STEM Games

# Technology Arena

Day 2


11 May 2023, Umag, Croatia

# Task 1:

# Ensuring Safety in Sports: Using Technology to Guide Return-to-Play Decisions

## 1.1 Introduction

Injuries occur daily in both professional, and amateur sports. Return-to-play is the process of determining when an athlete, who has been injured or ill, is able to safely return to sports participation. The return-to-play process typically involves a series of steps that an athlete must complete before being cleared to return to competition. This decision depends on several factors, including the injury location; type of the injury and its severity. In other words, the graver the injury - the longer it takes to recover.

Muscle injuries are common occurrences in professional football, affecting players of all levels of skill and experience. These injuries can range from minor strains to more severe tears, and can significantly impact a player's ability to perform at their best. Given the fast-paced nature of the game, it's no surprise that muscle injuries are so prevalent in football. Effective treatment and management of these injuries is important for players to return to play safely.

Determining the right time to return to play is crucial to prevent reinjury, as reinjuries often take longer to recover than the index injury. This is why the process is usually carefully overseen by medical professionals, however, their return-to-play predictions might not always be accurate. In this day and age, it can be assumed that their predictions could be further aided by new advances in technology.

## 1.2 Problem Statement

You've been hired by a football club to help them create a system which can automatically predict the length of recovery based on different parameters describing a muscle injury. After an athlete is injured, they are examined by the team physician, who then provides you with various criteria that outline the details of the injury. It is your task to predict return-to-play time (in days) using these parameters.

## 1.3 Data

### 1.3.1 Input

The input is given in a CSV file and contains parameters described in the table below.

| Parameter name(s) | Description | Data type |
|---|---|---|
| 'Id' | Injury identifier | integer |
| 'age' | How old is the athlete? | integer |
| 'is_contact' | Is the injury the result of a tackle? | boolean |
| 'has_stopped' | After the injury, did the player remain on the field and continue playing? | boolean |
| 'swelling' | What is the swelling level? | 0=None / 1=low / 2=moderate / 3=high |
| 'tone' | What is the tone level? | 0=None / 1=low / 2=moderate / 3=high |
| 'palpatation' | How painful is palpatation? | 0=None / 1=low / 2=moderate / 3=high |
| 'is_contraction_painful' | Is contraction painful? | boolean |
| 'is_stretching_painful' | Is stretching painful? | boolean |
| 'class' | Injury classification according to BAMIC. | Integers in range 0-4 |
| 'is_proximal', 'is_abdominal', 'is_distal' | Position of the injury according to the injured muscle | boolean |
| 'fascia_depth' | Depth of the injury | intergers in range 0-3, 0 is shallowest and 3 is deepest |
| 'is_hamstring', 'is_quadriceps', 'is_add_abd', 'is_calf', 'is_belly' | Location of the injury (hamstring, quadriceps, adductors / abductors, calf, belly) | boolean |

#### Example Input

Five example input samples are shown below in table.

| Id | age | is_contact | has_stopped | swelling | ... | is_add_abd | is_calf | is_belly |
|---|---|---|---|---|---|---|---|---|
| 1 | 23 | 1 | 0 | 1 | ... | 1 | 1 | 0 |
| 2 | 28 | 0 | 0 | 1 | ... | 1 | 1 | 1 |
| 3 | 29 | 1 | 1 | 2 | ... | 0 | 1 | 0 |
| 4 | 13 | 1 | 1 | 3 | ... | 1 | 0 | 0 |
| 5 | 19 | 1 | 1 | 2 | ... | 0 | 1 | 0 |

### 1.3.2 Output

| Parameter name(s) | Description | Data type |
|---|---|---|
| 'Id' | Injury identifier | integer |
| 'injury_duration' | Return-to-play time, measured **in days** | numerical |

The output should contain two columns, the first column should be the injury identifier 'Id', whereas the second column should contain predicted values, measured **in days**.

**Example Output**

For the five injury instances given in the example input, their respective example output (i.e. predicted return-to-play) is given below.

| Id | injury_duration |
|---|---|
| 1 | 19 |
| 2 | 19 |
| 3 | 21 |
| 4 | 19 |
| 5 | 7 |

### 1.3.3 Files

- `kaggle_x_train.csv` - contains the observed independent values, as given to you by the team physician.
- `kaggle_y_train.csv` - contains the actual return-to-play duration measured in days, for each of the injuries recorded in `kaggle_x_train.csv`
- `kaggle_x_test.csv` - this is the test data for which you need to submit your RTP duration predictions, measured in days.

## 1.4 Evaluation

Solutions will be ranked by the obtained RMSE value.

### 1.4.1 Mean Squared Error (MSE)

Mean Squared Error (MSE) is a metric used to measure the average squared difference between the actual and predicted values in a regression problem. It calculates the average of the squared differences between predicted and actual values.

Regression analysis uses MSE as a performance indicator to assess how well a model fits the data. When comparing projected and actual values, a lower MSE value suggests a better fit, whereas a larger MSE value implies a worse fit. As a result, MSE is frequently employed as a gauge of how well a predictive model can forecast values for brand-new data.

MSE is formulated as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where $n$ is the number of samples in the dataset, $y_i$ is the actual value of the $i$-th sample, and $\hat{y}_i$ is the predicted value of the $i$-th sample.

### 1.4.2 Root Mean Squared Error(RMSE)

RMSE stands for Root Mean Squared Error. Like MSE, it measures the average squared difference between the predicted values and the actual values. However, RMSE takes the

square root of the MSE, which makes it easier to interpret because it is in the same units as the dependent variable.

RMSE is calculated as:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

### 1.4.3 Example

To see how MSE and RMSE differ, let us see a general example. Let's observe some fictitious dependent variable $y$ which is measured in seconds.

Let us assume the real, observed values are:

$$y = [20, 87, 44]s$$

Our fictional regression algorithm has predicted the following values:

$$\hat{y} = [17, 89, 42]s$$

For these two sets of values, MSE and RMSE scores are:

$$MSE = \frac{1}{3}\left[(20-17)^2 + (87-89)^2 + (44-42)^2\right] \approx 5.67s^2$$

$$RMSE = \sqrt{MSE} \approx 2.38s$$

It should be highlighted how RMSE is expressed in seconds, which is the same as the dependent variable $y$, but MSE is expressed in squared seconds.

# Task 2:

# Ensuring Safety in the Pharmaceutical Industry: Calibration model for crystallization process

## 2.1 Introduction

The Laboratory for Automation and Measurements announced a new project in cooperation with the pharmaceutical industry. Through this project, they bought a new process analytical spectrometer with which they want to measure the concentration of API (Active Pharmaceutical Ingredient – the main component of the medicine) in the crystallizer (crystallization reactor) in real time. This process equipment enables in-line determination of essential properties of the crystallization process, which in the case of most APIs, enables safer work for all scientists and technicians working on the plant, as APIs are toxic and can lead to greater health consequences with prolonged exposure.

A spectrometer is a device that measures the amount of absorbed light rays of a certain range of wavelengths. These readings are named spectrums and differ depending on the concentration and temperature of the solution we are analyzing. The higher the concentration of the solution, the greater the absorption of the wavelength characteristic of that compound. In order for scientists to be able to obtain the actual concentration of the solution from these spectra, they must conduct a calibration experiment. The experiment is based on the examination of spectra obtained for a known concentration at different temperatures. Practically, this means that we make a solution of a certain concentration, known to us, and cool the solution in the reactor to a certain temperature, and heat it again to the initial temperature, at the same speed. We repeat the experiment for different concentrations.

## 2.2 Problem Statement

Scientists of the Automation and Measurement Laboratory have conducted a calibration experiment and now want to develop a model that will be able to calculate the actual concentration for any spectrum recorded in the reactor. They tried various commercial applications for this purpose, but they soon realized that these commercial applications do not give sufficiently accurate results, and since they develop this method for the pharmaceutical

industry, the accuracy criteria are extremely high. They decided that they would have to create the model themselves. When they started to get information and realized how much time would be spent on creating such a model, preprocessing and selection of relevant data for that model, they decided to hire a team of programmers to do it for them.

## 2.3 Data

All input and output variables are floating-point values.

### 2.3.1 Input

Input data is a set of spectral data (1 - 1042) gathered at a certain temperature (Temp).

| Parameter name(s) | Description | Data type |
|---|---|---|
| 'Id' | Sample identifier | integer |
| 'Temp' | Temperature of the solution, in Celsius | float |
| '1', '2','3', ..., '1040', '1041', '1042' | Each parameter represents absorption value recorded at that wavelength (spectrum) | float |

#### Example input

| Id | Temp | 1 | 2 | 3 | 4 | ... | 1040 | 1041 | 1042 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 20.03 | 0.10237 | 0.10745 | 0.11825 | 0.12769 | ... | 0.122708 | 0.14160 | 0.13809 |
| 2 | 17.82 | 0.103714 | 0.108472 | 0.119639 | 0.129942 | ... | 0.100451 | 0.096718 | 0.117435 |
| 3 | 30.03 | 0.102719 | 0.106607 | 0.117073 | 0.127244 | ... | 0.11679 | 0.082568 | 0.099011 |
| 4 | 27.03 | 0.077873 | 0.080579 | 0.091036 | 0.102673 | ... | -0.02511 | -0.02891 | -0.03958 |
| 5 | 43.88 | 0.075807 | 0.078954 | 0.088894 | 0.099848 | ... | -0.00912 | -0.00745 | 0.001699 |

### 2.3.2 Output

The output should contain two columns, the first column should be the sample identifier 'ID', whereas the second column should contain predicted values, the concentration of the solution, measured in **g/kg**.

| Parameter name(s) | Description | Data type |
|---|---|---|
| 'Id' | Sample identifier | integer |
| 'Conc' | Concentration of the solution, measured in **g/kg** | floating-point |

#### Example output

This is the output of the entries given in the example input.

| Id | Conc |
|---|---|
| 1 | 89.1 |
| 2 | 98.7 |
| 3 | 110.2 |
| 4 | 140.3 |
| 5 | 159.7 |

### 2.3.3 Files

- `kaggle_x_train.csv` - contains the observed independent values, the measurements obtained from the spectrometer.
- `kaggle_y_train.csv` - contains the dependent variable, the actual concentration of the solution, for each of the samples found in `kaggle_x_train.csv`.
- `kaggle_x_test.csv` - data for which you need to predict the output concentration and submit the result.

## 2.4 Evaluation

Solutions will be ranked by the obtained Root Mean Square Error (RMSE) value. For details on how it is calculated, please refer to section 1.4 *Evaluation*

# Task 3:

# How to Outsmart Big Joseph's Jazzy Jukebox: Tackling Melodies with Varying Amplitudes

## 3.1 Introduction

In the depths of Istria lies a giant known as Big Joseph. He guards the sacred server room of the GOAT company, and his job is to make sure that only the chosen ones can enter. How does he do it? By wielding an ancient Istrian instrument called the "Sopele". Those who seek entry must recognize the tune and play it back as a password. But beware, the Sopele's sound is so ghastly, it could make a grown man weep and flee in terror! Think you're tough enough to handle it? Give it a shot, if you dare!

Listen to *ODE TO JOY in Istrian scale*

The owners of the GOAT company want to put their security system to the ultimate test! Luckily for you, they've equipped you with a device that shields you from the most terrifying frequencies of the dreaded Sopele. With this nifty gadget, you'll be able to filter out the terror and convert the sound into a neat little array of amplitudes that you can work with. So, put on your detective hat, and let's see how many of Big Joseph's melodies you can crack! The durability of the system is at stake, and the fate of the company rests in your hands!

Ah, the plot thickens! It seems that Big Joseph likes to keep things interesting by experimenting with his playing style and position, which means that even the same melody (tone) could have slightly different arrays of amplitudes. But fear not, for we can build a model to bypass this issue!

We'll need to take into account the variations in playing style and position and come up with a way to detect and recognize the key elements of each melody. One way to do this would be to use machine learning algorithms to analyze the amplitudes of each tone and look for patterns in the variations.

We can also train the model on a wide range of melodies played by Big Joseph, to help it learn how to distinguish between different playing styles and positions. With enough data and a well-designed model, we should be able to crack the code and bypass any variations

in the amplitudes of Big Joseph's melodies.

So, let's get to work and build a model that will make even the mighty Big Joseph quiver in his boots!

## 3.2 Problem Statement

The melody always contains a single tone which is represented by an array of 441 amplitudes. An amplitude is called the harmonic in musical terms. It was derived from the original recording by using the fast Fourier transform (FFT) algorithm. The amplitudes can range from 0 to 1. It is your task to use these harmonic amplitudes and recognize the right tone played by Big Joseph.

## 3.3 Data

### 3.3.1 Input

The input is given in a CSV file and contains parameters described in the table below.

The first row represents ID of the sample. The rest of the numbers represent the amplitude of harmonics of the played tone (441 of harmonics). All given amplitudes are in range [0, 1].

| Parameter name(s) | Description | Data type |
|---|---|---|
| 'Id' | Identifier of the played tone | integer |
| '0','1','2',..., '438', '439','440' | Each of these 441 parameters describe harmonic amplitude | float |

#### Example Input

Example values for five different input samples are given below.

| Id | 0 | 1 | ... | 439 | 440 |
|---|---|---|---|---|---|
| 1 | 0.008092296 | 0.0051510693 | ... | 0.008900967 | 0.0051510693 |
| 2 | 0.005733023 | 0.007744874 | ... | 0.09058686 | 0.007744874 |
| 3 | 0.00015769545 | 3.9955437e-05 | ... | 0.0002253624 | 3.9955437e-05 |
| 4 | 0.00041158788 | 3.494455e-05 | ... | 0.00013812435 | 3.494455e-05 |
| 5 | 0.027826631 | 0.00055753894 | ... | 0.0008550674 | 0.00055753894 |

### 3.3.2 Output

The output is a label that represents a single Sopela tone. The possible labels are: **[m0, m1, m2, m3, m4, m5, v0, v1, v2, v3, v4. v5]**.

| Parameter name(s) | Description | Data type |
|---|---|---|
| 'Id' | Indentifier of the played tone | integer |
| 'labels' | The predicted tone Big Joseph played | categorical |

**Example Output**

For the five given example inputs, their respective predicted labels are given below.

| Id | labels |
|---|---|
| 1 | m1 |
| 2 | v5 |
| 3 | m1 |
| 4 | m1 |
| 5 | m4 |

### 3.3.3 Files

- `x_train.csv` - contains the observed independent values, the amplitudes of harmonics comprising a single tone.
- `y_train.csv` - contains the dependent variable, the actual played Sopela tone, for each of the samples found in `x_train.csv`
- `x_test.csv` - data for which you need to predict the played Sopela tone and submit the result.

## 3.4 Evaluation

The models will be evaluated by their accuracy score. Classification accuracy is calculated by comparing the predicted class labels to the actual class labels and counting the number of correct predictions. Then, the number of correct predictions is divided by the total number of predictions made by the model. The classification accuracy $A$ of the model is given by:

$$A = \frac{1}{n}\sum_{i=1}^{n}[y_i = \hat{y}_i]$$

where $n$ is the total number of instances in the data set, $y_i$ is the true label of the $i$-th instance, and $\hat{y}_i$ is the predicted label of the $i$-th instance. The indicator function $[y_i = \hat{y}_i]$ evaluates to 1 if the true label $y_i$ and the predicted label $\hat{y}_i$ are equal, and 0 otherwise.

### 3.4.1 Example

Suppose we have a dataset of 10 instances, where each instance belongs to one of 4 classes ("A", "B", "C", "D"), and the observed $y_i$ values are:

$$y = [A, B, A, B, C, C, D, A, D, B]$$

We have a classification model that predicts the class label for each instance, and we want to evaluate its accuracy. After running the model, we obtain the following predicted classes:

$$\hat{y} = [A, B, A, B, D, C, B, A, C, B]$$

The indicator function will tell us what the correct predictions were (1 means that the prediction is correct, 0 means the model didn't predict correctly):

$$[y = \hat{y}] = [1, 1, 1, 1, 0, 1, 0, 1, 0, 1]$$

Which means that our model predicted correctly 7 out of 10 times. Accuracy $A$ can then be calculated as:

$$A = \frac{1}{n} \sum_{i=1}^{n} [y_i = \hat{y}_i] = \frac{7}{10} = 0.7 = 70\%$$