Machine Learning Techniques for Embedded Control Software

Alessia Bozzini
Franco Fummi

UNIVERSITÀ
di VERONA
Dipartimento
di INFORMATICA

Version 1.1

Agenda

- Machine learning
- MATLAB
  - Classifier
  - Regressor
  - LSTM
- Control system
- Use cases
  - Bang bang controller
  - PID controller
  - Water tank controller

Machine Learning: Supervised, Unsupervised, Reinforcement learning and RNN

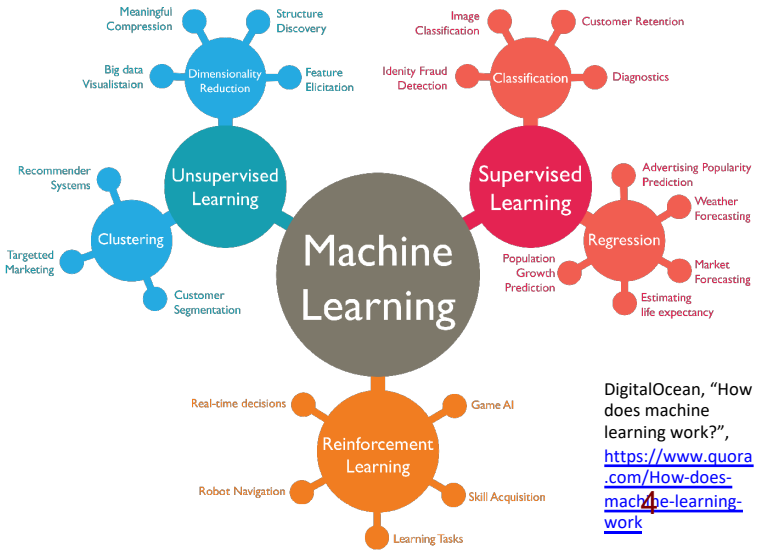
BACKGROUND

Machine Learning

We need a large data sets

- **Machine learning** represents a set of statistical methods to progressively improve the performance of an algorithm in identifying patterns in data
  - **Supervised**:
  - **Unsupervised**
  - **Reinforcement learning**

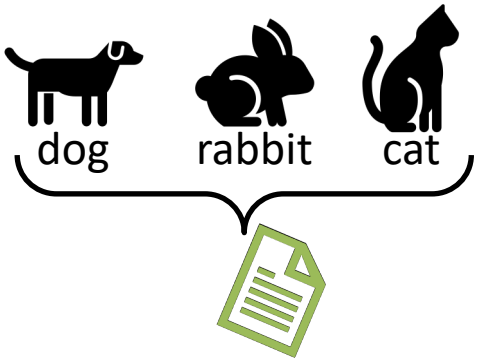
If you know an algorithm on how to solve a problem, just use it instead of using A.I.

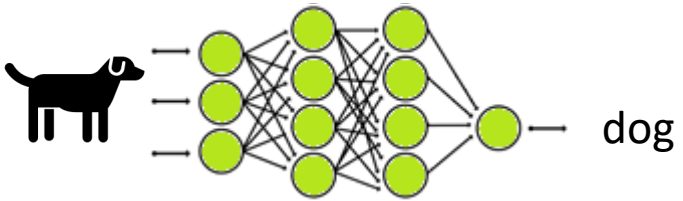


DigitalOcean, "How does machine learning work?", https://www.quora.com/How-does-machine-learning-work

Machine Learning: **Supervised**

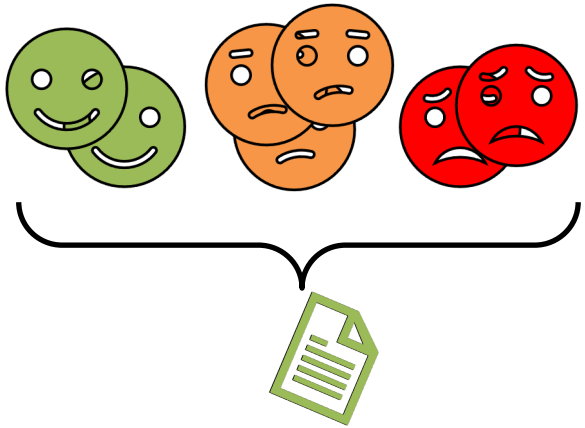
- Having **inputs** and related **outputs**, extract the **rule** that associates the inputs to the outputs
  - Classification
  - Regression

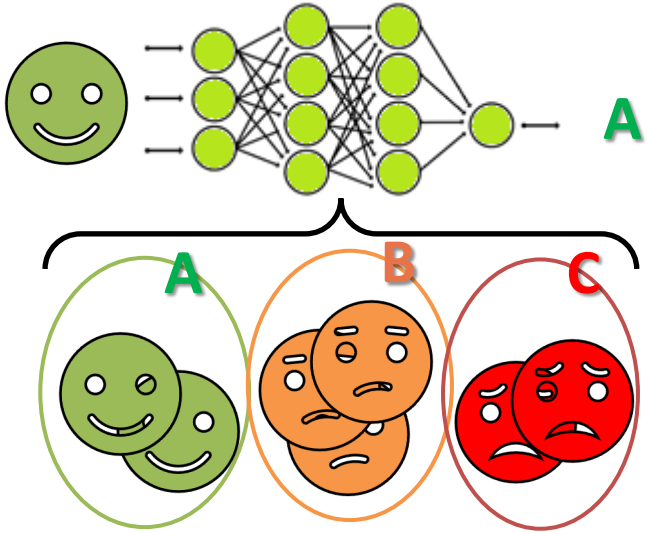




Machine Learning: **Unsupervised**

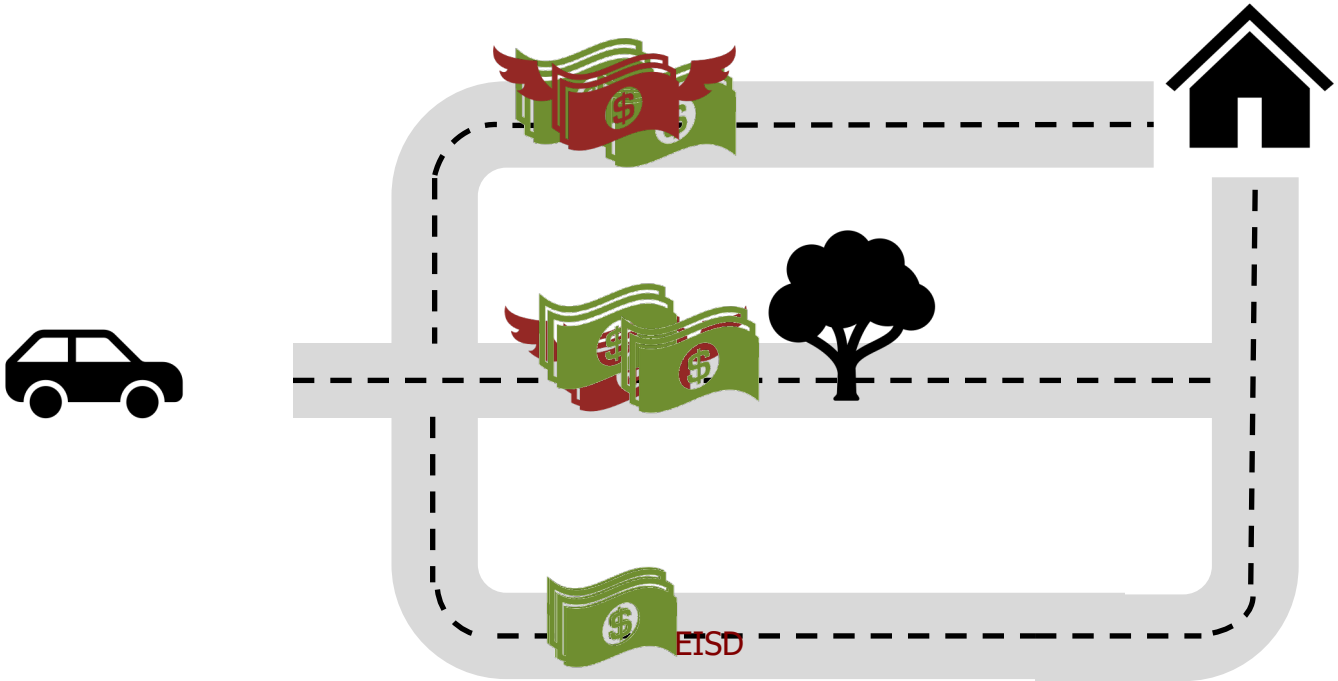
- Having unlabeled inputs, finding the structure
  - Clustering => Data is organized in different sets





Machine Learning: **Reinforcement learning**

- Allows the system to be able to learn and **adapt** to changes in the environment in which it is immersed



Artificial Intelligence

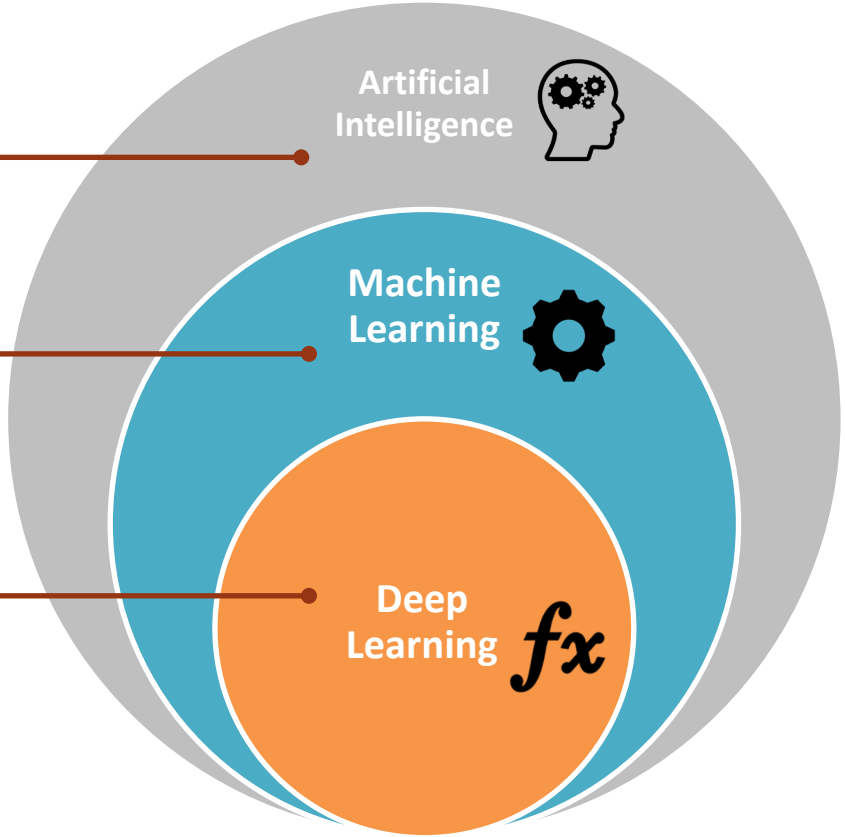
Any technique which enables computers to mimic human behavior

Current state of the art ⇐ **Machine Learning**

Subset of AI techniques which use statistical methods to enable machines to improve with experience

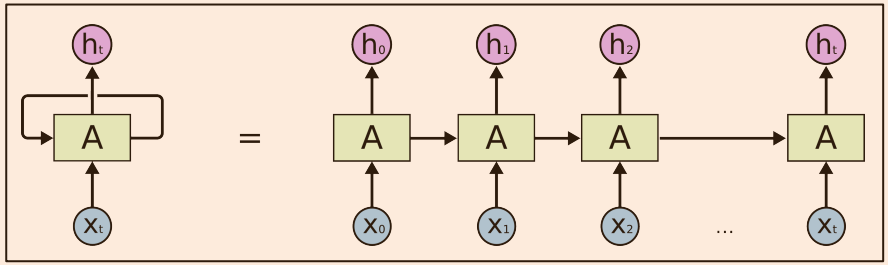
Generally not used because it takes too many resources ⇐ **Deep Learning**

Subset of ML which make the computation of multi-layer neural networks feasible



RNN: LSTM

- Recurrent neural networks (RNN) are networks **with loops** in them
  - Allow the persistence of information
- Variety of problems
  - speech recognition
  - language modeling
  - Translation
  - image captioning
  - …
- Look at recent information to perform the present task



If more context is needed → **LSTM**

Long Short Term Memory networks (LSTMs) are a special kind of RNN, capable of learning long-term dependencies

How to build a classifier

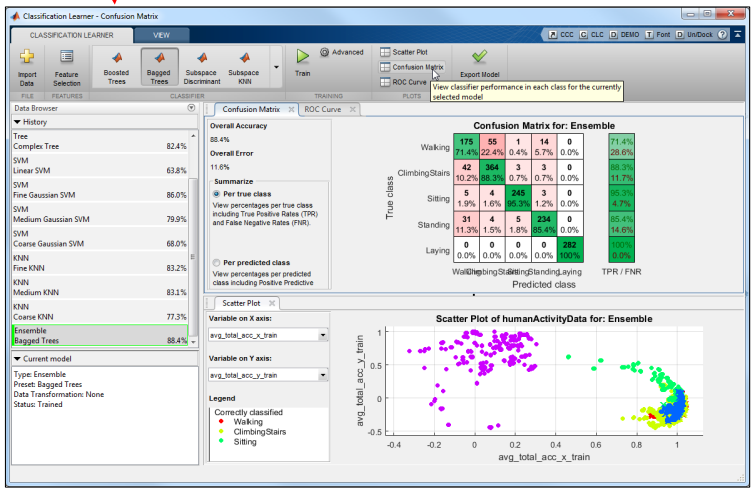
How to build a regressor

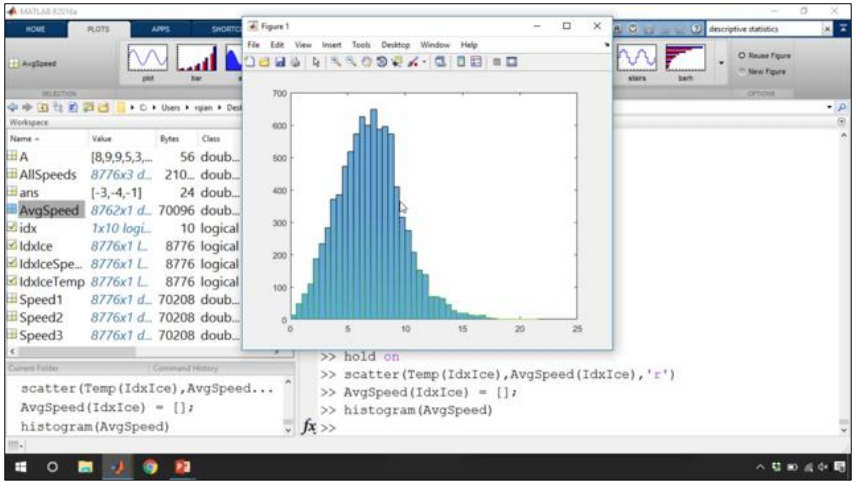
How to build a LSTM

MACHINE LEARNING IN MATLAB

Development environment

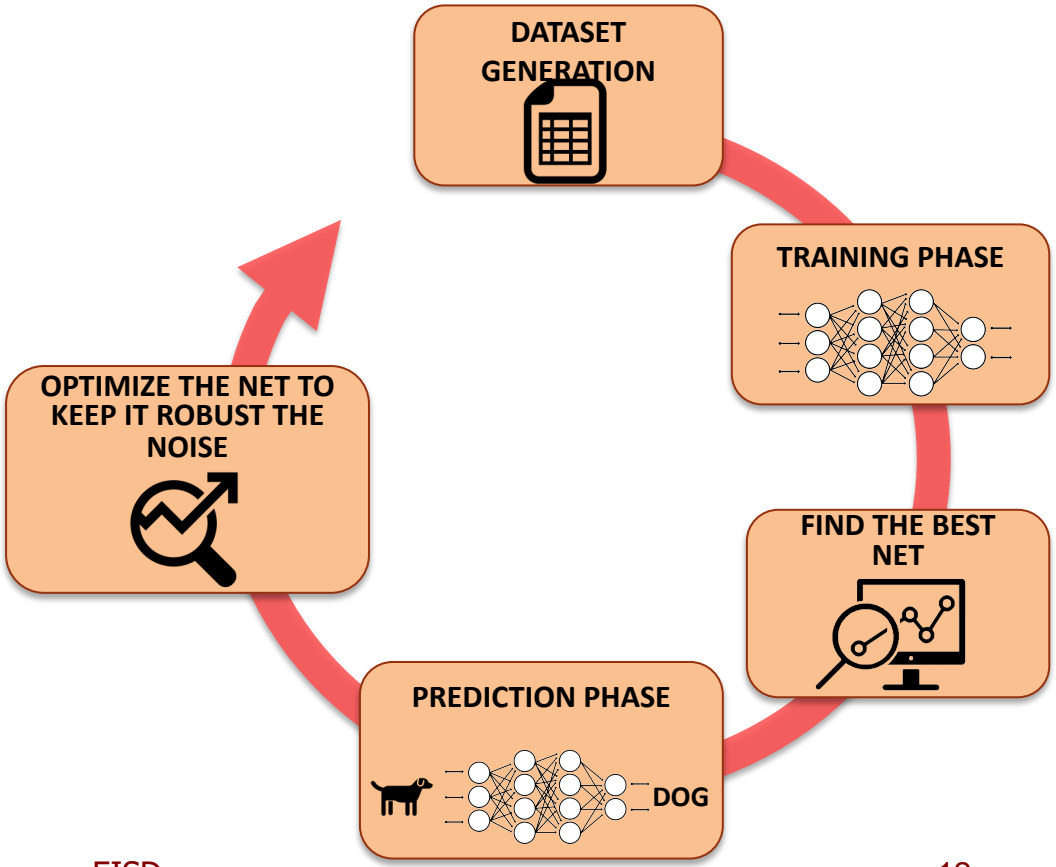
- The **MATLAB** environment
  - Is an environment for numerical calculation and statistical analysis
  - Allows the use of machine learning techniques → Unsupervised techniques
    - Classification Learner App
    - Regression Learner App





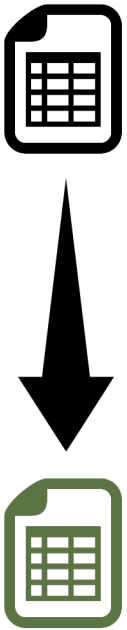
How to build a NET in MATLAB

- Main steps are:
  - Dataset generation
  - Build of the trained NET
  - Prediction phase
  - Optimize the NET



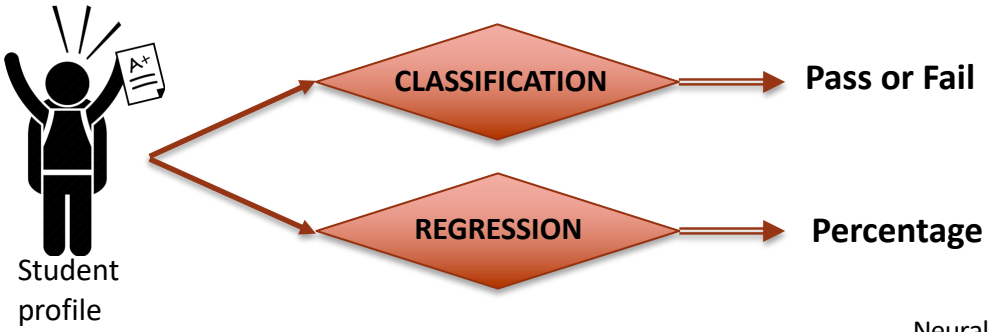
MATLAB: Dataset generation

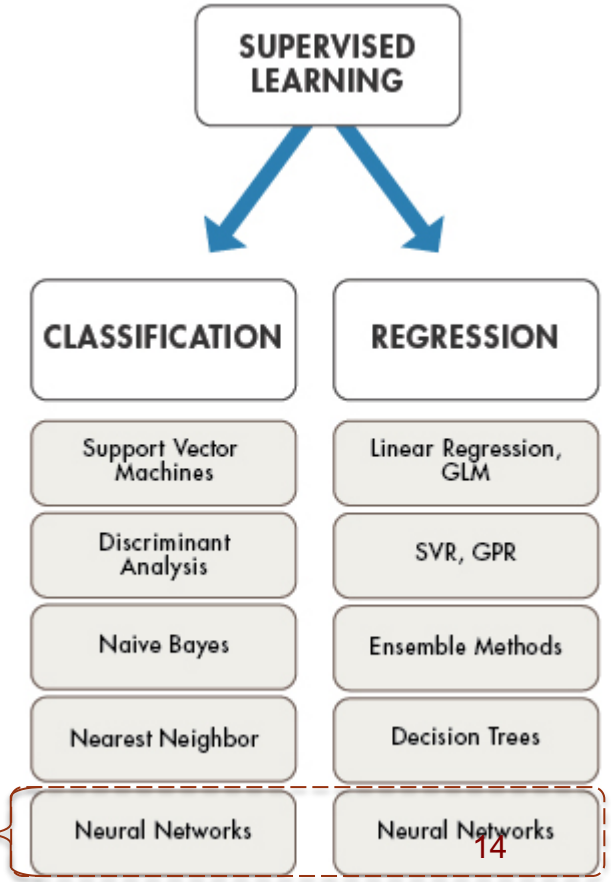
- Preprocess the data
  - Normalizing the data
  - Remove ambiguities
- Extract feauters
  - Find the minimum of feauters that will capture the essential patterns in the data



MATLAB: Training phase

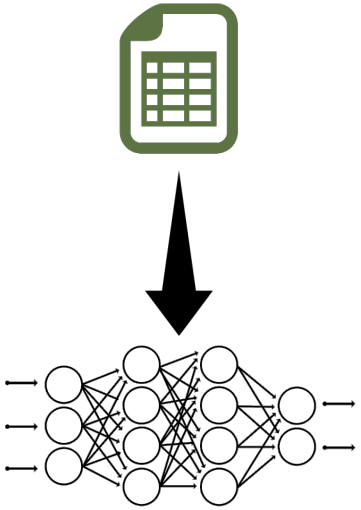
- Supervised Learning
  - CLASSIFICATION techniques predict *categorical* response
    - Support Vector Machines, Discriminant Analysis, Naive Bayes, Nearest Neighbor, Neural Networks
  - REGRESSION techniques predict *continuous* response
    - Linear Regression, Generalized Linear Models(GLM), Support Vector Machine Regression(SVR), Gaussian Process Regression(GPR), Ensemble Methods, Decision Trees, Neural Networks



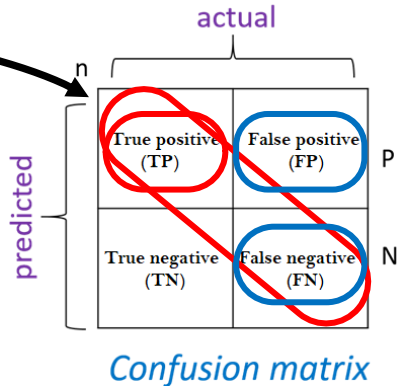


MATLAB: Training phase for classifier

- Two ways
  - Manual method
  - Automatic method
    - *Classification Learner App*



- How to choose the best classifier: use ***confusion matrix***
  - $Accurancy = \frac{\sum_i ConfusionMatrix[i][i] * 100}{\#observation}$
  - $precision = \frac{TP}{TP+FP}$
  - $recall = \frac{TP}{TP+FN}$



Confusion matrix

MATLAB: Training phase

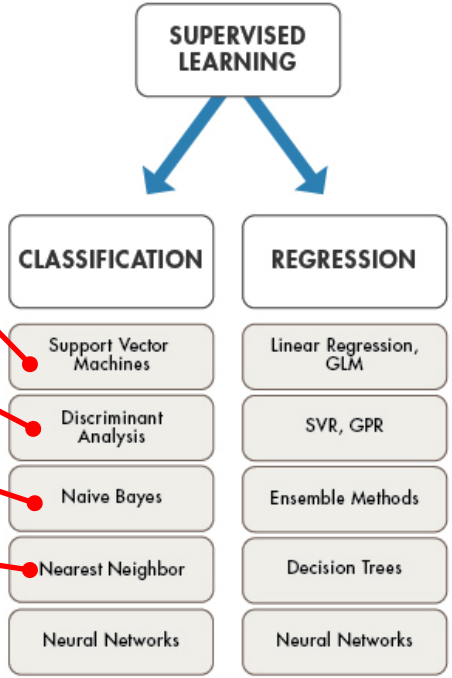
Manual method

fitcsvm, fitcecoc, fitckernel …

Fitcdiscr, …

Fitcnb, …

Fitcknn, …



MATLAB: Training phase for classifier

Manual method

Import Data

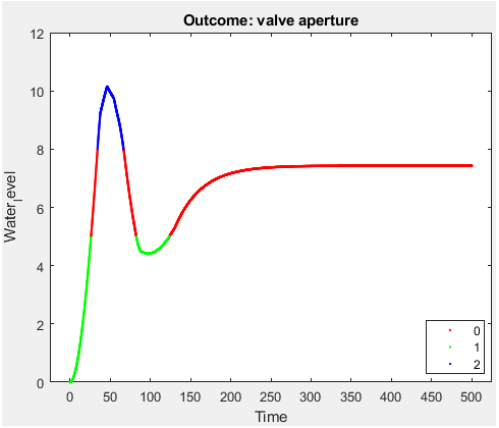
```
waterTank = ImportData('dataser.csv');
% Response
Y = waterTank.OPEN_CLOSE;
% Predictor
X = waterTank.WATER_LEVEL;
```

Partition the data into training set (60%) and test set (40%)

```
cv = cvpartition(length(waterTank),'holdout',0.40);
% Training set and Test set
Xtrain = X(training(cv),:); Xtest = X(test(cv),:);
Ytrain = Y(training(cv),:); Ytest = Y(test(cv),:);
```

Build the various classifiers with their confusion matrix, for example

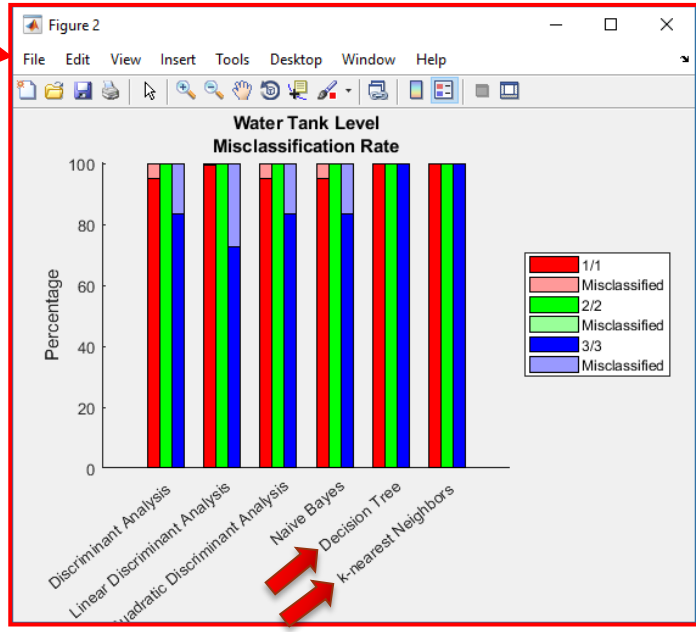
```
%% Linear Discriminant Analysis
lda = fitcdiscr(Xtrain,Ytrain);
     %lda is trained model of type ClassificationDiscriminant
Y_lda = lda.predict(Xtest);
C_lda = confusionmat(Ytest,Y_lda)
C_lda = bsxfun(@rdivide,C_lda,sum(C_lda,2)) * 100
```



MATLAB: Training phase for classifier

Manual method

Compare the confusion matrices graphically



Calculate accurancy

```
summ = 0;
 for i = 1:3
     summ = summ + C_lda(i,i);
 end
 accuracy_lda= summ*100/3;
```

Calculate precision

```
precision_0 = (C_lda(2,2)+C_lda(2,3)+C_lda(3,2)+C_lda(3,3))
              /(C_lda(1,2)+C_lda(1,3)+C_lda(2,2)
                +C_lda(2,3)+C_lda(3,2)+C_lda(3,3));
precision_1 = …
precision_2 = …
precision = precision_0 +  precision_1 + precision_2;
```

Calculate recall

```
recall_0 = (C_lda(1,1))/(C_lda(1,1)+C_lda(2,1)+C_lda(3,1));
recall_1 = …
recall_2 = …
recall = recall_0 + recall_1 + recall_2;
```

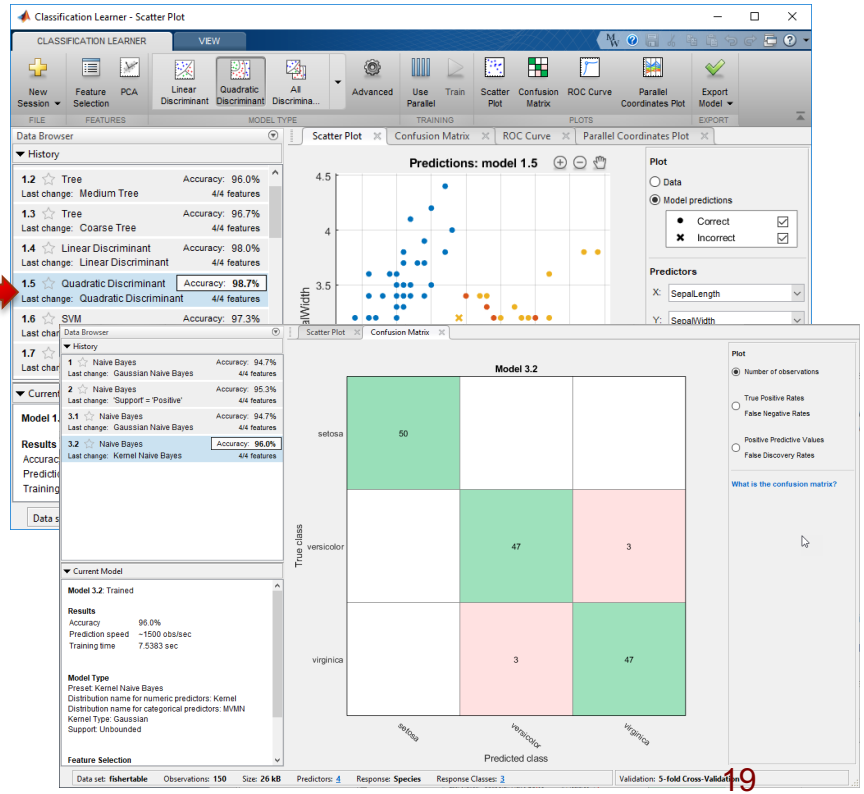
And then choose the best classifier to save in a MAT-file

MATLAB: Training phase for classifier

Automatic method *Classification Learner App*

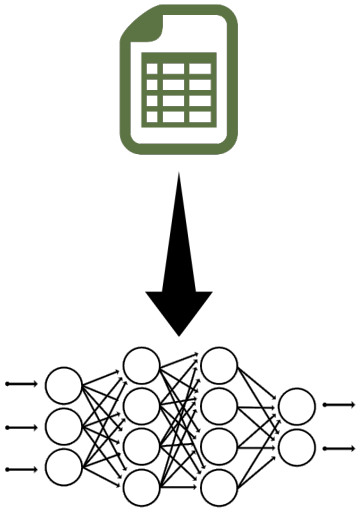
- Import dataset
- Try it on all types of classifiers
- Choose the best classifier
  - Only accuracy is calculated automatically
- Export it in a MAT-file

Precision and recall are very important as well !



MATLAB: Training phase for regressor

- Two ways
  - Manual method
  - Automatic method
    - *Regression Learner App*

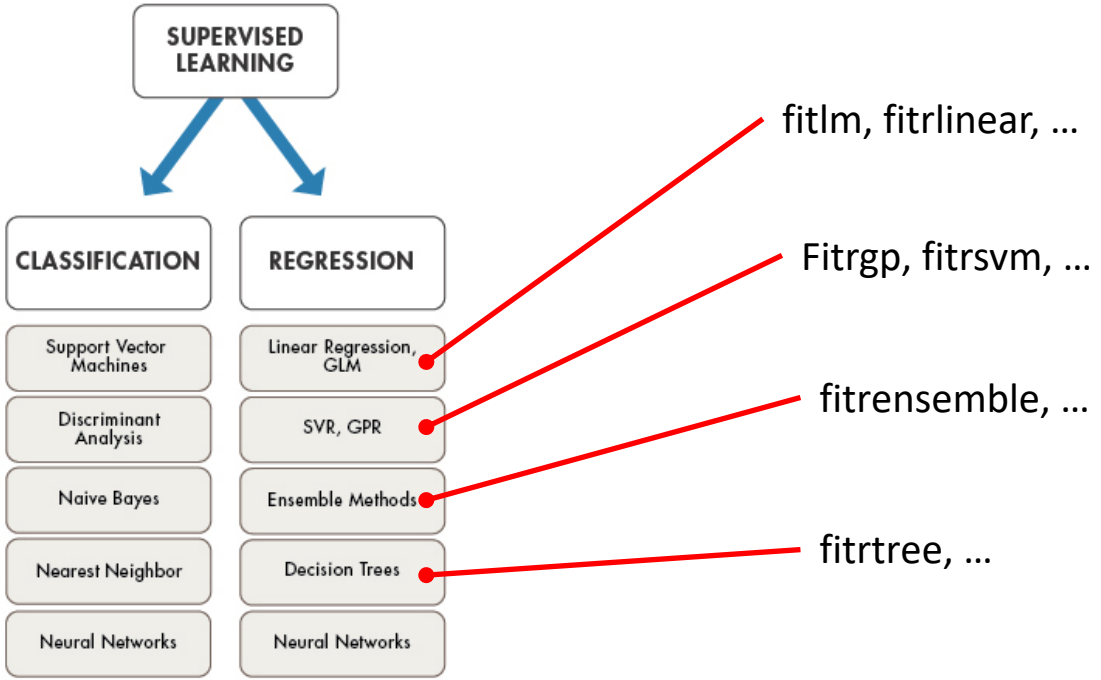


- How to choose the best regressor:
  use **root-mean-square error (RMSE)**

$$RMSE = sqrt(sum(Dates - Scores).\hat{}2)./Dates$$

MATLAB: Training phase

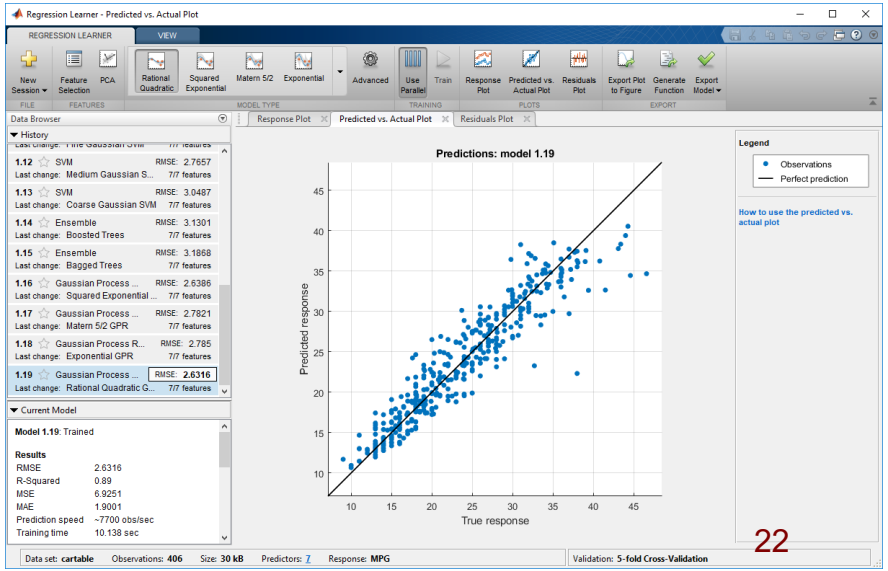
Manual method



MATLAB: Training phase for regressor

Automatic method *Regression Learner App*

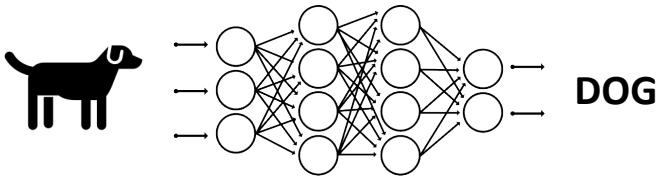
- Import dataset
- Try it on all types of classifiers
- Choose the best classifier
  - Only **root-mean-square error (RMSE)** is calculated automatically
- Export it in a MAT-file



MATLAB: Prediction phase

- After train, predict responses using **predict**

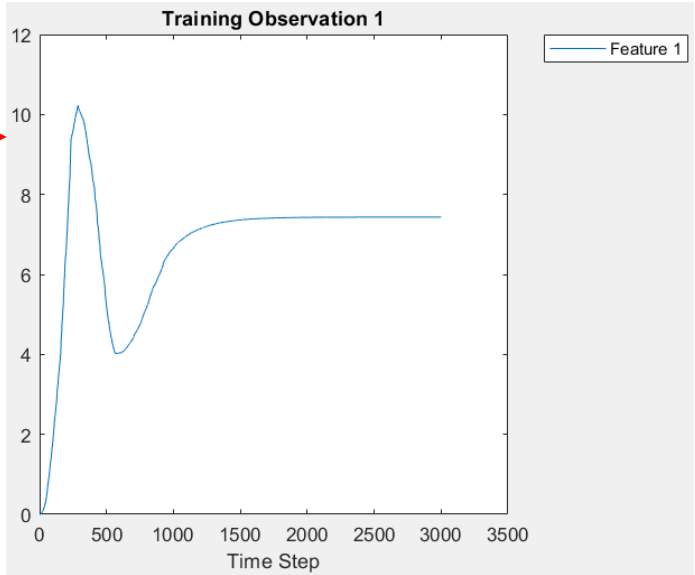
```
y = myClassifier.predict(feature1);
```



How to build a LSTM

Import Data

```
dataset = csvread("dataset_openclose.csv");
% Response
Y = dataset(:,3); %OPEN_CLOSE
% Predictor
X = dataset(:,2); %WATER_LEVEL
```



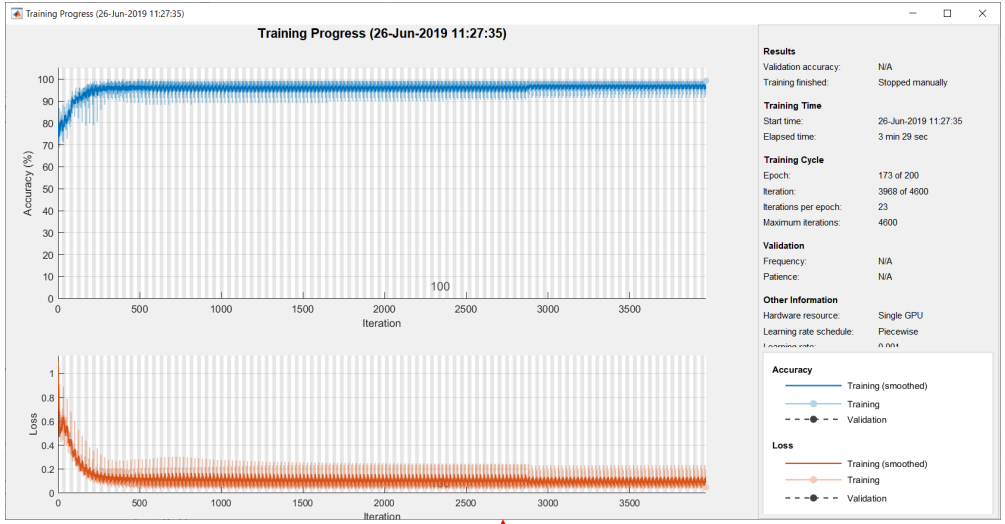
Partition the data into training set (60%) and test set (40%)

```
cv = cvpartition(length(dataset_openclose),'holdout',0.40);
% Training set and Test set
XTrain = X(training(cv),:); XTrain = num2cell(XTrain);
XTest = X(test(cv),:); XTest = num2cell(XTest);
YTrain = categorical(Y(training(cv),:));
YTest = categorical(Y(test(cv),:));
```

How to build a LSTM

Build network

```
%% Define LSTM Network Architecture
inputSize = 1;
numHiddenUnits = 100;
numClasses = 3;  % number of categories

layers = [ ...
    sequenceInputLayer(inputSize)
    bilstmLayer(numHiddenUnits,'OutputMode','last')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer] %regressionLayer

maxEpochs = 100;
miniBatchSize = 27;

options = trainingOptions('rmsprop', ...
    'MaxEpochs',200, ...
    'GradientThreshold',1, ...
    'InitialLearnRate',0.005, ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropPeriod',125, ... % default 10
    'LearnRateDropFactor',0.2, ... % default 0.1
    'Verbose',0, ...
    'Plots','training-progress')
```




```
%% Train LSTM Network
net = trainNetwork(XTrain,YTrain,layers,options);
```

How to build a LSTM

Prediction phase

```
%% Prediction
YPred = classify(net,XTest, ...
    'MiniBatchSize',miniBatchSize, ...
    'SequenceLength','longest');
```

Calculate accuracy

```
%% Calculate the classification accuracy of the
predictions
accuracy = sum(YPred == YTest)./numel(YTest)
```

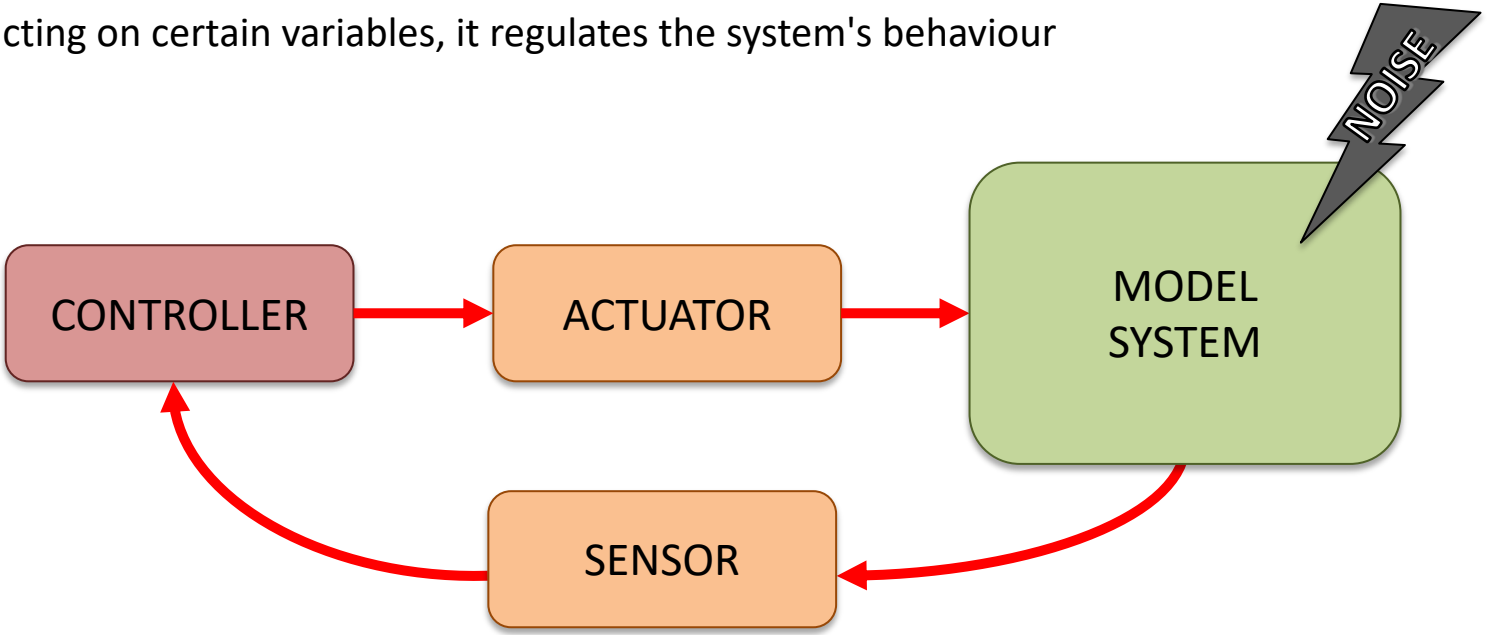
```
accuracy =

    0.9590

>> tabulate(YPred)
  Value    Count   Percent
      0     1571     78.55%
      1      290     14.50%
      2      139      6.95%
>> tabulate(YTest)
  Value    Count   Percent
      0     1557     77.85%
      1      304     15.20%
      2      139      6.95%
```

Methodology

MACHINE LEARNING IN CONTROL SYSTEM

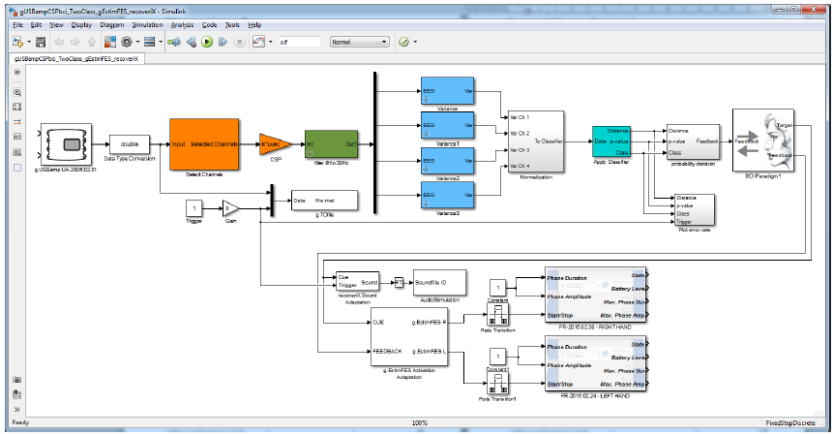
Control system

- A control system manages, commands, directs, or regulates the behavior of other devices or systems using control loops
  - acting on certain variables, it regulates the system's behaviour



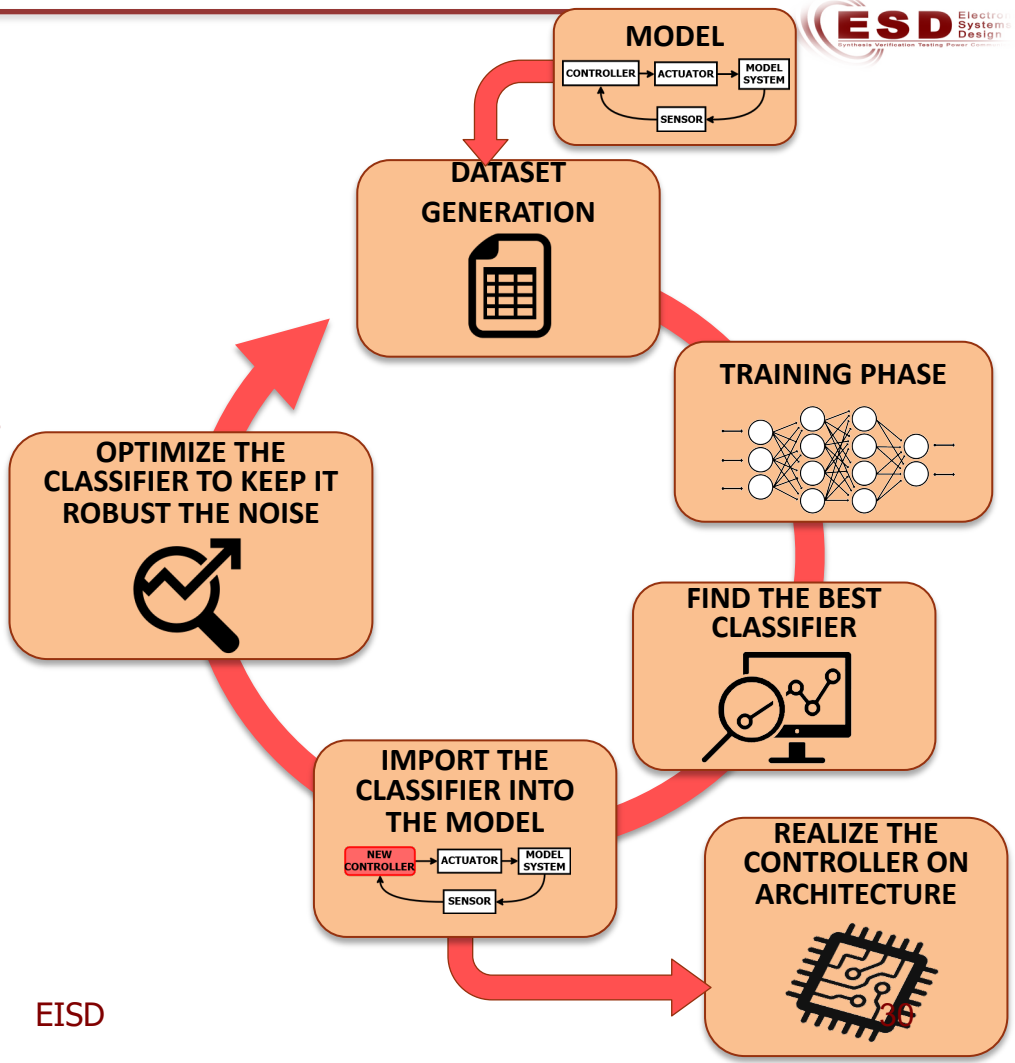
Development environment

- The **Simulink** environment
  - Allows construction of the simulation environment for the hardware and software design of an system
  - Is closely integrated with MATLAB



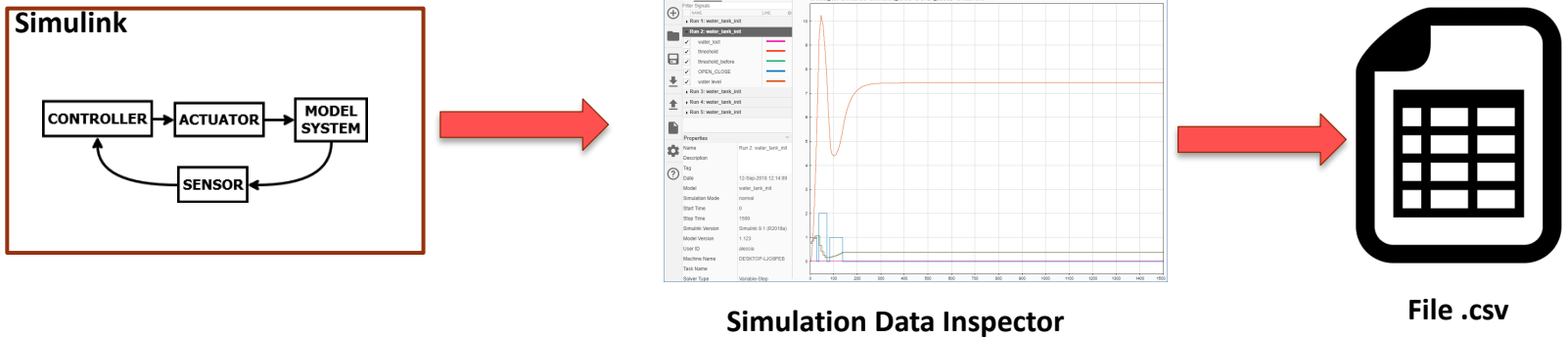
Methodology

- Main steps are:
  - Export the dataset
  - Build of the trained classifier
  - Implement in system
  - Optimize the classifier
  - Generate the code



Export dataset

- Use Simulation Data Inspector
  - Observe the interested signals
  - Export and save them in the Matlab workspace
  - Export the data in a CSV-file

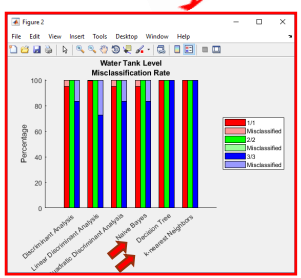


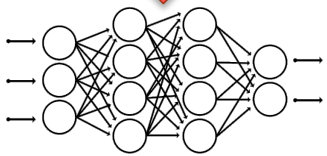
Build the trained classifier

Manual method

Statistics and Machine Learning Toolbox

- Import Dataset
- Build the various classifiers with their confusion matrix
- Compare the confusion matrices graphically
- Or calculate accuracy, precision and recall and choose the best classifier



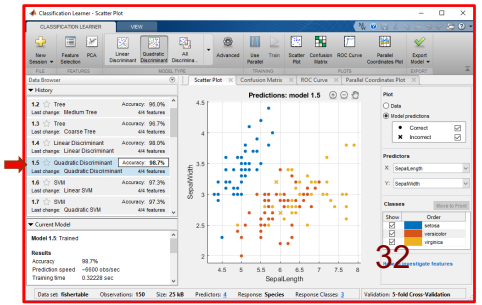


NEW CONTROLLER

Automatic method

Classification Learner App

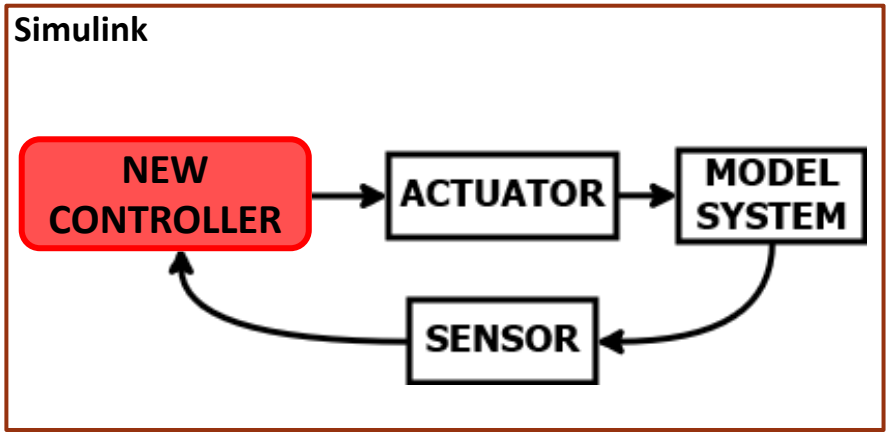
- Import dataset
- Try it on all types of classifiers
- Choose the best classifier



Implementation in the system

- Put the classifiers in
  - **M-function** blocks
  - **S-function** blocks
    - Simulating the **same result** is obtained with the M-functions
    - The S-function is **faster** than M-function

```
function OPEN_CLOSE = fcn(u1, u2, …)
persistent mdl;
if isempty(mdl)
    mdl = loadCompactModel('my_classifier.mat');
end
OPEN_CLOSE = int8(predict(mdl,[u1, u2, …]));
end
```

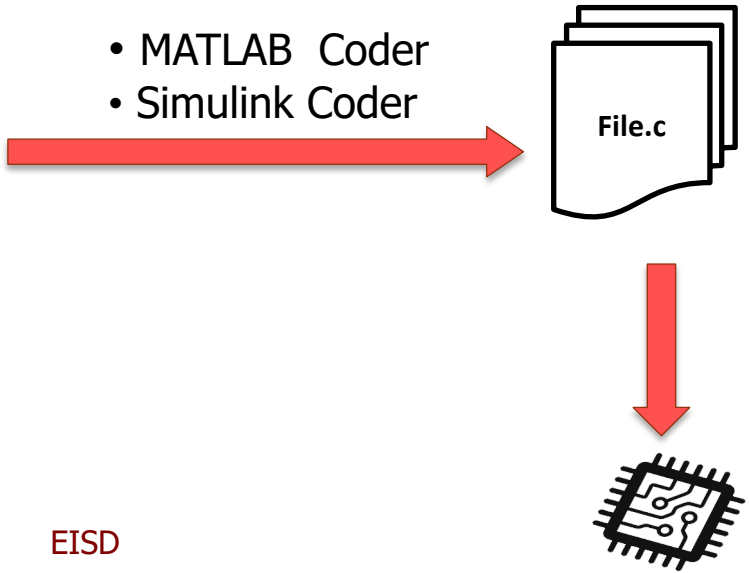


Code generation

- From M-file to C-file
  - MATLAB Coder
  - Simulink Coder

```
function OPEN_CLOSE = fcn(u1, u2, …)
persistent mdl;
if isempty(mdl)
    mdl = loadCompactModel('my_classifier.mat');
end
OPEN_CLOSE = int8(predict(mdl,[u1, u2, …]));
end
```

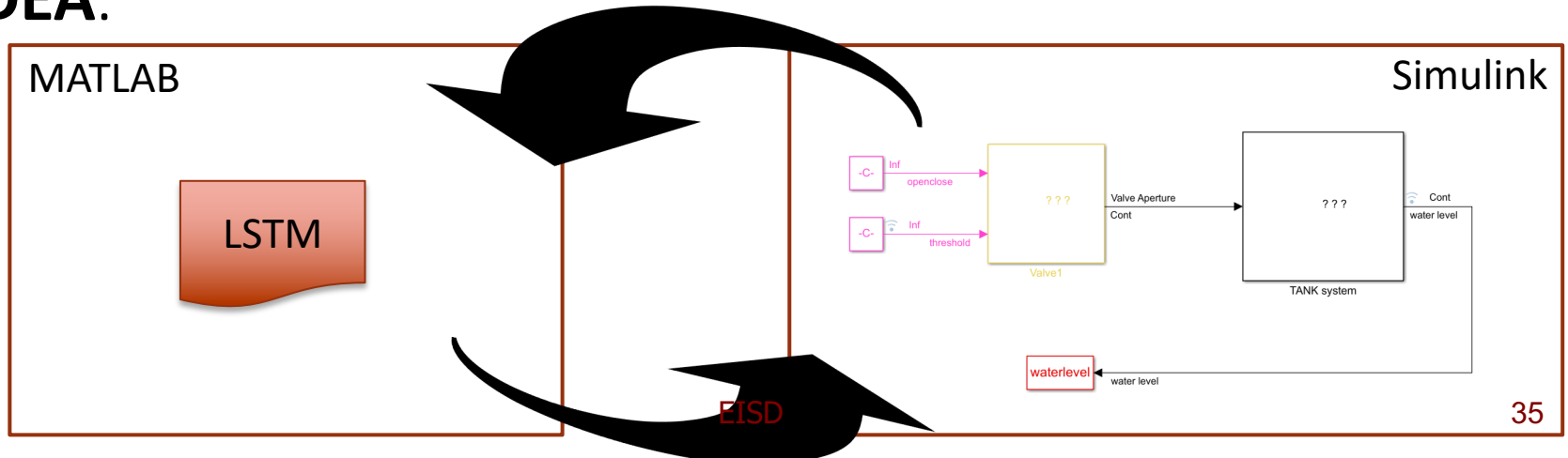
classifyX.m



LSTM limitations in MATLAB

- LSTM in MATLAB code
  - cannot be imported in Simulink
  - cannot be exported in C/C++

IDEA:

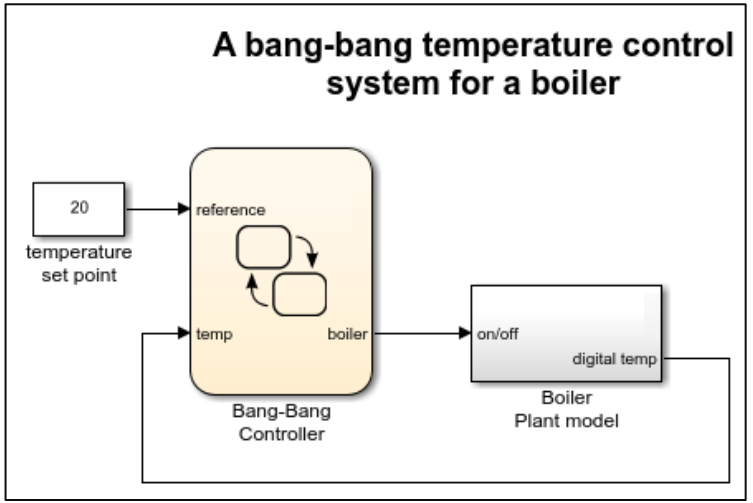


1. Bang bang controller
2. PID controller
3. Water tank controller

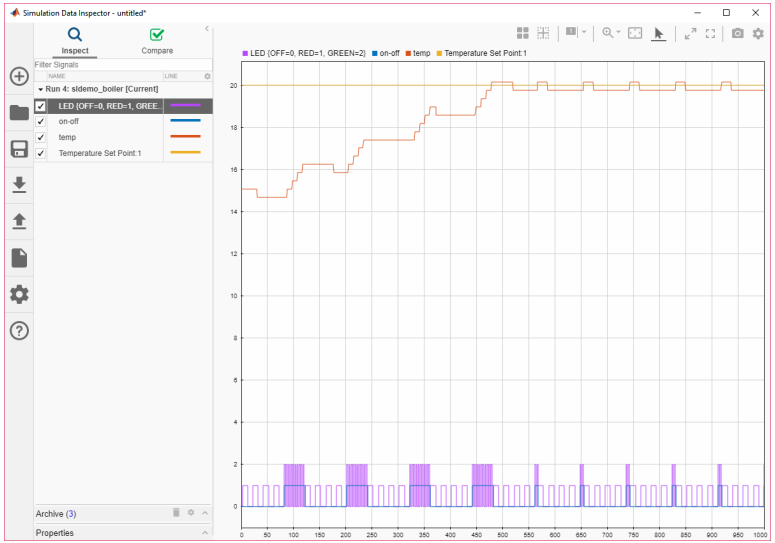
EXPERIMENTAL RESULTS

Bang bang controller

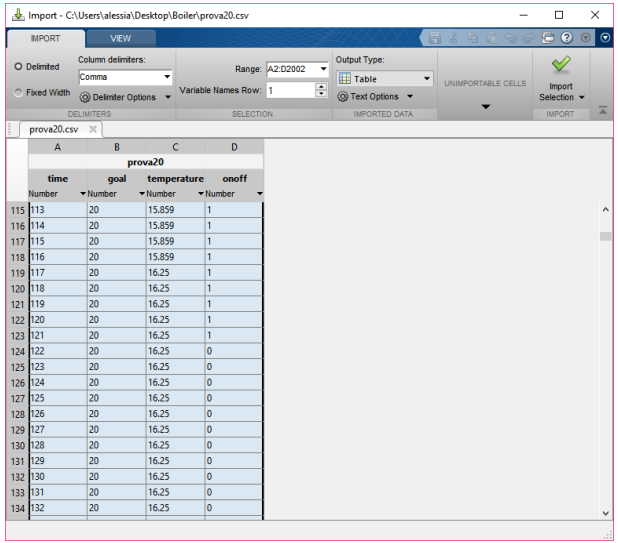
- The Bang-Bang control-boiler model is composed of
  - **Controller**
    - Inputs
      - Temperature set point
      - Digital temperature
    - Outputs
      - ON-OFF $\begin{cases} 0 & \text{the boiler is activated to warm up} \\ 1 & \text{the boiler is switched off} \end{cases}$
  - **Boiler plant model**



Bang bang controller



csvwrite()



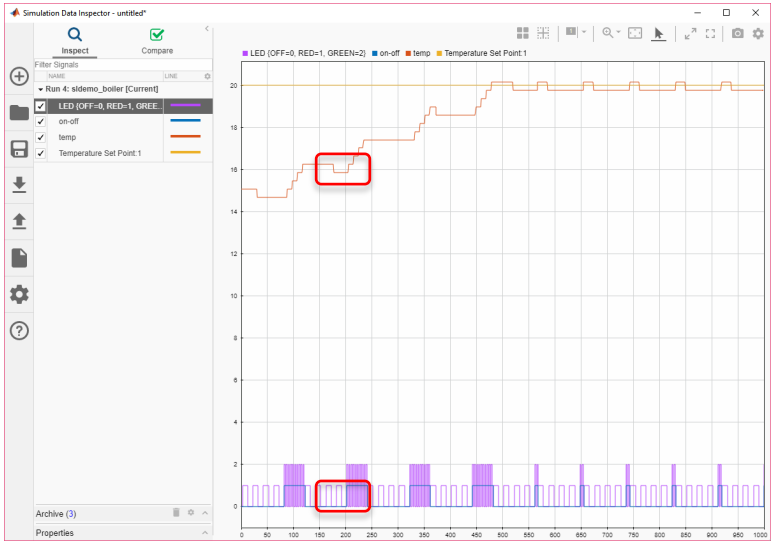
Simulation Data Inspector

File.csv

Bang bang controller

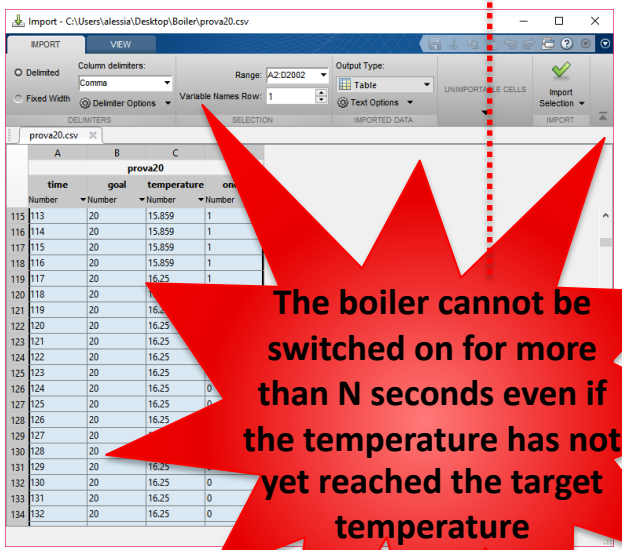
- Problem n. 1

it looks like the boiler is off for 80 seconds, then turns on for up to 40 seconds and then back off for another 80 seconds



Simulation Data Inspector

csvwrite()



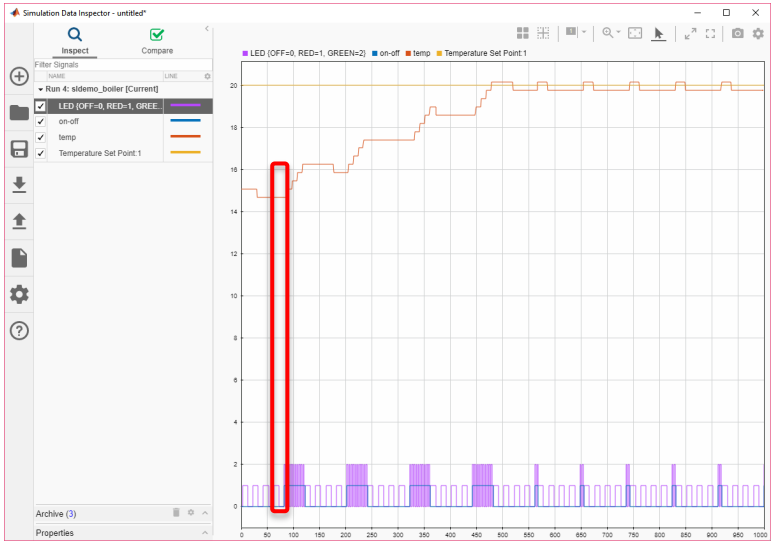
The boiler cannot be switched on for more than N seconds even if the temperature has not yet reached the target temperature

F csv

Bang bang controller

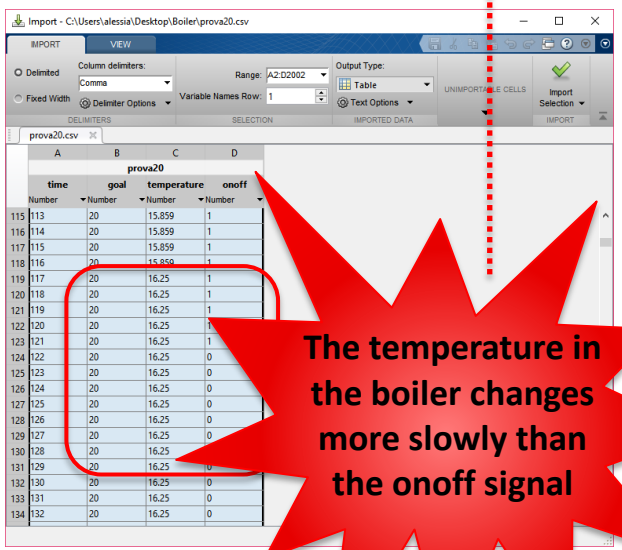
- Problem n. 2

When goal = 20 and temperature = 16.25, onoff signal is 1 or 0?



Simulation Data Inspector

csvwrite()

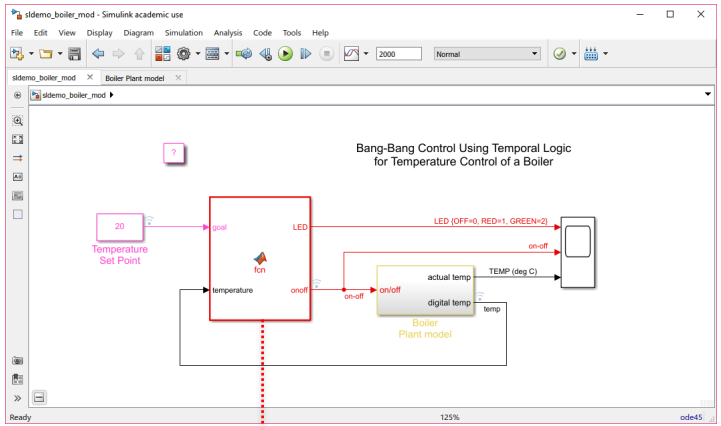


File .csv

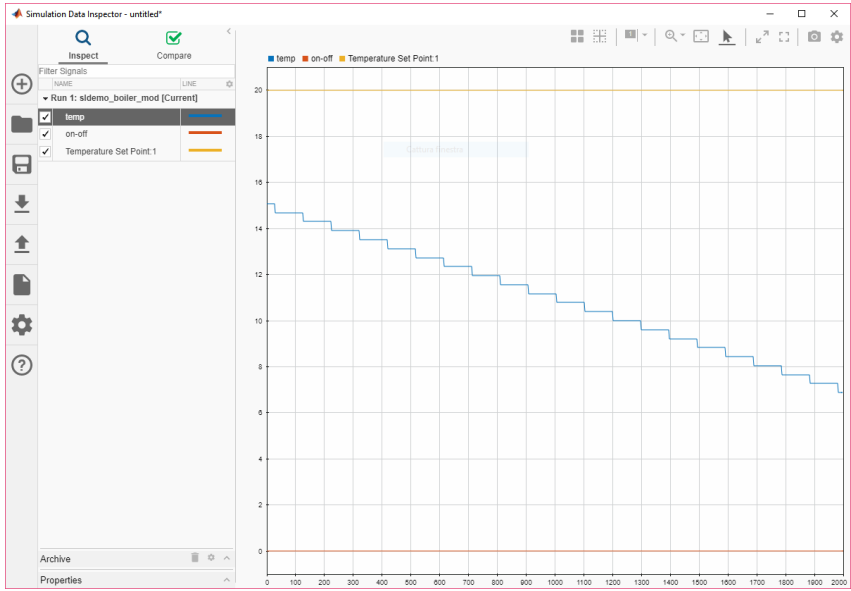
The temperature in the boiler changes more slowly than the onoff signal

Bang bang controller

- It is necessary to build an ad hoc dataset otherwise it does not work!



Simulation

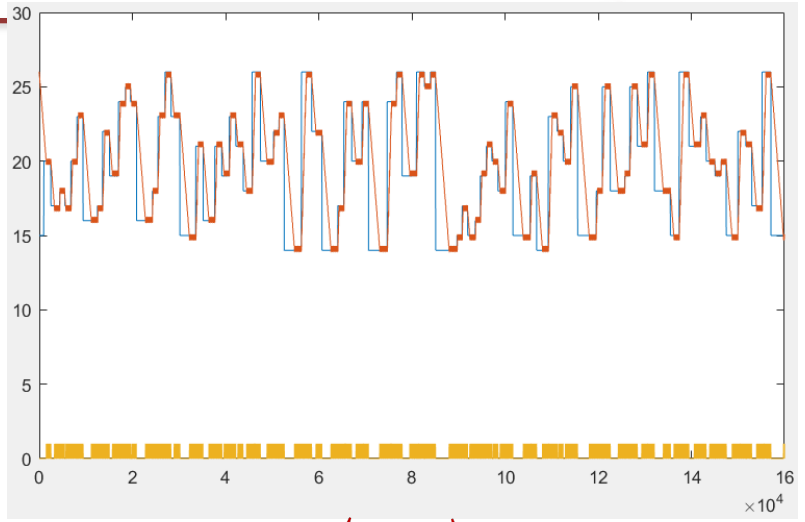



```
function [LED, onoff] = fcn(goal, temperature)
    persistent mdl;
    mdl = loadCompactModel('fineTree882.mat');
    u = [goal double(temperature)];
    LED = int8(0);
    onoff = int8(predict(mdl,u));
end
```

Training dataset

- Build initial dataset, by observing the signals
  - Temperature goal signal
  - Actual temperature signal
  - Onoff signal
- Then, the training dataset is composed as follows
  - Count_0: how much the on-off signal is 0
  - Count_1: how much the on-off signal is 1
  - $\begin{cases} diff: actualtemperature - goal \\ signdiff: \begin{cases} -1 & if\ actualtemperature - goal < 0 \\ 1 & otherwise \end{cases} \end{cases}$
  - Onoff: Controller's output

Rows: 160000

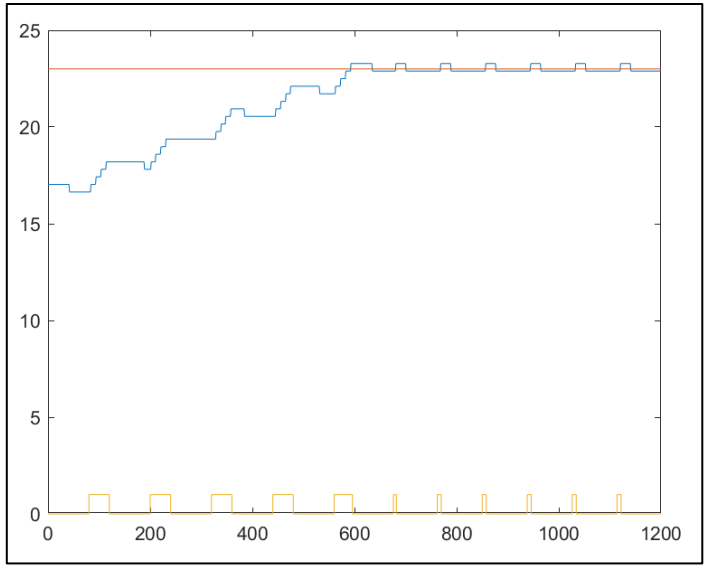


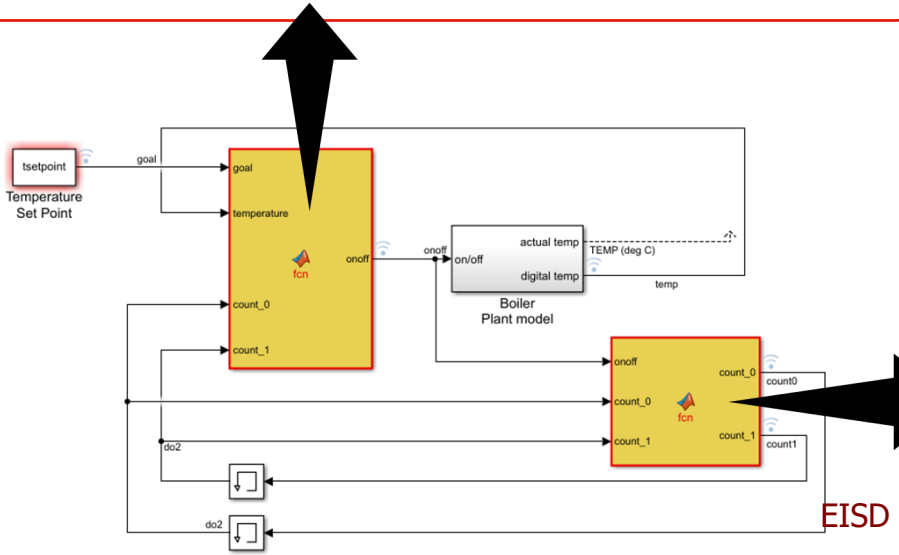
count_0	count_1	diff	onoff
Number	Number	Number	Number
75	0	-0.234	0
76	0	-0.234	0
77	0	-0.234	0
78	0	-0.234	0
79	0	-0.234	0
80	0	-0.234	1
0	1	-0.234	1
0	2	-0.234	1
0	3	-0.234	1

count_0	count_1	sign_diff	onoff
Number	Number	Number	Number
75	0	-1	0
76	0	-1	0
77	0	-1	0
78	0	-1	0
79	0	-1	0
80	0	-1	1
0	1	-1	1
0	2	-1	1
0	3	-1	1

Classifiers C.L.App and dataset with sign

```
function onoff  = fcn( goal, temperature, count_0, count_1)

    persistent mdl;
    u = [double(count_0), double(count_1), sign(double(temperature)-goal)];
    if isempty(mdl)
        mdl = loadCompactModel('classifier.mat'); %Boosted Trees
    end
    onoff = int8(predict(mdl,u));

end
```



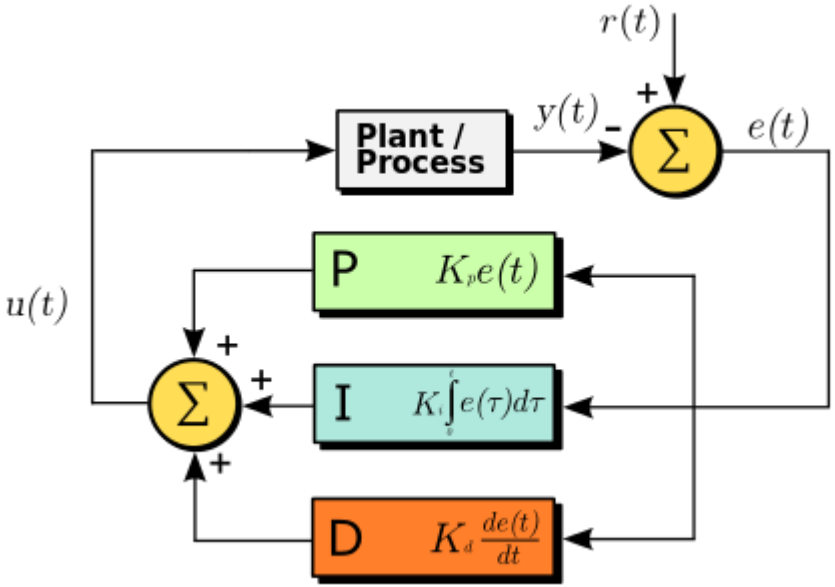


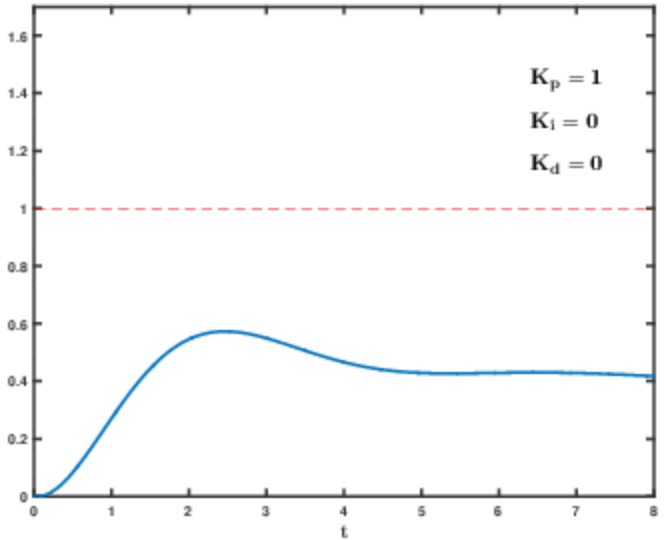

```
function [count_0, count_1] = fcn(onoff, count_0, count_1 )
    if onoff == 0
        count_0 = count_0 + 1;
        count_1 = 0;
    else
        count_0 = 0;
        count_1 = count_1 + 1;
    end
end
```

EISD

PID

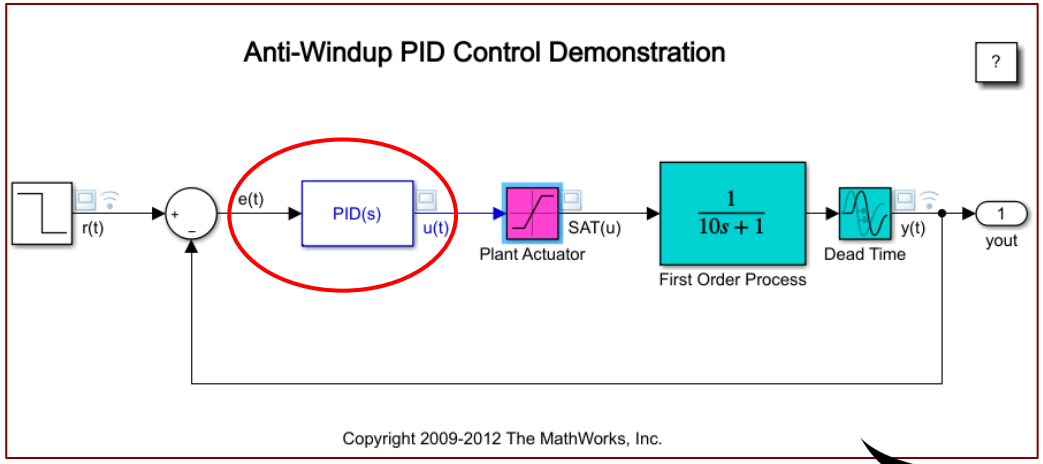
- The **proportional–integral–derivative (PID) controller** acquires a value as input from a process and compares it with a reference value. The difference, the so-called error signal, is then used to determine the value of the controller output variable, which is the manipulable variable of the process.

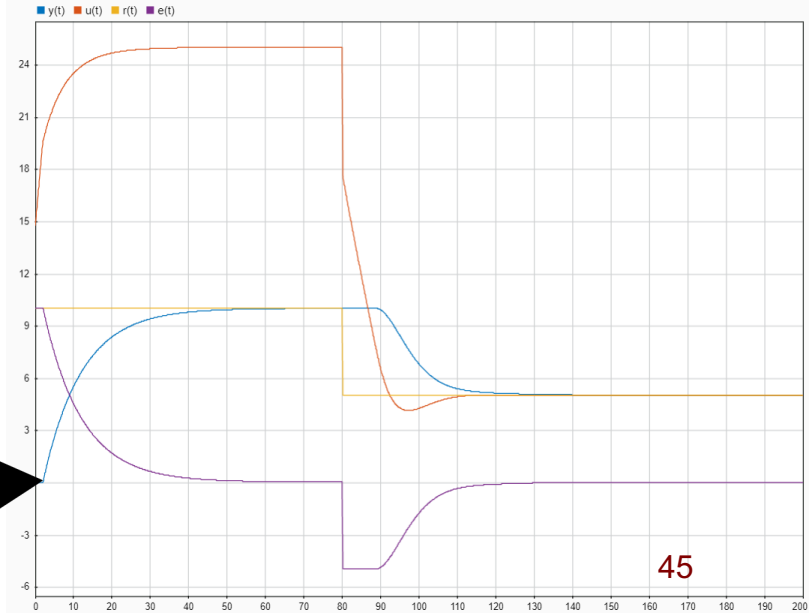




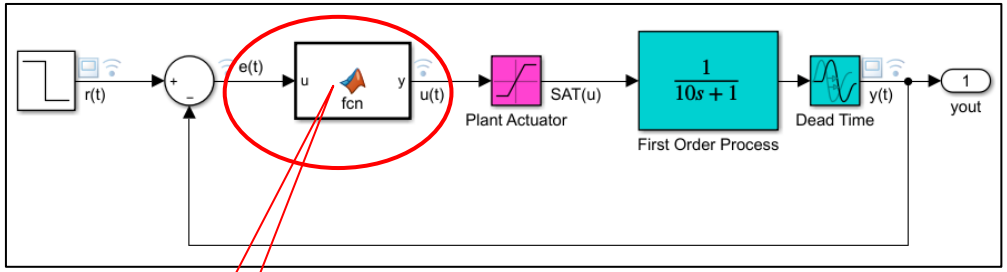
PID

- This example shows how to use anti-windup schemes to prevent integration wind-up in PID controllers when the actuators are saturated
  - https://it.mathworks.com/help/simulink/slref/anti-windup-control-using-a-pid-controller.html



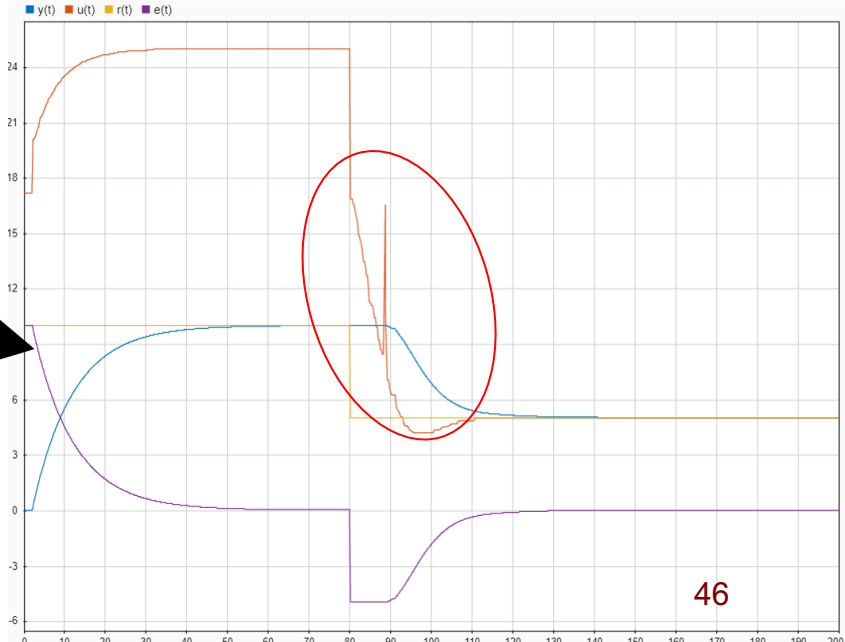


PID example: PID controller

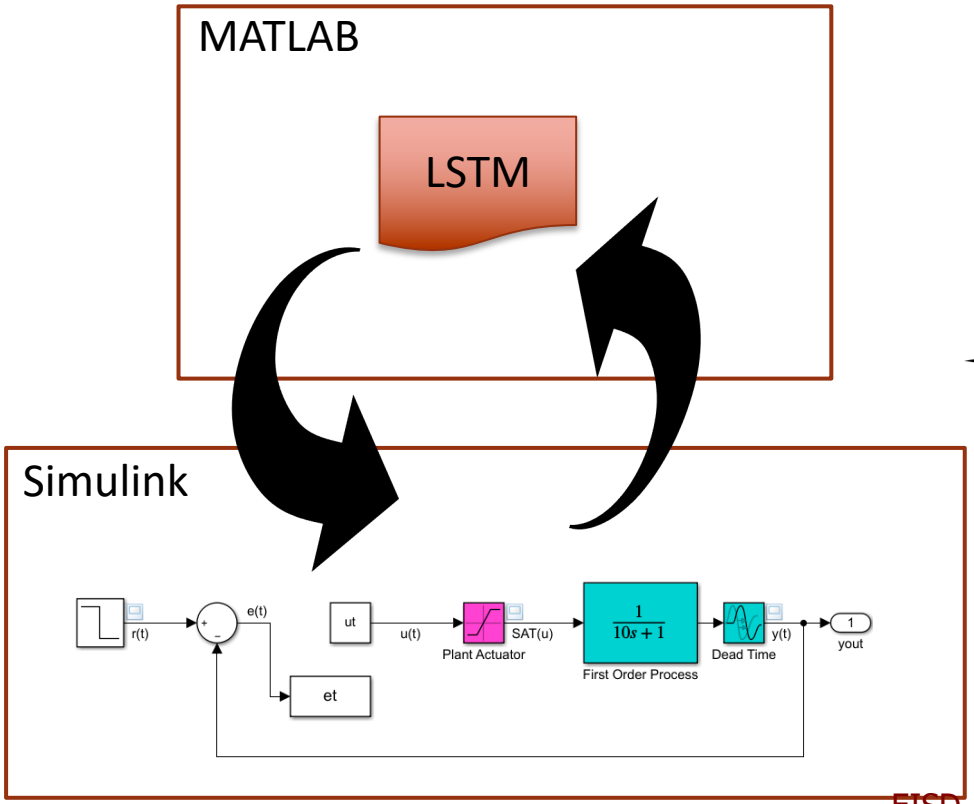


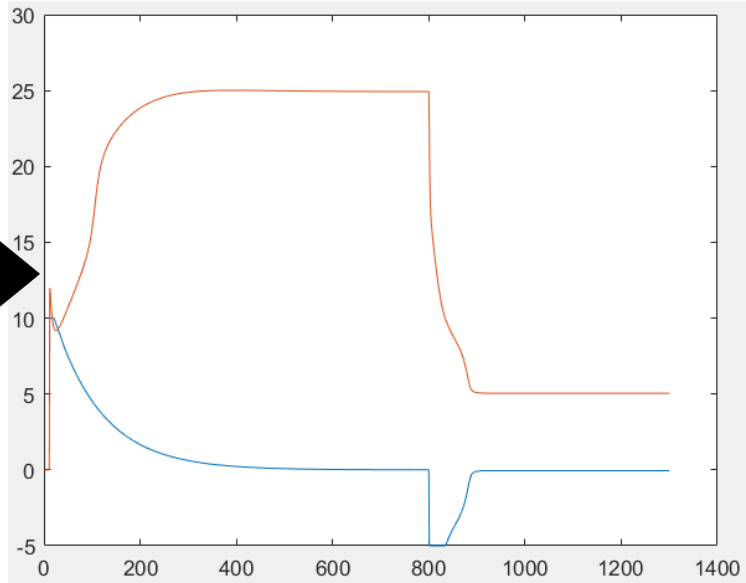

```
function y = fcn(u)
mdl = loadCompactModel(«myRegressor.mat");

y = predict(mdl,u);
```

Bagged tree regressor with Regression Learner App



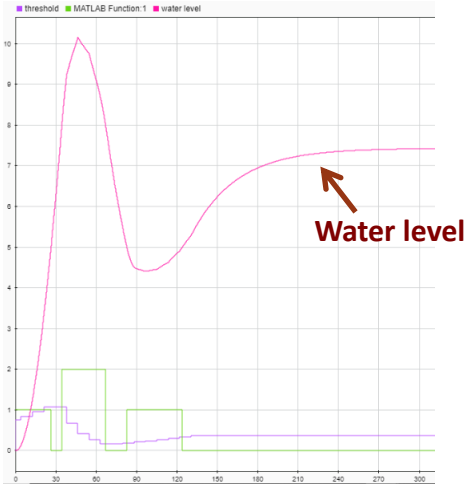
PID simulation

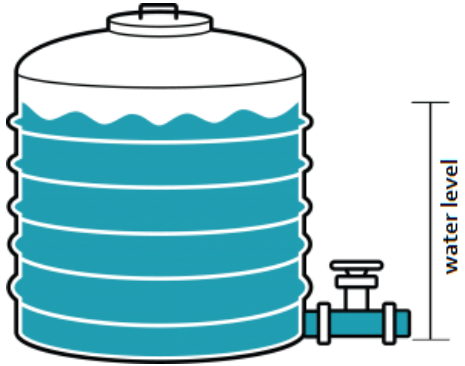


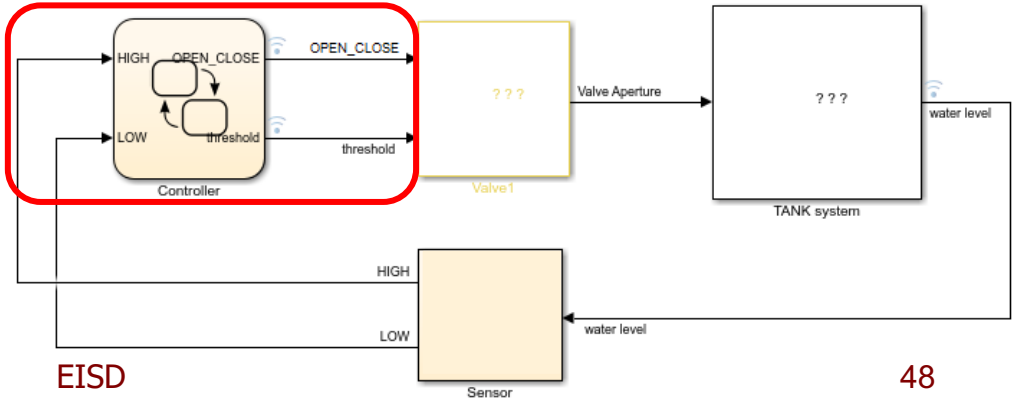


Water tank

- Water tank is a tank that leaks water
- Controller manages the opening of the valve
  - Water level is between two values
  - The signals to be managed are
    - OPEN_CLOSE $\begin{cases} 0 & 5.5 \leq waterLevel \leq 8 \\ 1 & waterLevel < 5.5 \\ 2 & waterLevel > 8 \end{cases}$
    - Threshold → maximum valve opening value

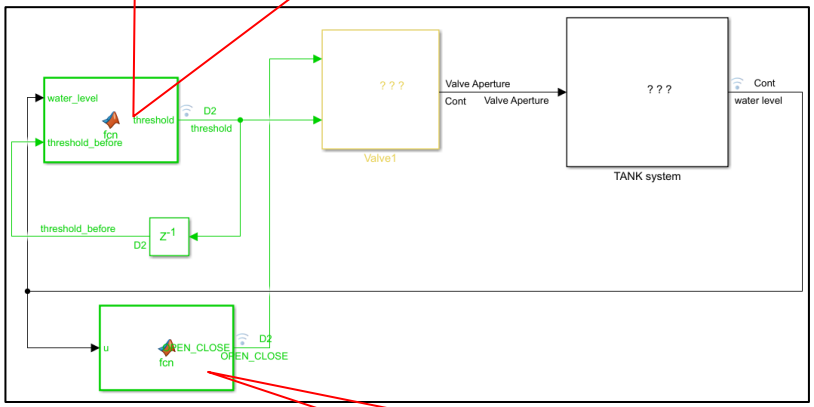






Water tank simulation with regressor

```
function threshold  = fcn(water_level, threshold_before)
persistent mdl;
init_threshold = 0.75;
if isempty(mdl)
     mdl = loadCompactModel('medTree0018.mat');
     threshold = init_threshold;
else
    X = [water_level threshold_before];
    threshold = double(mdl.predict(X));
end
end
```

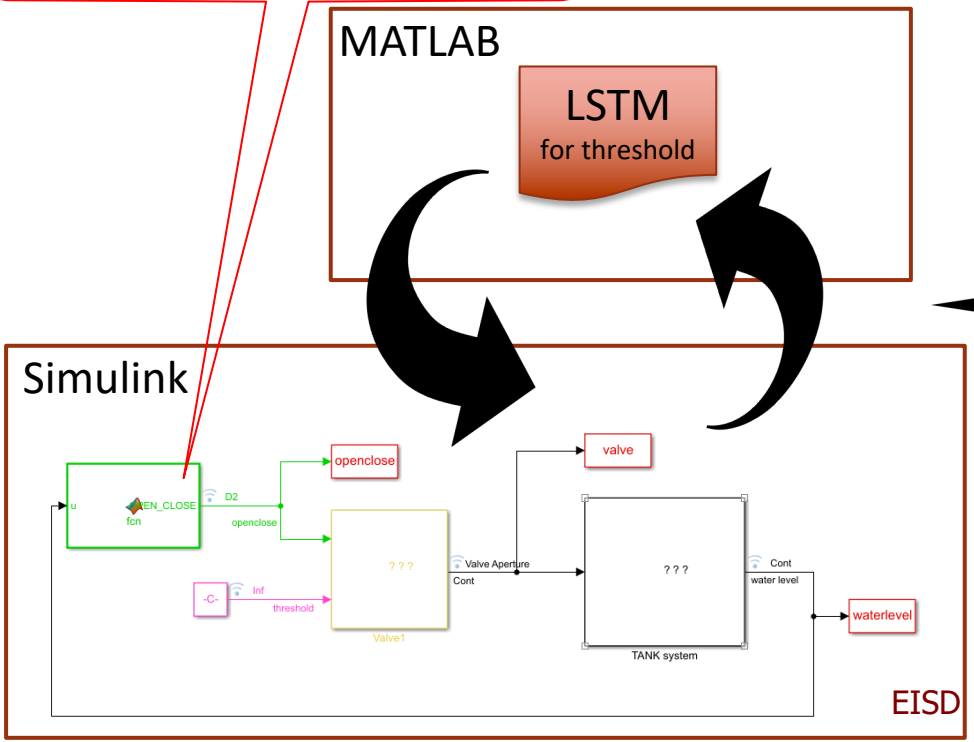


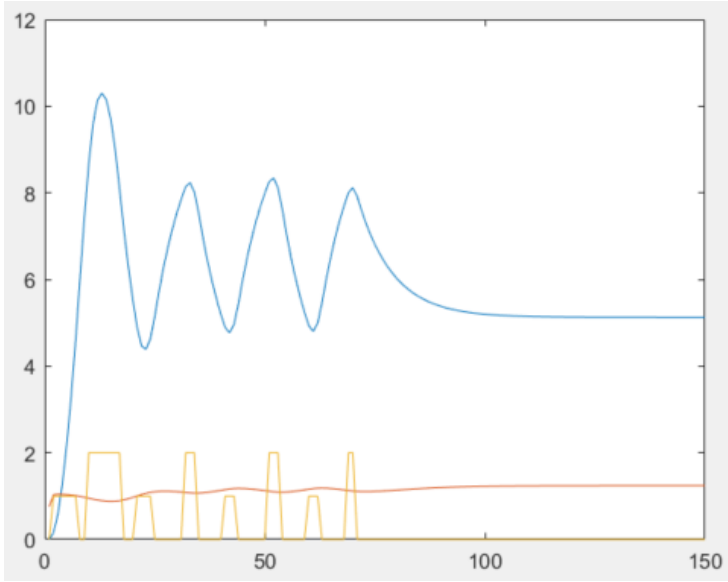

```
function OPEN_CLOSE = fcn(u)
persistent mdl;
if isempty(mdl)
    mdl = loadCompactModel('my_classificatorOC.mat');
end
OPEN_CLOSE = int8(predict(mdl,u));
end
```

EISD

Water tank simulation with LSTM

```
function OPEN_CLOSE = fcn(u)
persistent mdl;
if isempty(mdl)
    mdl = loadCompactModel('myClassifier.mat');
end
OPEN_CLOSE = int8(predict(mdl,u));
end
```





Objectives

- Assumptions
  - Machine learning is often used in those fields of computer science where designing and programming explicit **algorithms is impractical**
  - **Building a control system is not easy** and it is necessary to have a engineer specializing who has an extensive knowledge in all areas of engineering, and an overview of the physical laws
  - The controller **code** of a system is **not always accessible**

- **PROPOSED SOLUTION**
  - Replicate the behavior of an embedded control system that is not known how it works, with **supervised** machine learning techniques

Objectives

- Assumptions
  - The **MATLAB/Simulink** environment allows
    - [Simulink] construction of the simulation environment for the hardware and software design of an embedded system
    - [Matlab] use of machine learning techniques

- Is it possible to implement a controller with machine learning techniques in MATLAB/Simulink environment?

References

- *"Understanding LSTM Networks"*, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

- The MathWorks, Inc.,"*Generate Code Using Simulink® Coder™*", https://it.mathworks.com/help/rtw/examples/generating-code-using-simulink-coder.html