Predictive Maintenance

Machine Learning, MQTT and json/protobuf

Fondazione Bruno Kessler

Marco Roveri

roveri@fbk.eu

Outline

- Recap on Predictive Maintenance
- Introduction to machine learning in python
- Introduction to MQTT and json/protobuf in python

Outline

- Recap on Predictive Maintenance
- Introduction to machine learning in python
- Introduction to MQTT and protobuf in python

Condition Based Maintenance

- The **goal of CBM** is to **spot upcoming equipment failures** so maintenance can be **proactively scheduled when it is needed**.
- Asset conditions need to **trigger maintenance** with a **long enough time period before failure**.
  - So work can be finished before the asset fails or performance falls below the optimal level

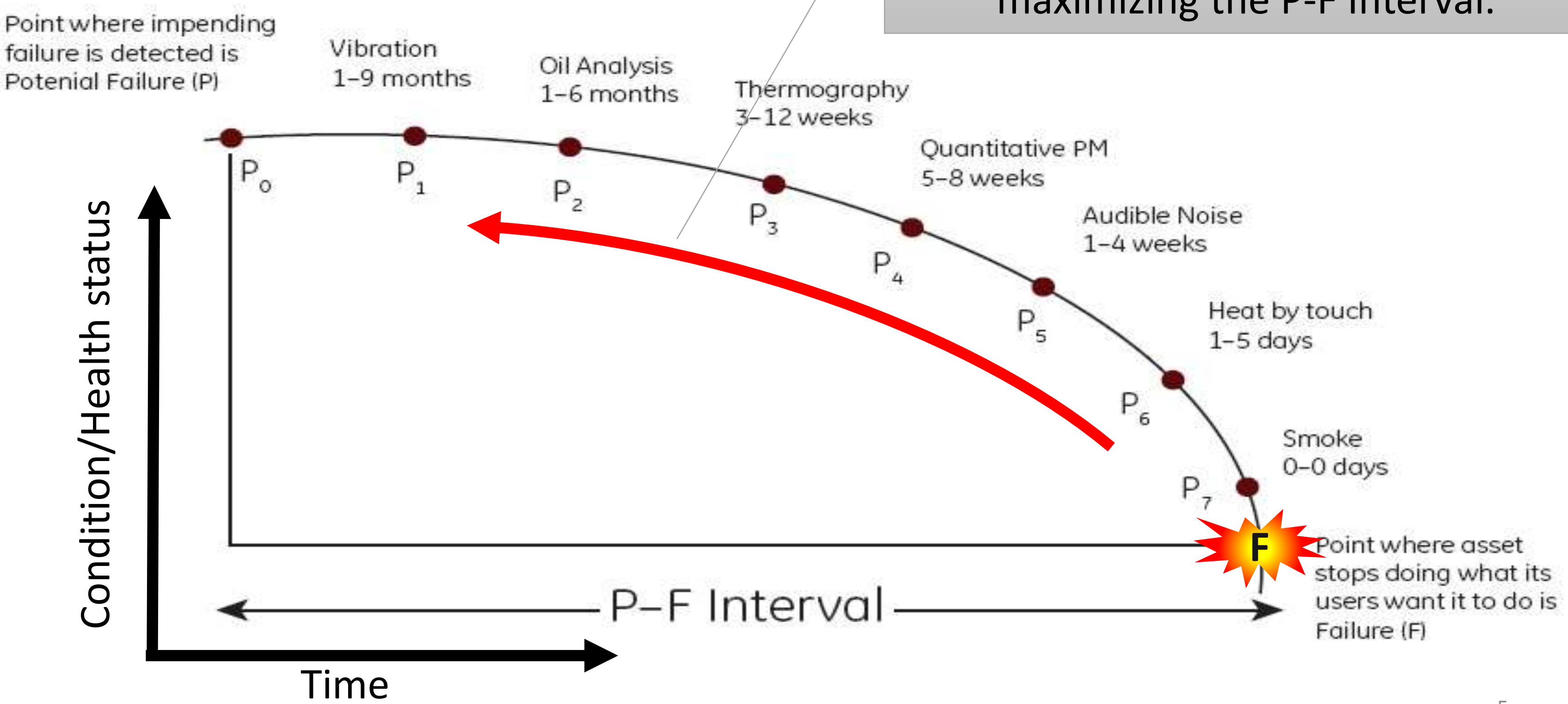
CBM = Cost Savings + Higher system reliability

- CBM allows preventive and **corrective actions** to be **scheduled at the optimal time**, thus **reducing** the **total cost** of ownership.
- Today **improvements in technology** are making it easier to **gather**, **store**, and **analyze data** for CBM.
- CBM is highly effective where safety and reliability is the paramount concern such as **Industry 4.0** aware plants, aircraft industry, semiconductor manufacturing, nuclear, oil and gas, and so on

Potential-Failure Curve

- The P-F curve chart is one of the most important tools for reliability centered maintenance plan.

CBM move point P (potential failure) to the earliest time possible maximizing the P-F Interval.



Predictive maintenance

- Relies on maintenance based on trends acquired by equipment data.
- It is based on predicting when an asset needs attention rather than simply replacing a part, when it could last longer.
- It deals with online data
  - machine conditions are constantly monitored
  - data is constantly analyzed.

Data acquisition

- Data are acquired from the monitored equipment.

Data processing

- The acquired data are validated and transformed properly according to the requirements of the decision making techniques.

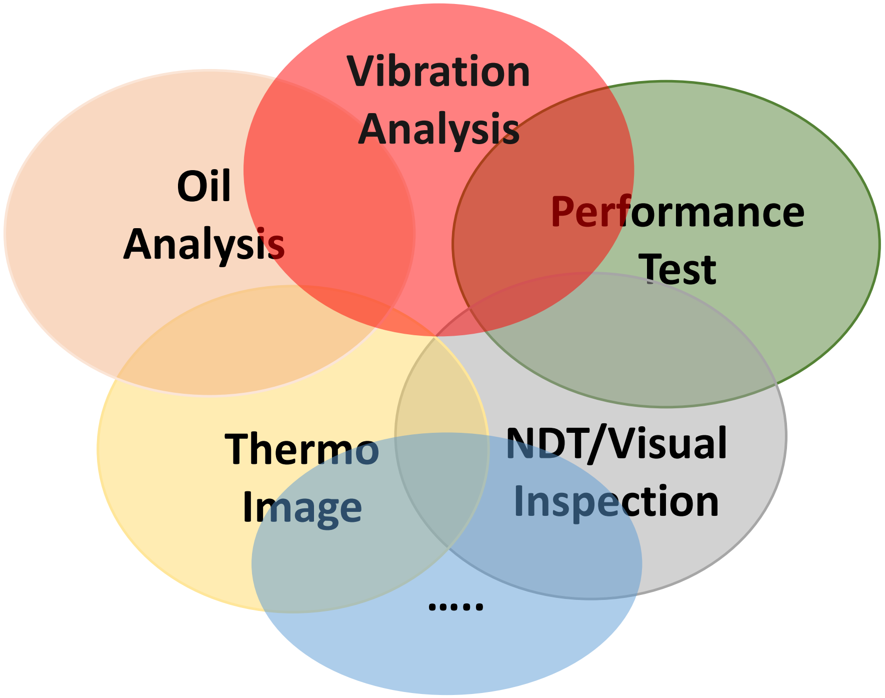
Decision Making

- A number of methods and techniques are applied to obtain the current health condition of the monitored equipment and recommend maintenance plans.

Predictive Maintenance Tools

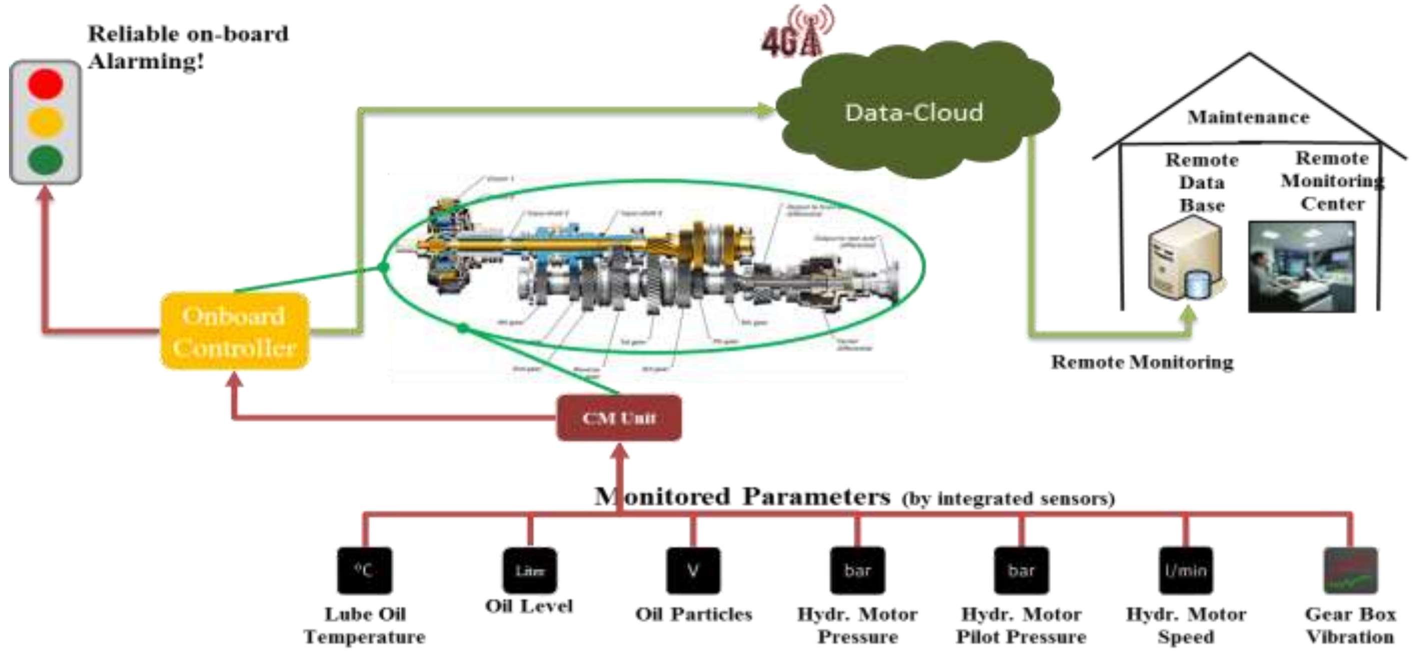
- The **predictive maintenance** principle is to **use a combination of measurements** on process and machinery to **evaluate equipment conditions**.

- Several technologies are available:



Data Collection and Storage

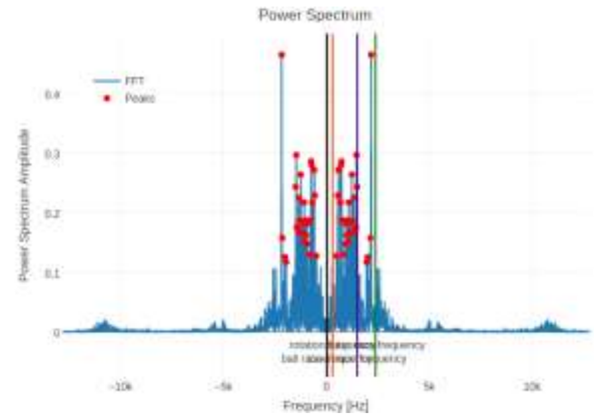
- All the methods described before need to analyze machine data.
- Data can be collected from the system by two different methods:
  - <u>Spot readings</u>: can be performed at regular intervals by using **portable instruments**.
  - <u>Continuous readings</u>: Sensors can be retrofitted to equipment or installed during manufacture for **continuous data collection**.
- Collected data is sent to the **cloud** for **storage** and for **remote monitoring and processing**

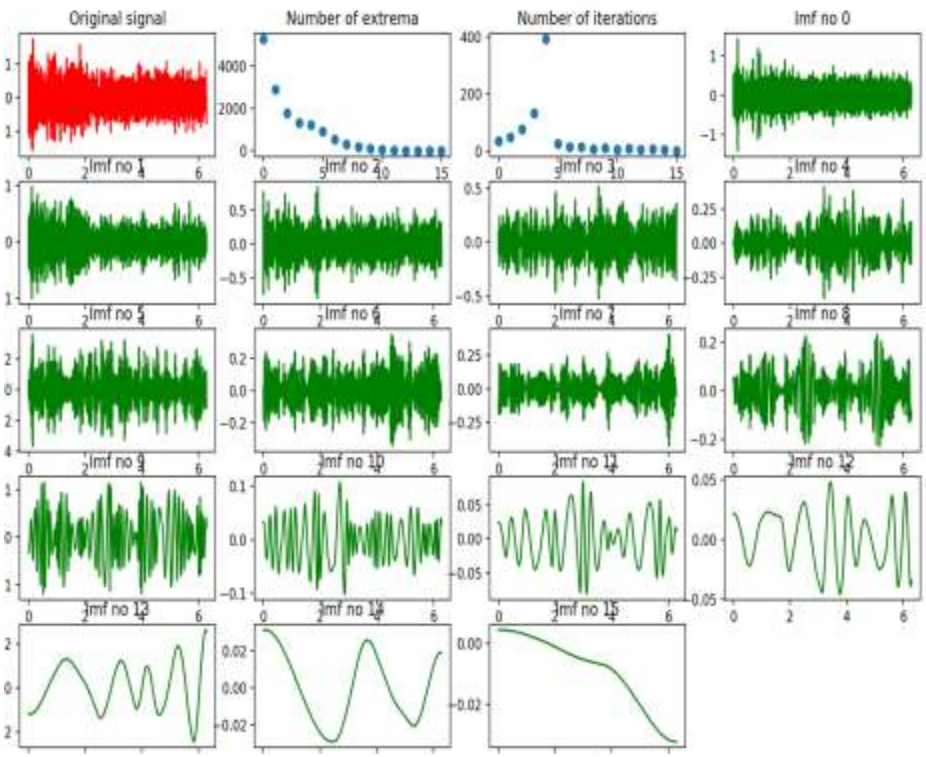


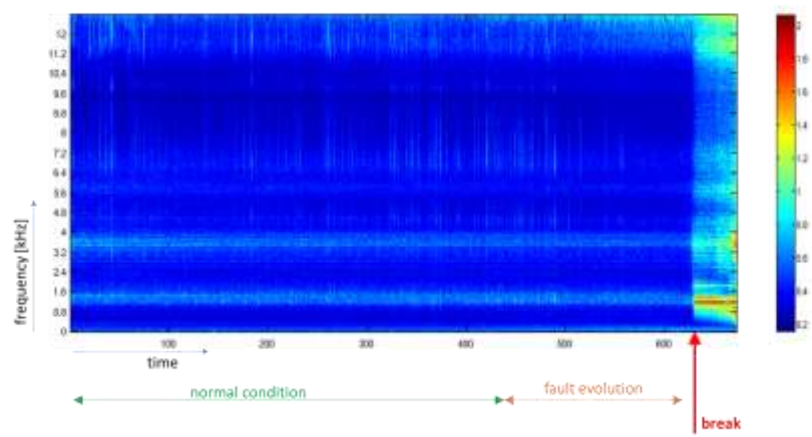
Data Processing

- Collected data need to be analyzed to extract **useful information** to infer health status:

  - ➢ **Digital Signal Processing**







Decision Making

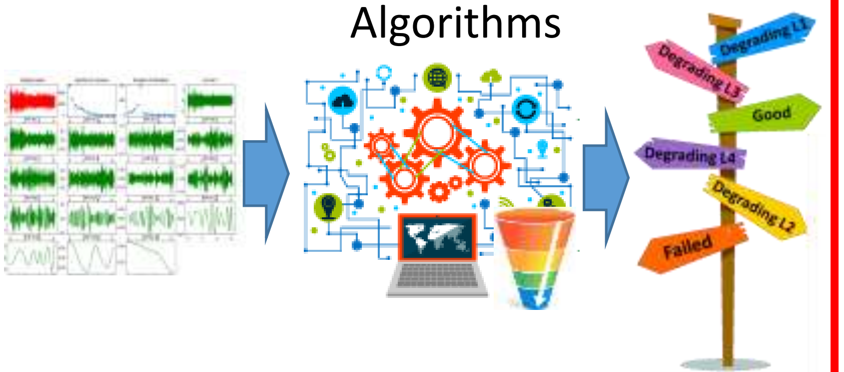
- The processed data needs to be analyzed to extract health status and to identify the time to failure

Domain Expert
Human Operator

➢ Manual analysis: a human operator visually inspects the acquired and processed data and leveraging on his experience guess the lath status and remaining useful life.

Machine Learning
Algorithms

➢ Computer based analysis: Leveraging on decision making support systems like e.g. (deep) machine learning classify the acquired data with health status and predict the remaining useful life.



Machine (deep) learning approach

Input

Raw data (eg vibration, temperature, …) about

1. the behaviour of n components (often obtained by ad hoc platforms) until degradation.
2. Current (n+1) running component

Output.

Prediction on:

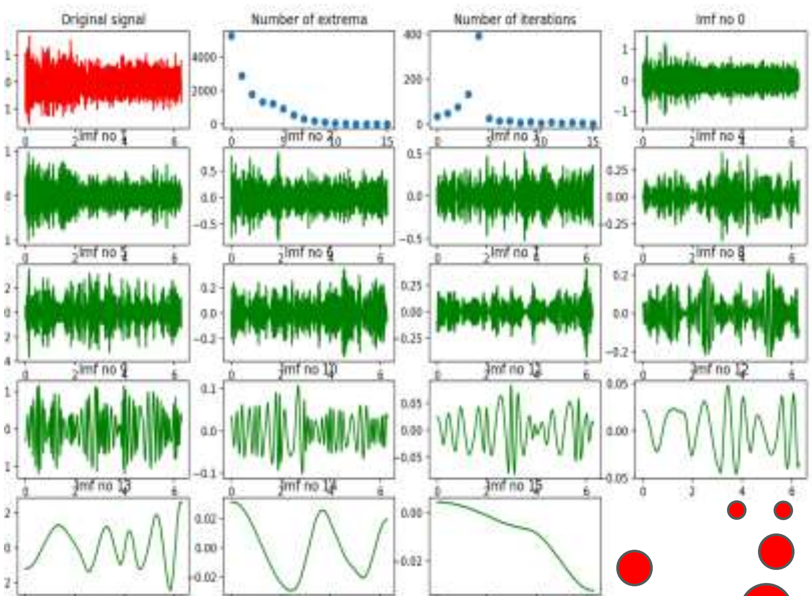
1. Health condition (good/degrading/bad/…)
2. Remaining useful life (RUL)
3. Cause of fault
4. …

Two steps + some pre-processing:

1. Pre-processing of data
2. Learning to classify (based on ML technique)
3. Learning to estimate RUL (support vector regression)
4. …

Pre-processing and feature extraction

Step 1: IMF Decomposition through EMD



Step 2: Health Indicator Computation

IMF_1 IMF_2 IMF_3 IMF_n

Hilbert transformation HT_i 1<=i<=n

Hilbert Marginal Spectrum $h_i(f)$, 1<=i<=n

IMF selection: $\mathbf{h_{max}(f_{or}), h_{max}(f_{ir}), h_{max}(f_b)}$

Bearing component frequencies: f_{or}, f_{ir}, f_b

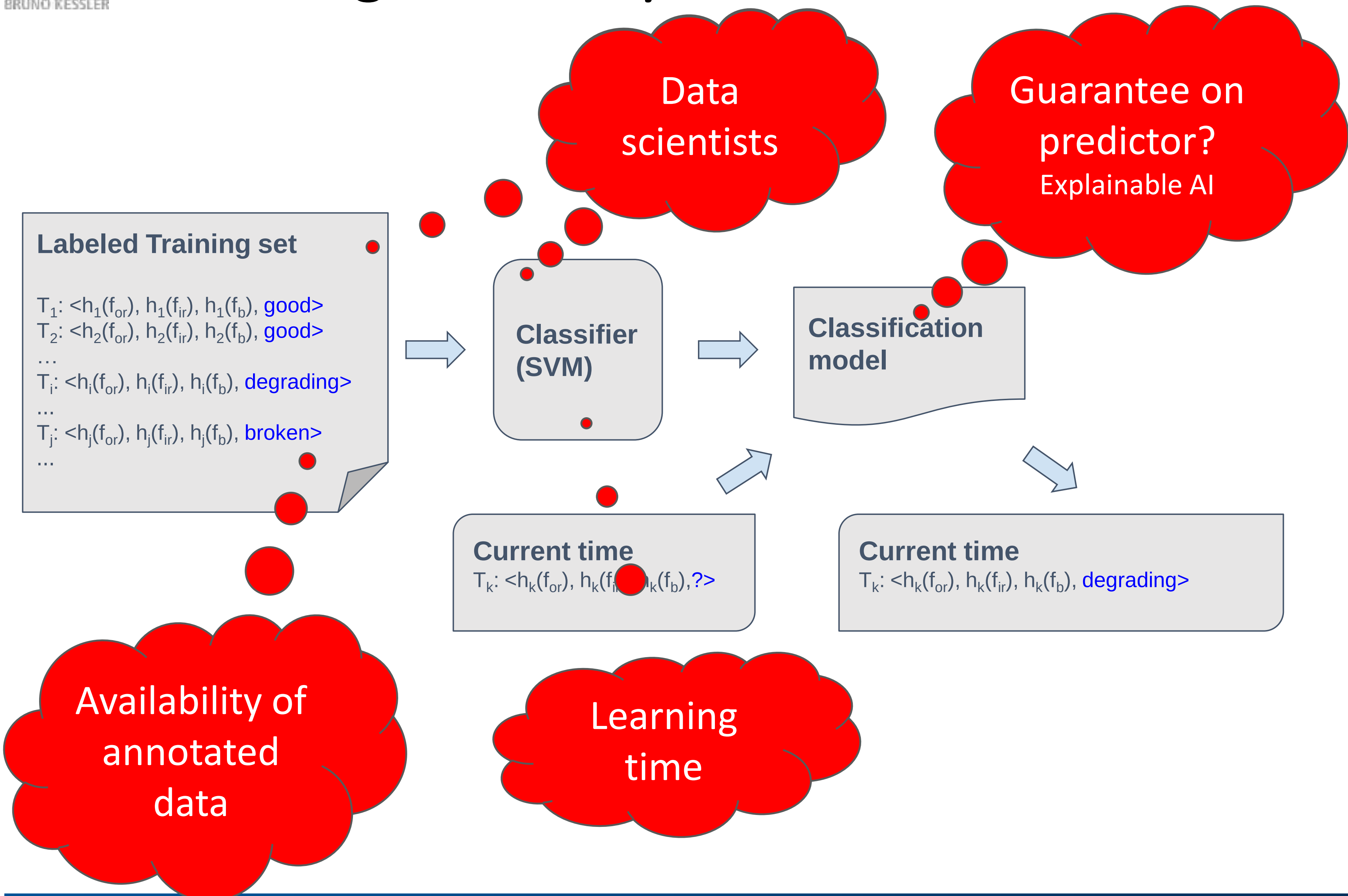
Three features (**health indicators**), one per component (outer rate, inner rate, balls) frequency.
For ea time interval t:
$\mathbf{(h_t(f_{or}), h_t(f_{ir}), h_t(f_b))}$

Domain Experts

Data scientists

DSP Experts

Learning to classify



Learning to predict RUL

Training sets

$\{<h_{i-nr}(f_{ir}), \dots ,h_{i-r}(f_{ir}),h_i(f_{ir}),h_{i+1}(f_{ir})>\}$

$\{<h_{i-nr}(f_{ir}), \dots ,h_{i-r}(f_{ir}),h_i(f_{ir}),h_{i+2}(f_{ir})>\}$

.....

Regression (SVR)

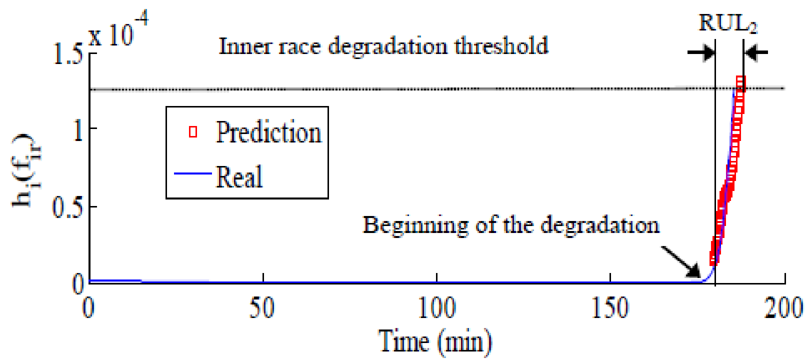
Regression models

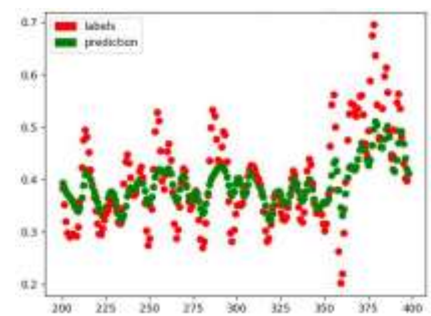
Current trace

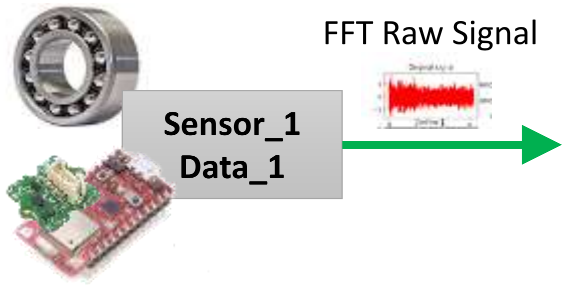
$h_{k-nr}(f_{ir}), \dots ,h_{k-r}(f_{ir}),h_k(f_{ir}),?$

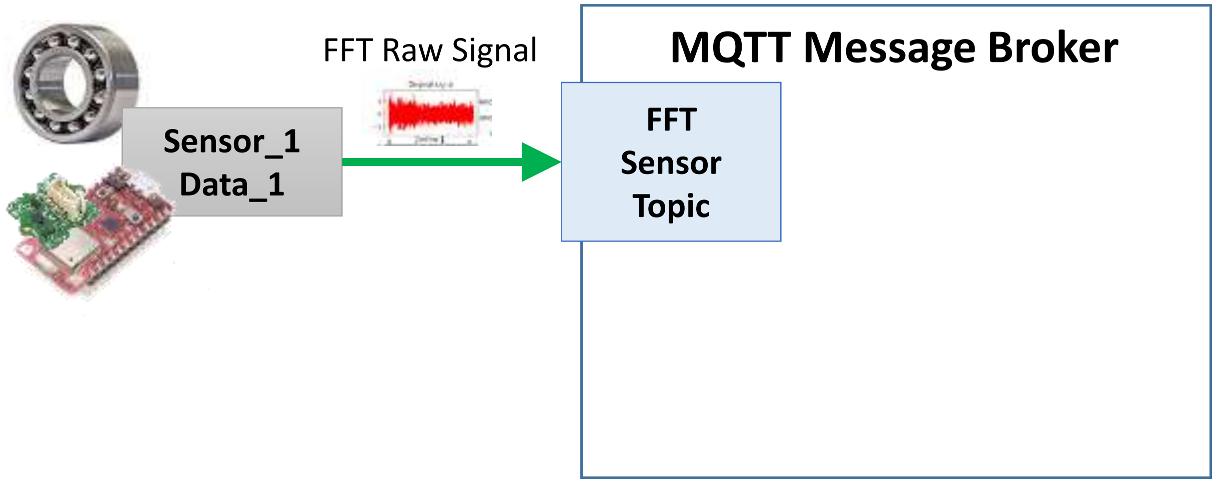
Current trace

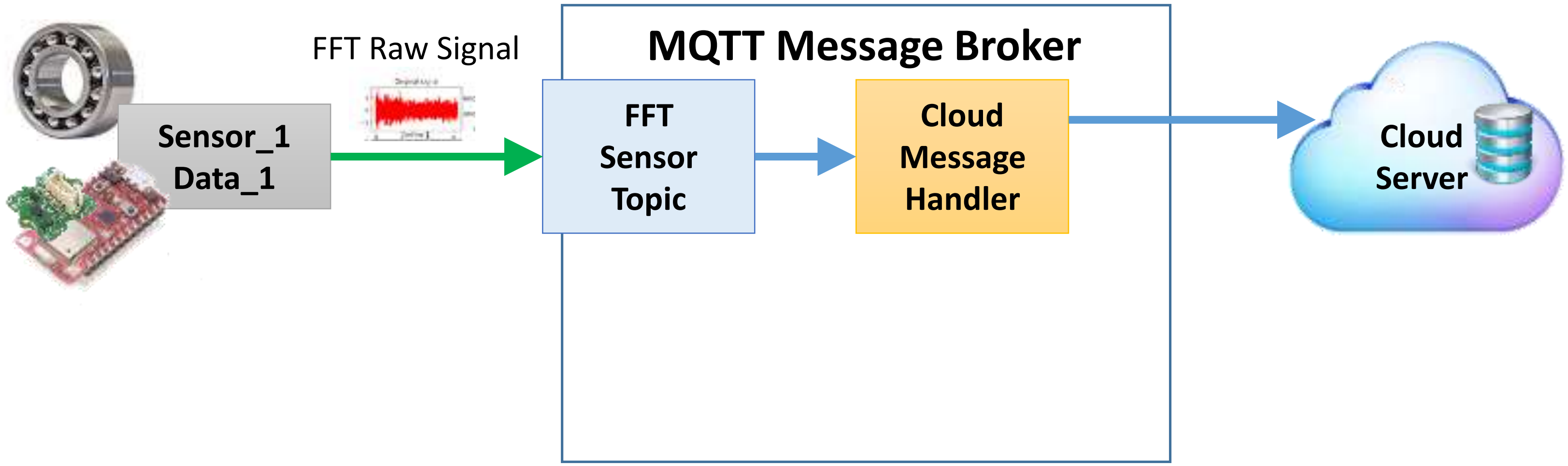
$h_{k-nr}(f_{ir}), \dots ,h_{k-r}(f_{ir}),h_k(f_{ir}), h_{k+1}(f_{ir}),h_{k+2}(f_{ir}), \dots$



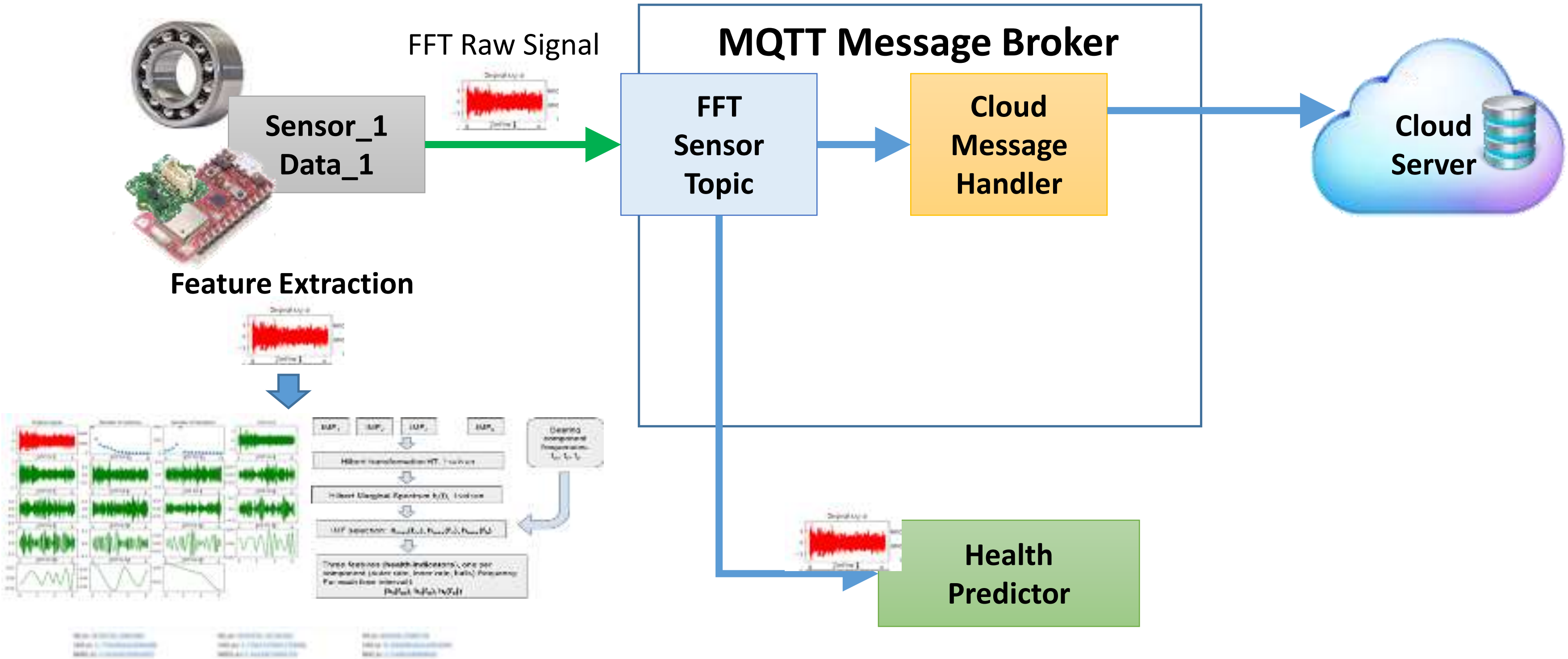


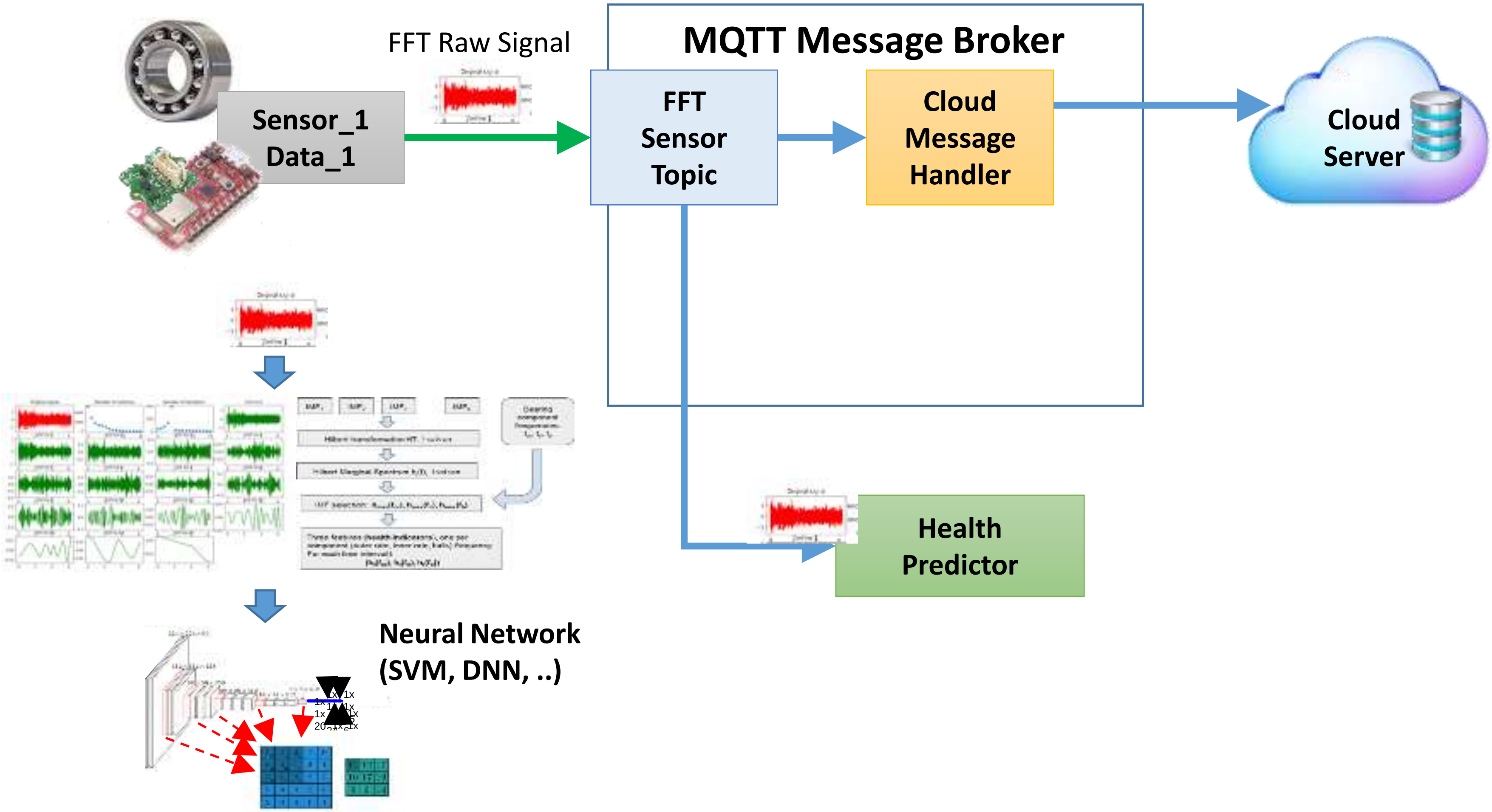

FFT Raw Signal
Sensor_1
Data_1


FFT Raw Signal
MQTT Message Broker
Sensor_1
Data_1
FFT
Sensor
Topic

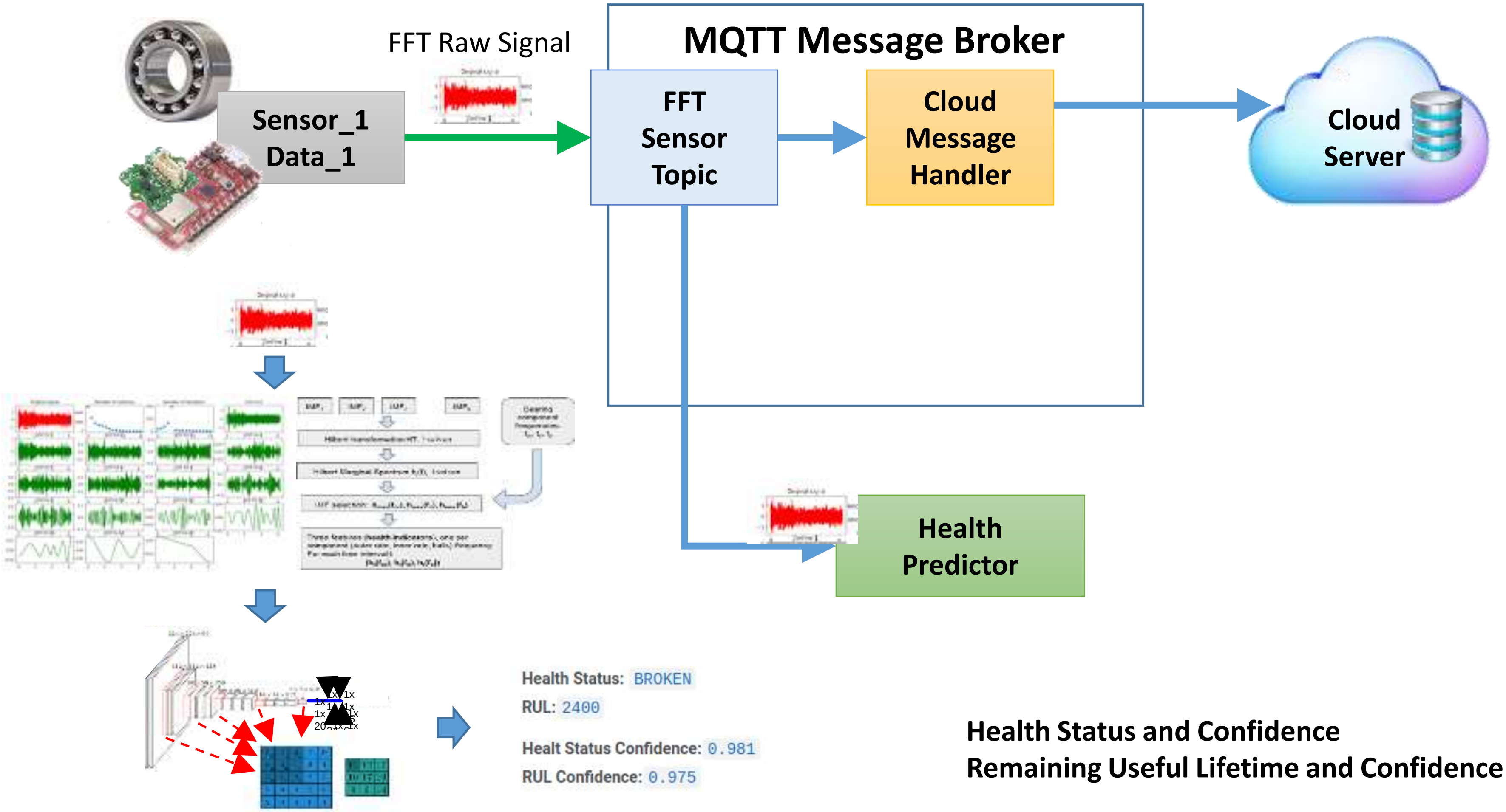

FFT Raw Signal
MQTT Message Broker
Sensor_1
Data_1
FFT
Sensor
Topic
Cloud
Message
Handler
Cloud
Server

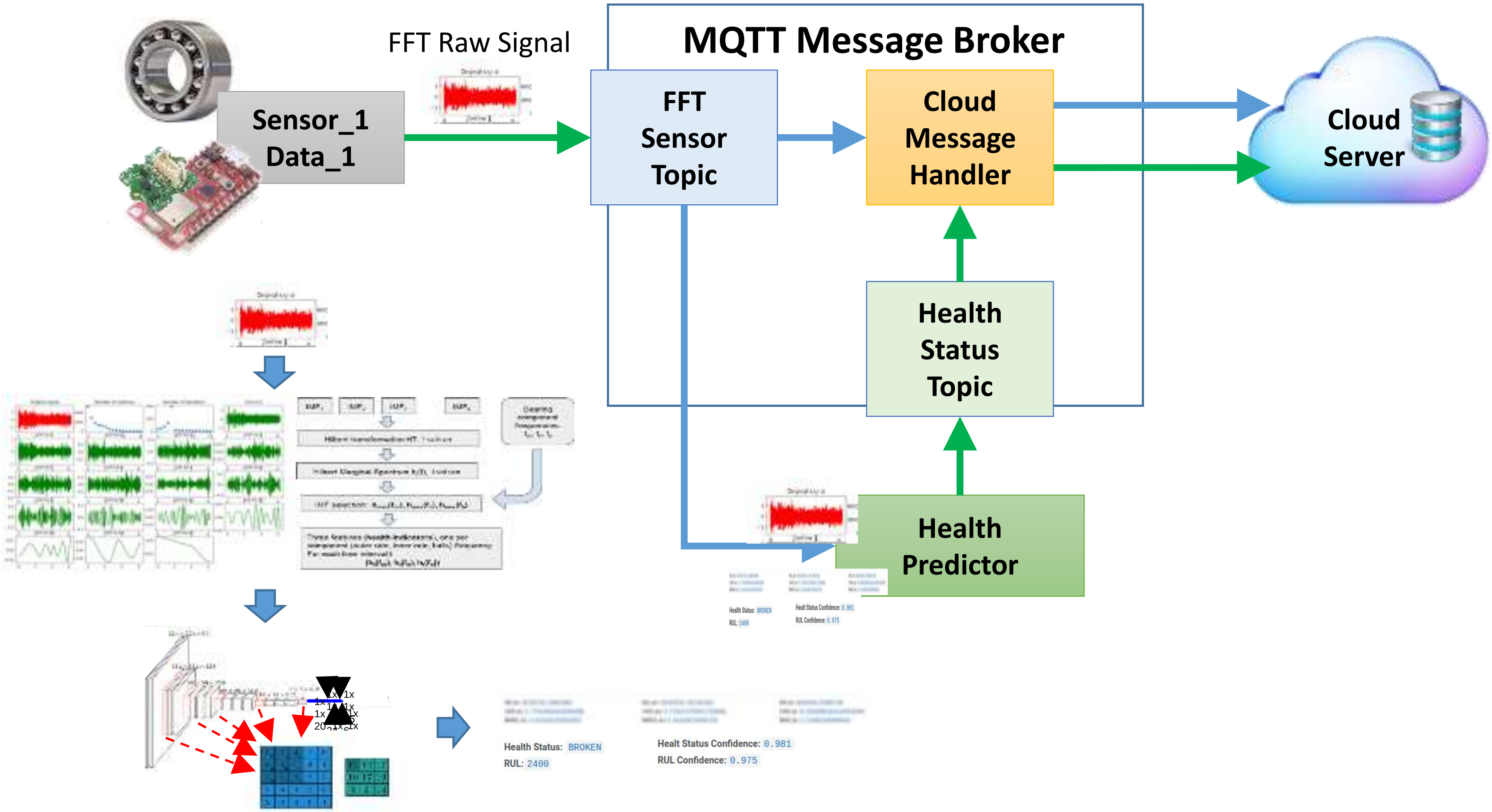
FONDAZIONE BRUNO KESSLER



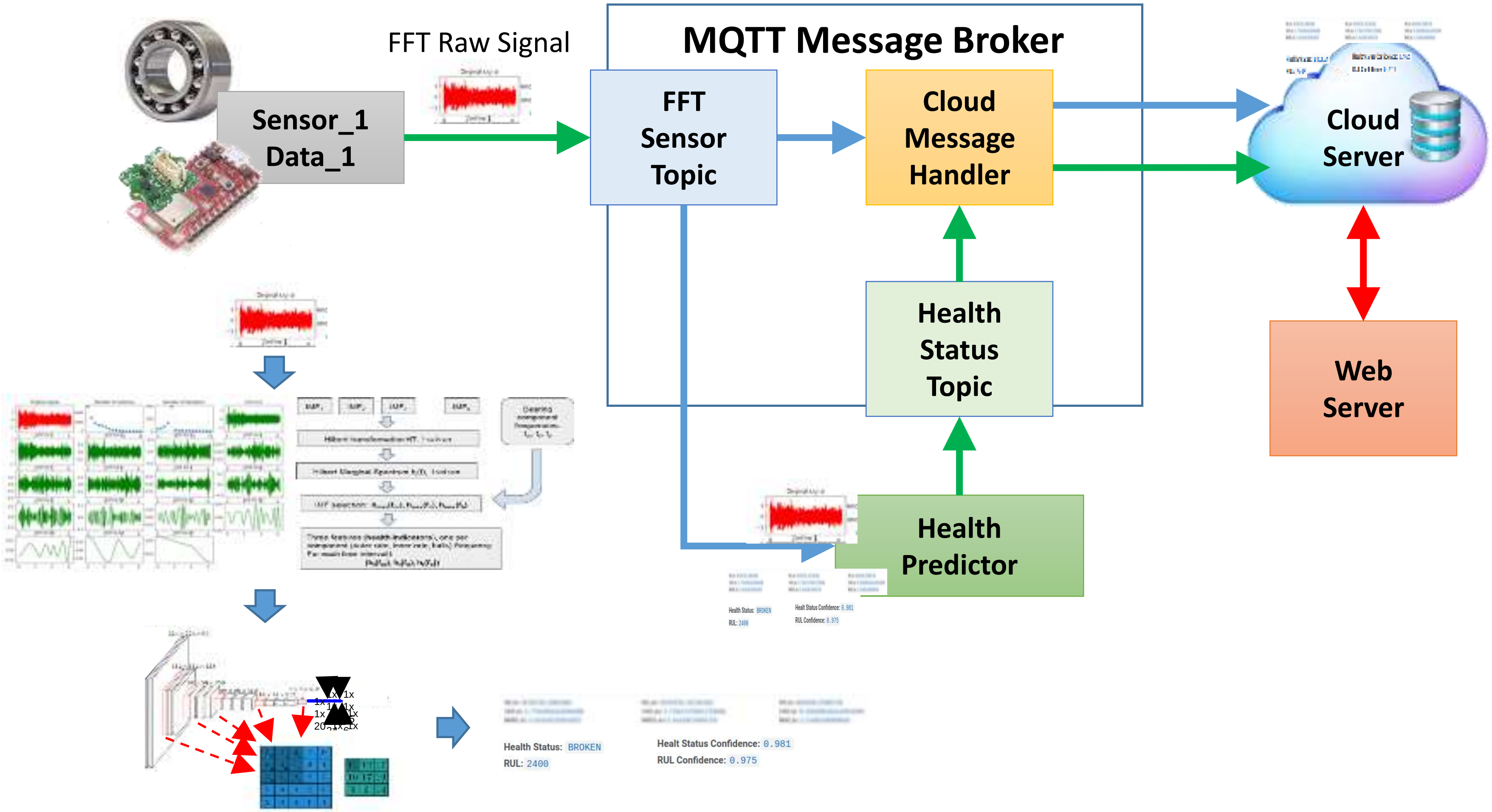


FFT Raw Signal
MQTT Message Broker
Sensor_1
Data_1
FFT
Sensor
Topic
Cloud
Message
Handler
Cloud
Server
Health
Predictor
Neural Network
(SVM, DNN, ..)

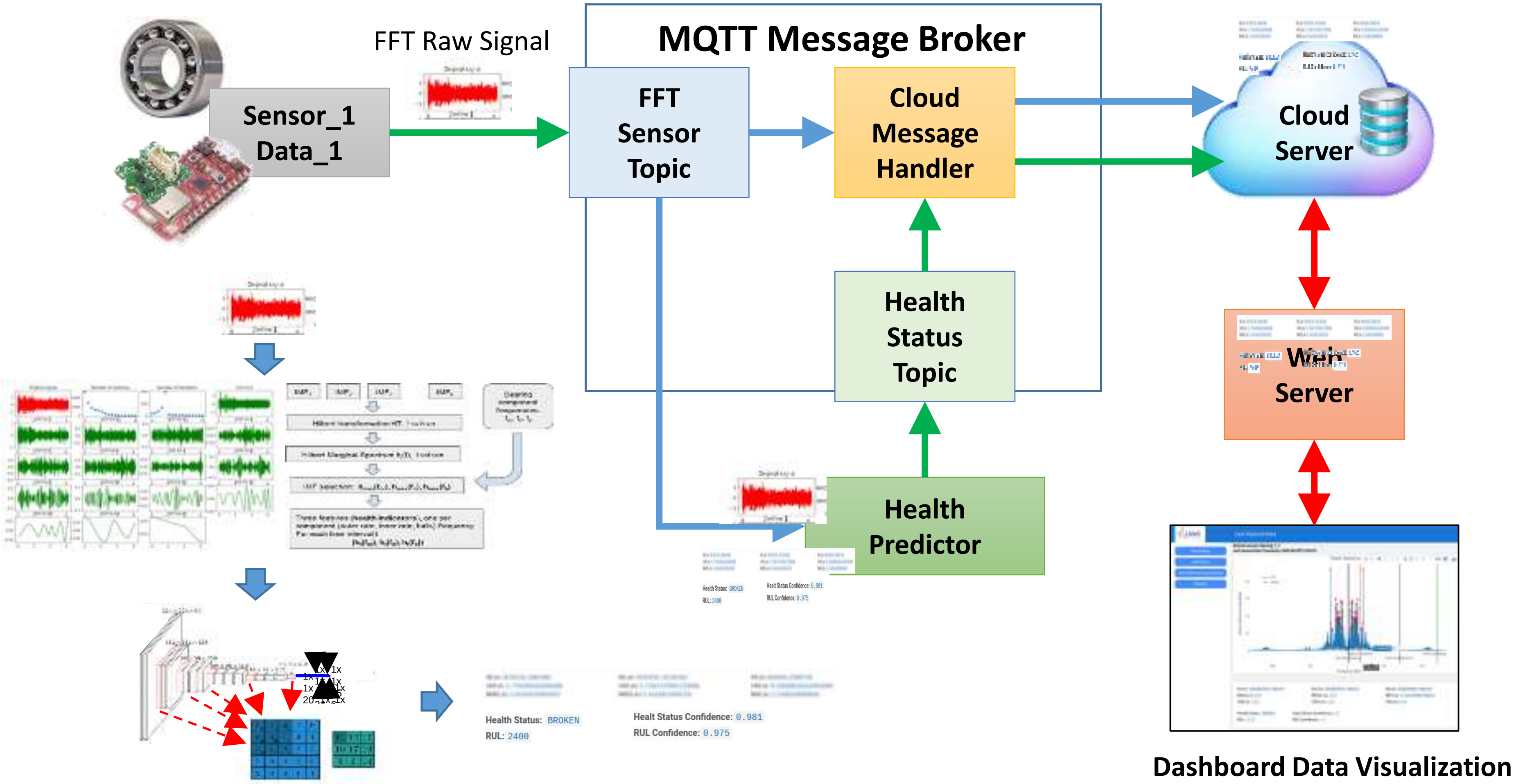

MQTT Message Broker
FFT Raw Signal
Sensor_1
Data_1
FFT
Sensor
Topic
Cloud
Message
Handler
Cloud
Server
Health
Predictor
Health Status: BROKEN
RUL: 2400
Healt Status Confidence: 0.981
RUL Confidence: 0.975
Health Status and Confidence
Remaining Useful Lifetime and Confidence



FFT Raw Signal
MQTT Message Broker
Sensor_1
Data_1
FFT
Sensor
Topic
Cloud
Message
Handler
Cloud
Server
Health
Status
Topic
Health
Predictor
Health Status: BROKEN
RUL: 2400
Healt Status Confidence: 0.981
RUL Confidence: 0.975



FFT Raw Signal
MQTT Message Broker
Sensor_1
Data_1
FFT
Sensor
Topic
Cloud
Message
Handler
Cloud
Server
Health
Status
Topic
Web
Server
Health
Predictor
Health Status: BROKEN
RUL: 2400
Healt Status Confidence: 0.981
RUL Confidence: 0.975


FFT Raw Signal
MQTT Message Broker
Sensor_1
Data_1
FFT
Sensor
Topic
Cloud
Message
Handler
Cloud
Server
Health
Status
Topic
Health
Predictor
Web
Server
Health Status: BROKEN
RUL: 2400
Healt Status Confidence: 0.981
RUL Confidence: 0.975
Dashboard Data Visualization

FONDAZIONE
BRUNO KESSLER

Outline

- Recap on Predictive Maintenance
- Introduction to machine learning in python
- Introduction to MQTT and json/protobuf in python

Introduction to machine learning in python

- What is Machine Learning?
  - Machine learning is the process of extracting knowledge from data automatically, usually with the goal of **making predictions on new, unseen data**.
    - Example: a spam filter
      - The user keeps labeling incoming mails as either spam or not spam.
      - A machine learning algorithm then "learns" a predictive model from data that distinguishes spam from normal emails, a model which can predict for new emails whether they are spam or not.
  - Central to machine learning is the concept of **automating decision making** from data **without the user specifying explicit rules** how this decision should be made.
    - For the case of emails, the user doesn't provide a list of words or characteristics that make an email spam. Instead, the user provides examples of spam and non-spam emails that are labeled as such.
  - The second central concept is **generalization**. The goal of a machine learning model is to predict on **….**
    - For the case of emails, we are not interested in marking an already labeled email as spam or not. Instead, we want to make the user's life easier by automatically classifying new incoming mail.

Introduction to machine learning in python

- There are two kinds of machine learning we will talk about today:
  - supervised learning
  - unsupervised learning

- Supervised Learning: Classification and regression
  - In Supervised Learning, we have a dataset consisting of both input features and a desired output, such as in the spam / no-spam example. The task is to construct a model (or program) which is able to predict the desired output of an unseen object given the set of features.

  - Some more complicated examples are:
    - Given a multicolor image of an object through a telescope, determine whether that object is a star, a quasar, or a galaxy.
    - Given a photograph of a person, identify the person in the photo.
    - Given a list of movies a person has watched and their personal rating of the movie, recommend a list of movies they would like.
    - Given a persons age, education and position, infer their salary

  - What these tasks have in common is that there is one or more unknown quantities associated with the object which needs to be determined from other observed quantities.

Introduction to machine learning in python

- Supervised learning is further broken down into two categories, classification and regression:

  - In classification, the label is discrete, such as "spam" or "no spam". In other words, it provides a clear-cut distinction between categories. Furthermore, it is important to note that class labels are nominal, not ordinal variables. Nominal and ordinal variables are both subcategories of categorical variable. Ordinal variables imply an order, for example, T-shirt sizes "XL > L > M > S". On the contrary, nominal variables don't imply an order, for example, we (usually) can't assume "orange > blue > green".

  - In regression, the label is continuous, that is a float output. For example, in astronomy, the task of determining whether an object is a star, a galaxy, or a quasar is a classification problem: the label is from three distinct categories. On the other hand, we might wish to estimate the age of an object based on such observations: this would be a regression problem, because the label (age) is a continuous quantity.

- In supervised learning, there is always a distinction between
  - a training set for which the desired outcome is given, and
  - a test set for which the desired outcome needs to be inferred.

  The learning model fits the predictive model to the training set, and we use the test set to evaluate its generalization performance.

Introduction to machine learning in python

- Unsupervised Learning
  - In Unsupervised Learning there is no desired output associated with the data. Instead, we are interested in extracting some form of knowledge or model from the given data. In a sense, you can think of unsupervised learning as a means of discovering labels from the data itself. Unsupervised learning is often harder to understand and to evaluate.
  - Unsupervised learning comprises tasks such as dimensionality reduction, clustering, and density estimation. For example, in the iris data discussed later, we can use unsupervised methods to determine combinations of the measurements which best display the structure of the data. Some more involved unsupervised learning problems are:
    - Given detailed observations of distant galaxies, determine which features or combinations of features summarize best the information.
    - Given a mixture of two sound sources (for example, a person talking over some music), separate the two (this is called the blind source separation problem).
    - Given a video, isolate a moving object and categorize in relation to other moving objects which have been seen.
    - Given a large collection of news articles, find recurring topics inside these articles.
    - Given a collection of images, cluster similar images together (for example to group them when visualizing a collection)
- Sometimes the two may even be combined: e.g. unsupervised learning can be used to find useful features in heterogeneous data, and then these features can be used within a supervised framework.

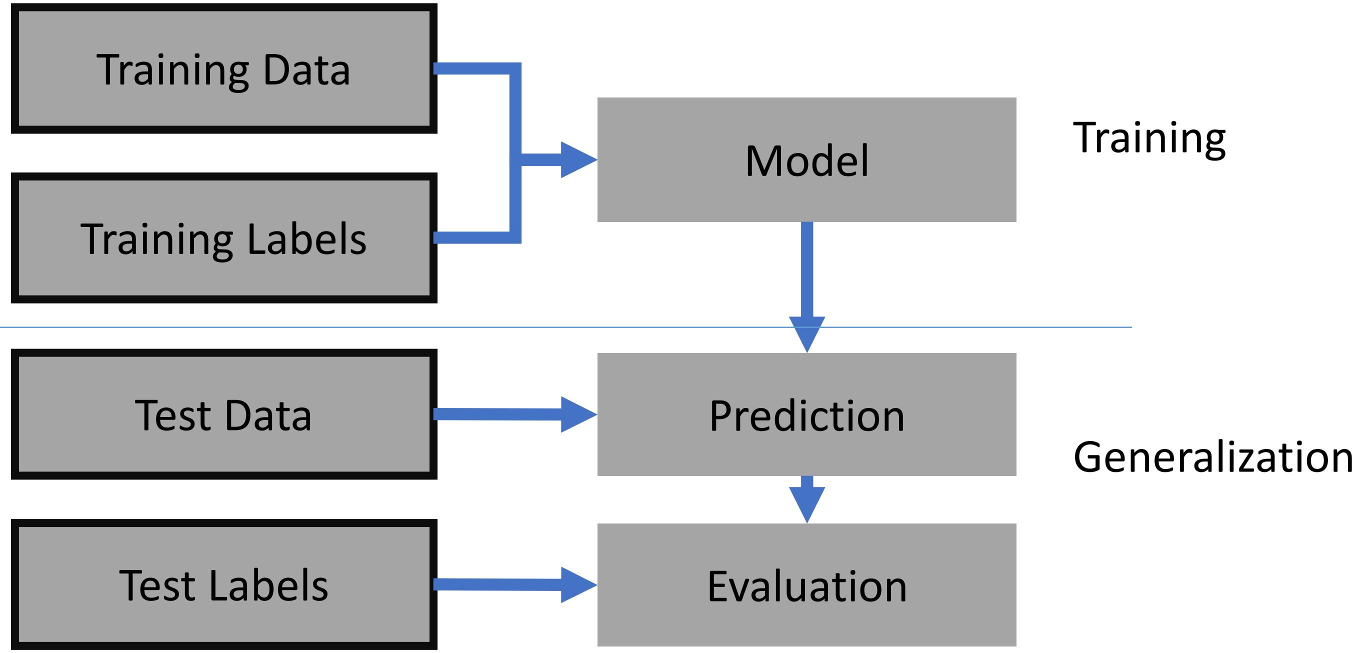
Introduction to machine learning in python

$$X = \begin{bmatrix} x_1^{(1)} & \cdots & x_N^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(K)} & \cdots & x_N^{(K)} \end{bmatrix}$$

$$Y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(K)} \end{bmatrix}$$

$$X' = \begin{bmatrix} x_1^{(1)} & \cdots & x_N^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(L)} & \cdots & x_N^{(L)} \end{bmatrix}$$

$$Y' = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(L)} \end{bmatrix}$$



The data is presented to the algorithm usually as a two-dimensional array (or matrix) of numbers. Each data point (also known as a sample or training instance) that we want to either learn from or make a decision on is represented as a list of numbers, a so-called feature vector, and its containing features represent the properties of this point.

Introduction to machine learning in python

0 = Iris Setosa

1 = Iris Versicolor

2 = Iris Virginica

We represent each flower sample as one row in our data array, and the columns (features) represent the flower measurements in centimeters. For instance, we can represent this Iris dataset, consisting of 150 samples and 4 features, as a 2-dimensional array or matrix $\mathbb{R}^{150\times 4}$

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & x_4^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(150)} & x_2^{(150)} & x_3^{(150)} & x_4^{(150)} \end{bmatrix} \qquad Y = \begin{bmatrix} 0 \\ \vdots \\ 2 \end{bmatrix}$$

Introduction to machine learning in python

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

iris = load_iris()
X, y = iris.data, iris.target

classifier = KNeighborsClassifier()

train_X, test_X, train_y, test_y = train_test_split(X, y,
                                                    train_size=0.5,
                                                    test_size=0.5,
                                                    random_state=123,
                                                    stratify=y)
```

Introduction to machine learning in python

```
print("Labels for training and testing data")
print(train_y)
print(test_y)
[1 1 1 0 0 2 1 1 1 0 1 0 2 0 0 2 0 2 1 1 0 0 2 1 2 1 0 1 1 1 2 1 2 2 0 0 2
 2 0 0 2 2 2 2 0 2 0 2 1 1 0 2 2 0 2 1 2 1 2 1 1 0 0 1 2 0 0 2 2 1 0 1 0 0  1]
[0 2 1 0 2 0 1 2 0 0 2 1 2 0 1 2 2 2 2 2 1 2 1 1 2 2 0 0 1 0 0 2 0 1 0 0 1
 1 2 2 0 1 0 1 1 2 0 1 1 1 0 2 2 2 1 0 0 1 1 0 2 1 0 2 0 2 1 1 2 0 2 1 0 0  1]

print('All:', np.bincount(y) / float(len(y)) * 100.0)
print('Training:', np.bincount(train_y) / float(len(train_y)) * 100.0)
print('Test:', np.bincount(test_y) / float(len(test_y)) * 100.0)
('All:', array([ 33.33333333,  33.33333333,  33.33333333]))
('Training:', array([ 33.33333333,  33.33333333,  33.33333333]))
('Test:', array([ 33.33333333,  33.33333333,  33.33333333]))
```

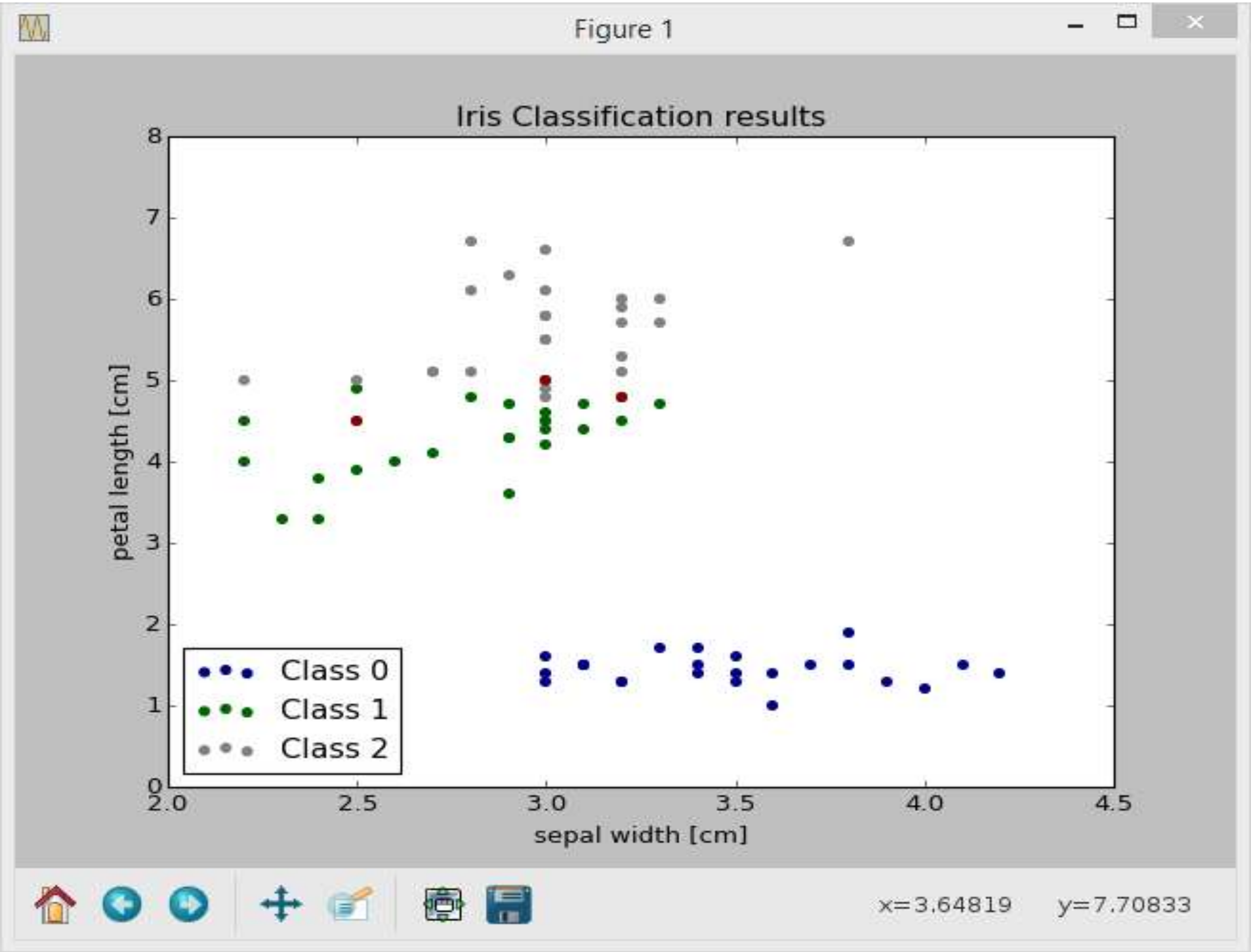
Introduction to machine learning in python

```
classifier.fit(train_X, train_y)
pred_y = classifier.predict(test_X)

print("Fraction Correct [Accuracy]:")
print(np.sum(pred_y == test_y) / float(len(test_y)))
Fraction Correct [Accuracy]:
0.96

print('Samples correctly classified:')
correct_idx = np.where(pred_y == test_y)[0]
print(correct_idx)
Samples correctly classified:
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 45 46 47 48 50 51
 52 53 54 55 56 57 58 59 61 62 63 64 65 66 67 68 69 70 71 72 73 74]
print('\nSamples incorrectly classified:')
incorrect_idx = np.where(pred_y != test_y)[0]
print(incorrect_idx)
Samples incorrectly classified:
[44 49 60]
```

Introduction to machine learning in python

```
# Plot two dimensions

colors = ["darkblue", "darkgreen", "gray"]

for n, color in enumerate(colors):
    idx = np.where(test_y == n)[0]
    plt.scatter(test_X[idx, 1], test_X[idx, 2], color=color, label="Class %s" % str(n))

plt.scatter(test_X[incorrect_idx, 1], test_X[incorrect_idx, 2], color="darkred")

plt.xlabel('sepal width [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc=3)
plt.title("Iris Classification results")
plt.show()
```

Introduction to machine learning in python



Introduction to machine learning in python

```
from __future__ import print_function
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import make_blobs

X, y = make_blobs(centers=2, random_state=0)

print('X ~ n_samples x n_features:', X.shape)
print('y ~ n_samples:', y.shape)

print('\nFirst 5 samples:\n', X[:5, :])
print('\nFirst 5 labels:', y[:5])

plt.scatter(X[y == 0, 0], X[y == 0, 1], c='blue', s=40, label='0')
plt.scatter(X[y == 1, 0], X[y == 1, 1], c='red', s=40, label='1', marker='s')

plt.xlabel('first feature')
plt.ylabel('second feature')
plt.legend(loc='upper right');
plt.show()
```

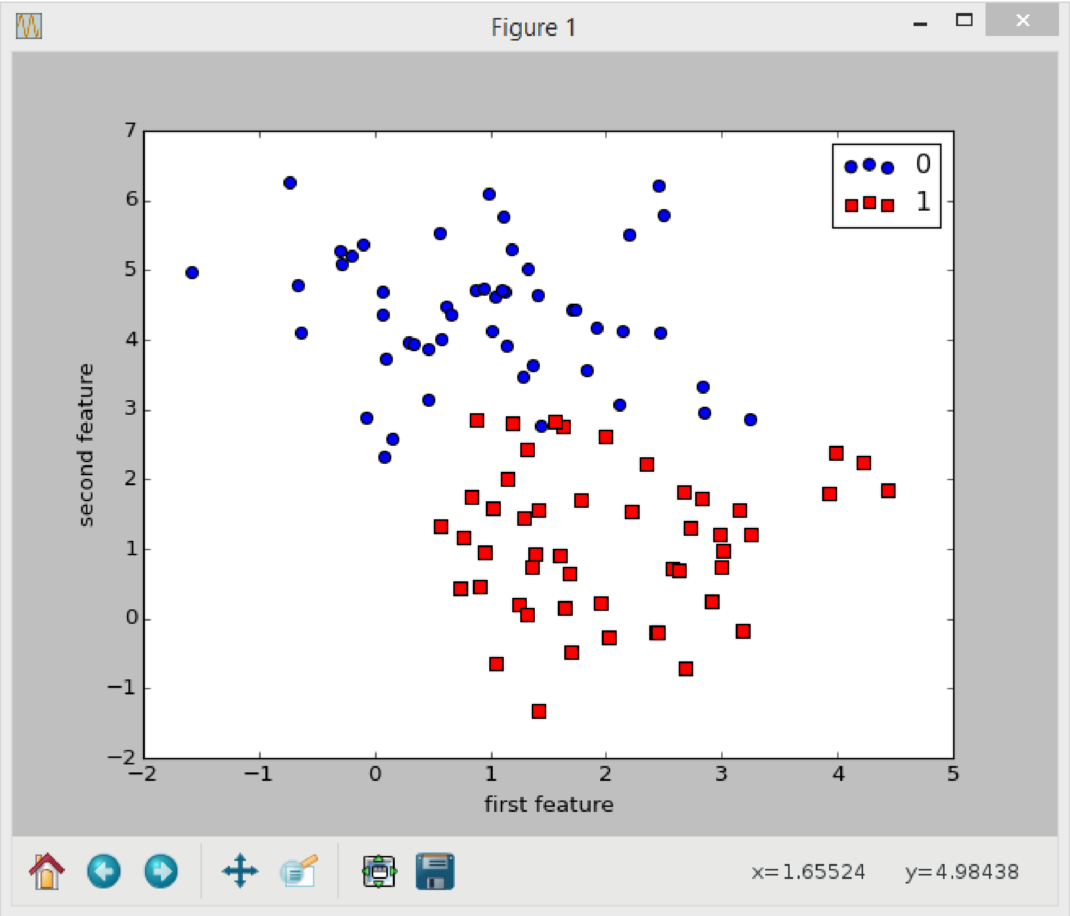
There are many classifiers.
Here we see also another dataset

Introduction to machine learning in python

X ~ n_samples x n_features: (100, 2)
y ~ n_samples: (100,)

First 5 samples:
[[4.21850347 2.23419161]
[0.90779887 0.45984362]
[-0.27652528 5.08127768]
[0.08848433 2.32299086]
[3.24329731 1.21460627]]

First 5 labels: [1 1 0 0 1]



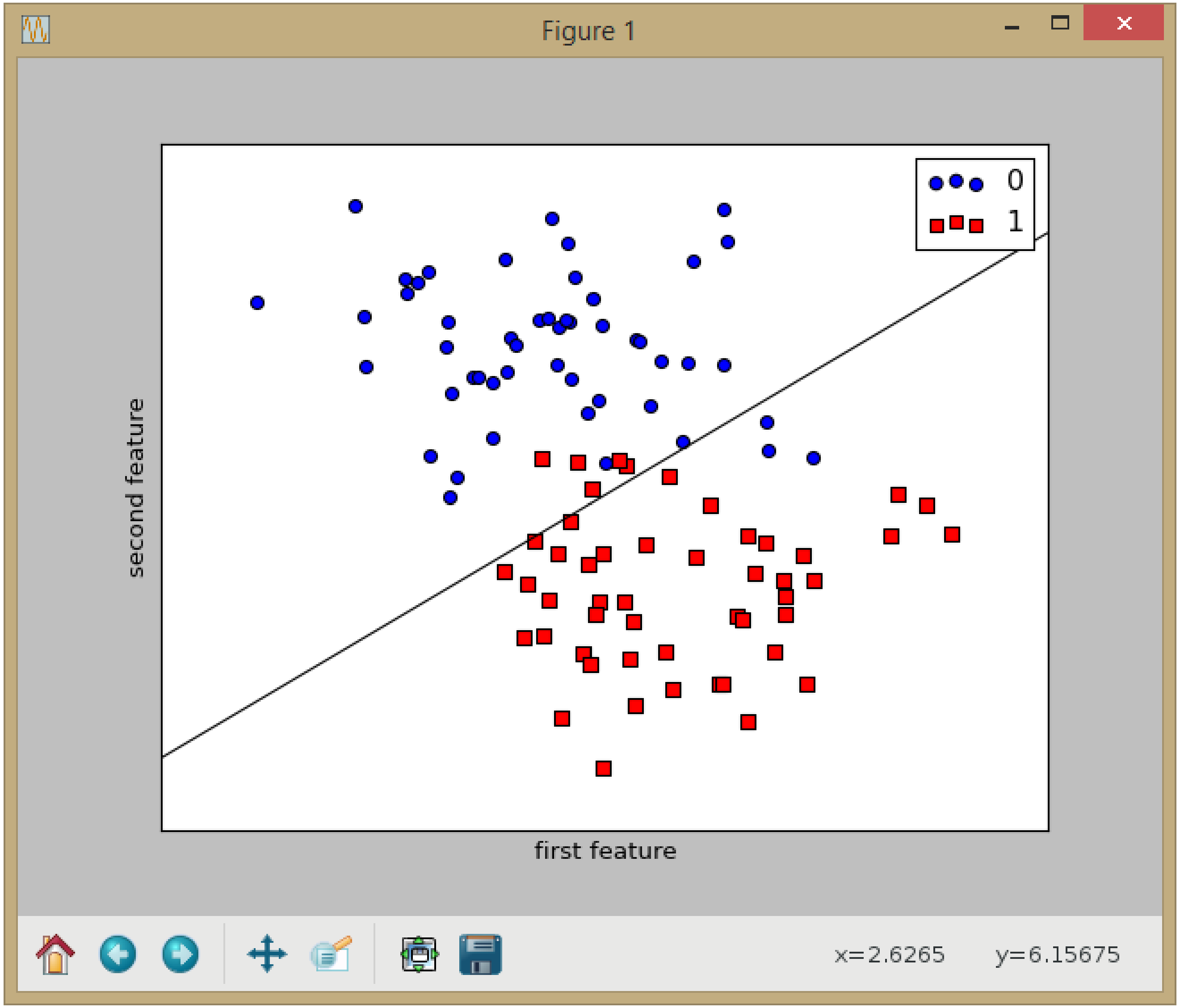
Introduction to machine learning in python

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
                                                    random_state=1234, stratify=y)

classifier = LogisticRegression()
classifier.fit(X_train, y_train)

# We can then apply the model to unseen data and use the model to
# predict the estimated outcome using the predict method:
prediction = classifier.predict(X_test)

# We can compare these against the true labels:
print(prediction) print(y_test)
[1 0 1 0 1 1 1 1 1 1 1 0 0 0 0 1 0 0 1 0 0 0 1 1 0]
[1 1 1 0 1 1 0 1 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 1 0]
```

Introduction to machine learning in python

```
# We can evaluate our classifier quantitatively by measuring what
# fraction of predictions is correct. This is called accuracy:
np.mean(prediction == y_test)

# There is also a convenience function , score, that all scikit-learn
# classifiers have to compute this directly from the test data:
classifier.score(X_test, y_test) classifier.score(X_test, y_test)

# It is often helpful to compare the generalization performance (on the test set) to the
# performance on the training set:
classifier.score(X_train, y_train)

# LogisticRegression is a so-called linear model, that means it will create a decision that is
# linear in the input space. In 2d, this simply means it finds a line to separate the blue from the red:
from figures import plot_2d_separator

plt.scatter(X[y == 0, 0], X[y == 0, 1], c='blue', s=40, label='0')
plt.scatter(X[y == 1, 0], X[y == 1, 1], c='red', s=40, label='1', marker='s')
plt.xlabel("first feature") plt.ylabel("second feature")
plot_2d_separator(classifier, X)
plt.legend(loc='upper right')
plt.show()
```

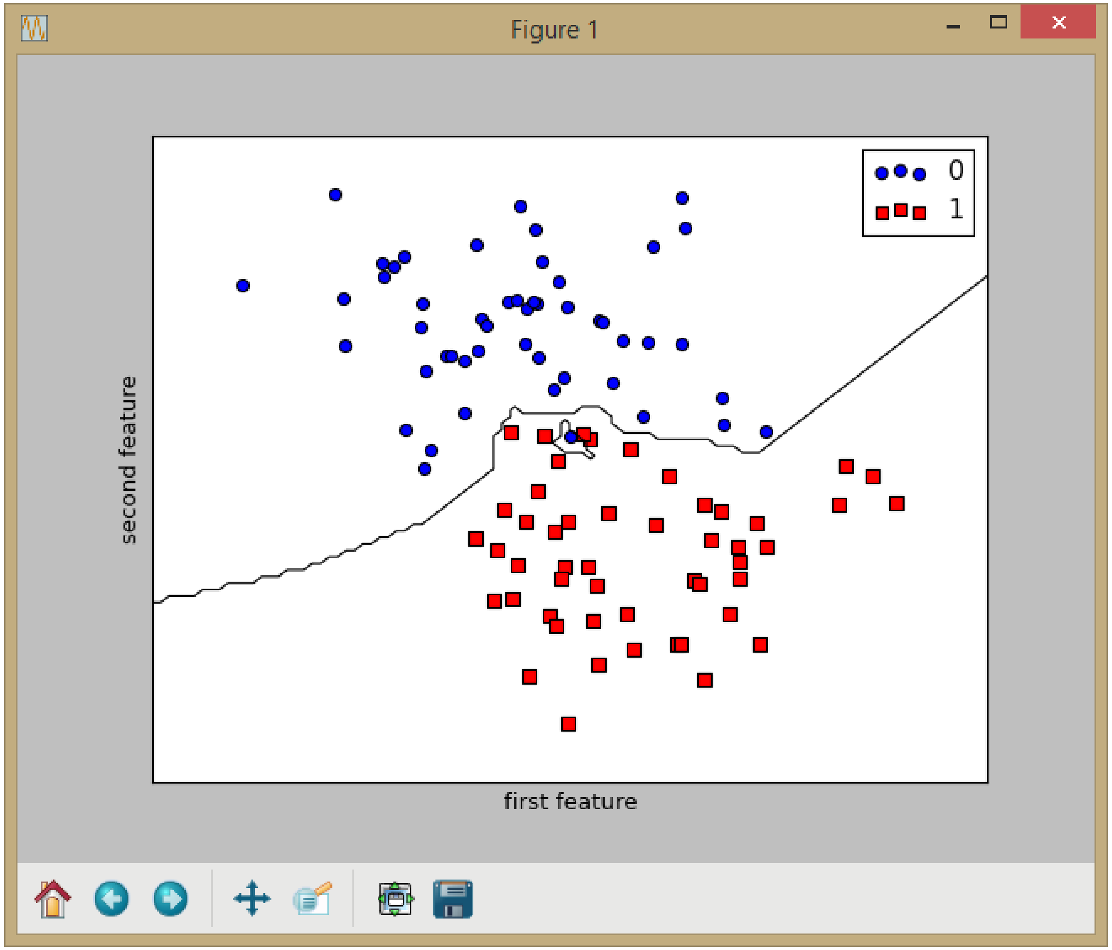
Introduction to machine learning in python



Introduction to machine learning in python

```
# Estimated parameters: All the estimated model parameters are attributes of the estimator object
# ending by an underscore. Here, these are the coefficients and the offset of the line:
print(classifier.coef_) print(classifier.intercept_)

# Another popular and easy to understand classifier is K nearest neighbors (kNN). It has one of the
# simplest learning strategies: given a new, unknown observation, look up in your reference database
# which ones have the closest features and assign the predominant class.
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)

# We fit the model with out training data
knn.fit(X_train, y_train)

plt.scatter(X[y == 0, 0], X[y == 0, 1], c='blue', s=40, label='0')
plt.scatter(X[y == 1, 0], X[y == 1, 1], c='red', s=40, label='1', marker='s')

plt.xlabel("first feature")
plt.ylabel("second feature")
plot_2d_separator(knn, X)
plt.legend(loc='upper right');
plt.show()

knn.score(X_test, y_test)
```

Introduction to machine learning in python



[[1.38092515 -1.49993172]]
[1.54995538]

Introduction to machine learning in python

- Scikit-learn strives to have a uniform interface across all methods. Given a scikit-learn estimator object named model, the following methods are available (not all for each model):

- Available in all Estimators
  - model.fit() : fit training data. For supervised learning applications, this accepts two arguments: the data X and the labels y (e.g. model.fit(X, y)). For unsupervised learning applications, fit takes only a single argument, the data X (e.g. model.fit(X)).
  - model.predict() : given a trained model, predict the label of a new set of data. This method accepts one argument, the new data X_new (e.g. model.predict(X_new)), and returns the learned label for each object in the array.
  - model.predict_proba() : For classification problems, some estimators also provide this method, which returns the probability that a new observation has each categorical label. In this case, the label with the highest probability is returned by model.predict().
  - model.score() : for classification or regression problems, most (all?) estimators implement a score method. Scores are between 0 and 1, with a larger score indicating a better fit. For classifiers, the score method computes the prediction accuracy. For regressors, score computes the coefficient of determination (R2) of the prediction.

Introduction to machine learning in python

- After training a scikit-learn model, it is desirable to have a way to persist the model for future use without having to retrain.
  - pickle
  - joblib

```
from sklearn import svm
from sklearn import datasets
clf = svm.SVC()
iris = datasets.load_iris()
X, y = iris.data, iris.target
clf.fit(X, y)
```

```
import pickle
pickle.dump(clf, 'filename.pkl')
clf1 = pickle.load('filename.pkl')
clf1.predict(X[0:1])

from sklearn.externals import joblib
joblib.dump(clf, 'filename.pkl')
clf2 = joblib.load('filename.pkl')
clf2.predict(X[0:1])
```

Introduction to machine learning in python

- There are many estimators, and each estimator has a possibly large set of parameters.
- Typically machine learning libraries (e.g. sklearn) provide techniques and tools to perform exhaustive search over specified parameter values for an estimator.
- All estimators at least provide the two important member function: fit and predict.
- GridSearchCV implements a “fit” and a “score” method.
  - It also implements “predict”, “predict_proba”, “decision_function”, “transform” and “inverse_transform” if they are implemented in the estimator used.
  - The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

Introduction to machine learning in python

```
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV
iris = datasets.load_iris()
# parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4], 'C': [1, 10, 100, 1000]},
              {'kernel': ['linear'], 'C': [1, 10, 100, 1000]},
              {'kernel': ['poly'],  'C': [1, 10, 100, 1000]} ]
svc = svm.SVC()
clf = GridSearchCV(svc, parameters)
clf.fit(iris.data, iris.target)
sorted(clf.cv_results_.keys())
clf.best_estimator_
```

Introduction to machine learning in python

```
from sklearn import svm
from sklearn.model_selection import GridSearchCV
import pikle

# The list of features and labels (all with same size)
f_or = [] # array of values for f_or
f_ir = [] # array of values for f_ir
f_b  = [] # array of values for f_b
a_or = [] # array of values for a_or
a_ir = [] # array of values for a_ir
a_b = []  # array of values for a_b
label = []  # array of values for label

parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                     'C': [1, 10, 100, 1000]},
           {'kernel': ['linear'], 'C': [1, 10, 100, 1000]},
           {'kernel': ['poly'],  'C': [1, 10, 100, 1000]} ]

svc = svm.SVC()
clf = GridSearchCV(svc, parameters)
clf.fit(list(zip(f_or,f_ir,f_b,a_or,a_ir,a_b))), label)
clf = clf.best_estimator_

pickle.dump(clf, 'filename.pkl')

clf1 = pickle.load('filename.pkl')

newv = [0.45, 0.22, 0.33, 300.4, 200.1, 20.45]
clf1.predict(list(newv))
```

FONDAZIONE
BRUNO KESSLER

Outline

- Recap on Predictive Maintenance
- Introduction to machine learning in python
- Introduction to MQTT and json/protobuf in python

Introduction to mqtt and protobuf

- MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol.
- It was designed as an extremely lightweight publish/subscribe messaging transport.
- It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium.
- It is also ideal for mobile applications because of its small size, low power usage, minimised data packets, and efficient distribution of information to one or many receivers
- You can Install the python client using **PIP** with the command
  - pip install **paho-mqtt**

Introduction to mqtt and protobuf

- The core of the client library is the **client class** which provides all of the functions to **publish messages** and **subscribe to topics**.

- The paho mqtt client class has several methods.
  - The main ones are:
    - connect() and disconnect()
    - subscribe() and unsubscribe()
    - publish()

Introduction to mqtt and protobuf

- Creating a Client Instance
  - The client constructor takes 4 optional parameters, but only the client_id is necessary, and should be unique.
  - Client(client_id=””, clean_session=True, userdata=None, protocol=MQTTv311, transport=”tcp”)

- Connecting to an MQTT Broker or Server
  - Before you can publish messages or subscribe to topics you need to establish a connection to a broker. To do this use the **connect** method of the Python mqtt client. The method can be called with 4 parameters. The connect method declaration is shown below with the default parameters.
  - connect(host, port=1883, keepalive=60, bind_address="")

- Publishing Messages
  - Once you have a connection you can start to publish messages. To do this we use the **publish** method. The publish method accepts 4 parameters. The parameters are shown below with their default values.
  - publish(topic, payload=None, qos=0, retain=False)

Introduction to mqtt and protobuf

- Subscribing To Topics
  - To subscribe to a topic you use the **subscribe** method of the Paho MQTT Class object. The subscribe method accepts 2 parameters – A topic or topics and a QOS (quality of Service) as shown below with their default values.
  - **subscribe**(topic, qos=0)

- When a client subscribes to a topic it is basically telling the broker to send messages sent to that topic to itself.
  - The broker is, in effect, publishing messages on that topic.
  - When the client receives messages it invokes the **on_message** callback.
  - To handle those messages we need to activate and process the **on_message** callback.
  - To process callbacks you need to:
    1. Create callback functions to Process any Messages
    2. Start a loop to check for callback messages.

Introduction to mqtt and protobuf

```
def on_message(client, userdata, message):
    print("message received ", str(message.payload.decode("utf-8")))
    print("message topic=", message.topic)
    print("message qos=", message.qos)
    print("message retain flag=", message.retain)

# Now we need to attach our callback function to our client object as follows:

client.on_message=on_message        #attach function to callback

# and finally we need to run a loop otherwise we won’t see the callbacks.
# The simplest method is to use loop_start() as follows.

client.loop_start()    #start the loop
```

Introduction to mqtt and protobuf

```
import paho.mqtt.client as mqtt #import the client1
import time
############
def on_message(client, userdata, message):
    print("message received ", str(message.payload.decode("utf-8")))
    print("message topic=", message.topic)
    print("message qos=", message.qos)
    print("message retain flag=", message.retain)
########################################
broker_address="192.168.1.184"
print("creating new instance")
client = mqtt.Client("P1") #create new instance
client.on_message=on_message #attach function to callback
print("connecting to broker")
client.connect(broker_address) #connect to broker
client.loop_start() #start the loop
print("Subscribing to topic", "house/bulbs/bulb1")
client.subscribe("house/bulbs/bulb1")
print("Publishing message to topic", "house/bulbs/bulb1")
client.publish("house/bulbs/bulb1", "OFF")
```

Introduction to mqtt and protobuf

- MQTT message payload is a “string”.
- Typical data to be sent to a topic is a complex structure
  - e.g. the message with the accelerometer contains
    - The id of the sensor to uniquely identify the sensor
    - The accelerometer values over the sample period
    - The sampling period to properly interpret the sampled data (i.e. the interval among two samples)
    - Timestamp of the acquired data
- The complex structure needs to be serialized to be sent over MQTT, and de-serialized to properly reinterpret the message to analyze it
- There are many data serialization approaches with support for different programming languages
    - https://en.wikipedia.org/wiki/Comparison_of_data_serialization_formats
  - JSON
  - ASN. 1
  - Pickle
  - SOAP
  - Google Protocol Buffer

Introduction to JSON in python

- JSON (JavaScript Object Notation) is a lightweight data interchange format inspired by JavaScript object literal syntax (although it is not a strict subset of JavaScript).
- There are several bindings for the different languages. Proper serialization, deserialization are needed for complex structures.

```
import json
data = { "x" : [10, 20, 30],  "y" : [3, 4, 5],   "z" : {"a" : 2, "b" : 4}}
print "Original: ", data
data_str = json.dumps(data)
data_str
print "Encoded: ", data_str
dec_str = json.loads(data_str)
dec_str
print "Decoded: ", dec_str
client.publish("sensor_topic", data_str)
…
data = json.loads(msg.payload)
```

Introduction to mqtt and protobuf

- What are protocol buffers?
  - Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data, but smaller, faster, and simpler.
  - You define how you want your data to be structured once, then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages.
- Pick your favorite language
  - Protocol buffers currently supports generated code in
    - Java
    - Python
    - C
    - Objective-C
    - C++
    - Go,
    - JavaNano
    - RubyC#

Introduction to mqtt and protobuf

```
package eu.fbk.promcamp.proto;
message BearingPredictiveData {
 required string id = 1;
 required int64 timestamp = 2;
 required double MHS_f_ir = 4;
 required double MHS_f_or = 5;
 required double MHS_f_b = 6;
 required double H_fr = 7;
 required double H_f_ir = 8;
 required double H_f_or = 9;
 required double H_f_b = 10;
 required double HP_fr = 11;
 required double HP_f_ir = 12;
 required double HP_f_or = 13;
 required double HP_f_b = 14;
 required double ff = 15;
 required double amp = 16;
 enum BearingStatus {  GOOD = 0;  DEGRADING = 1;  BROKEN = 2;  }
 required BearingStatus HealthStatus = 17 [default = GOOD];
 required double HealthStatusConfidence = 18 [default = -1];
 required double RUL = 19 [default = -1];
 required double RULConfidence = 20 [default = -1];
};
```

```
syntax = "proto2";

package eu.fbk.promcap.proto;

message BearingData {
  required string id = 1;
  required int64 timestamp = 2;
  required double delta = 3;
  repeated double raw = 4 [packed = true];
};
```

Introduction to mqtt and protobuf

- To compile the python library

  protoc –I. file.protoc

- This generates a file file_pb2.py
- To use it in python

```
Import file_pb2 as pbuf
import datetime
ts = datetime.datetime.now().timestamp()

data = ibm.BearingData()
data.id = “Sensor_ID”
data.timestamp = ts
data.delta = 25641
data.raw[:] = hvibration

client.publish(“sensor_topic", data.SerializeToString())
```

```
Import file_pb2 as pbuf

data = ibm.BearingData()

data.ParseFromString(msg.payload)

print(data.id)
print(data.raw)
```

FONDAZIONE
BRUNO KESSLER

Thanks for the attention!!