TRACTIAN

Data Scientist Challenge

Condition Monitoring

This challenge is designed to assess the core skills required of a data scientist. It presents hypothetical scenarios in which you can demonstrate your knowledge and abilities by proposing effective solutions to the given problems. Before beginning the challenge, please download the files associated with each section.

Note: Each task described here must be submitted using a different Google Colaboratory notebook.

Part 1 - Signal manipulation

Associated files: *part_1.zip*.

The files contain raw vibration signals sampled with the SmartTrac accelerometer sensor in *csv* format. The filenames are formatted as "*{start}-{interval}-{sensor_id}.csv*". For the example *1623535615-3006-IAJ9206.csv*, the meaning of each field is:

- *start*: datetime of the sample. Example: 1623535615 [epoch Unix].
- *interval*: total duration of the signal. Example: 3006 [ms].
- *sensor_id*: a string that identifies the sensor. Example: IAJ9206.

Those samples refer to acceleration measurements in *g* for three axes of measurement (*X, Y, Z*).

Using this data:

1. Plot each sensor data in the time domain.
2. Apply any processing step you find relevant given the signal visualization. Then, plot each sensor data in the frequency domain.
3. Write a function that extracts the most relevant frequencies from the signal.

4. When there is excessive looseness or a bearing localized failure, there can be mechanical impacts that manifest as periodic peaks in the vibration signal and excite a natural frequency. Considering the signals from all sensors and axes, find the one containing impacts that excite natural frequencies between 300 Hz and 500 Hz. Use a statistical metric to quantify the presence and extent of the impacts.

Part 2 - Carpet detector

Associated files: *part_2.zip*.

Lubrication issues, whether due to contaminants, incorrect lubricant, or lack of lubricant, can cause distributed wear on the bearings. This anomalous condition manifests in the vibration spectrum by generating random noise (background noise), typically in the form of a **'carpet'** as it approaches the natural frequencies. Carpet patterns can be perceived as a series of spectral peaks that are randomly close to each other, and its detection is of great importance in order to diagnose lubrication problems. Figure 1 shows an example of a carpet region, along with an illustration of what does not qualify as a carpet—namely, a series of regularly spaced spectral peaks.

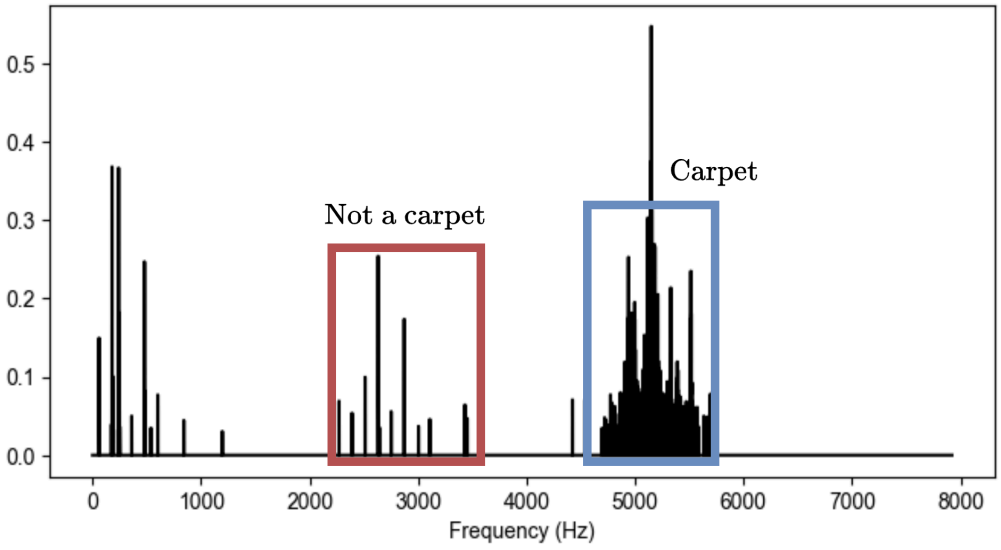

Figure 1: Example of a carpet region.

The files from *part_2.zip* contain raw vibration signals sampled with the SmartTrac accelerometer sensor in *csv* format, where column '*t*' refers to the time in seconds and '*data*' is the acceleration in *g*.

Using this data:

1. Develop a model to detect the carpet regions that might be present in the signal. Requisites:
    a. The model must be implemented according to the following Python template. You must use the classes provided to represent the data.
    b. Valid carpets must be above 1000 Hz.

c. There can be more than one disjoint carpet region in the same spectrum. Example: 1500 Hz to 1700 Hz and 3000 Hz to 5000 Hz.
d. There cannot be overlapping carpet regions.

```python
from pydantic import BaseModel, Field
from typing import List


class CarpetRegion(BaseModel):
    start_hz: float = Field(..., description="Start frequency in Hz")
    end_hz: float = Field(..., description="End frequency in Hz")


class Wave(BaseModel):
    time: List[float] = Field(..., description="Time points of the wave")
    signal: List[float] = Field(..., description="Signal values")


class Model:
    def __init__(self, **params):
        # Store hyperparameters if needed
        self.params = params

    def predict(self, wave: Wave) -> List[CarpetRegion]:
        """
        Predict carpet regions from a given wave.
        This should be implemented with actual logic.
        """
        # Example placeholder
        raise NotImplementedError("Predict method not implemented.")
```

2. Detect carpets on all the signals provided and plot each spectrum with the corresponding identified region.

3. Using the results, propose a metric to quantify the carpet severity and find the sample with the worst carpet symptom.

Part 3 - Looseness prediction

Associated files: *part_3.zip*.

Looseness is a common fault condition that can lead to excessive vibration. Structural looseness typically involves loose bolts in non-rotating components. It is important to address this issue promptly to prevent the development of additional failures caused by the resulting vibrations.

The files from *part_3.zip* are described below:

- *data*: contains raw vibration signals in *csv* format, where column '*X-Axis'* refers to the time in seconds and columns '*Ch1 Y-Axis', 'Ch2 Y-Axis'* and *'Ch3 Y-Axis*' are the acceleration signals in *g* corresponding to axes *X*, *Y* and *Z*, respectively. Each filename corresponds to the ***sample_id*** of the sample.
- *part_3_metadata.csv*: contains the mapping between each ***sample_id*** and the corresponding condition for a specific subset of samples. Note that different sensors were used, which is identified in column ***sensor_id***, and each sensor has a different ***orientation***, i.e., mapping between sensor axis (*X, Y* or *Z*) and orientation (*horizontal, vertical* or *axial*).
- *test_data*: contains a few raw acceleration signals from different assets in *g*, where columns '*x*', '*y*', and '*z*' correspond to axes *X*, *Y* and *Z*, respectively, and *t* is the time in seconds. Each filename corresponds to the ***sample_id*** of the sample.
- *test_metadata*: contains information about each test ***sample_id***, including the rotational speed, orientation mapping, and the type of asset.

The content from *data* was produced by inducing a structural looseness condition in laboratory experiments, using a motor, bearings and a coupling that are usually found in industrial applications, as shown in Figure 2.

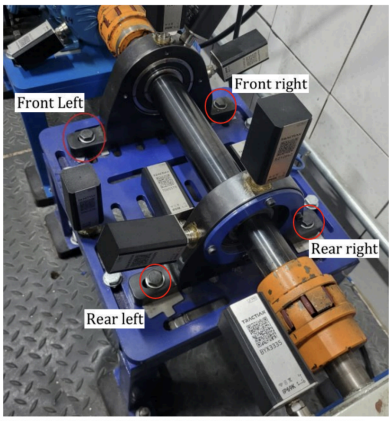

Figure 2: Laboratory test bench for structural looseness.

Using the content from *data*:

1. Extract features that might be relevant for distinguishing between a healthy and loose asset. Plot the features from the dataset using any data visualization you find relevant.
2. Develop a model that distinguishes between healthy and loose assets, and use it to predict the condition of the unlabeled samples. Requirements:
    a. The model must be implemented according to the following Python template. You must use the classes provided to represent the data.
    b. The output is a boolean, where *True* means there is structural looseness.
    c. The model must receive the waveforms corresponding to the horizontal, vertical and axial directions as input.
    d. The model can use a baseline condition if you think it is needed. However, this is not mandatory.
    e. You can use any method you find relevant (Machine Learning, Rule-based Logic, etc.), as long as you solve the problem and follow the template.
    f. [**BONUS**] Implement the *score* method that quantifies the looseness symptom with a float between *0* and *1*, where *0* means no symptom, *1* means very strong symptoms, and any value in between is proportional to the strength of the symptoms.

Using the content from *test_data*:

3. Run your model on each sample and predict its condition (loose or healthy). Compute the score for each sample (if implemented) and display the results in a table format.
4. Plot any information you find relevant about each sample (spectrum, waveform, features, etc.).

```python
from pydantic import BaseModel, Field
from typing import List


class Wave(BaseModel):
    time: List[float] = Field(..., description="Time points of the wave")
    signal: List[float] = Field(..., description="Signal values")


class LoosenessModel:
    def __init__(self, **params):
```

```
        # Store hyperparameters if needed
        self.params = params

    def predict(self, wave_hor: Wave, wave_ver: Wave, wave_axi: Wave) -> bool:
        """
        Predicts the presence of structural looseness based on horizontal,
        vertical, and axial wave data.

        Args:
            wave_hor (Wave): Horizontal wave data
            wave_ver (Wave): Vertical wave data
            wave_axi (Wave): Axial wave data

        Returns:
            bool: True if looseness is detected, False otherwise
        """
        raise NotImplementedError("The 'predict' method must be implemented.")

    def score(self, wave_hor: Wave, wave_ver: Wave, wave_axi: Wave) -> float:
        """
        Computes a confidence score (between 0 and 1) representing the
        likelihood of structural looseness. This is optional.

        Args:
            wave_hor (Wave): Horizontal wave data
            wave_ver (Wave): Vertical wave data
            wave_axi (Wave): Axial wave data

        Returns:
            float: score (0 = no looseness, 1 = high confidence of looseness)
        """
        raise NotImplementedError("The 'score' method must be implemented.")
```