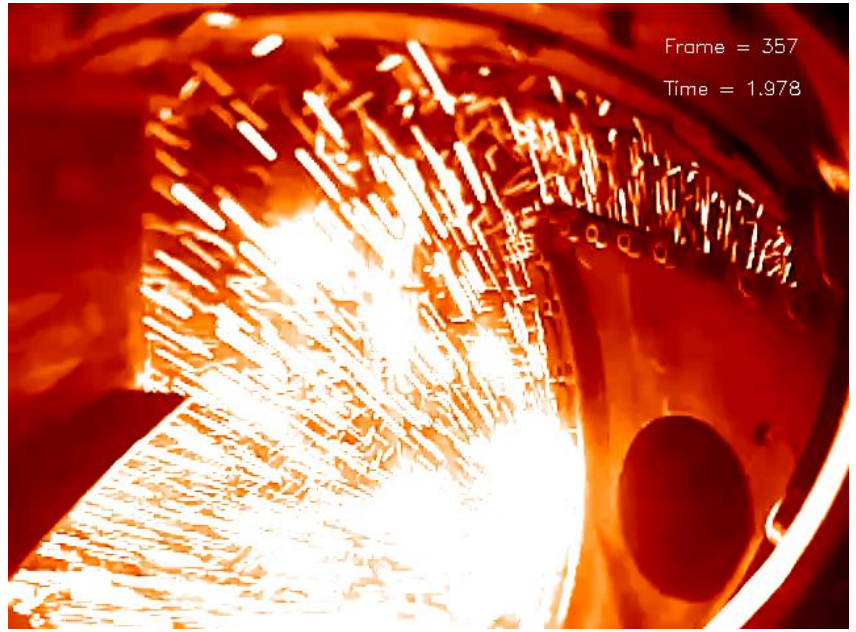
Progress on applying machine learning to disruption prediction

A summary of our group's work at the MIT Plasma Science & Fusion Center

PSFC
MIT Plasma Science & Fusion Center

R.S. Granetz, C. Rea, K. Montes,
J. Zhu, A. Tinguely

TSD workshop 2019
PPPL
2018/08/05-07



View from visible camera of disruption on Alcator C-Mod.

Introduction

There are a number of AI methods that have been applied to disruption prediction: neural networks (NN, RNN, CNN), support vector machines (SVM), random forests (RF), generative topographic maps (GTM), etc.

- All these methods require extensive databases of disruption-relevant signals, which we have compiled for C-Mod, DIII-D, EAST, and KSTAR.
- We desire a low rate of missed disruptions *and* a low rate of false positives

Most of our group's efforts have been on the training, optimization, and testing of random forests algorithms for C-Mod, DIII-D, and EAST ***for real time use***.

- We now have an RF predictor running in real time in the DIII-D plasma control system, and will have the same on EAST soon.

We have also begun working with recurrent neural networks (RNN), and comparing their performance to random forests.

A bit more information

Avoiding a disruption is preferable to having to mitigate a disruption. This requires:

- a sufficiently long warning time to allow the plasma control system to modify the discharge appropriately
- knowing which of the plasma input signals are most responsible for the prediction of an impending disruption

This is known as “interpretability” or “feature contribution analysis”. (In AI, the input signals are called “features”.)

- This is notoriously difficult for most AI methods
- *We need to do it in real time*

SQL databases established on different tokamaks

- Time series (0D) of 40-60 plasma parameters of thousands of disrupted and non-disrupted discharges for different devices.

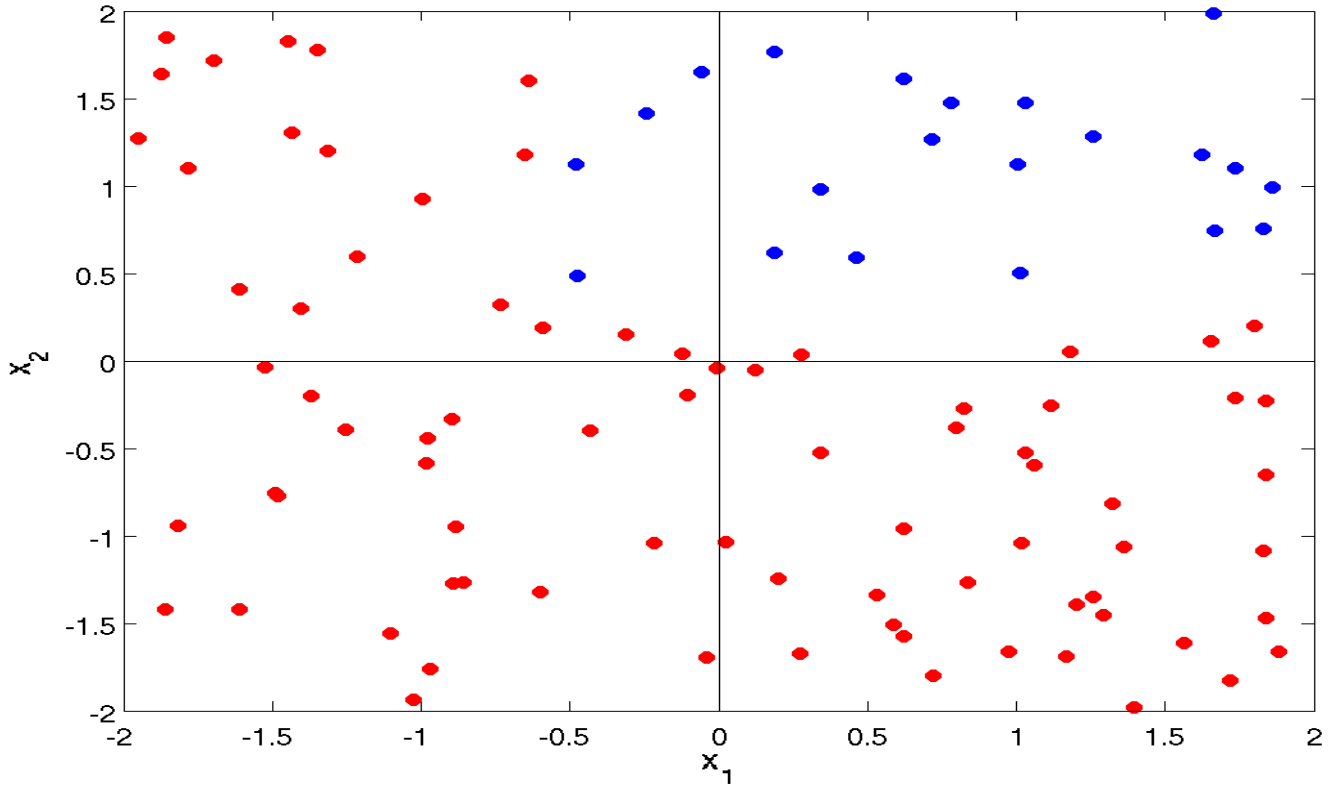
Device	Discharges	Samples/ Feature
C-Mod	5507	498,925
DIII-D	13245	3,018,096
EAST	18751	1,518,082
KSTAR	4219	773,083

Q: What is a ‘random forest’?

A: a collection of decision trees

Q: What is a ‘decision tree’ ?

Example of simple ‘training’ dataset

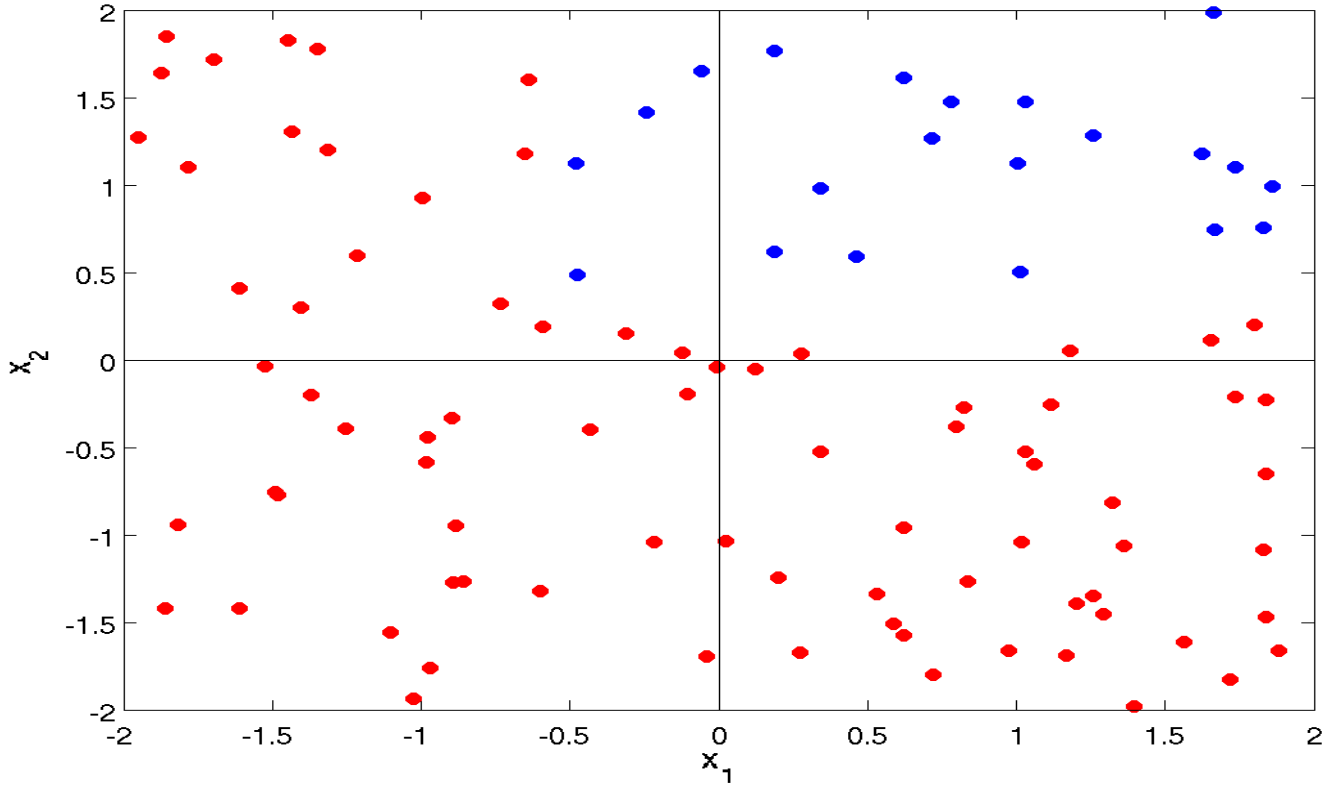


- Just 2 ‘physics’ parameters: x1, x2
  - Our disruption datasets have 40+ parameters: β_N, q_{95}, P_{rad}, …)
- 2 classes: red, blue (analogous to ‘stable’ or ‘close to disrupt’ classes)

Based on the training dataset, *can we develop a set of rules* about x_1 and x_2 that tell us when the data will be red, and when the data will be blue?

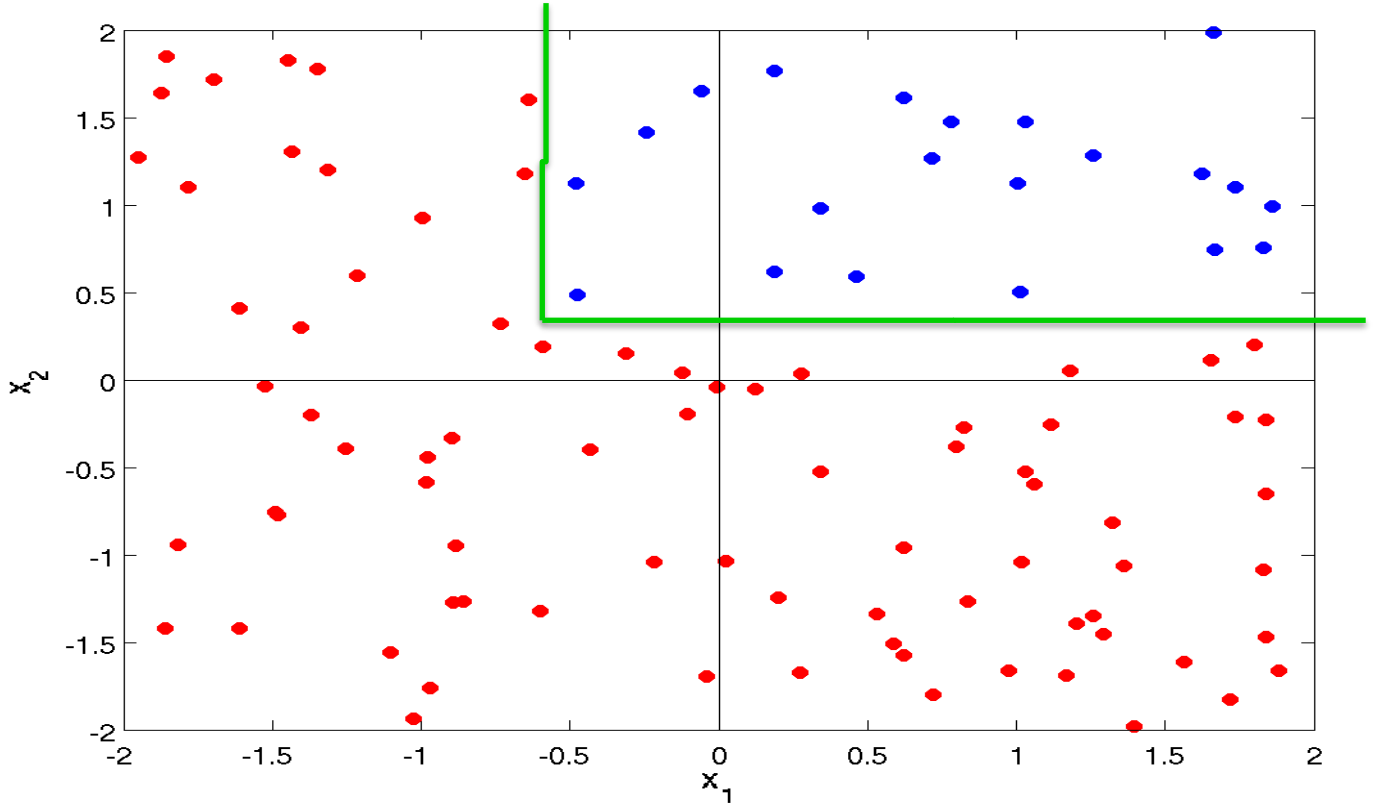
If yes, then if we are given new values for x_1 and x_2, we can apply those rules to predict whether the new data will be red or blue.

Example of simple ‘training’ dataset



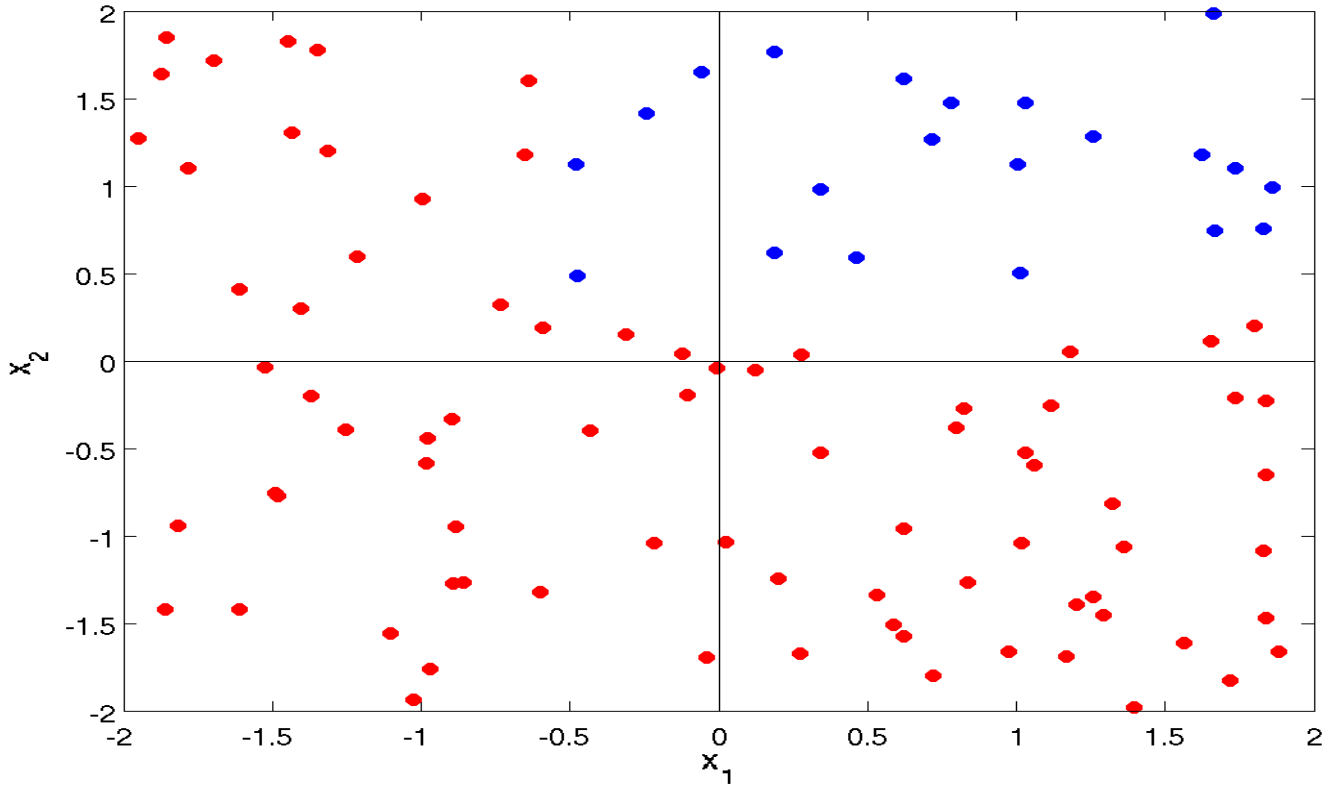
Humans: excellent at image processing and spatial recognition!

Example of simple ‘training’ dataset



Humans: excellent at image processing and spatial recognition!

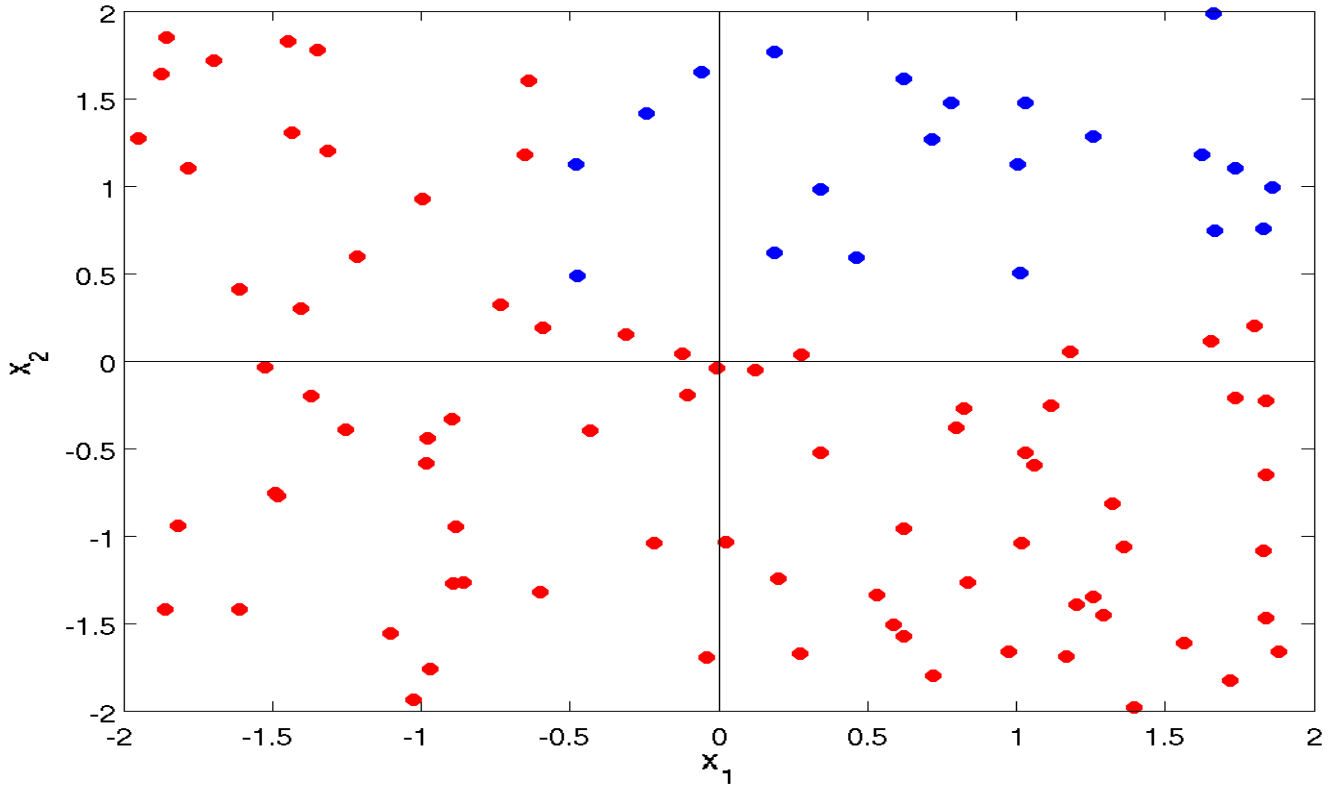
Example of simple ‘training’ dataset



Computers: quick and accurate at math, but terrible at spatial recognition!

Decision tree approach: Divide (x_1,x_2)-space into smaller and smaller sub-spaces, until each sub-space contains only red data or only blue data

Example of simple ‘training’ dataset

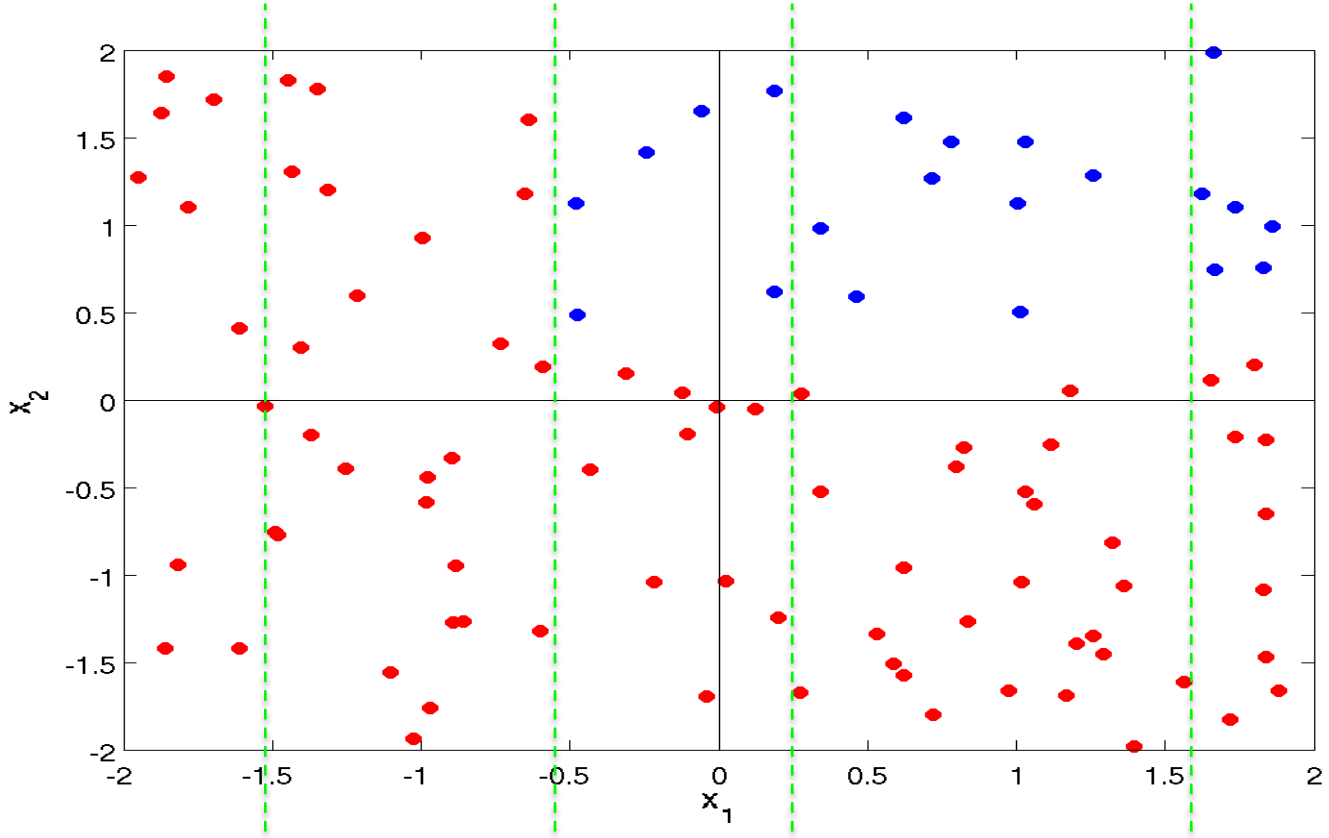


Computers: quick and accurate at math, but terrible at spatial recognition!

Decision tree approach: Divide (x_1,x_2)-space into smaller and smaller sub-spaces, until each sub-space contains only red data or only blue data

Example: start with x_1

Example of simple ‘training’ dataset



Computers: quick and accurate at math, but terrible at spatial recognition!

Decision tree approach: Divide (x_1,x_2)-space into smaller and smaller sub-spaces, until each sub-space contains only red data or only blue data

Example: start with x_1
Problem: at what value of x_1 should I divide the dataset?

There is a mathematical way to determine the best value to divide the dataset: Minimize the 'total impurity'

If node m is not pure, then the instances should be split to decrease impurity, and there are multiple possible attributes on which we can split. For a numeric attribute, multiple split positions are possible. Among all, we look for the split that minimizes impurity after the split because we want to generate the smallest tree. If the subsets after the split are closer to pure, fewer splits (if any) will be needed afterward. Of course this is locally optimal, and we have no guarantee of finding the smallest decision tree.

Let us say at node m, N_{mj} of N_m take branch j; these are $\mathbf{x}^t$ for which the test $f_m(\mathbf{x}^t)$ returns outcome j. For a discrete attribute with n values, there are n outcomes, and for a numeric attribute, there are two outcomes ($n = 2$), in either case satisfying $\sum_{j=1}^{n} N_{mj} = N_m$. N_{mj}^i of N_{mj} belong to class C_i: $\sum_{i=1}^{K} N_{mj}^i = N_{mj}$. Similarly, $\sum_{j=1}^{n} N_{mj}^i = N_m^i$.

Then given that at node m, the test returns outcome j, the estimate for the probability of class C_i is

$$\hat{P}(C_i|\mathbf{x}, m, j) \equiv p_{mj}^i = \frac{N_{mj}^i}{N_{mj}} \tag{9.7}$$

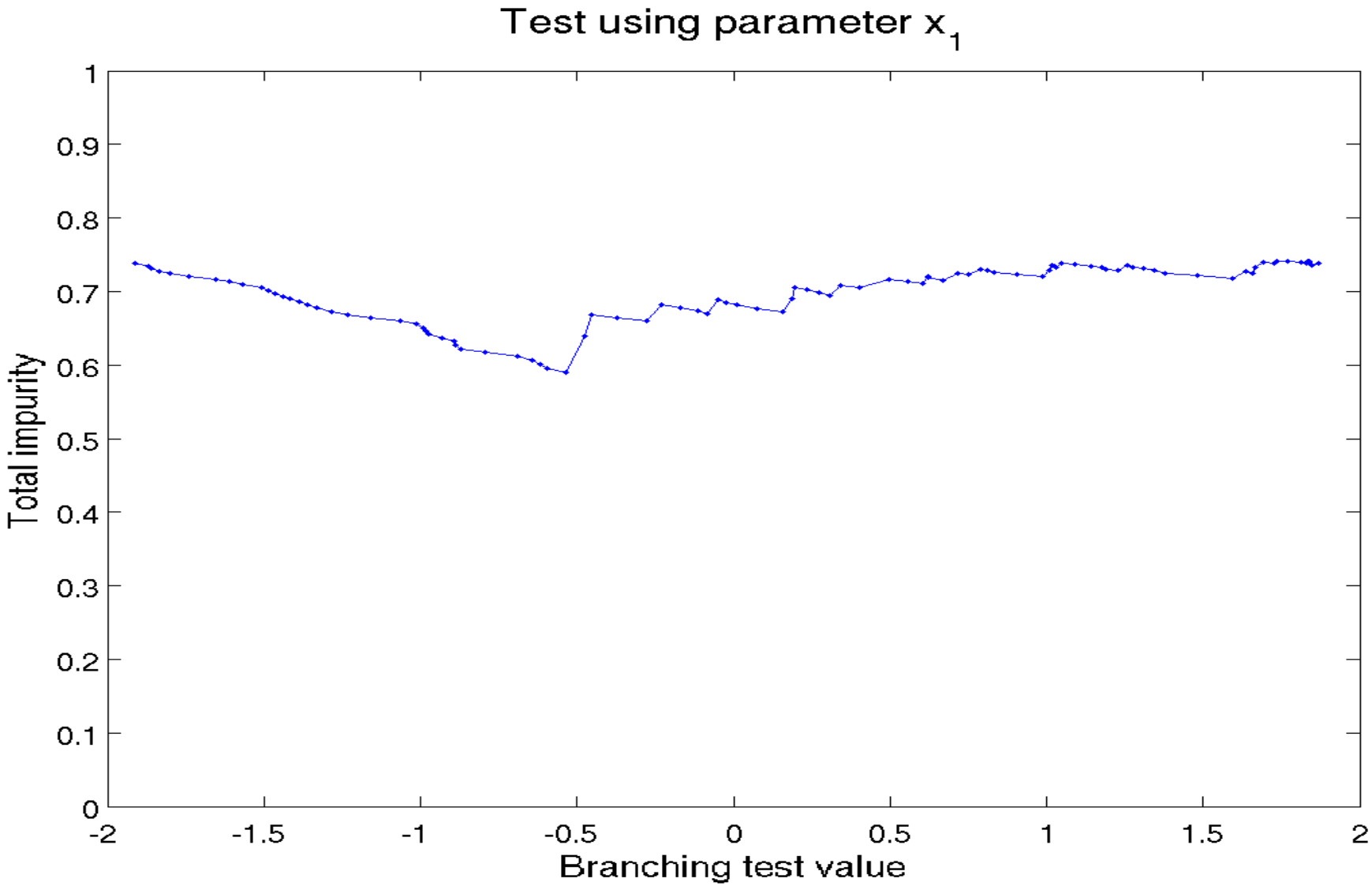
and the total impurity after the split is given as

$$\mathcal{I}'_m = -\sum_{j=1}^{n} \frac{N_{mj}}{N_m} \sum_{i=1}^{K} p_{mj}^i \log_2 p_{mj}^i \tag{9.8}$$

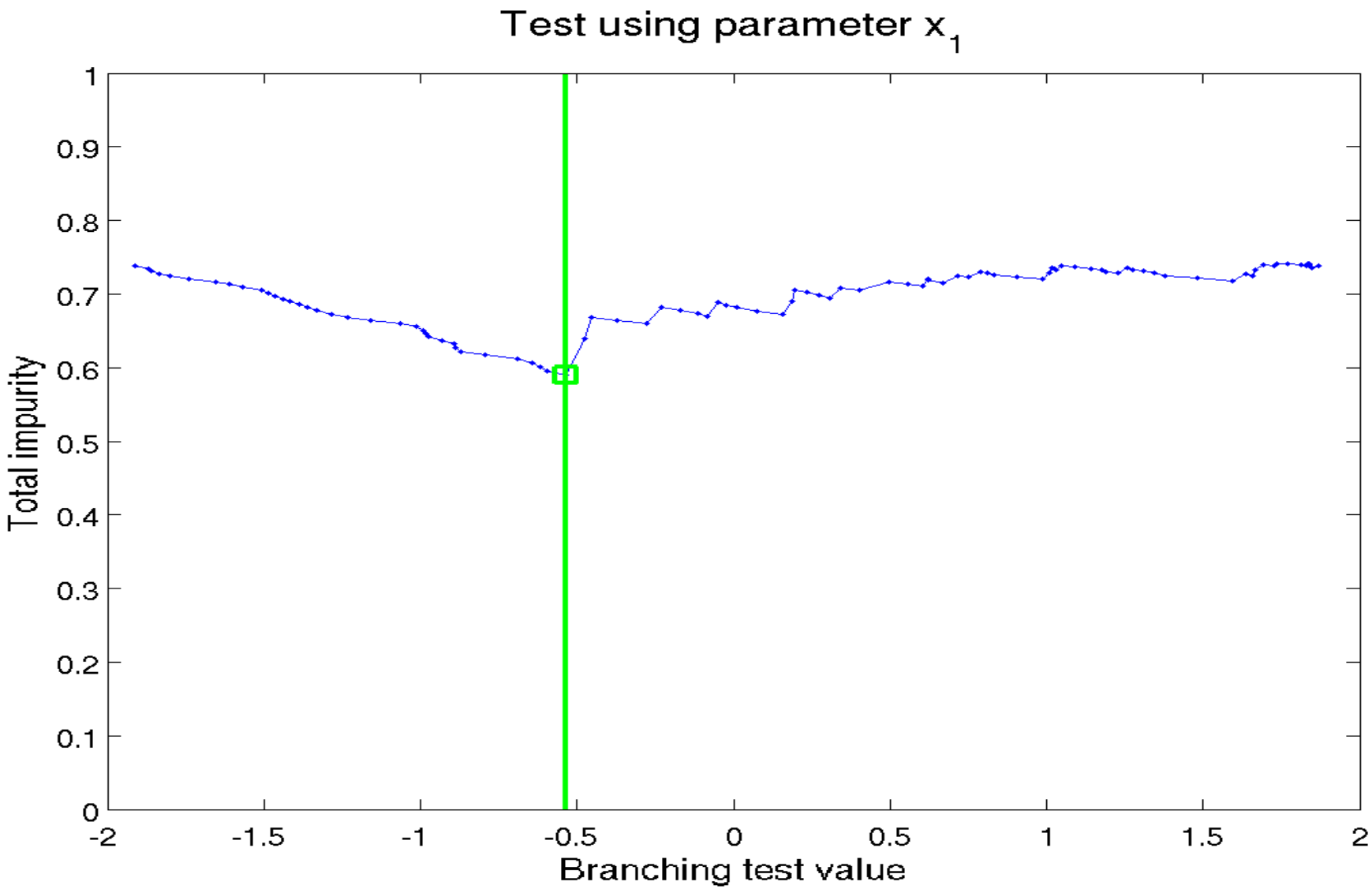
In the case of a numeric attribute, to be able to calculate p_{mj}^i using equation 9.1, we also need to know w_{m0} for that node. There are $N_m - 1$ possible w_{m0} between N_m data points: we do not need to test for all (possibly infinite) points; it is enough to test, for example, at halfway between points. Note also that the best split is always between adjacent points belonging to different classes. So we try them, and the best in terms of purity is taken for the purity of the attribute. In the case of a discrete attribute, no such iteration is necessary.

E. Alpaydin, "Introduction to Machine Learning", 2nd edition, MIT Press

There is a mathematical way to determine the best value to divide the dataset:
Minimize the ‘total impurity’

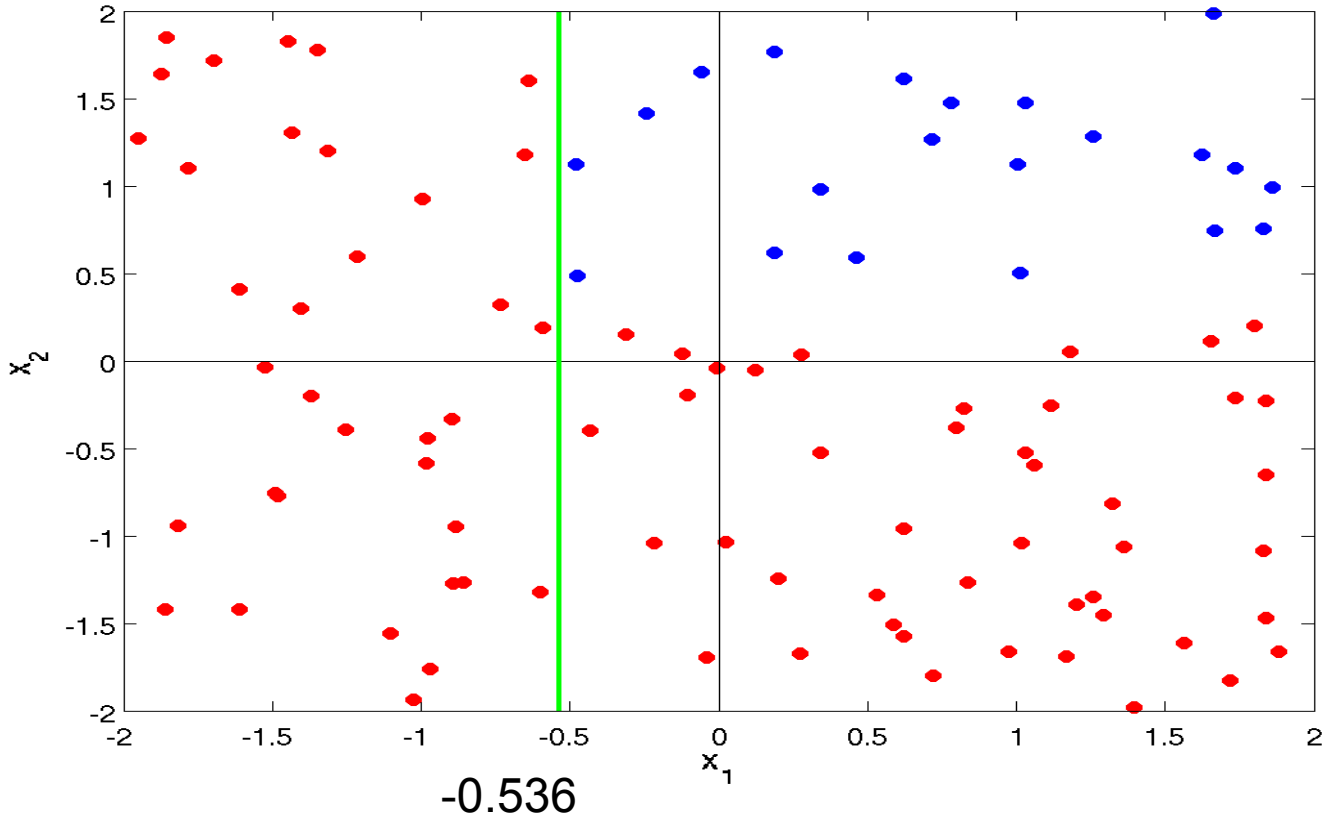


There is a mathematical way to determine the best value to divide the dataset:
Minimize the ‘total impurity’



-0.536

Example of simple ‘training’ dataset



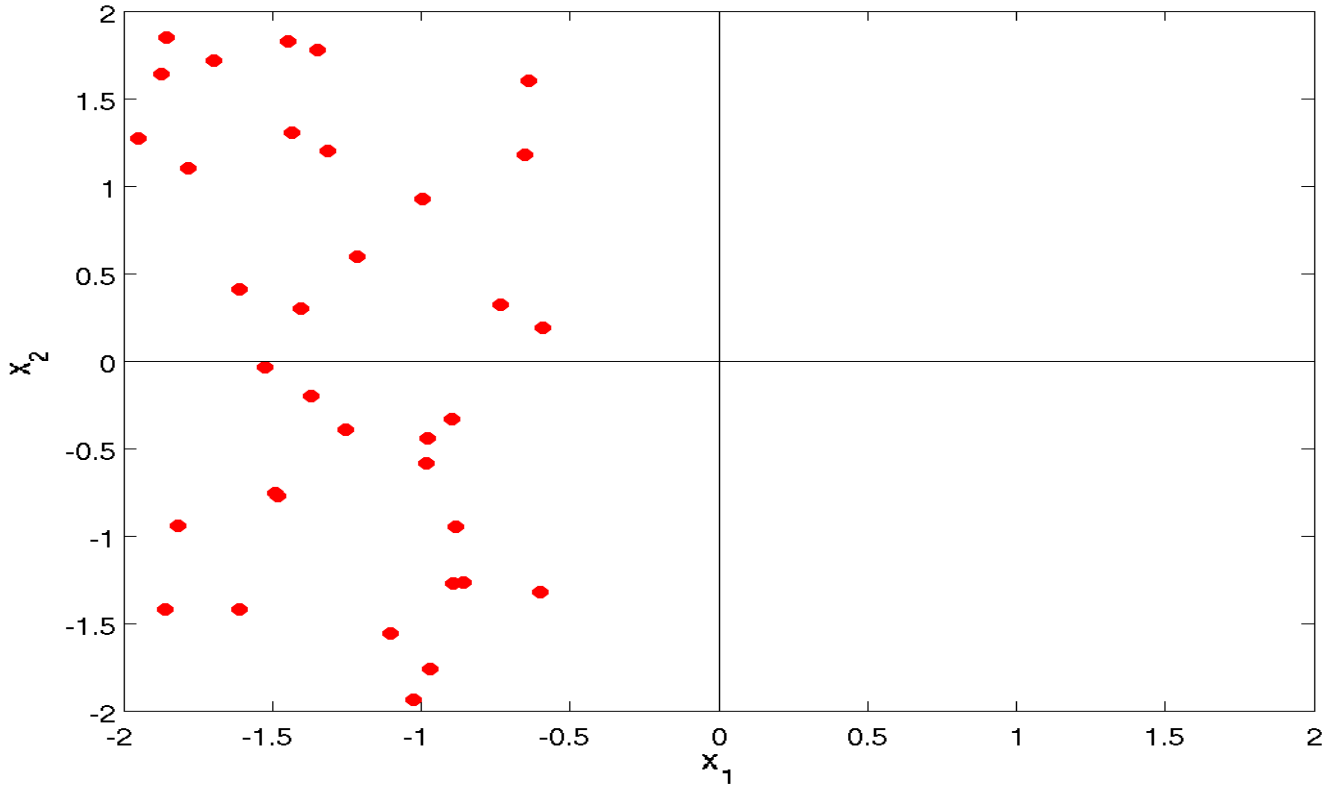
Computers: quick and accurate at math, but terrible at spatial recognition!

Decision tree approach: Divide (x_1,x_2)-space into smaller and smaller sub-spaces, until each sub-space contains only red data or only blue data

Example: start with x_1

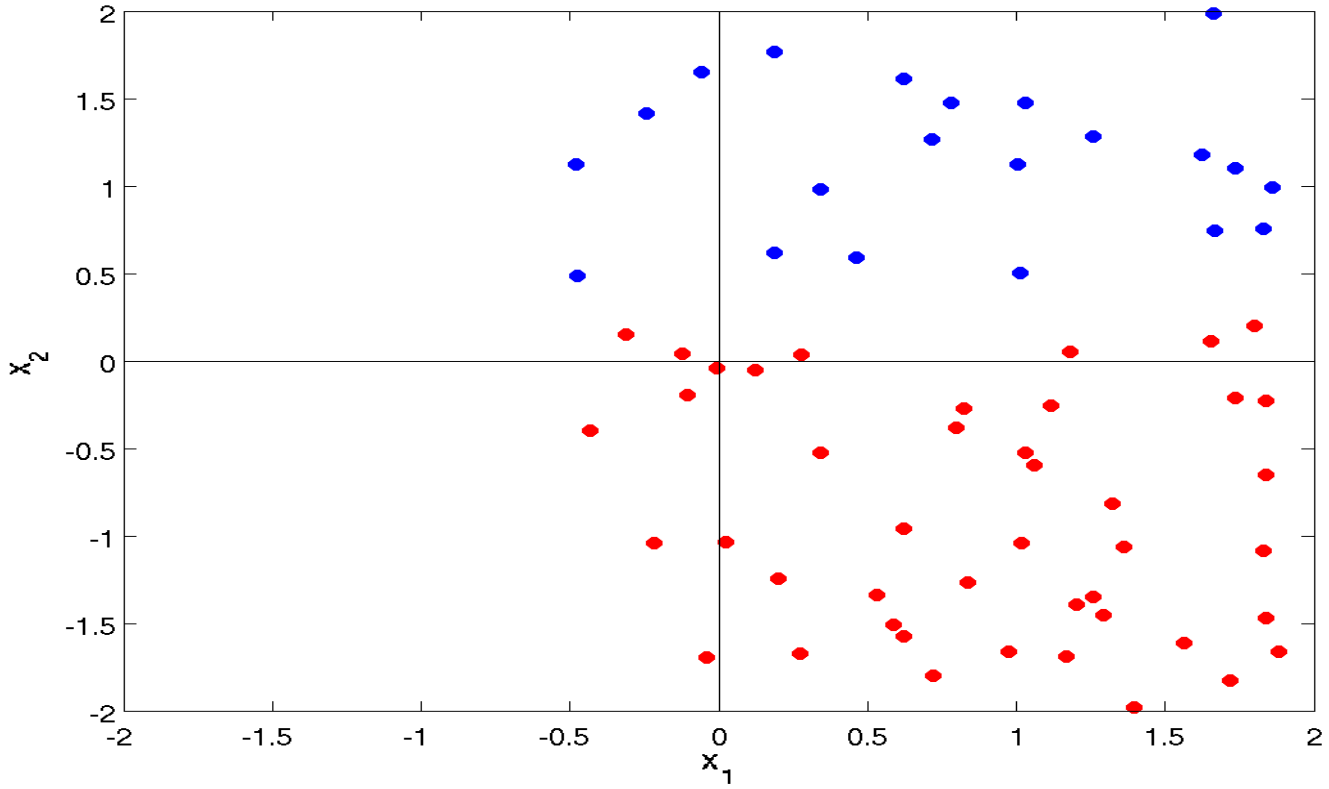
Divide dataset at x_1 value that minimizes the ‘total impurity’

Now we have two subdivided datasets:



This region of (x_1, x_2)-space only contains red data. We do not have to subdivide this region any more.

Now we have two subdivided datasets:

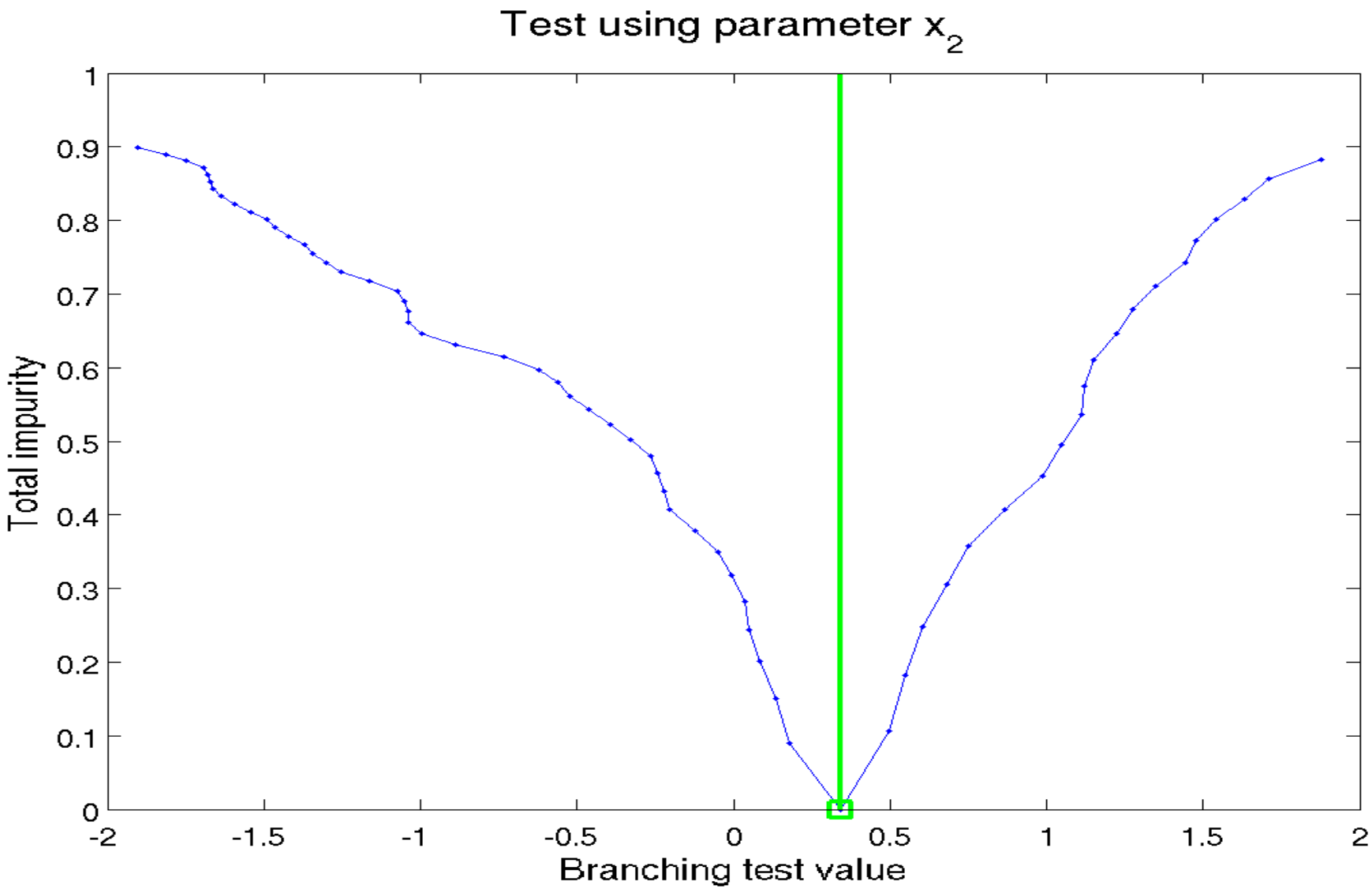


This region of (x_1, x_2)-space still contains both red and blue data. We need to subdivide this region again.

This time, use x_2 to subdivide the data*
At what value of x_2 should I divide the dataset?

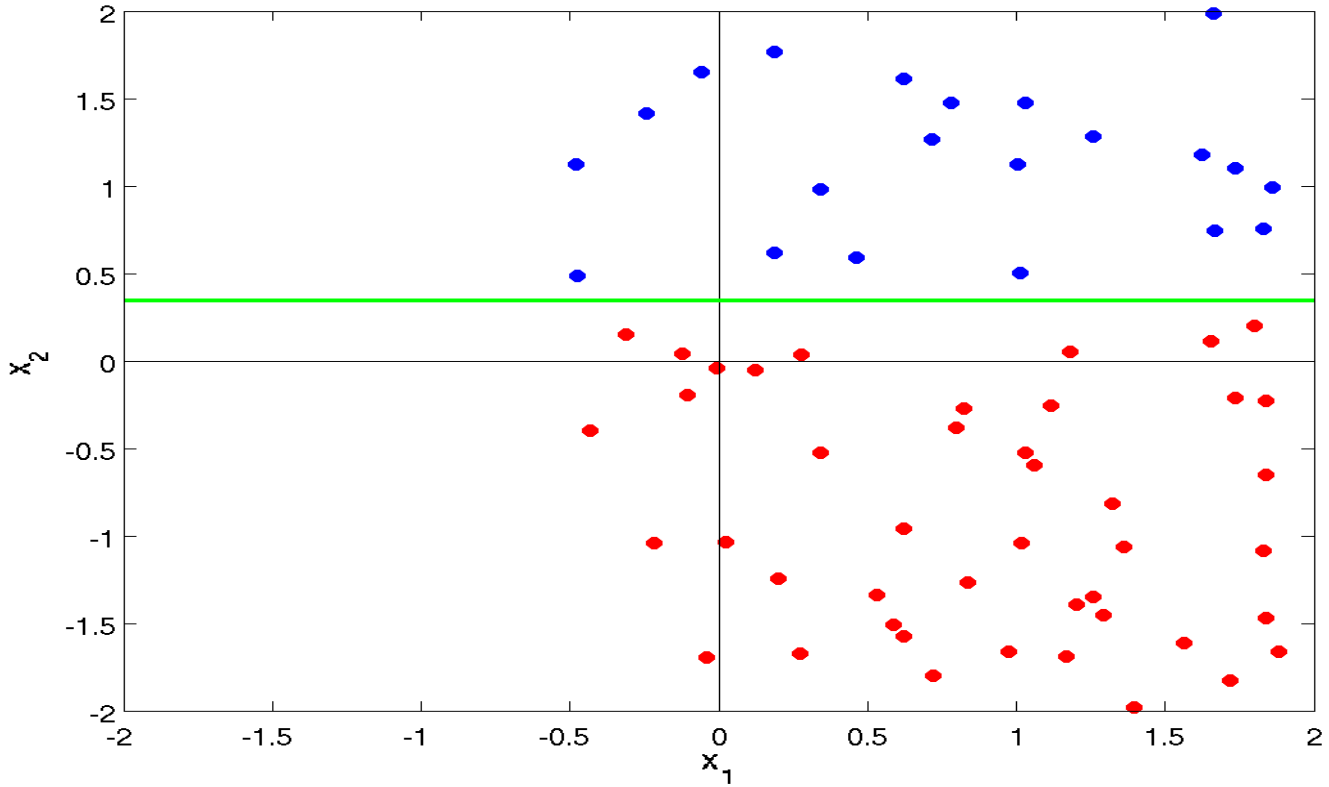
*The computer does not know which parameter is best to choose, especially when there are many parameters. Most decision tree algorithms pick the parameter randomly.

Find the value of x_2 that minimizes the ‘total impurity’



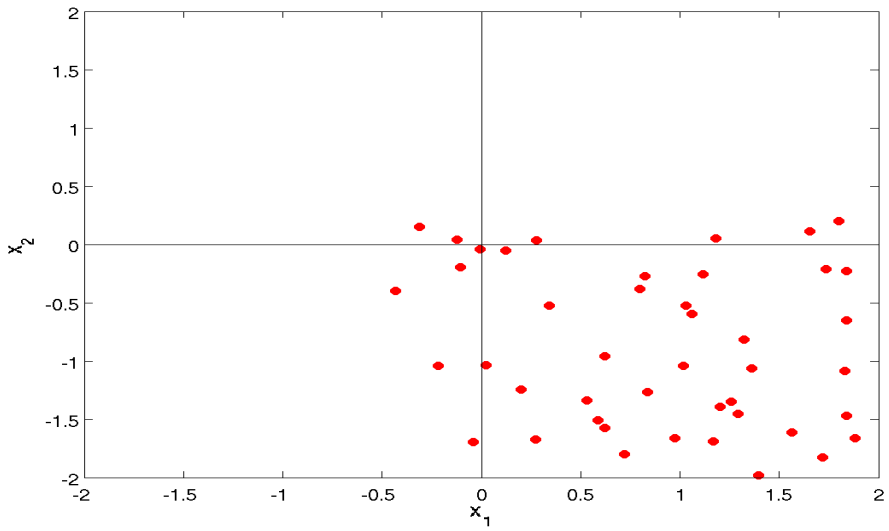
0.344

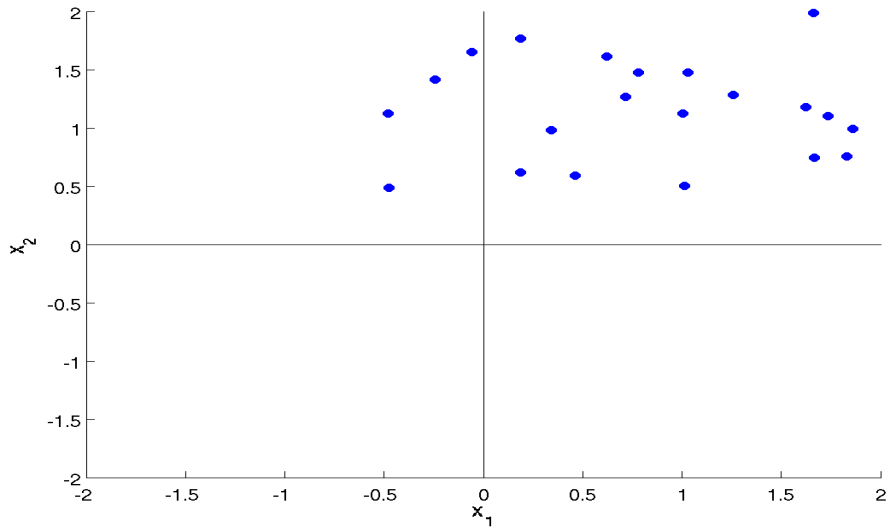
0.344



Divide dataset at x_2 value that minimizes the 'total impurity'

Now we have two more subspaces of the dataset:



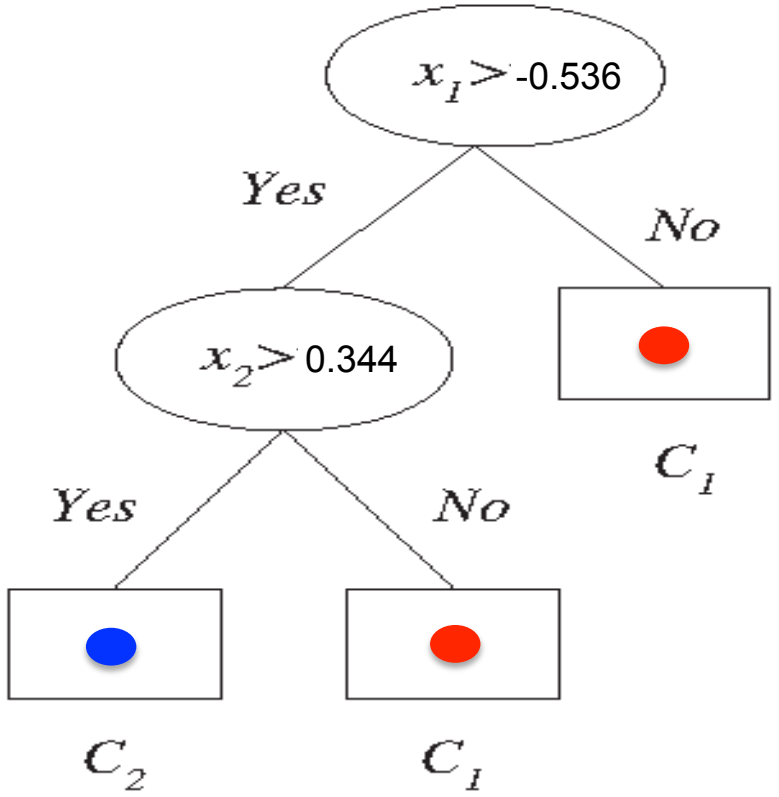


Both of these regions of the (x_1, x_2)-space are 'pure'. We do not need to subdivide any more. We have *learned* the rules (parameters and values) that define red regions and blue regions for our simple training set. For new values of (x_1, x_2), we can apply the same rules to predict whether they will be red or blue.

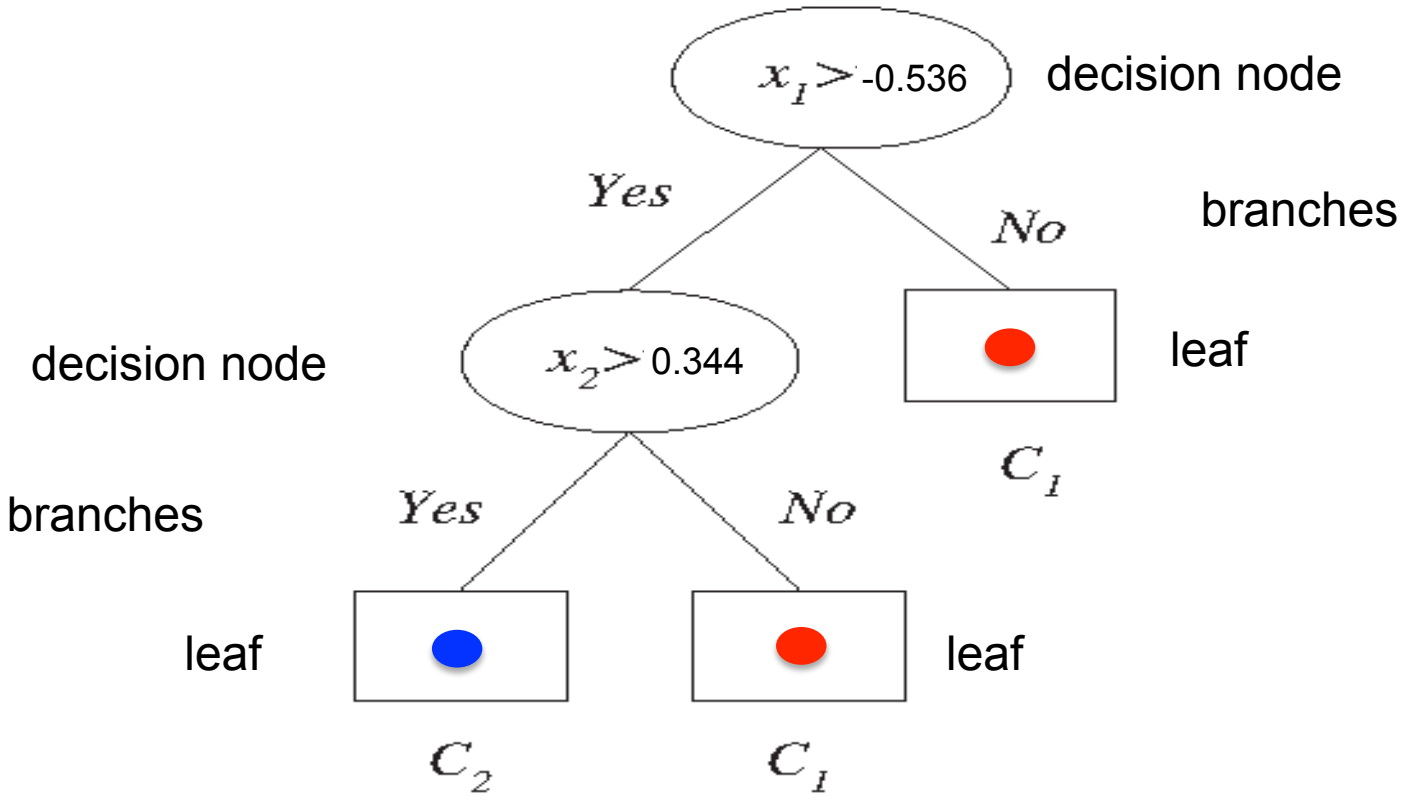
Rules: $x_1 > -0.536$ AND $x_2 > 0.344 \Longrightarrow$ blue

$x_1 < -0.536$ OR $x_2 < 0.344 \Longrightarrow$ red

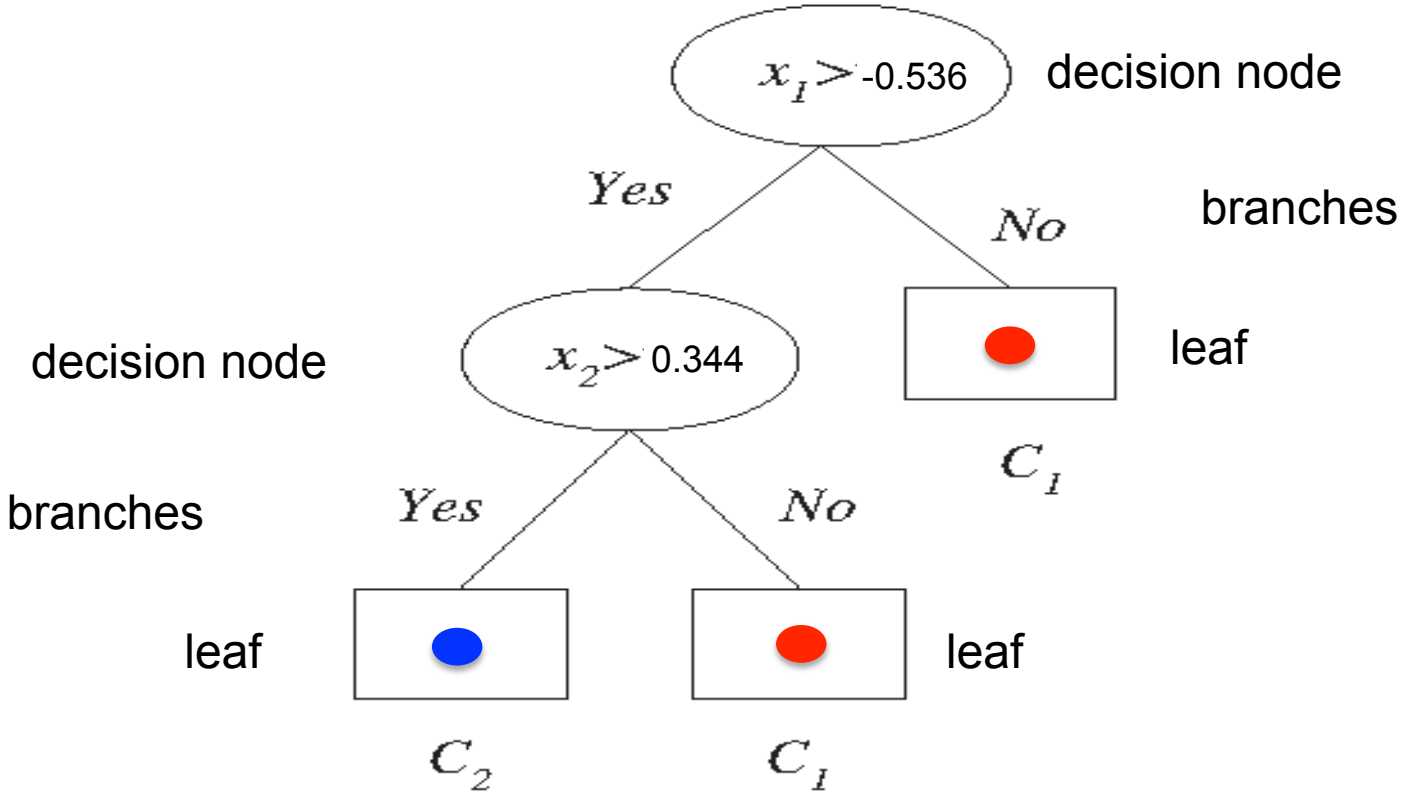
We can illustrate this process using a tree-like structure



We can illustrate this process using a tree-like structure

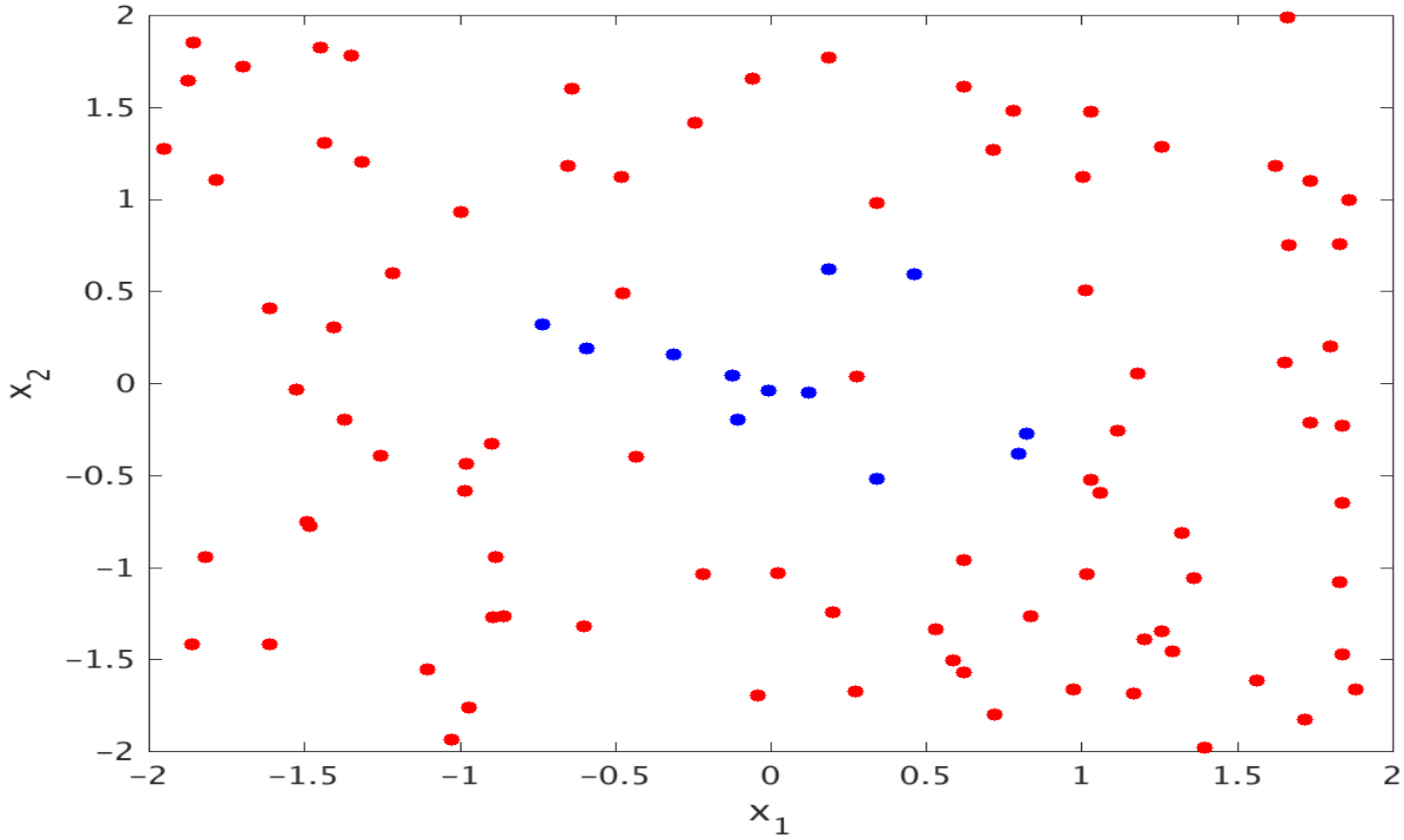


We can illustrate this process using a tree-like structure

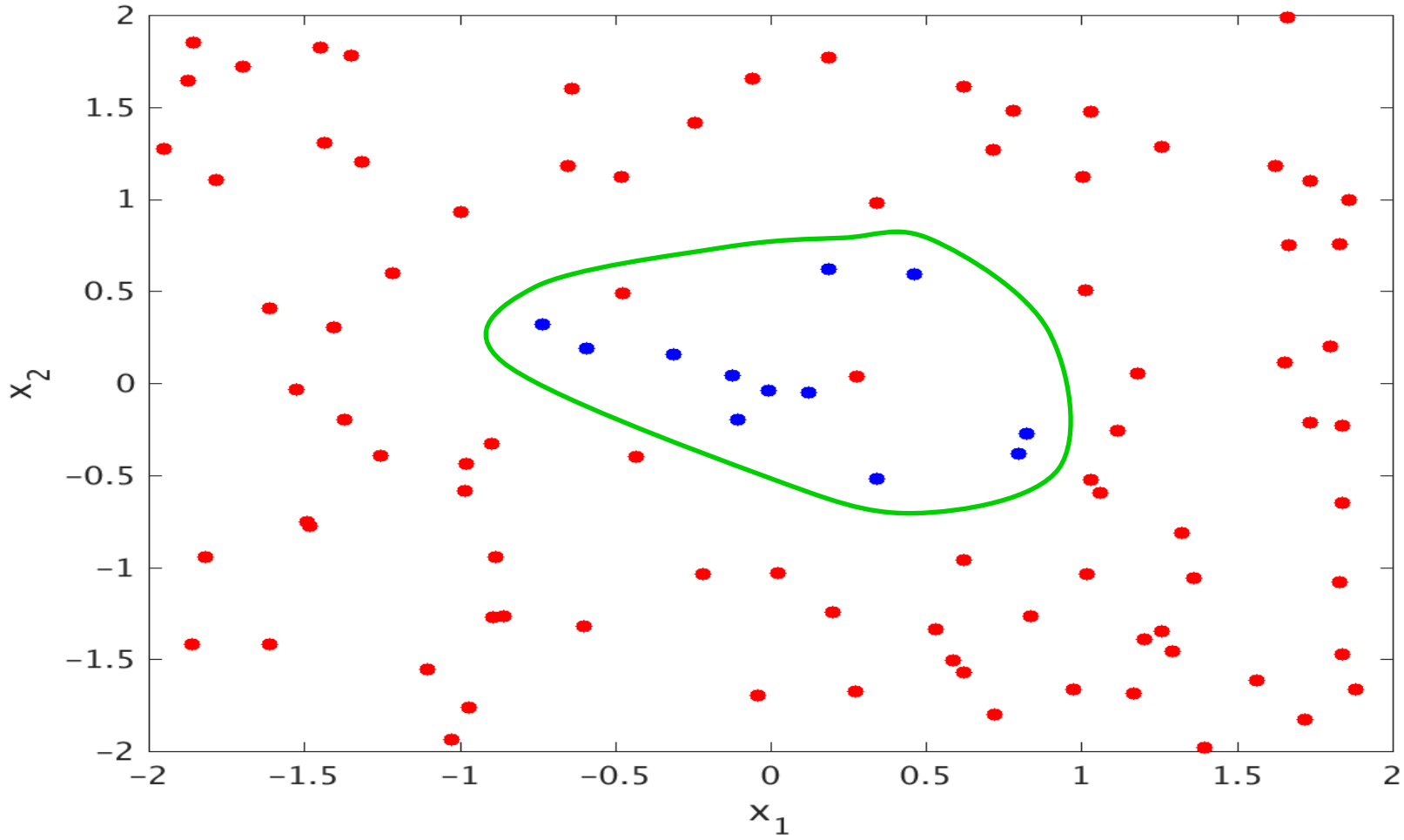


Our simple training dataset only requires a small decision tree. What if we had a more complicated dataset?

Example of a more complicated ‘training’ dataset

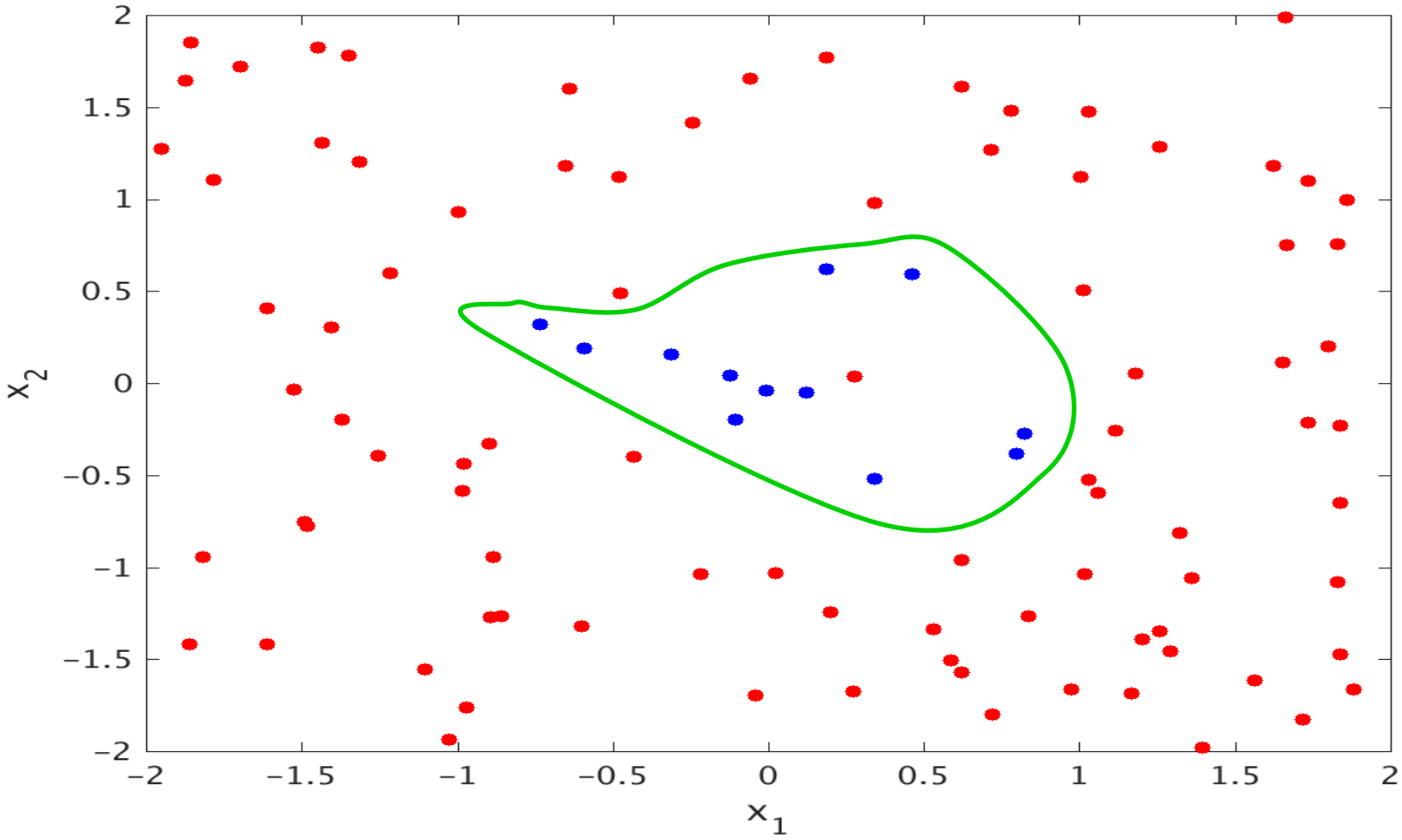


Example of a more complicated ‘training’ dataset



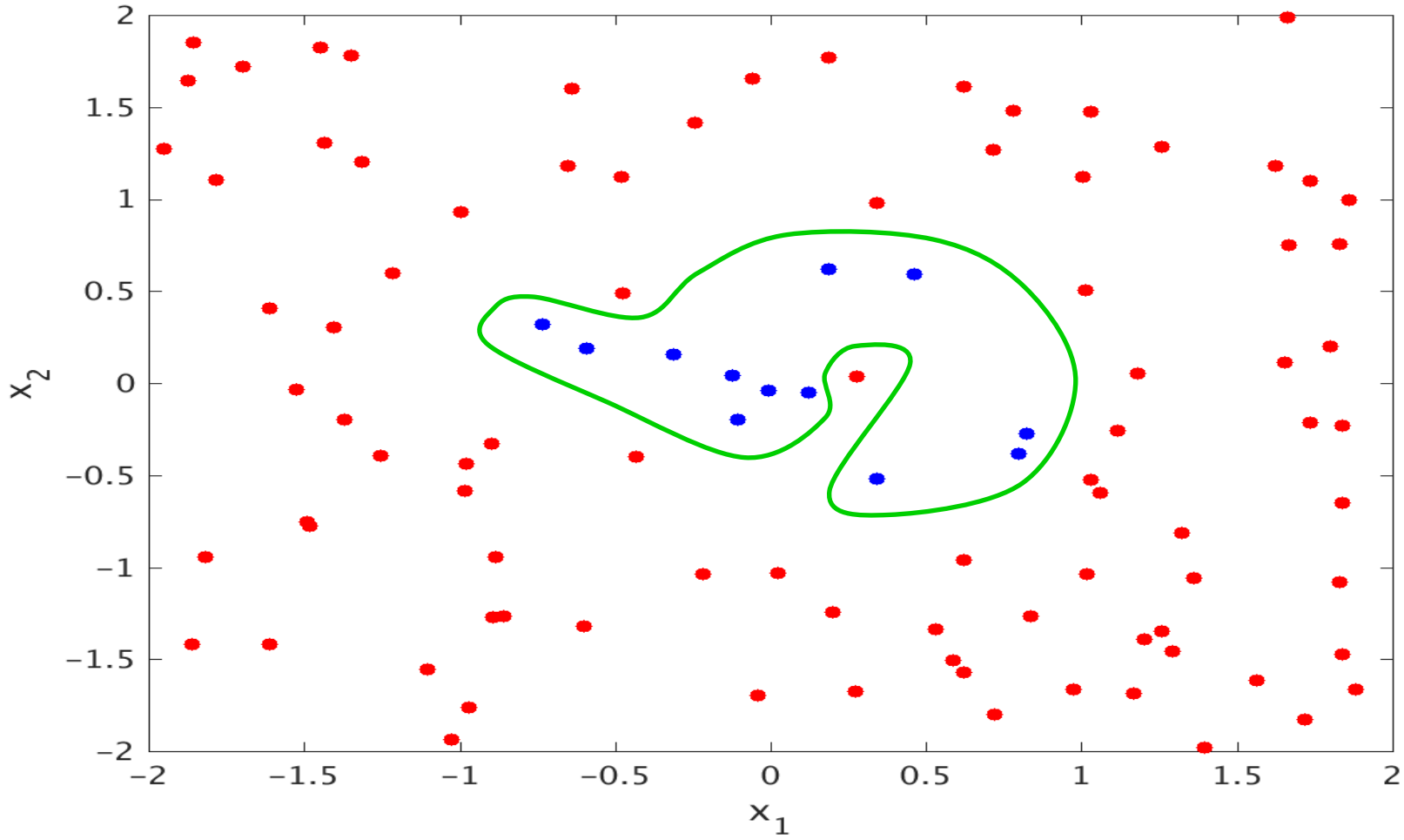
Possible pattern recognition by a human

Example of a more complicated ‘training’ dataset



or maybe this

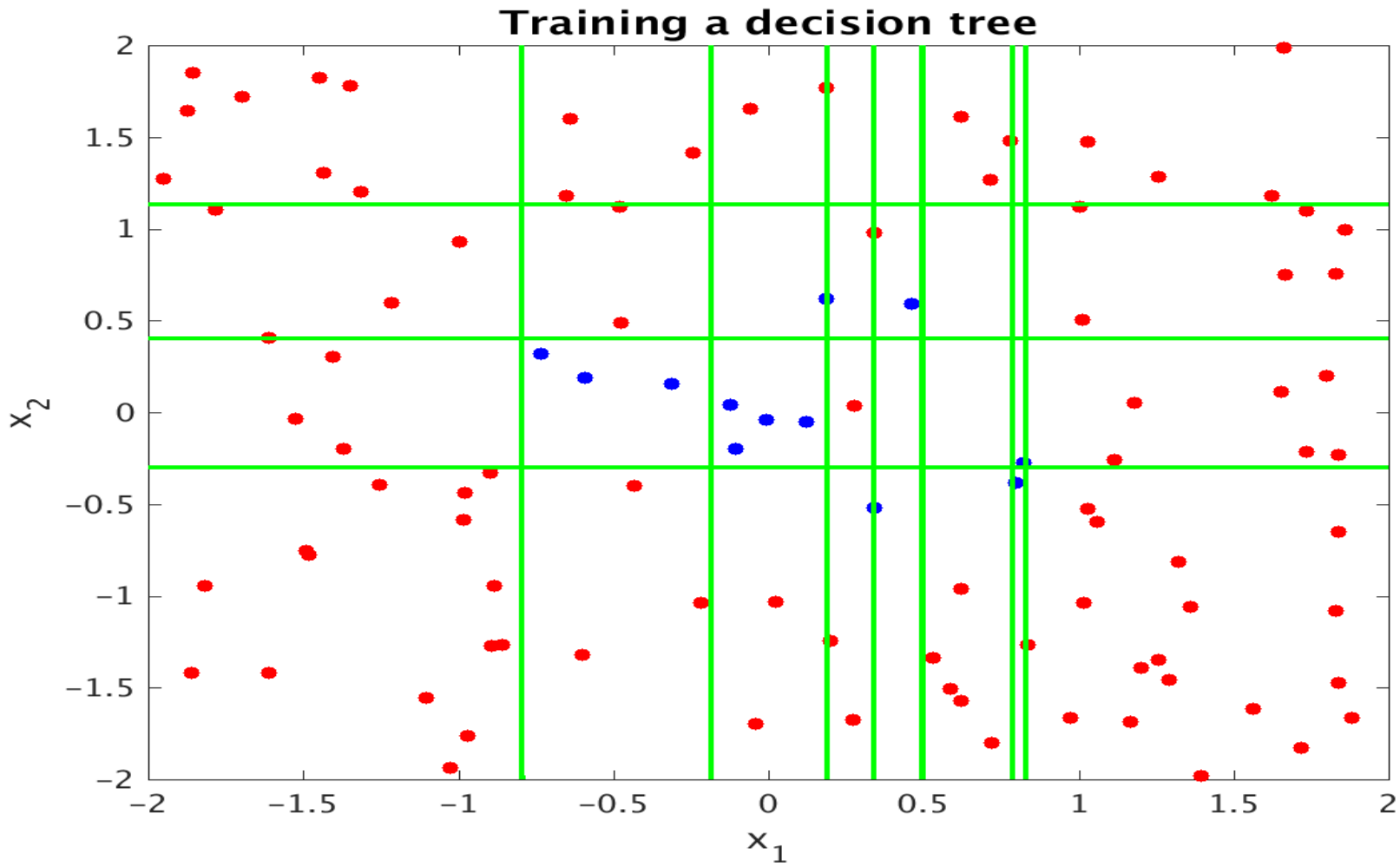
Example of a more complicated ‘training’ dataset



But probably not this

Overfitting?

Example of a more complicated ‘training’ dataset



A decision tree can always be fully “trained”. It may have many branches and many leaves. Some leaves may contain only a small number of points. This could be an example of “overfitting”.

Random Forests avoid overfitting

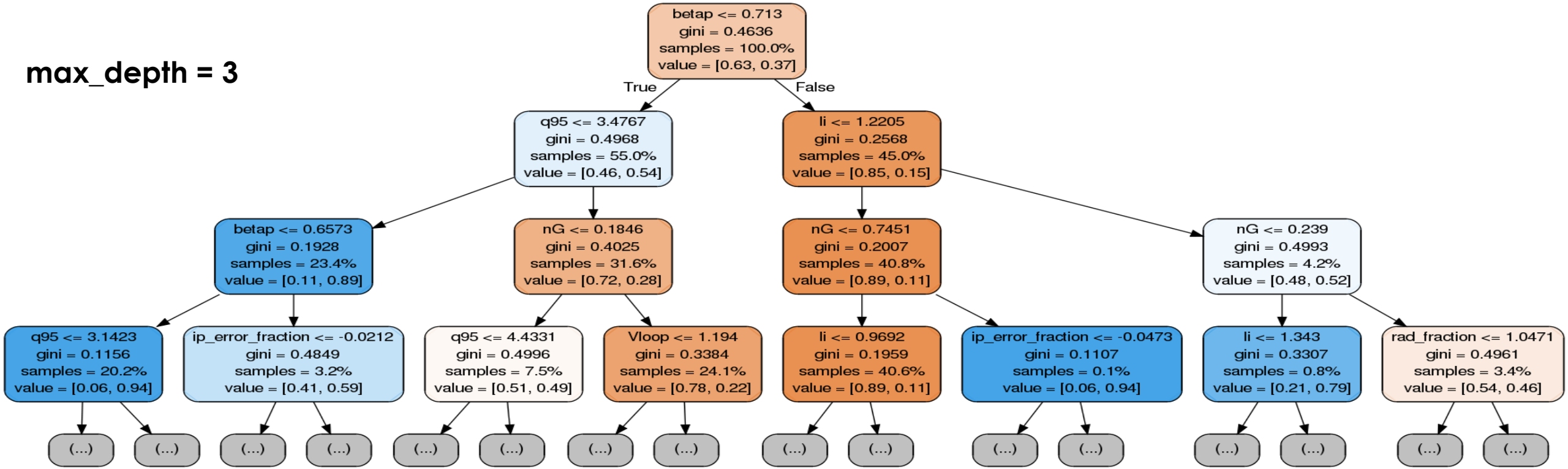
A fully grown decision tree (i.e. every branch ends in a pure leaf) is subject to overfitting the training data.

This problem can be overcome by growing many independent, uncorrelated decision trees, and averaging their test data predictions. This collection of independent trees is called a ***Random Forest***.

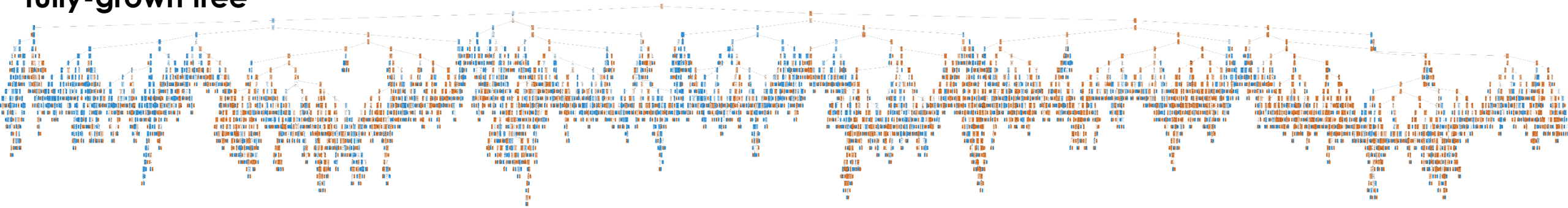
For our disruption prediction application, we selected the 10-12 most relevant plasma parameters. We grow (i.e. train) 500 independent trees. (It takes only 15-20 seconds to train using multiprocessing on 32 CPU cores.)

graphical depiction of a single tree in a Random Forests

max_depth = 3



fully-grown tree

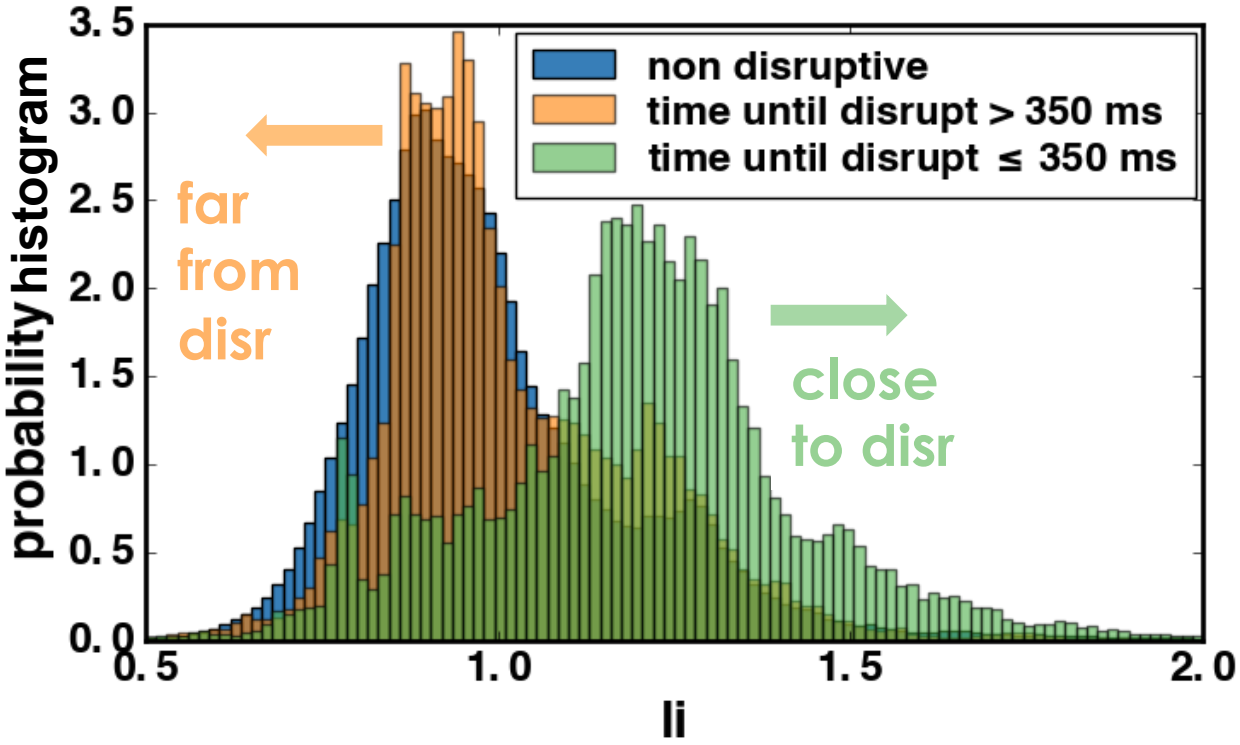


DPRF has run in DIII-D PCS during 2018 experimental campaign - τ_{class} set at 350 ms before the disruption

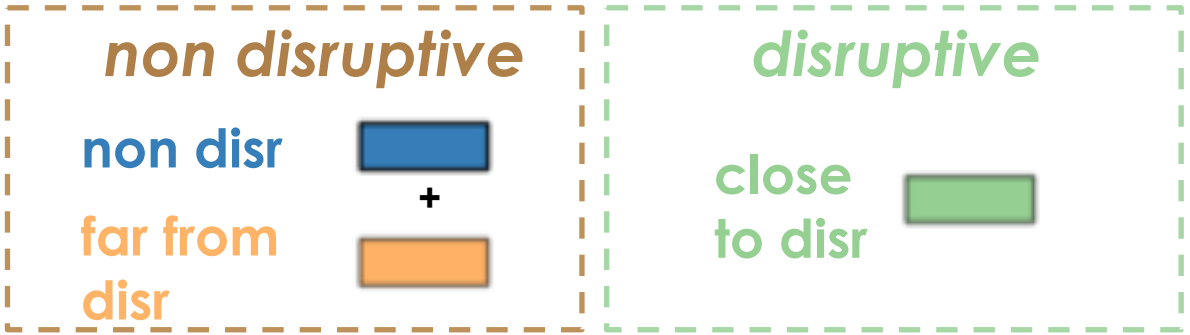
importance

n_equal_1 normalised
q95
n/nG
ip_error_ fraction
li
betap
Vloop
Wmhd
Te_width normalised

- Univariate analysis on input features: τ_{class} = **350 ms** as threshold for **transition from safe to disruptive** phase in the plasma parameter space.
- **9 features**, mainly dimensionless or machine-independent parameters, **available in real-time**.



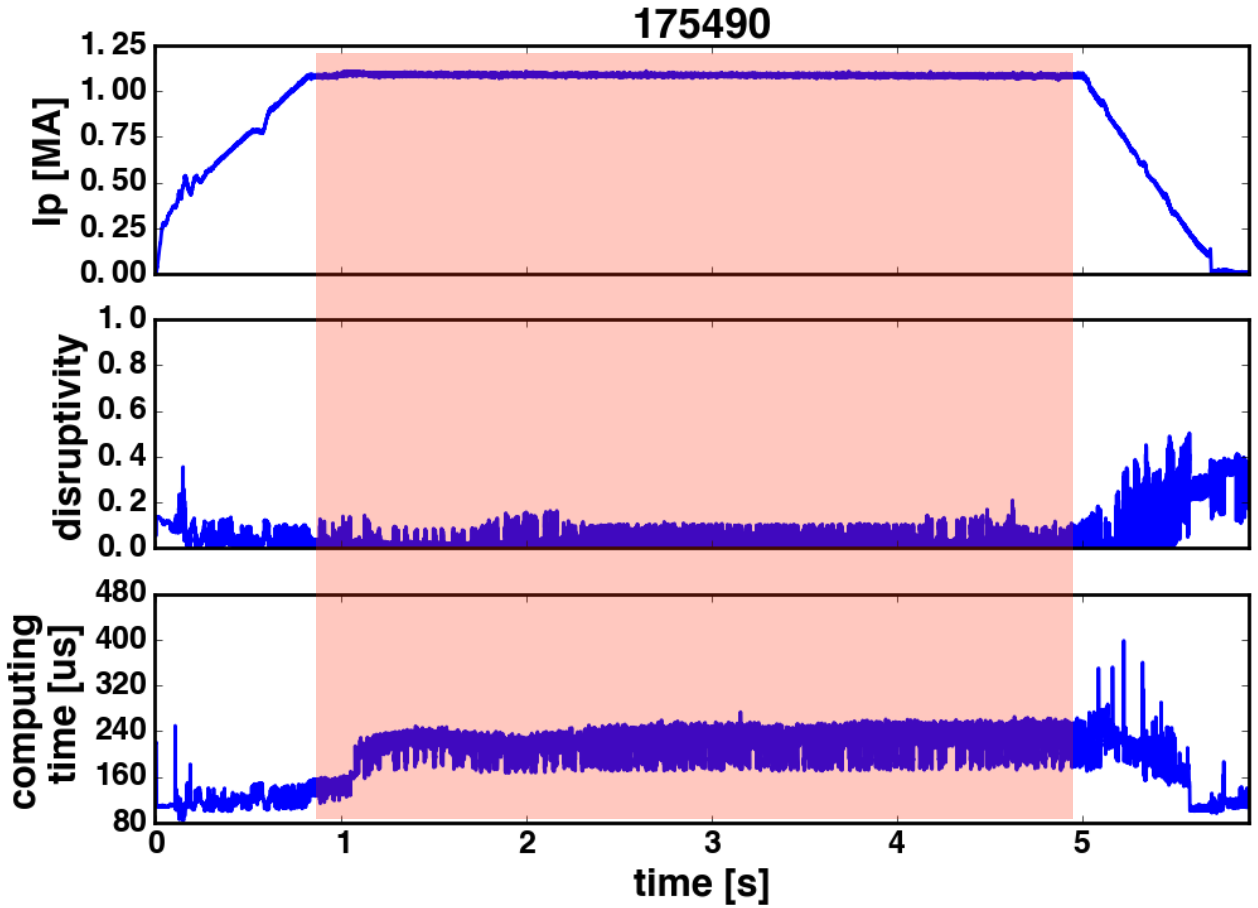
- **Binary** classification:



DPRF was trained on ~5300 DIII-D discharges, 16% disruptive, from the 2014-2017 experimental campaigns

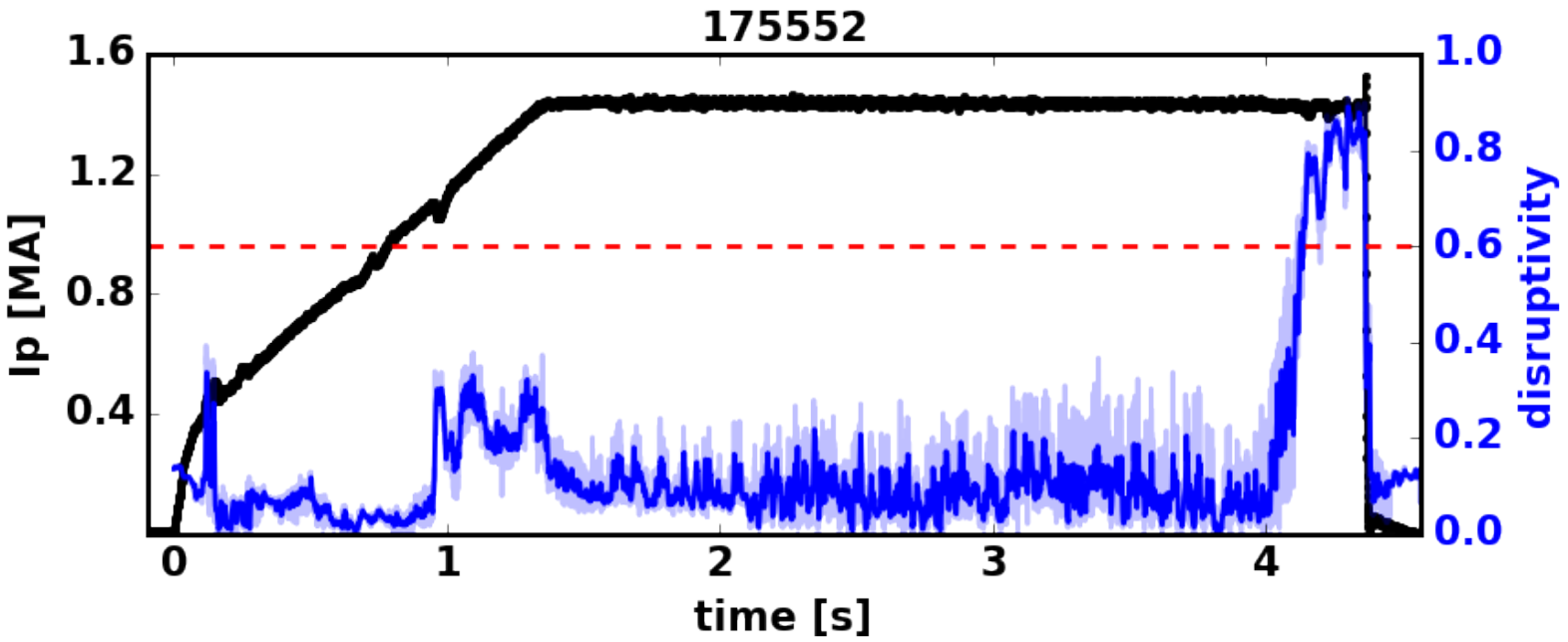
- **Training** only on **flattop** data;
  - No rampup or rampdown (disruptions).
  - **Not tailored on specific disruption dynamics**;
  - Only causal filtering;
  - Only unintentional disruptions;
  - No hardware-related disruptions.

- **Real-time inference** on unseen shots:
  - **Offline training**, then **translation to C*** for real-time integration;
  - **Each CPU cycle**, the 9-features vector is evaluated (250-300 μs)and inference stored as **disruptivity** signal.



*sklearn-porter, https://github.com/nok/sklearn-porter, D. Morawiec.
C. Rea et al., submitted to Nuclear Fusion (2019).

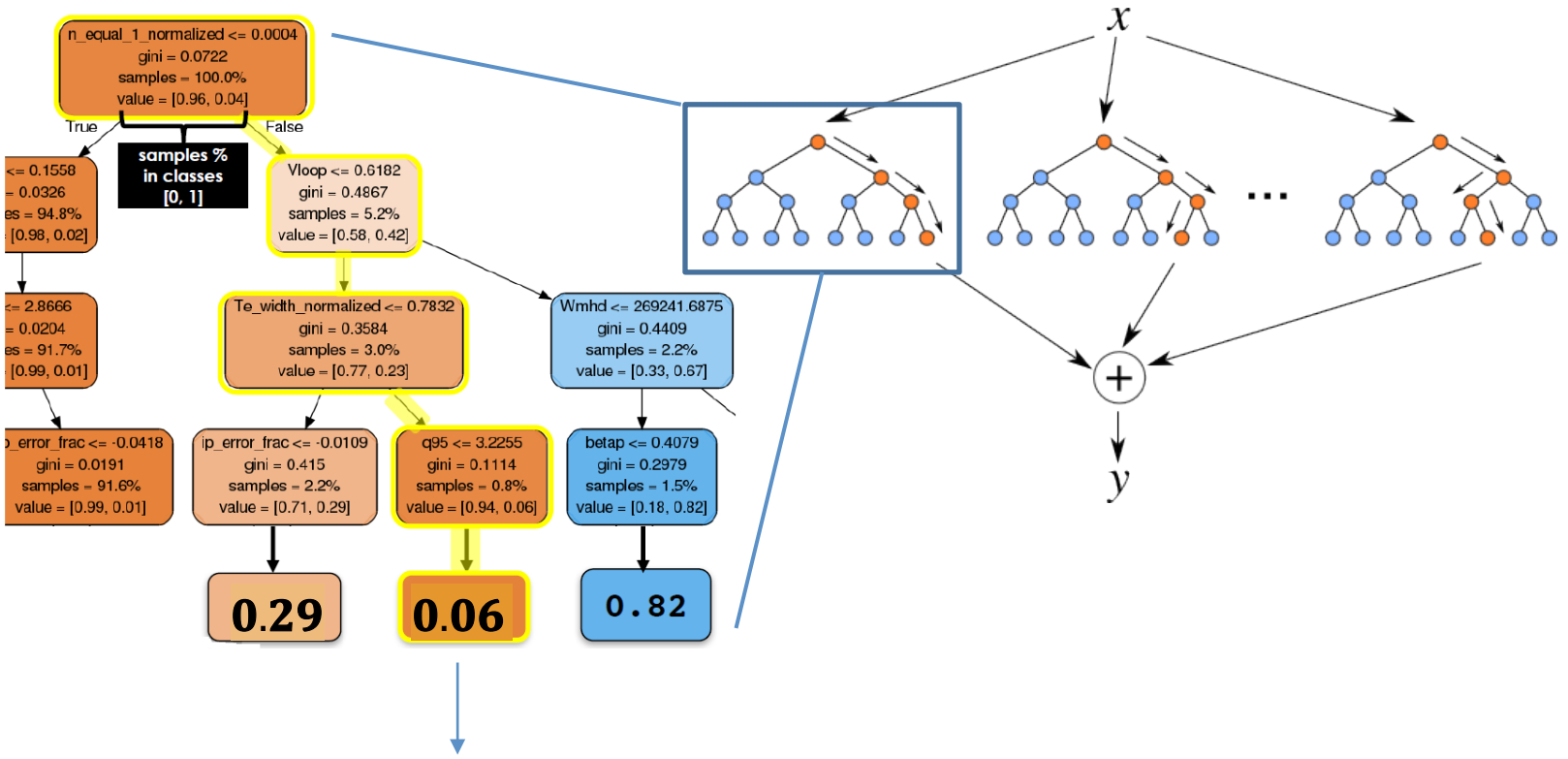
DPRF disruptivity for flattop disruption starts to rise above 60% about 240ms before disruption event



Which are the drivers of high disruptivity?

Disruption Prediction via Random Forests (DPRF) Algorithm

- Decision tree ensemble algorithm currently implemented for real-time use on DIII-D PCS
- Maps physics inputs x at each time sample to a disruption probability y
- Predictions can be interpreted in real-time by examining the **decision path** taken by each new sample



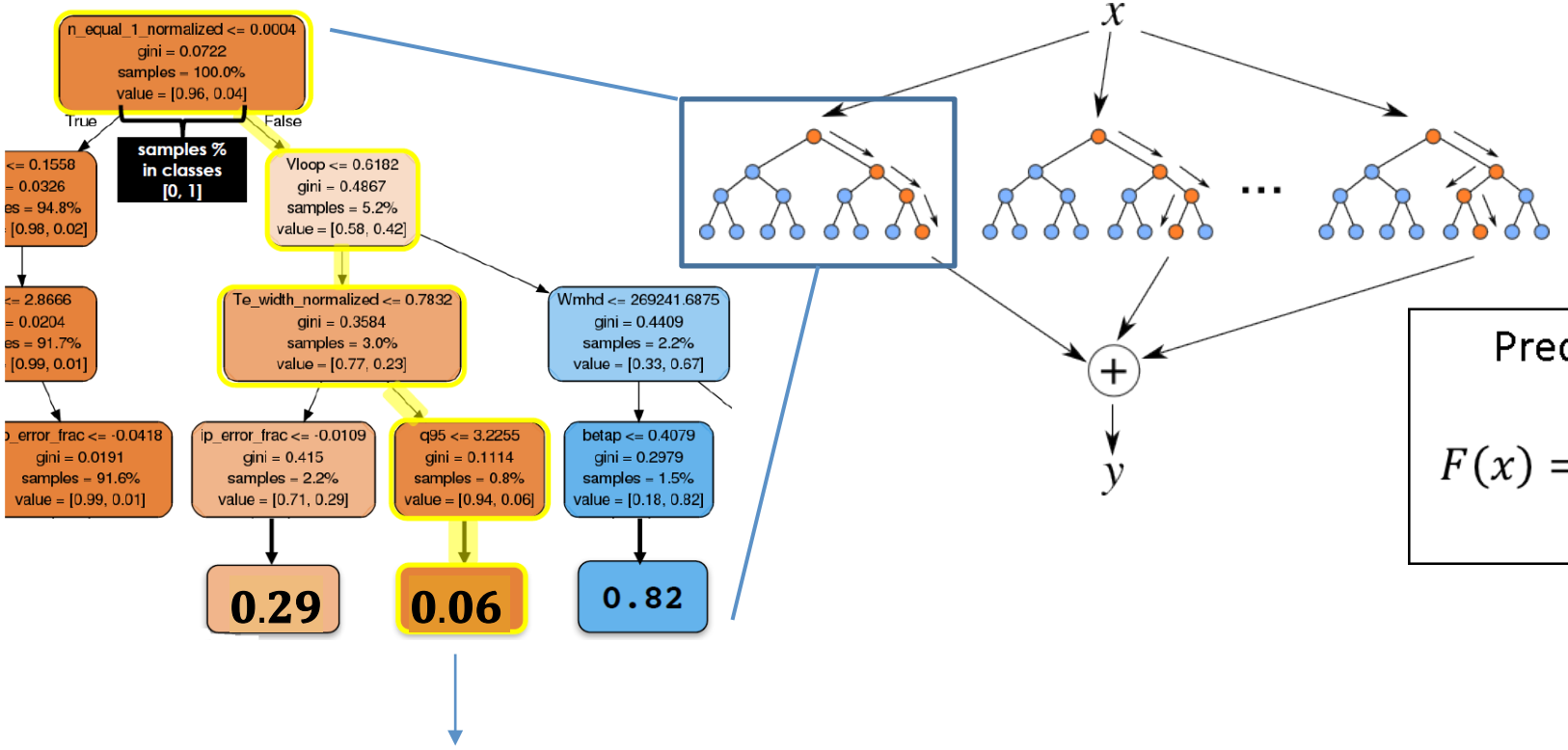
Prediction for 1 tree:

$$f(\vec{x}) = b + \sum_{k=1}^{K} contrib(x_k)$$

Prediction: $\underline{0.06} \approx 0.04$ (**model bias**, b) + 0.38 (**gain**, V_{loop}) - 0.19 (**loss**, r_{T_e}/a) - 0.17 (**loss**, q_{95})

Disruption Prediction via Random Forests (DPRF) Algorithm

- Decision tree ensemble algorithm currently implemented for real-time use on DIII-D PCS
- Maps physics inputs x at each time sample to a disruption probability y
- Predictions can be interpreted in real-time by examining the **decision path** taken by each new sample



Prediction for 1 tree:

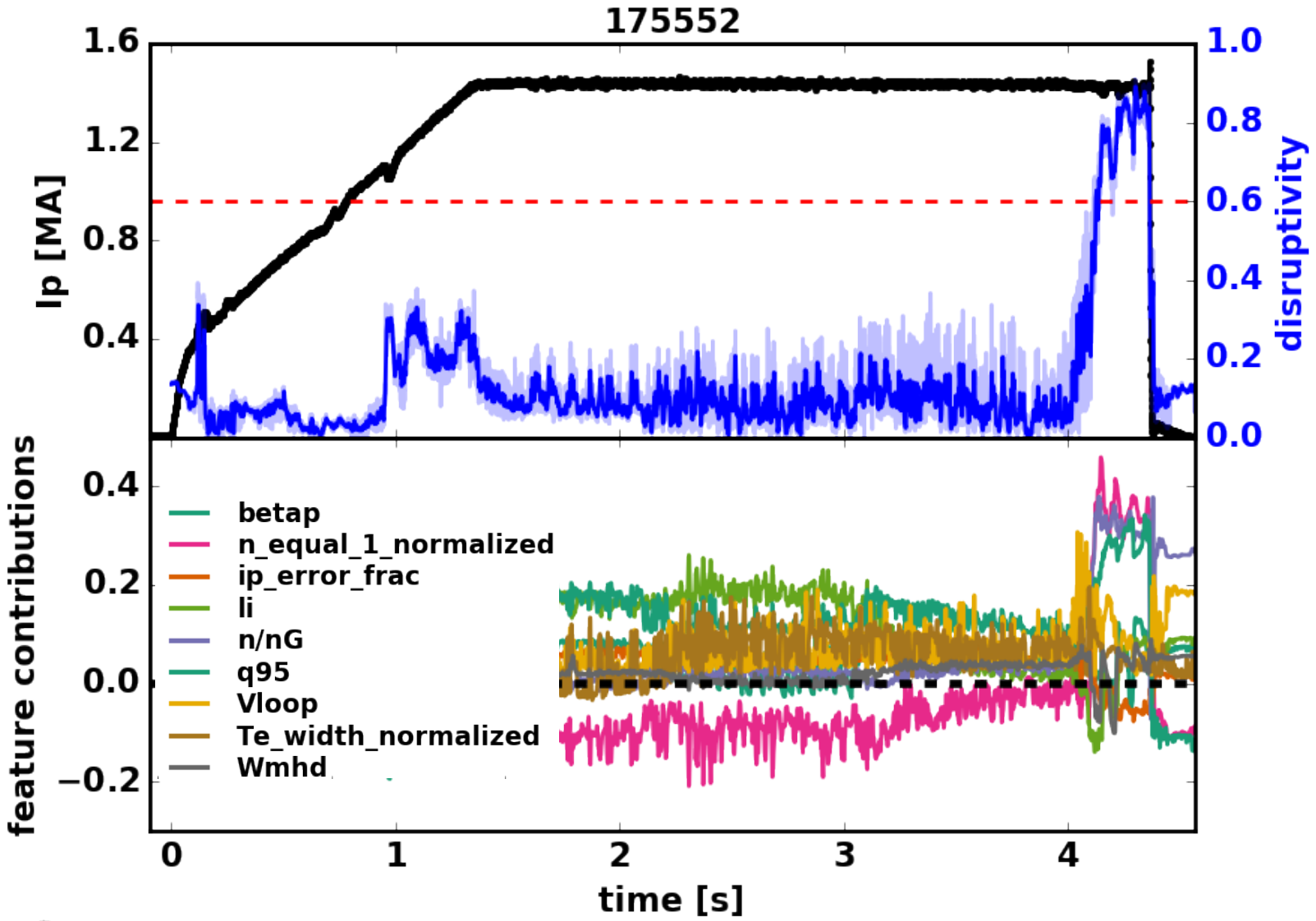
$$f(\vec{x}) = b + \sum_{k=1}^{K} contrib(x_k)$$

Prediction function for a *forest* of J trees:

$$F(x) = \frac{1}{J}\sum_{j=1}^{J} b_j + \sum_{k=1}^{K}\left(\frac{1}{J}\sum_{j=1}^{J} contrib_j(x,k)\right)$$

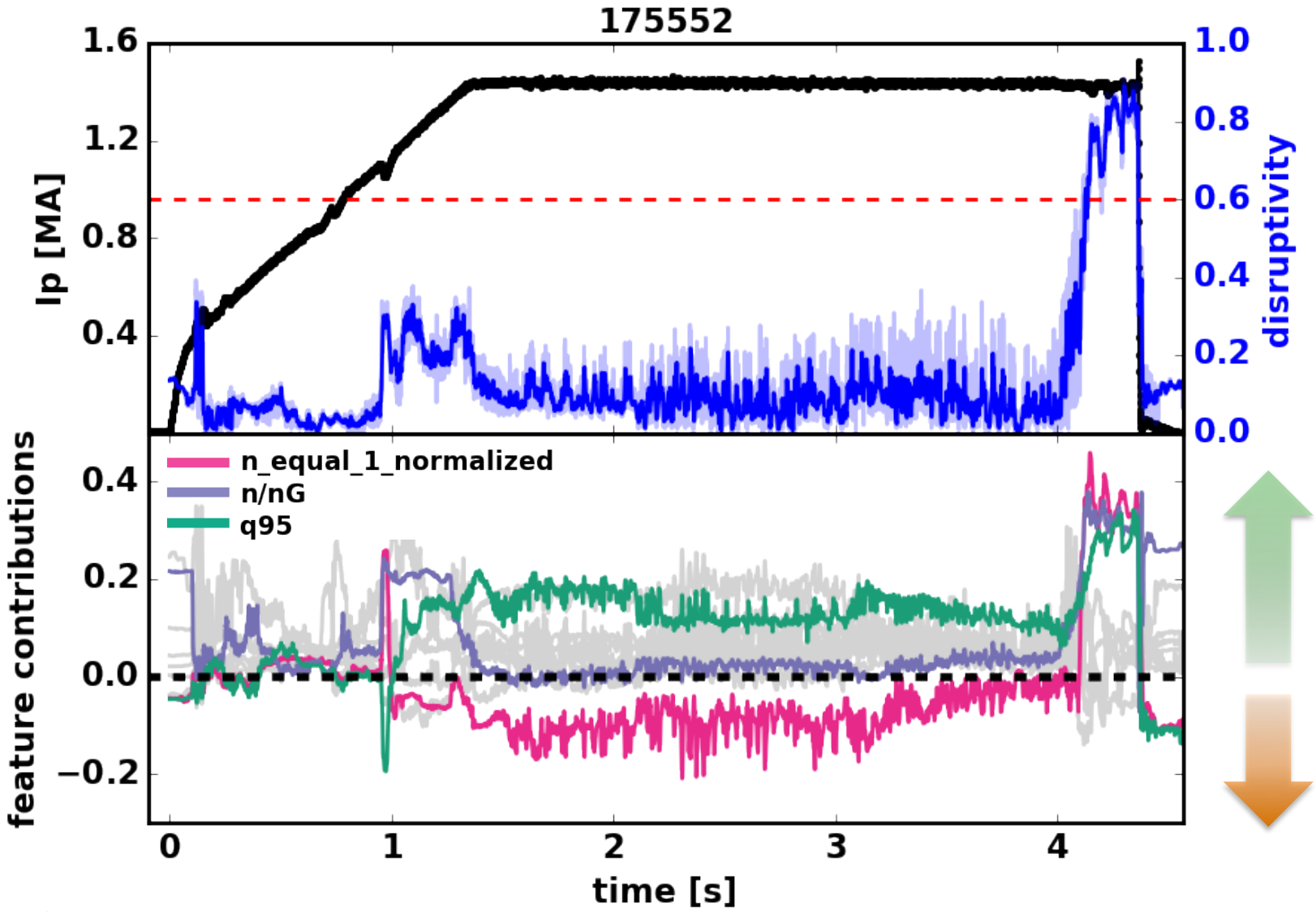
Feature contributions

DPRF disruptivity for flattop disruption starts to rise above 60% about 240ms before disruption event



- **Disruptivity breakdown** via **feature contributions** analysis: **Positive/negative contributions** are extracted from the **decision paths** traversed by the data sample along the individual estimators (trees) in the forest during inference.

Locked mode proxy, safety factor and Greenwald density fraction contribute to most of the increasing disruptivity

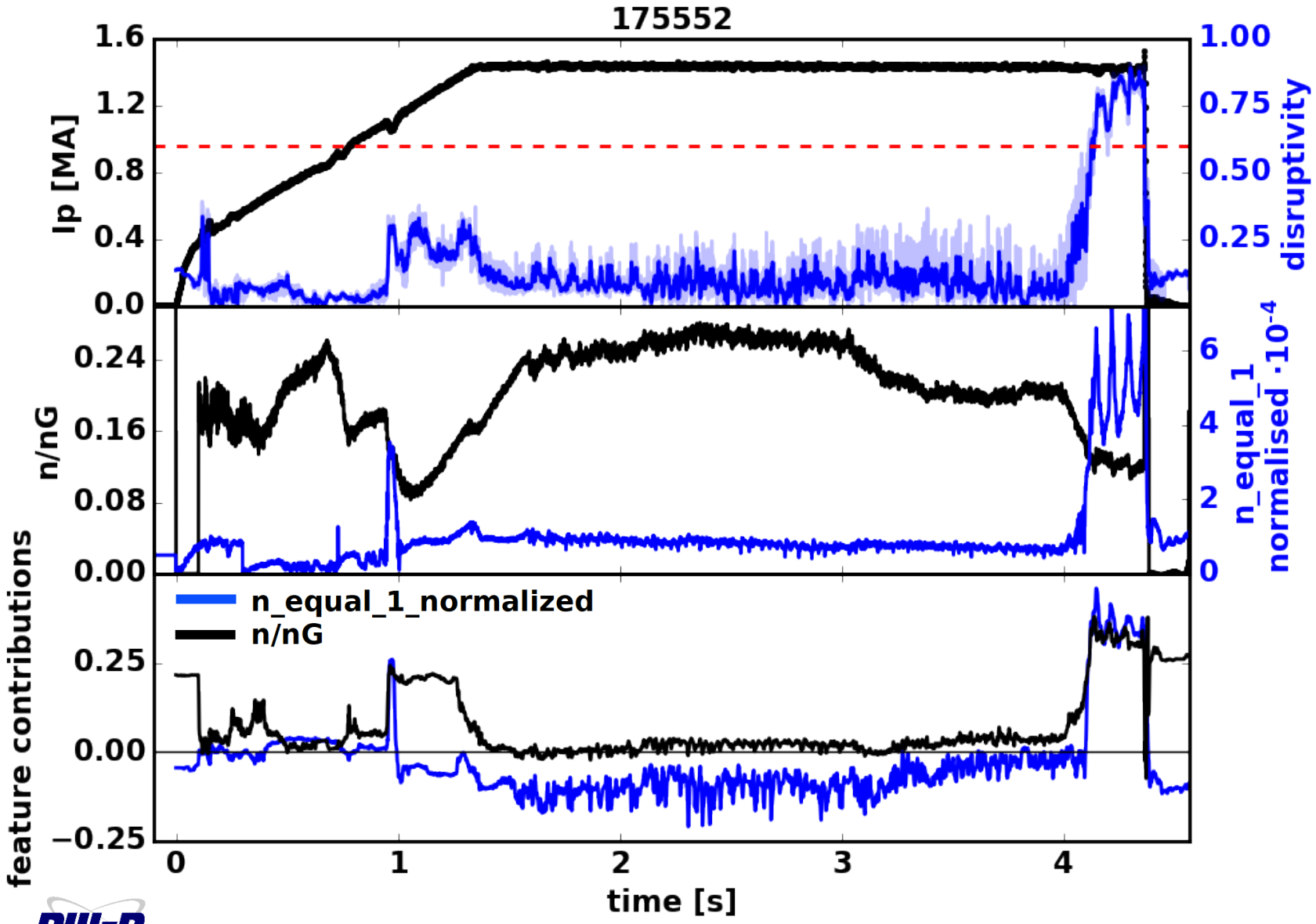


- Increasing DPRF disruptivity reflects input features deviations.
- **Positive** and **negative contributions** push towards **disruptive** and **non disruptive** operational spaces.

disruptive

non disruptive

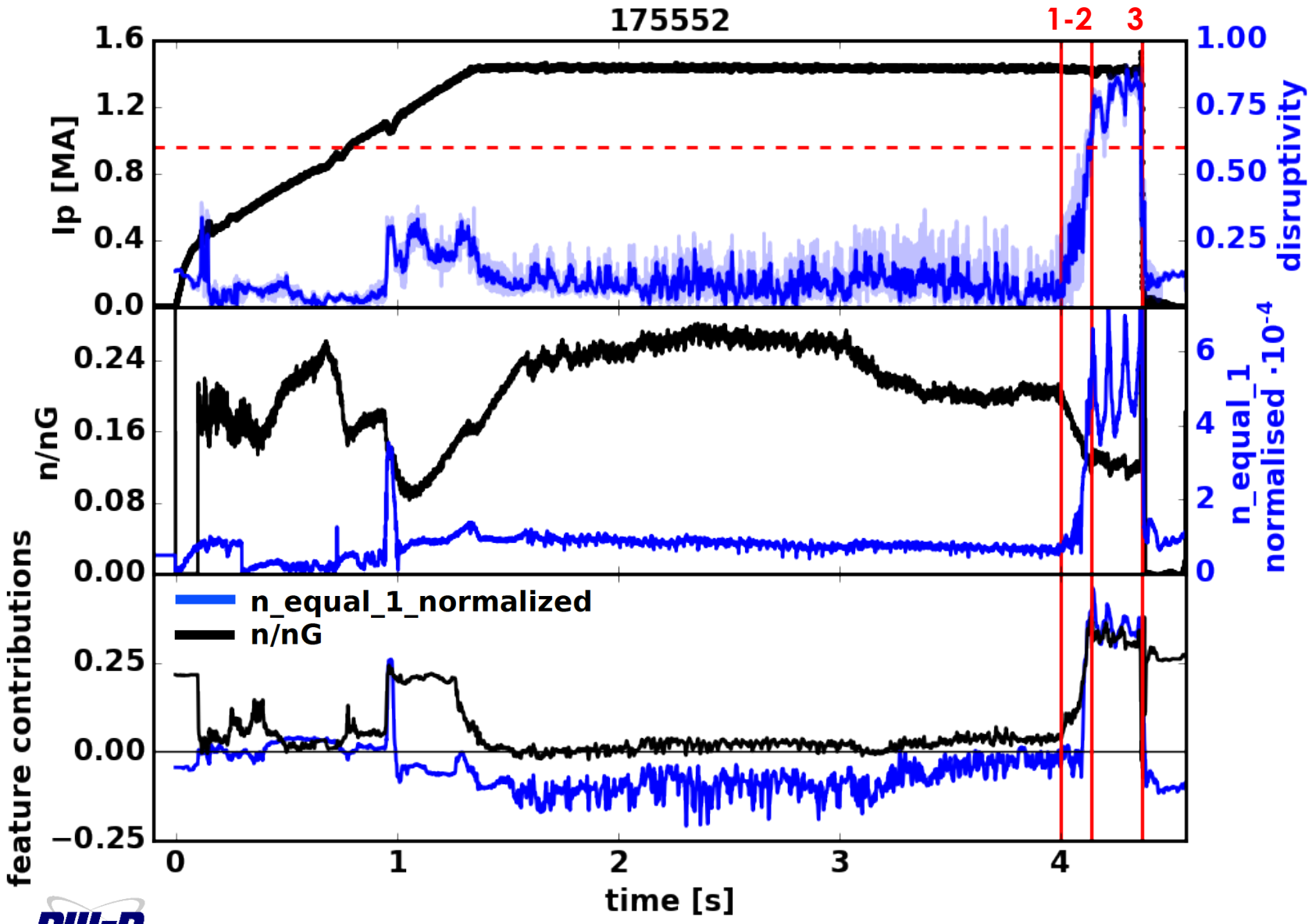
Increasing disruptivity, interpreted via feature contributions, reflects plasma parameters deviations to disruptive scenario



actual time traces used for inference in real-time

feature contributions of raw signals

Increasing disruptivity, interpreted via feature contributions, reflects plasma parameters deviations to disruptive scenario



[1] LON (LOQ) q95 ~ 3.2 — 4s

[2] ML — 4.14s

[3] DISRUPTION — 4.37s

- **Feature contributions** analysis successfully **identifies chain of events** leading to disruption;
- PCS implementation could enable **real-time scenario detection**.

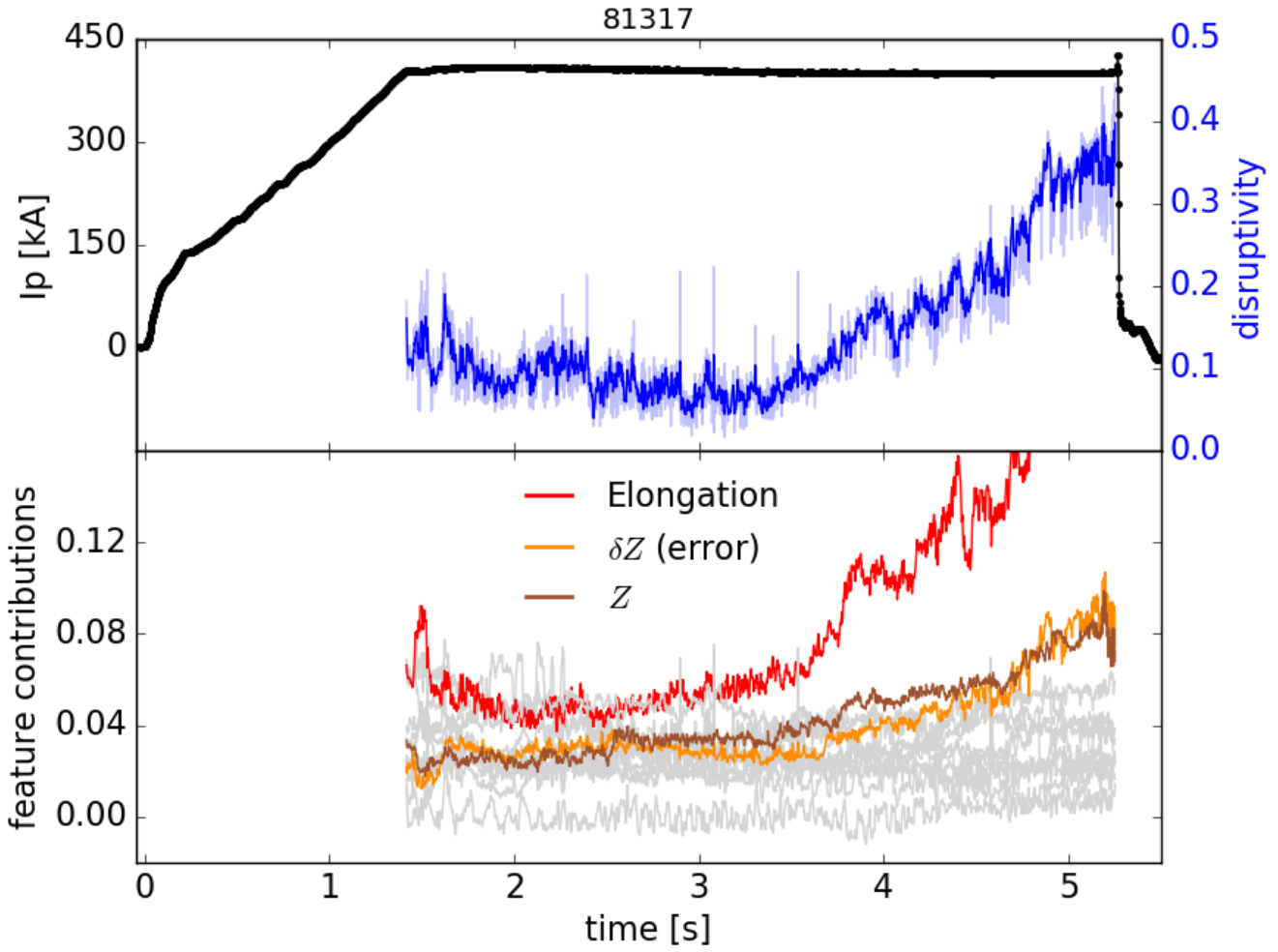
EAST DPRF trained on ~7500 shots in the 2014-2017 campaigns - τ_{class} set at 100 ms before the disruption

importance

ip_error fraction
Vloop
radiated fraction
li
kappa
q95
n/nG
Wmhd
βp
δZ
Z
Te_width
n_equal_1 mode

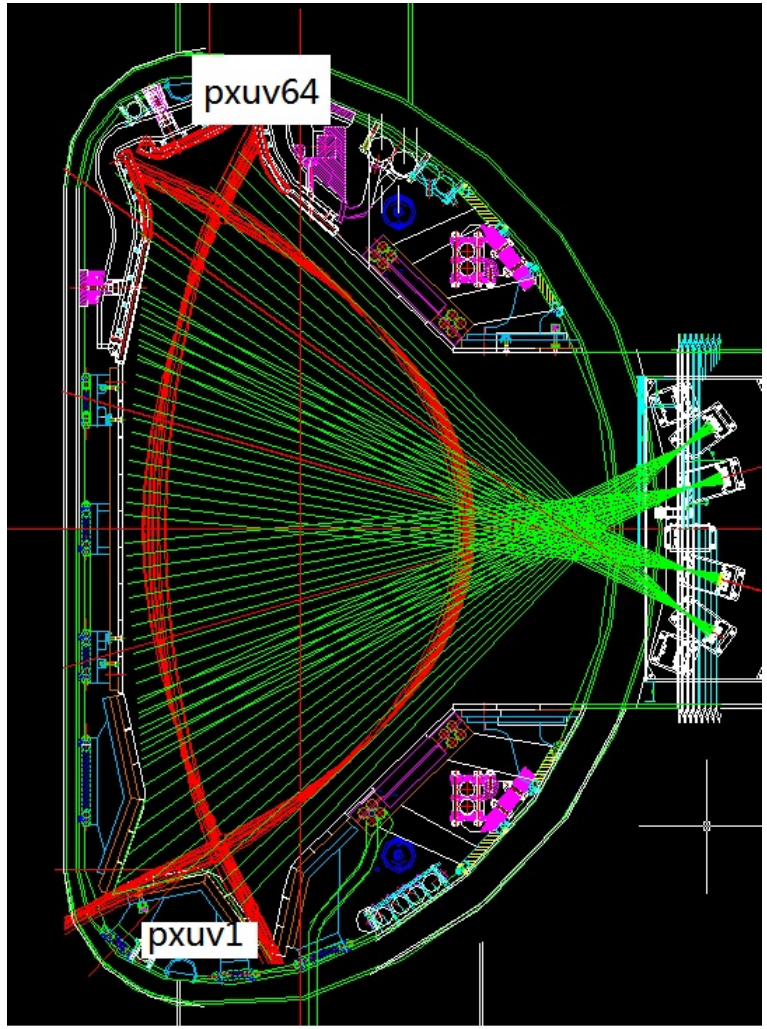
Inter-shot disruptivity predictions:

- DPRF trained on GA clusters;
- Data preprocessed on EAST servers once availble;
- DPRF disruptivity and features contributions available ~4-5 min after shot.



Real-time implementation under way

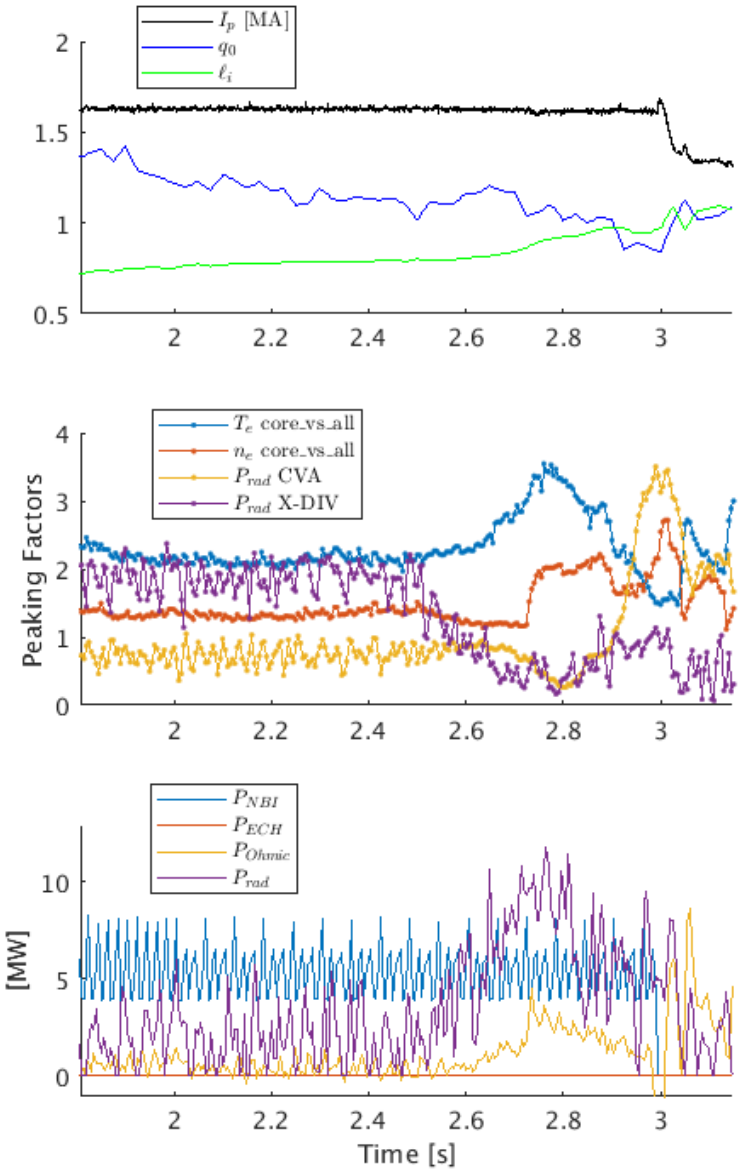
Spatial profile information can be condensed into ‘profile width’ or ‘profile peaking’ signals

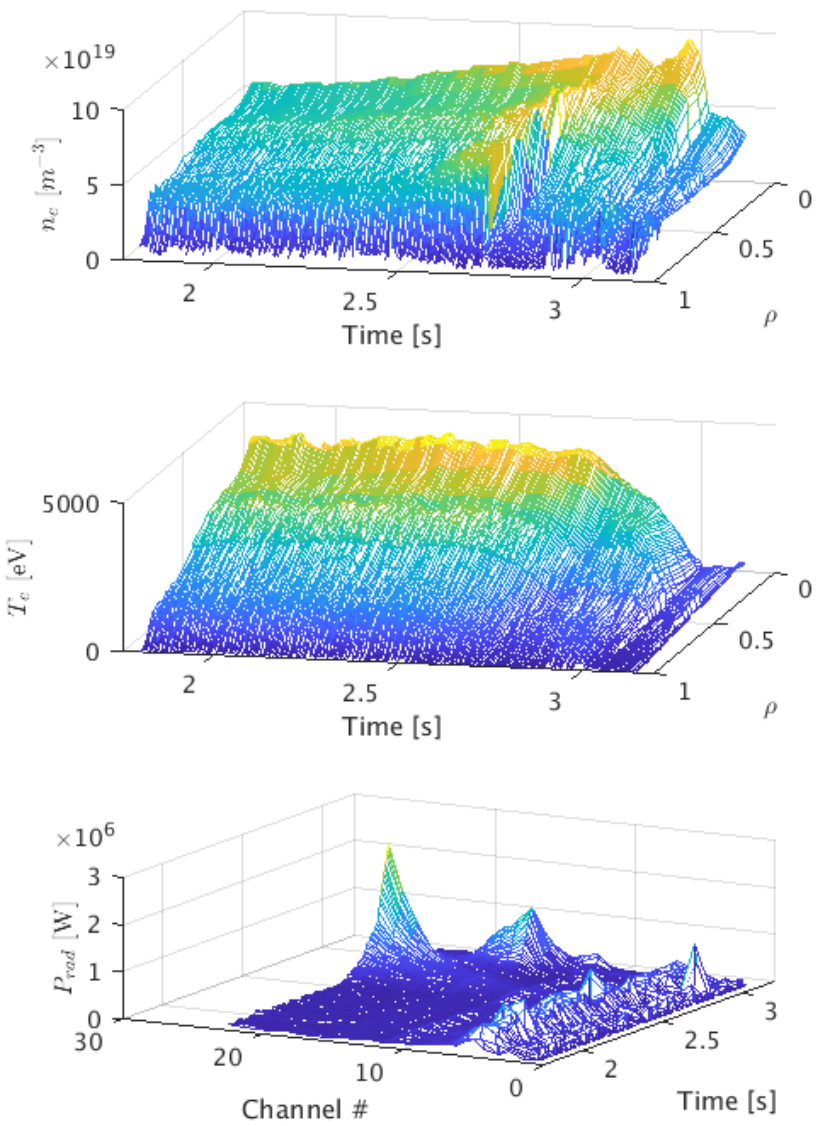


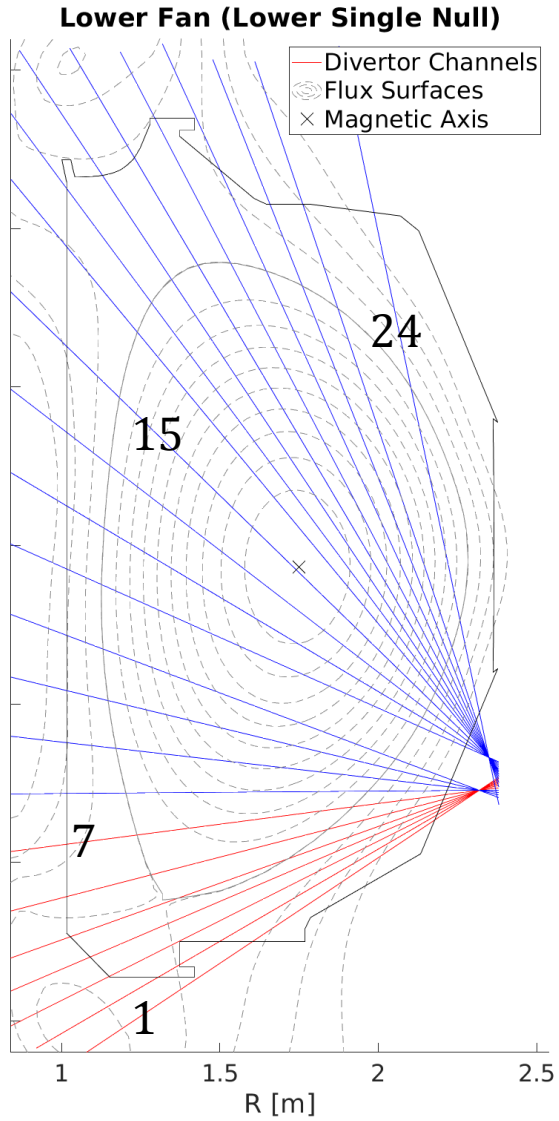
AXUV arrays on EAST

- T_e width from ECE and/or Thomson scattering
- T_e peaking from ECE and/or Thomson scattering
- n_e peaking from Thomson scattering
- Pressure peaking from n_e x T_e
- P_{rad} peaking from AXUV arrays
- divertor vs core P_{rad}

'Peaking Factors' Add Spatial Information to Inputs for Prediction

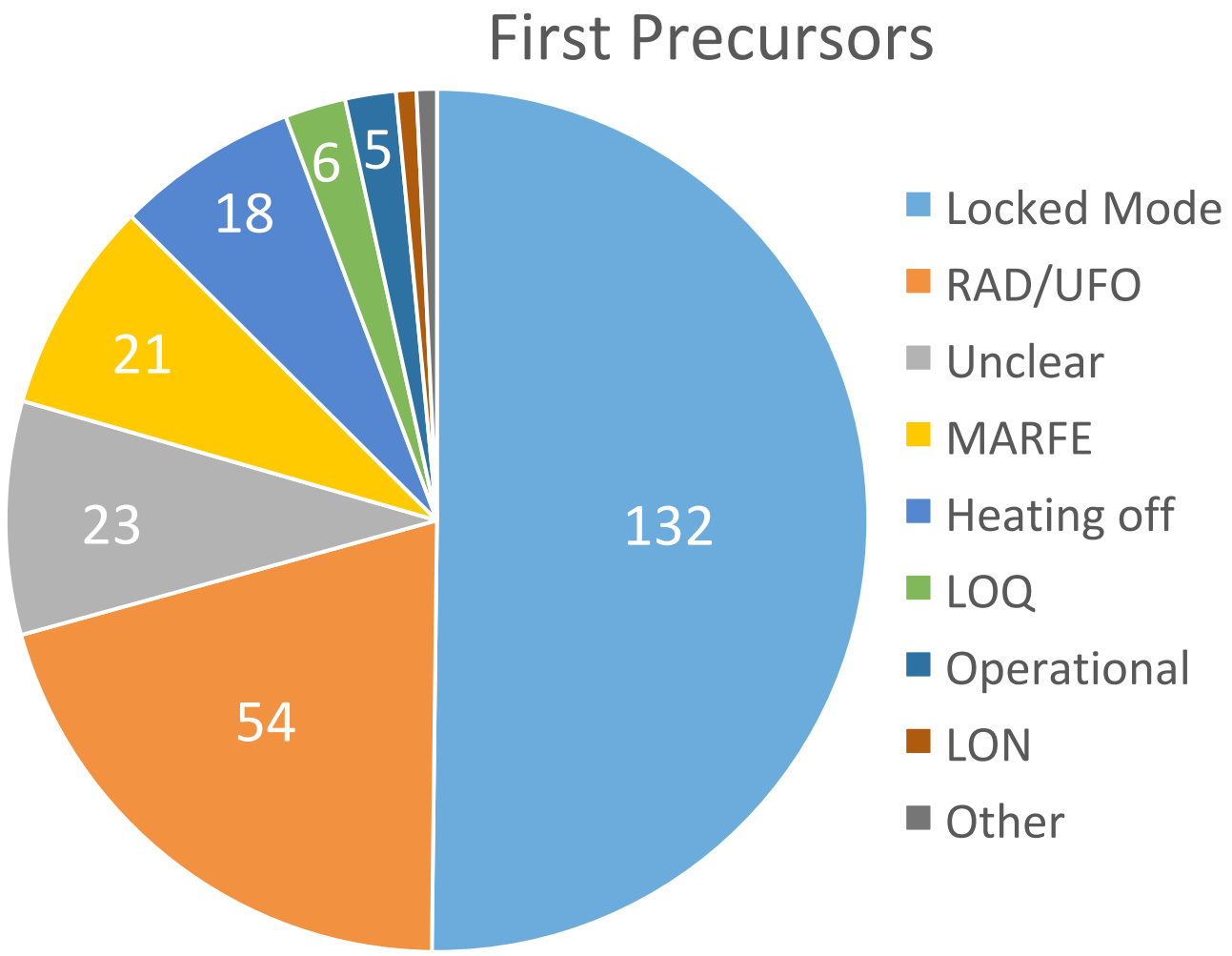






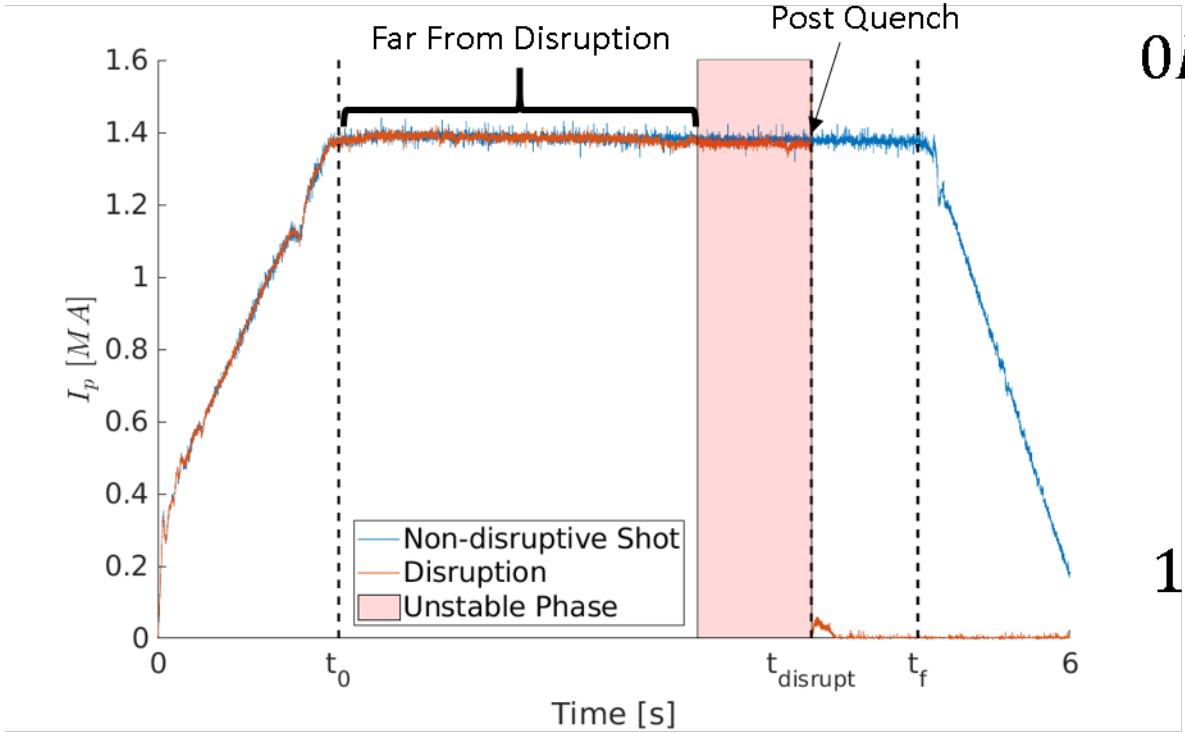
Can Feature Contributions Be Indicative of Disruption Precursors?

- Implemented post-mortem manual analysis of > 250 disruptions from DIII-D 2015/2016
- Identified physics precursors in disruptive event chains
- Strong correlation between type of first precursor and experimental run
  - 19 of 21 MARFE disruptions during experiments studying detachment
  - 74% of RAD/UFO disruptions during metal ring campaign
  - 79% of Locked Mode disruptions are during runs studying LM avoidance, TM control, and ELM control with RMPs



Compared Manual Analysis with DPRF Predictions Using 13 Inputs

- Split into 2 classes using manually tagged $t_{unstable}$
- Excluded time slices 'far from disruption' and 'post quench'



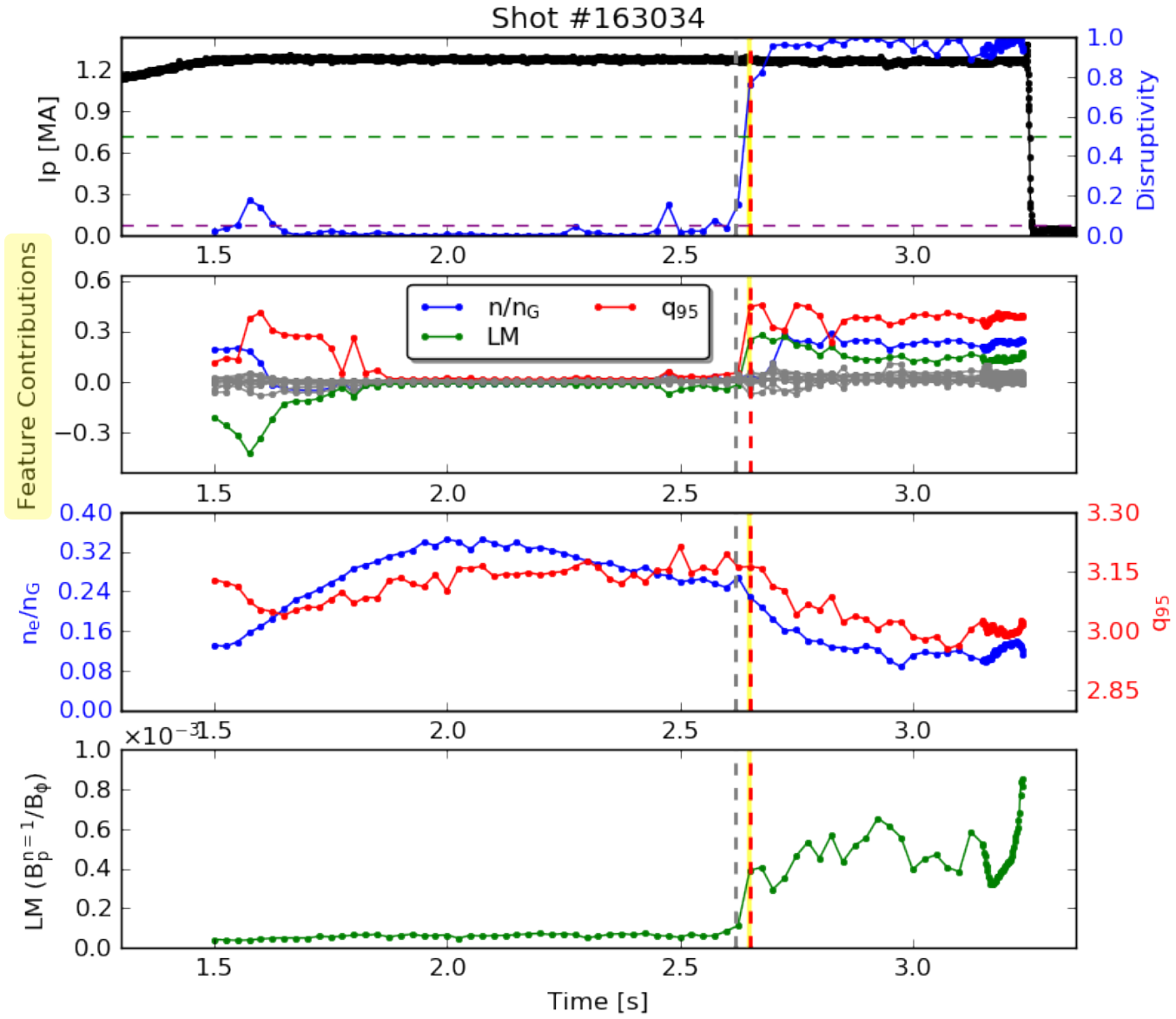
$0D$

$1D$

Feature	Description
n/n_G	Greenwald fraction
W_{mhd}	Stored energy
β_p	Poloidal beta
$(I_p - I_{prog})/I_{prog}$	I_p error fraction
ℓ_i	Internal inductance
$B_p^{n=1}/B_\phi$ (LM)	Locked mode proxy
q_{95}	Safety factor @ 95% flux
P_{rad}/P_{input}	Radiated power fraction
V_{loop}	Loop voltage [V]
P_{rad} Core Peaking	Core vs all ratio
P_{rad} Divertor Peaking	Divertor vs all ratio
T_e peaking	Core vs all ratio
n_e peaking	Core vs all ratio

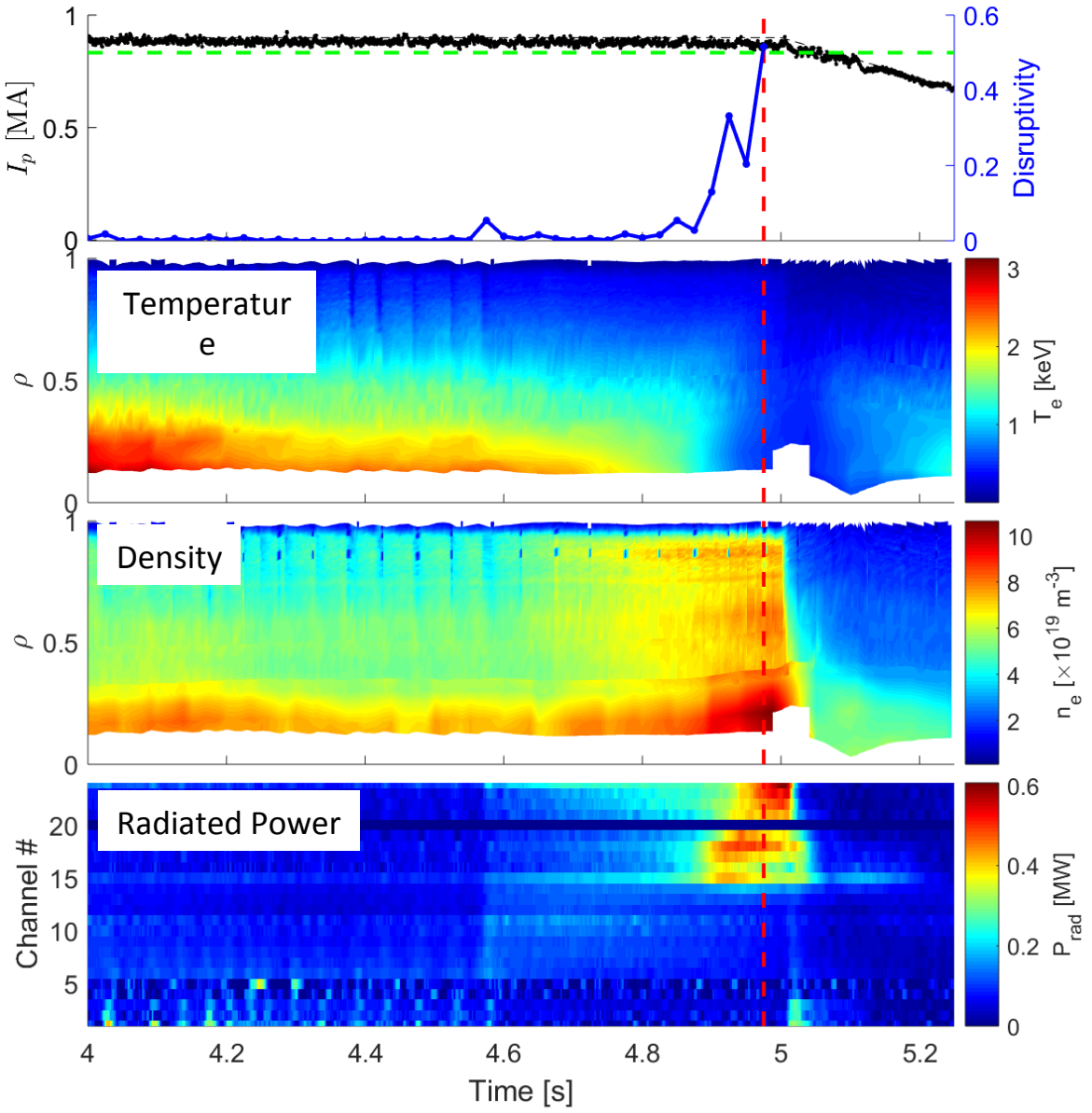
True Positive Example: A clearly triggered LM disruption

- 2/1 rotating mode locks and disrupts during a rotating TM control experiment
- Contributions from LM signal and low q_{95} trigger disruption warning alarm
- Disruptivity further increases due to dropping n_e/n_G



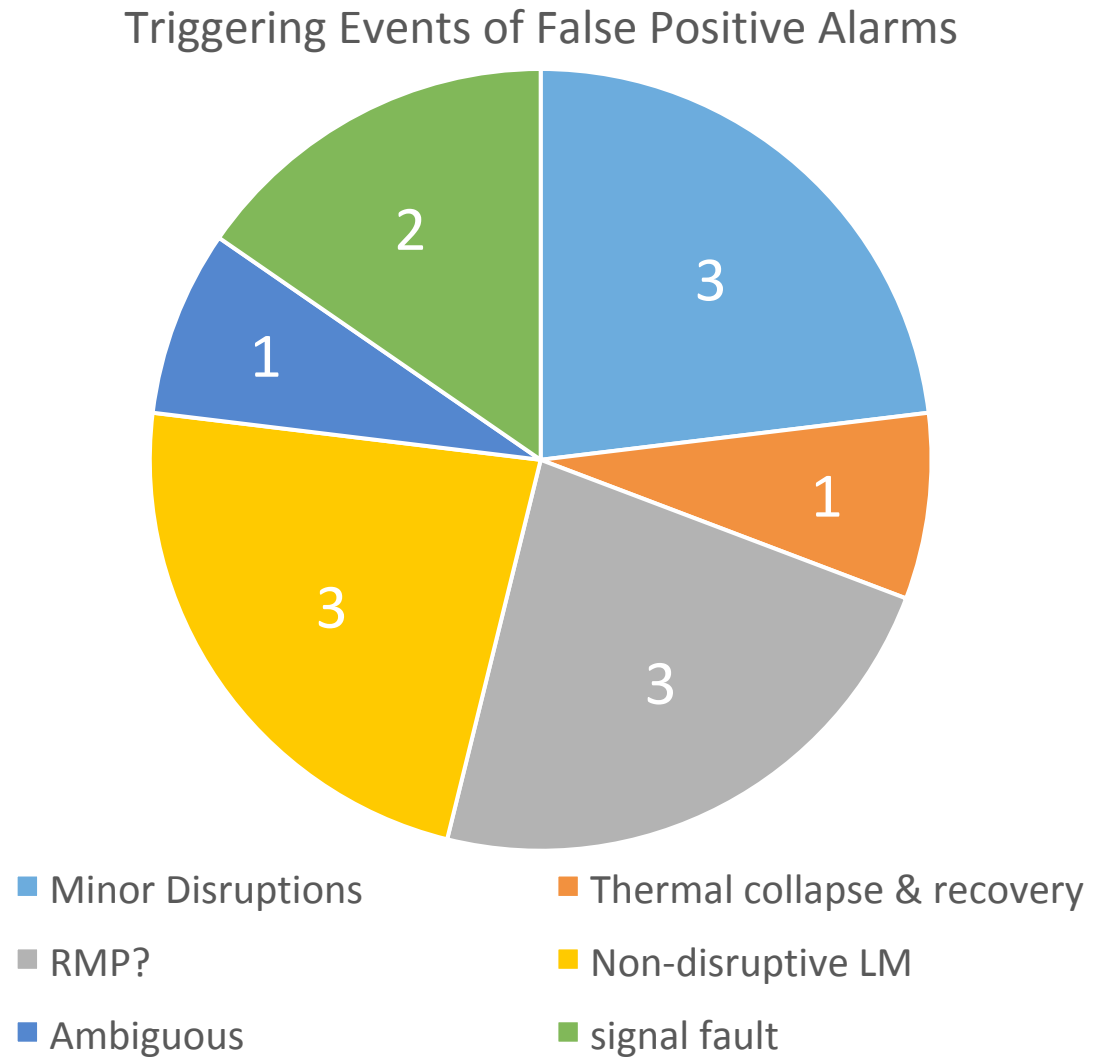
False Positive Example: Crash and Recovery?

- DPRF fooled by a large crash and radiative event near $t{\sim}5\ s$
- 1D information contributes most to the prediction
- Soft landing & rampdown classifies this shot as a false positive



For which non-disruptive discharges did we trigger an alarm?

- False positive rate (FPR) $\approx 7\%$
- If minor disruptions and thermal collapses/recoveries are excluded, FPR reduces to $\approx 5\%$
- Decoupling the pickup from the RMP coils and the n=1 LM signal could further reduce FPR to $\approx 3\%$



Random Forest vs Recurrent Neural Network

There are a number of reasons why Random Forest is an attractive Machine Learning method:

- The architecture of a Random Forests involve only a couple design parameters, which are easily optimized
- Different features (plasma parameters), with vastly different numerical ranges, present no issues
- Data can be sampled at non-uniform rates
- For Random Forests, the degree to which each feature contributes to the classification decision can be characterized in a straightforward deterministic way (“white box”)

HOWEVER:

RF classification is done on each time slice independently

- Information from previous classification decisions is not used in determining the classification of the current time slice
- The classification of the current time slice is not used for classification decisions of future time slices.

Neural Networks are another AI method for classification problems

Many design parameters that can be difficult to determine:

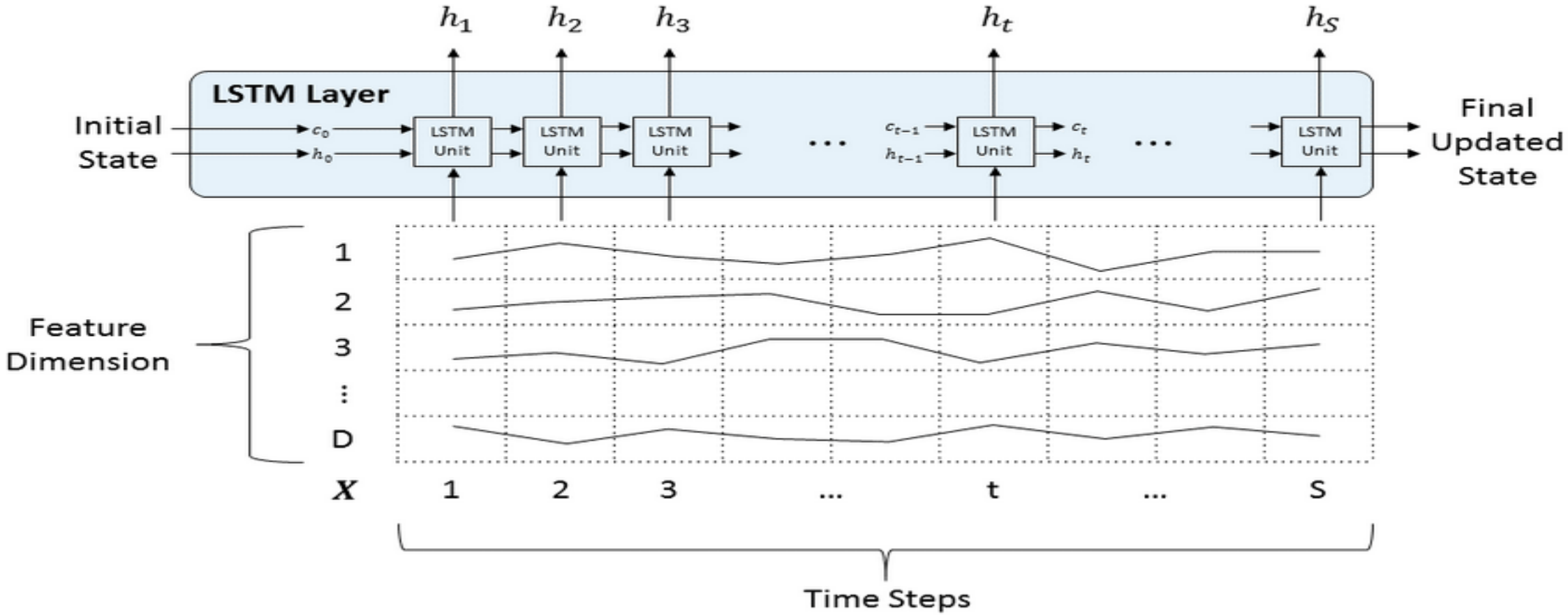
- How many hidden layers?
- How many nodes in each hidden layer?
- 1000's or millions of weights to determine/optimize
  - Deep Learning; back-propagation;
- Difficult to determine the degree to which each feature contributes to the classification decision ("black box")

But their complexity can incorporate features such as temporal history

Recurrent Neural Networks (RNN's) have the capability to include past classification information in current and future decisions

LSTM Layer Architecture

This diagram illustrates the flow of a time series X with D features of length S through an LSTM layer. In this diagram, h denotes the output (also known as the *hidden state*) and c denotes the *cell state*.

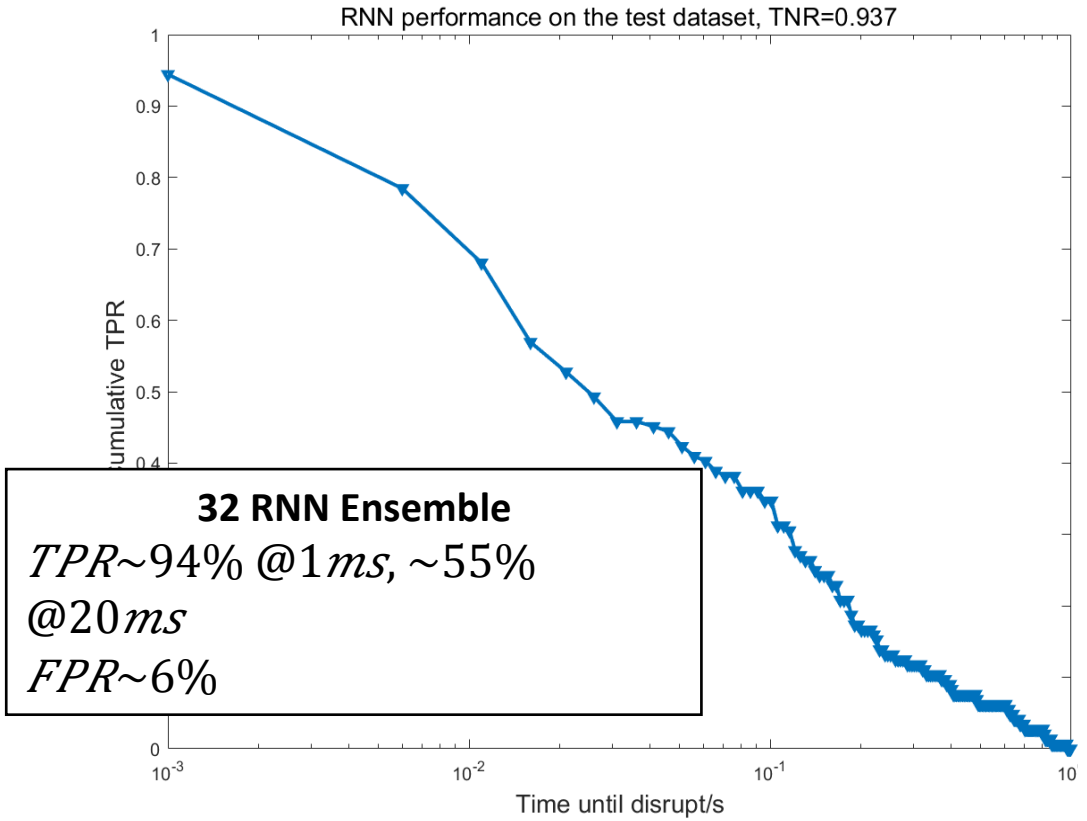


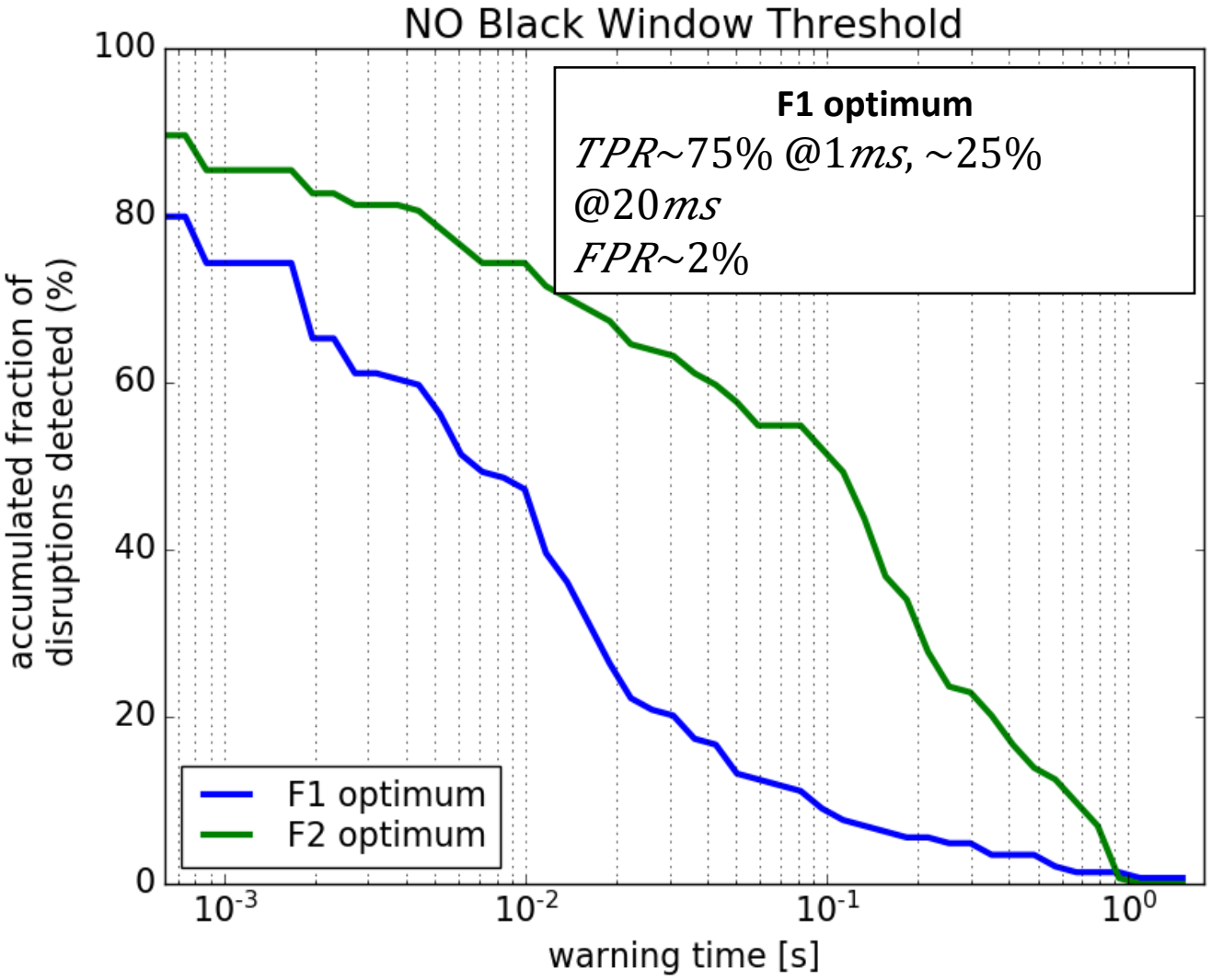
Our initial experiences working with RNN's

- Data had to be re-sampled to be on a uniform timebase
- All plasma input parameter numerical values must be normalized and shifted to have similar numerical ranges
- Training and optimization takes several days, as opposed to a few minutes for RF's
- Determining feature importance is very cumbersome, and very slow compared to RF's
  - Each input parameter's value is varied by a small amount, one parameter at a time. The observed changes in the output are used to determine the Jacobian for each parameter, for each time slice. The sum of each parameter's Jacobians over all time slices gives a relative feature importance.

But, using an ensemble of 32 RNN's, our initial finding is that they perform better than RF's, at least on C-Mod data (which is the most difficult of our set of tokamaks to predict accurately)

Comparison of RNN and Random Forests performance on C-Mod data





Some next steps

- Install our trained RF in the EAST PCS and run in real time. Possibly use the output to fire MGI valve
- Upgrade our RF predictor that's currently running in the DIII-D PCS to determine feature importance in real time
- Develop RNN's for DIII-D and EAST, and compare to RF's
- After training disruption prediction algorithms on one tokamak's database, try applying it to the other tokamaks' data. The long-term goal is to determine if a universal disruption warning algorithm can be realized using AI methods.