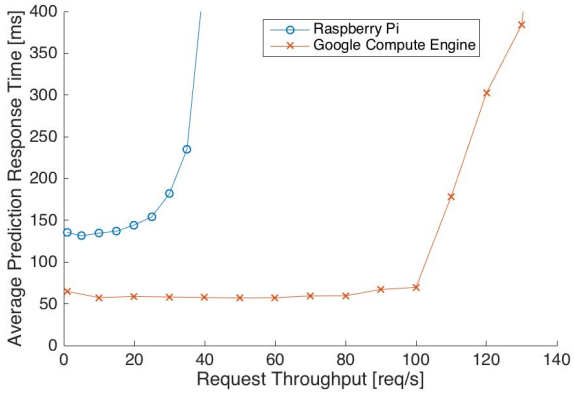

Figure 11. Response time of both scoring engines with the tool condition model, including network latency.

It can be seen that the performance of the scoring machine on the cloud server greatly exceeds that of the scoring engine on the microcomputer, with respect to both response time and throughput capacity. In both manufacturing examples, the time required to perform the linear algebra operations for the scoring process was non-trivial. The experimental results demonstrate that the response time and throughput capacity of a GPR scoring engine on a microcomputer board can be severely limited by the computational power of the device.

The response time remains reasonably constant as throughput increases, until the point where the computational demand of evaluating the model exceeds the computational capacity of the scoring engine hardware, as shown in Figure 10 and Figure 11. In all four cases the response time dramatically increases when the request throughput exceeds the capacity of the scoring engine.

VI. Discussion

In this work we demonstrated the performance of a GPR PMML scoring engine when installed on a Raspberry Pi microprocessor and a managed cloud service. The aim was to investigate the advantages and limitations of each strategy, using two examples derived from real manufacturing problems.

Purpose-built scoring engines will become increasingly important in the smart manufacturing industry, especially as internet-connected manufacturing machines become more mainstream. The response time and throughput capacity of these scoring engines are critical to the adoption of predictive modeling in smart manufacturing. For many real-time applications, the response time of the scoring machine must be sufficiently low to avoid manufacturing devices becoming idle whilst waiting for feedback from the scoring engine.

Both cloud-based and embedded scoring engines have their own advantages and limitations. Firstly, if a hardware failure occurs on the cloud-based system the scoring engine can be copied to another virtual machine quickly and easily. In contrast, if a hardware failure occurs on an embedded scoring engine, the faulty hardware must be manually replaced.

An embedded microcomputer scoring engine will excel when network reliability is considered important. Embedded scoring engines do not rely on wireless communication protocols such as Wi-Fi or Bluetooth, so they are less prone to network issues.

Whilst the cloud-based scoring engine outperformed the microcomputer in this study, the minimum response time of a cloud-based scoring engine will always be limited by network latency. The average response time of a cloud-based scoring engine will never be lower than the average round-trip network latency between the manufacturing machine and the virtual server.

Large manufacturing operations will likely need to evaluate predictive models at much higher rates than demonstrated in this work. One way to increase the prediction rate is to run multiple PMML scoring engines in parallel. Future studies could investigate how virtual machines in the cloud could be used in parallel to achieve the high throughput required for a modern predictive-model driven manufacturing plant.

The GPR scoring algorithm is computationally demanding for datasets with a large number of training data points. A more efficient algorithm such as Sparse Gaussian Process Regression could provide faster predictions with a similar prediction accuracy [27].

In conclusion, the development of standards-compliant scoring engines is critical to the widespread adoption of predictive models for smart manufacturing. The throughput capacity and response time of such scoring engines is dependent on a number of factors, including the deployment strategy and the complexity of the predictive model.

Acknowledgment

The authors acknowledge the support by the Smart Manufacturing Systems Design and Analysis Program at the National Institute of Standards and Technology (NIST), Grant Numbers 70NANB12H225 and 70NANB12H273 awarded to University of California, Berkeley, and to Stanford University respectively. In addition, the authors appreciate the support of the Machine Tool Technologies

Research Foundation (MTTRF), System Insights and Infinite Uptime for the equipment used in this research. Last but not least, the authors would like to acknowledge the valuable contributions by the late Prof. David Dornfeld of UC Berkeley and his support and collaboration on this research.

DISCLAIMER

Certain commercial systems are identified in this paper. Such identification does not imply recommendation or endorsement by NIST; nor does it imply that the products identified are necessarily the best available for the purpose. Further, any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NIST or any other supporting U.S. government or corporate organizations.

REFERENCES

[1] S. Wang, J. Wan, D. Li and C. Zhang, "Implementing smart factory of industrie 4.0: an outlook," *Int. J. Distrib. Sens. Netw.*, vol. 2016, p. 7, 2016.

[2] J. Davis *et al.*, "Smart Manufacturing," *Annu. Rev. Chem. Biomol. Eng.*, vol. 6, pp. 141–160, 2015.

[3] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola and J. M. Hellerstein, "Distributed GraphLab: a framework for machine learning and data mining in the cloud," *Proc. VLDB Endow.*, vol. 5, no. 8, pp. 716–727, 2012.

[4] J. Dean *et al.*, "Large Scale Distributed Deep Networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1223–1231.

[5] A. Guazzelli, M. Zeller, W.-C. Lin and G. Williams, "PMML: An open standard for sharing models," *R Journal*, vol. 1, no. 1, pp. 60–65, 2009.

[6] D. Gorea, "Dynamically integrating knowledge in applications. an online scoring engine architecture," *Proceedings of the International Conference on Development and Application Systems, 9th Edition, Suceava Romania*, 2008, vol. 3.

[7] J. Chaves, C. Curry, R. L. Grossman, D. Locke and S. Vejcik, "Augustus: the design and architecture of a PMML-based scoring engine," *Proceedings of the 4th international workshop on Data mining standards, services and platforms*, 2006, pp. 38–46.

[8] S. Krishnan and J. L. U. Gonzalez, "Google compute engine," in *Building Your Next Big Thing with Google Cloud Platform*, Springer, 2015, pp. 53–81.

[9] J. Park *et al.*, "A generalized data-driven energy prediction model with uncertainty for a milling machine tool using Gaussian Process," *ASME 2015 International Manufacturing Science and Engineering Conference*, 2015, pp. V002T05A010–V002T05A010.

[10] R. Bhinge *et al.*, "An intelligent machine monitoring system for energy prediction using a Gaussian Process regression," *2014 IEEE International Conference on Big Data*, 2014, pp. 978–986.

[11] C. E. Rasmussen and C.K. I. Williams, *Gaussian processes for machine learning*, MIT Press, 2006.

[12] "Documentation for GPML Matlab Code." [Online]. Available: http://www.gaussianprocess.org/gpml/code/matlab/doc/. [Accessed: 18-Aug-2016].

[13] U.S. Energy Information Administration, *U.S. EIA Monthly Energy Review*, May-2016.

[14] M. Helu, S. Robinson, R. Bhinge, T. Bänziger and D. Dornfeld, "Development of a machine tool platform to support data mining and statistical modeling of machining processes," *Proc MTTRF 2014 Annual Meeting, San Francisco, CA*, 2014.

[15] E. Kannatey-Asibu and D. A. Dornfeld, "A study of tool wear using statistical analysis of metal-cutting acoustic emission," *Wear*, vol. 76, no. 2, pp. 247 – 261, 1982.

[16] R. Silva, R. Reuben, K. Baker and S. Wilcox, "Tool wear monitoring of turning operations by neural network and expert system classification of a feature set generated from multiple sensors," *Mech. Syst. Signal Process.*, vol. 12, no. 2, pp. 319–332, 1998.

[17] A. Diniz, J. Liu and D. Dornfeld, "Correlating tool life, tool wear and surface roughness by monitoring acoustic emission in finish turning," *Wear*, vol. 152, no. 2, pp. 395–407, 1992.

[18] D. E. Dimla, "Sensor signals for tool-wear monitoring in metal cutting operations—a review of methods," *Int. J. Mach. Tools Manuf.*, vol. 40, no. 8, pp. 1073–1098, 2000.

[19] P. Welch, "The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms," *IEEE Trans. Audio Electroacoustics*, vol. 15, no. 2, pp. 70–73, Jun. 1967.

[20] Data Mining Group, "PMML 4.3 - General Structure." [Online]. Available: http://dmg.org/pmml/v4-3/GeneralStructure.html.

[21] *lxml - XML and HTML with Python*. [Online]. Available: http://lxml.de. [Accessed: 28-October-2016].

[22] S. Van Der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy array: a structure for efficient numerical computation," *Comput. Sci. Eng.*, vol. 13, no. 2, pp. 22–30, 2011.

[23] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[24] E. Anderson *et al.*, *LAPACK Users' Guide*, Third. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.

[25] M. Dory, A. Parrish, and B. Berg, *Introduction to Tornado*. O'Reilly Media, Inc., 2012.

[26] E. Upton and G. Halfacree, *Raspberry Pi User Guide*, 3rd Edition, Wiley, 2014.

[27] N. Lawrence, M. Seeger, and R. Herbrich, "Fast sparse Gaussian Process methods: The informative vector machine," *Proceedings of the 16th Annual Conference on Neural Information Processing Systems*, Vancouver, BC, 2003, pp. 609–616.