Fusion Engineering and Design 129 (2018) 179–182

Contents lists available at ScienceDirect

Fusion Engineering and Design

journal homepage: www.elsevier.com/locate/fusengdes

Real-time implementation with FPGA-based DAQ system of a probabilistic disruption predictor from scratch

J. Vega[a,*], M. Ruiz[b], E. Barrera[b], R. Castro[a], G.A. Rattá[a], S. Dormido-Canto[c], A. Murari[d], E. Bernal[b], JET Contributors[1,2]

[a] Laboratorio Nacional de Fusión, CIEMAT, Avda, Complutense, 40, Madrid, 28040, Spain
[b] Grupo de Investigación en Instrumentación y Acústica Aplicada: Universidad Politécnica de Madrid, Madrid, Spain
[c] Departamento de Informática y Automática, UNED, Madrid, Spain
[d] Consorzio RFX, Corso Stati Uniti 4, Padova, 35127, Italy

ARTICLE INFO

Keywords:
Disruption prediction
Real-time
Machine learning
Venn predictors
Data acquisition
Fpga

ABSTRACT

Real-time (RT) disruption prediction (DP) is essential to trigger mitigation actions that avoid irreversible damage to the devices. This paper deals with disruption mitigation alarms and performs the RT implementation of a probabilistic predictor. The RT implementation has been carried out with a fast controller with DAQ FPGA-based data acquisition devices corresponding to ITER catalogue (in particular, a reconfigurable Input/Output platform has been used). Up to three input signals have been used and relevant information for the prediction is extracted from the temporal and the frequency domains. The signals are read from the JET database. Then D/A conversions are carried out and used as inputs to the real time system. In this way, the whole process of digitization, data analysis and prediction is performed. The computation time for each prediction takes less than 200 μs.

1. Introduction

At present, important efforts to develop physics models for disruption prediction are being carried out. However, so far, they do not achieve success rates close to 100%. Instead, data-driven models can be used. The objective is to build classifiers to distinguish between disruptive and non-disruptive behaviours. The main drawback for DP is the need of large training sets to generate reliable classifiers. For example, JET APODIS predictor was trained with 7648 non-disruptive discharges and 521 unintentional disruptions [1]. Of course, ITER or DEMO cannot wait for such a number of discharges to have reliable predictions.

This article performs a specific implementation of the probabilistic *predictor from scratch* described in [2]. The name *predictor from scratch* comes from the fact that the learning process starts with only 1 disruptive discharge and at least 1 non-disruptive discharge. Its specific implementation uses an adaptive Venn predictor [3] that requires very few training examples and shows a high learning rate. The objective of the paper is not to analyse the predictor results but the real-time requirements to perform predictions in an FPGA-based data acquisition card.

Section 2 describes the steps to make Venn predictions, where these steps are the ones implemented in the FPGA. Section 3 shows the general architecture of the predictor. Section 4 details the implementation in an FPGA-based system that meets the requirements established in [4].

2. Venn predictors and a toy example

A general Venn predictor [3] requires a training set $\{z_1, \ldots, z_n\}$ that verifies the assumption of independent and identically distributed data. Each z_i is a pair $(\mathbf{x}_i, y_i)$ where $\mathbf{x}_i$ is a feature vector (whose components are characteristics of distinctive nature to distinguish between vectors) and $y_i \in \{Y_1, \ldots, Y_M\}$ is the label of the class to which $\mathbf{x}_i$ belongs to. The objective of a Venn predictor is the estimation of the probability that a new feature vector $\mathbf{x}$ belongs to one class from the possible M classes.

The Venn prediction framework assigns each of the possible classifications Y_j to $\mathbf{x}$ and divides all examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n), (\mathbf{x}, y)\}$ into a number of categories by means of a taxonomy function. This taxonomy function assigns to each pair $(\mathbf{x}_i, y_i)$ its category τ_i from a finite set of categories.

The Venn predictor implemented here uses the nearest centroid

* Corresponding author.
E-mail address: jesus.vega@ciemat.es (J. Vega).
[1] EUROfusion Consortium, JET, Culham Science Centre, Abingdon, OX14 3DB, UK.
[2] The paper "Overview of the JET results in support to ITER" by X. Litaudon et al. has been published. The reference is: Nuclear Fusion 57 (2017) 102001 (41pp).

https://doi.org/10.1016/j.fusengdes.2018.02.071
Received 11 June 2017; Received in revised form 16 January 2018; Accepted 17 February 2018
Available online 07 March 2018
0920-3796/

Fig. 1. Given the training set {(-1, *A*), (1, *A*), (4, *B*), (6, *B*)}, predict the class and probability interval of x = 5.5 with the nearest centroid taxonomy.

taxonomy (NCT) [5]. The NCT implies that the category of an element is equal to the label of the nearest centroid. Other types of taxonomies to implement Venn predictors can be logistic taxonomies [6] and taxonomies derived from multi-class SVM classifiers [7]. Reference [8] describes 5 different types of taxonomies where the underlying Venn predictors are based on neural networks.

A Venn predictor computes a probability matrix where the row j is the empirical probability distribution of the labels in a category τ that contains ($\mathbf{x}$, Y_j):

$$P_M = \begin{pmatrix} p^{Y_1}(Y_1) & p^{Y_1}(Y_2) & p^{Y_1}(Y_M) \\ p^{Y_2}(Y_1) & p^{Y_2}(Y_2) & p^{Y_2}(Y_M) \\ p^{Y_M}(Y_1) & p^{Y_M}(Y_2) & p^{Y_M}(Y_M) \end{pmatrix}$$

Once the above matrix has been formed, the Venn prediction process assigns as label of $\mathbf{x}$ the label of the column with the highest mean value. The prediction probability is the interval between the minimum of the winner column and the maximum of such column.

To fix ideas, let's consider the two-class problem of Fig. 1. It is a one-dimensional problem and the objective is to classify the point x = 5.5.

1st step: x = 5.5 is assumed to belong to class *A*.

Centroid(class A) = (–1 + 1 + 5.5)/3 = 1.83.

Centroid(class B) = (4 + 6)/2 = 5.

Table 1 summarizes the possible categories of the elements taking into account the NCT taxonomy.

Because the point to classify belongs to category τ_B, τ_B = {(4, *B*), (6, *B*), (5.5, *A*)}, the first row of matrix P_M is $P^A(A) = 1/3$, $P^A(B) = 2/3$.

2nd step: x = 5.5 is assumed to belong to class *B*.

Centroid(class A) = (–1 + 1)/2 = 0.

Centroid(class B) = (4 + 6 + 5.5)/3 = 5.17.

Table 2 summarizes the category of the elements.

Therefore, the second row of P_M is $P^B(A) = 0/3 = 0$ and $P^B(B) = 3/3 = 1$. In this way

$$P_M = \begin{pmatrix} 1/3 & 2/3 \\ 0 & 1 \end{pmatrix}$$

As the second column has a larger mean value, the Venn predictor predicts label *B* for x = 5.5 with a probability interval [2/3, 1].

3. General architecture of the disruption predictor

The high level predictor algorithm is described in Fig. 2. The block 'operation start' defines the starting point to collect data to develop the adaptive predictor. All signals required are stored from that moment until having one disruptive discharge and at least one non-disruptive discharge.

Table 1
categories with the assumption (5.5, *A*).

Element	Nearest centroid	Category
(-1, *A*)	(1,83, *A*)	τ_A
(1, *A*)	(1,83, *A*)	τ_A
(4, *B*)	(5, *B*)	τ_B
(6, *B*)	(5, *B*)	τ_B
(5.5, *A*)	(5, *B*)	τ_B

Table 2
categories with the assumption (5.5, *B*).

Element	Nearest centroid	Category
(-1, *A*)	(0, *A*)	τ_A
(1, *A*)	(0, *A*)	τ_A
(4, *B*)	(5.17, *B*)	τ_B
(6, *B*)	(5.17, *B*)	τ_B
(5.5, *B*)	(5.17, *B*)	τ_B

```
operation start
signal storage
wait for{
    one disruptive discharge
    one non-disruptive discharge
} end 'wait for'
first model creation{
    compute normalization factors
    signal normalization
    disruptive example
    non-disruptive example
} end 'first model creation'
wait for discharge {
    real-time prediction with last model created
    signal storage
    if missed alarm{
        new model creation {
            recompute normalization factors
            signal normalization
            add disruptive example
            add non-disruptive example
        } end 'new model creation'
    } end 'if missed alarm'
} end 'wait for discharge'
```

Fig. 2. predictor pseudo-code.

In the 'first model creation' block, a previous step is to normalise the signals to avoid a larger weight of quantities with larger absolute values (for instance, the plasma current in JET is $O(10^6)$ and the locked mode is $O(10^{-4})$). Quantities are normalised according to

$$Q_{norm}(t) = \frac{Q(t) - \min(Q)_{\text{from 'operationstart'}}}{\max(Q)_{\text{from 'operationstart'}} - \min(Q)_{\text{from 'operationstart'}}}$$

where max(·) and min(·) are the maximum and minimum values respectively. In this way, the variation range of the signals is [0,1].

At this point, it is important to describe the feature vectors of the predictor. According to the best results obtained in [2], three signals are used: plasma current (Ip), locked mode (LM) and internal inductance (LI). The signals are digitised at 1 ksample/s and are analysed in time windows 32 ms long. Each feature vector, $\mathbf{x}$, will have 4 components, two of them in the time domain of the signals and the others in the frequency domain:

$$\mathbf{x} = (std(|fft(Ip)|_+^*),\ mean(LM),\ std(|fft(LM)|_+^*),\ mean(LI)) \quad (1)$$