Federico Alvetreti
Ioan Corrias
Lucia Dicunta
Leonardo Di Nino

TherMike

Statistical Learning

Final Project

TherMike - Hearing hot loud

Data Collection

Feature Engineering

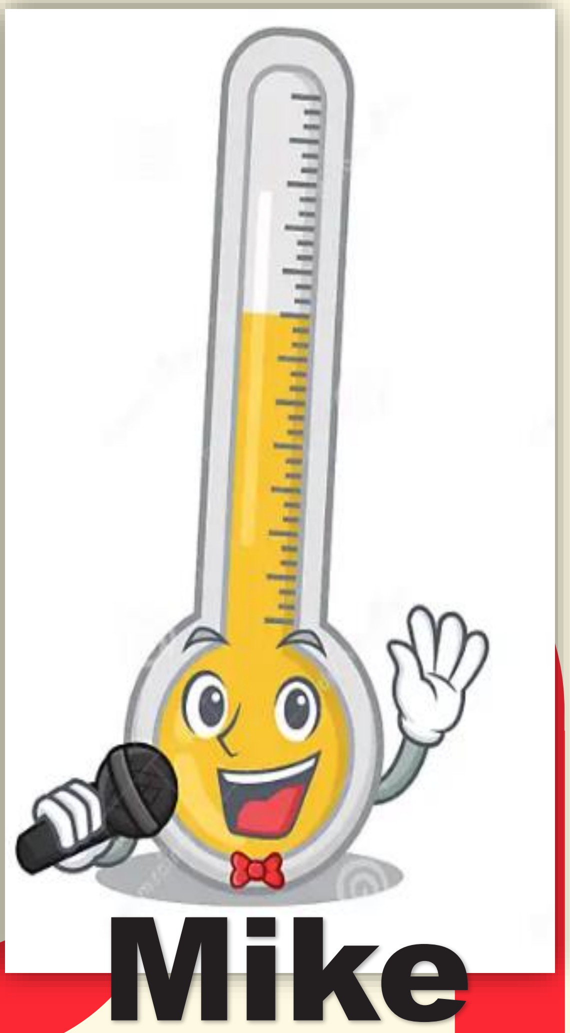
Statistical Framework

Failures and Successes

TherMike- Hearing hot loud

We were inspired by Mr.Brutti sound experiments and we decided to dig deeper in the problem. TherMike arises as the opportunity to deploy a machine learning model that can interpret or explain a phenomenon that is actually well known in neurosciences and psychoacoustics:

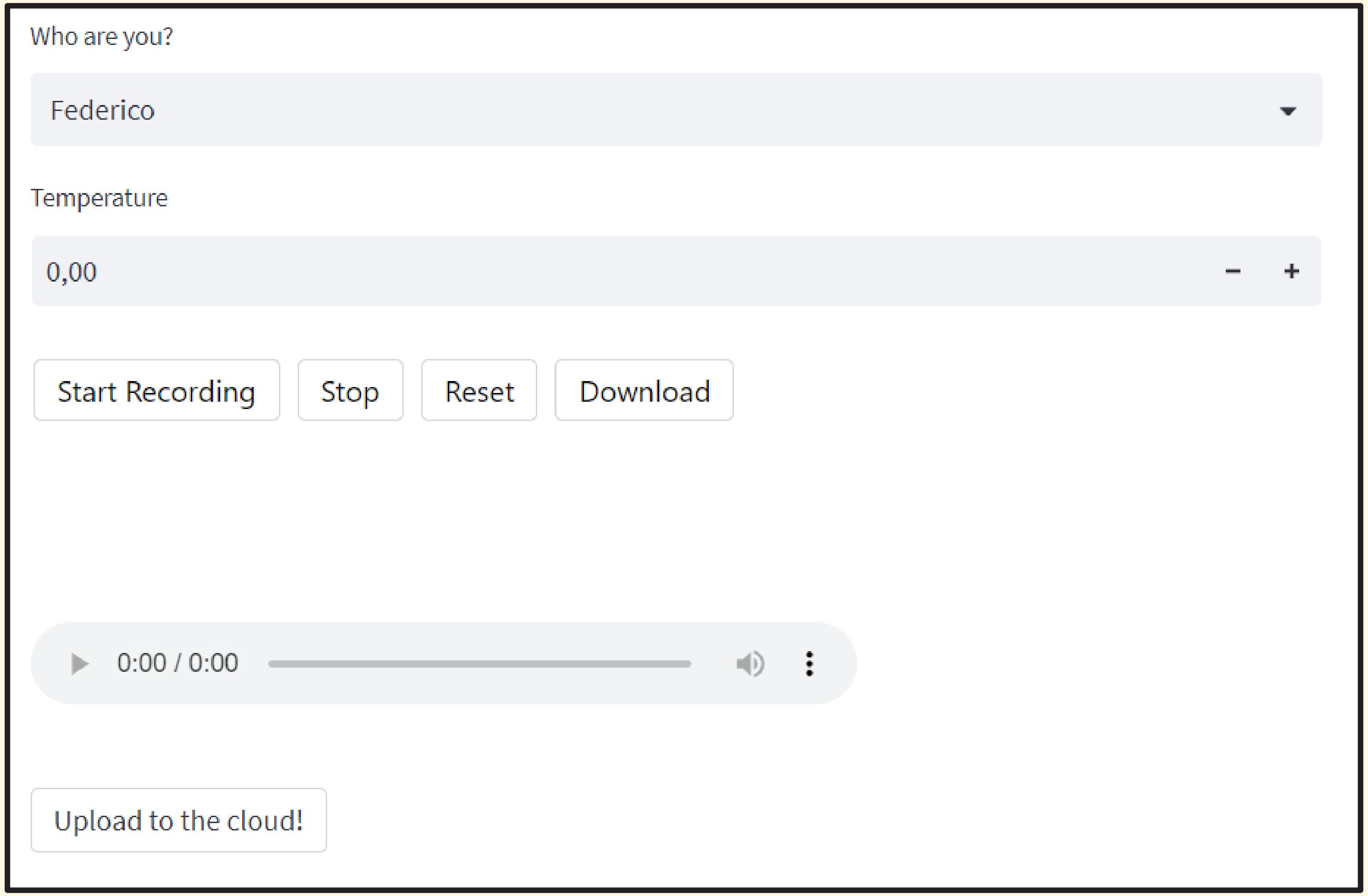
<<how is it possible for our ear and brain to collect and decode information that are apparently unrelated with what we are actually hearing? >>



So, we decided not to build a model "simply" capable of classifying if the water is cold or hot given its splashing sound: ***<u>we wanted to build a real thermometer able to listen to temperature.</u>***

This makes our problem a regression one: given a pouring sound, we want to predict the temperature. And now lots of questions arises!

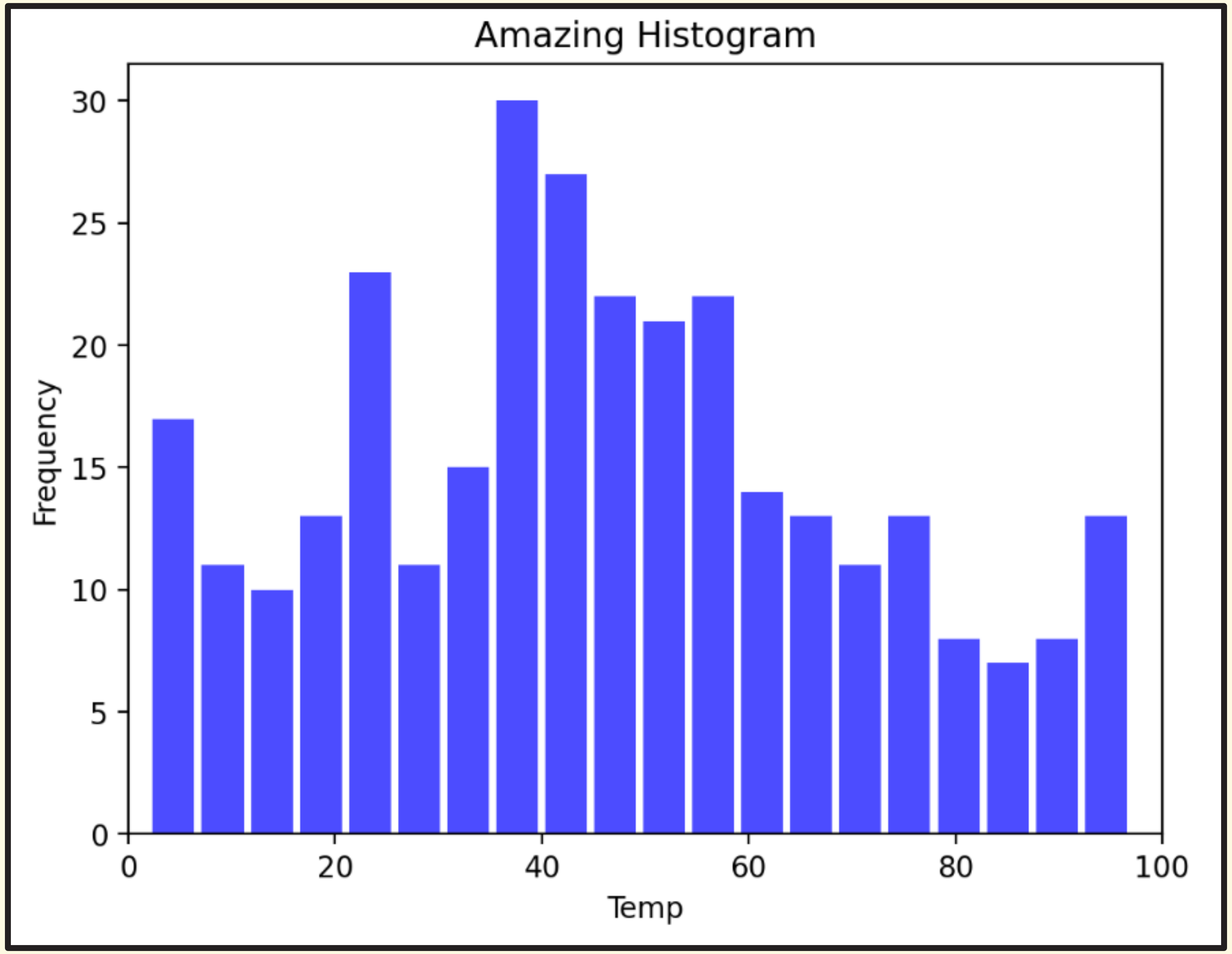
Data Collection



Strictly from a practical perspective, Ioan built an app on StreamLit to allow each of us to collect data on our own. The data stream was processed and stored on a shared Google Drive and each file has been labeled with the author name and the detected temperature.

Data Collection

A **histogram** updating after each update shows the **distribution of the labels**: this was an implementation that has been inserted in order to keep us from building an unbalanced dataset.

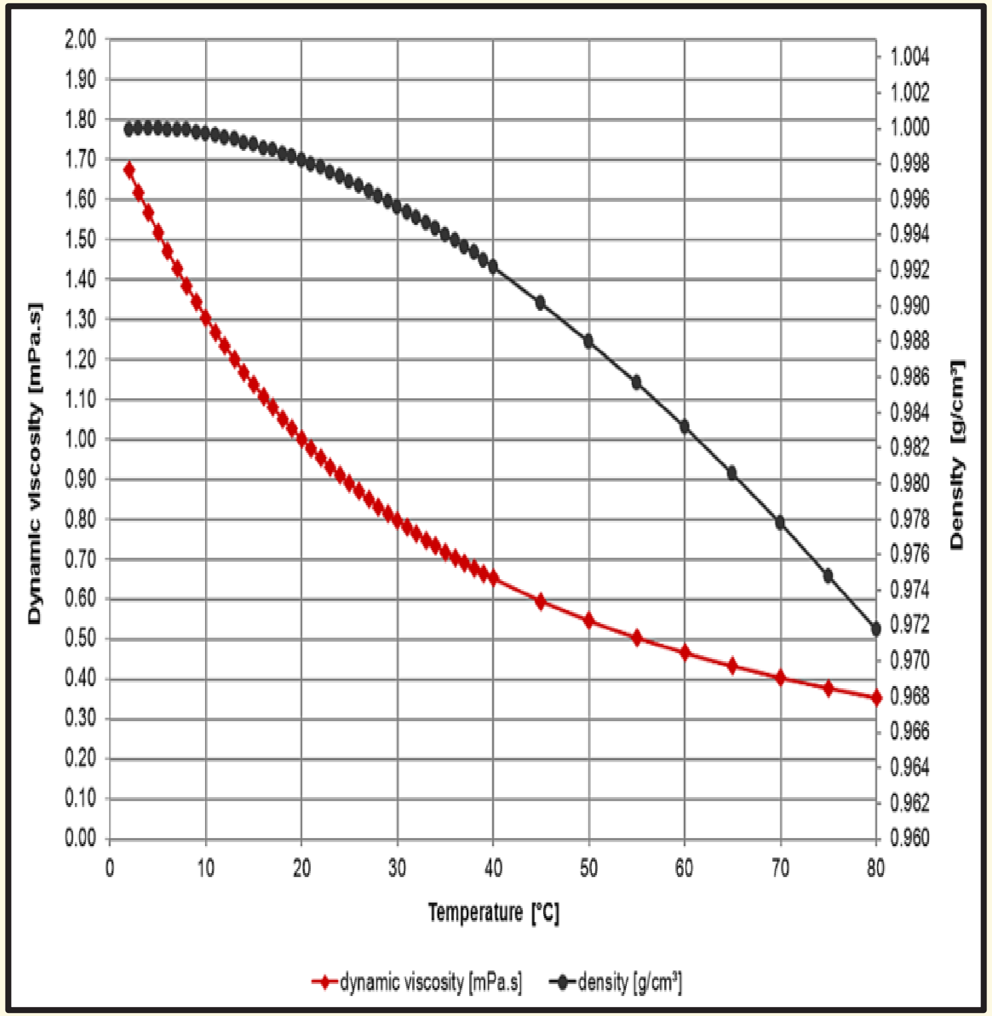


The experiment was easy in its guiding lines: pour some water, record its splash, detect the temperature. We just decided for some parameters to be similar in order to gather homogenenous data:

- We used two metal vessels;
- We poured water from no more than 10 cm away;
- We detected the temperature right after pouring the water;
- We recorded audio lasting between 5 and 7 seconds.

In the end we collected 309 audio samples on our own. Additionally, we also used as benchmark dataset the one from the similar study we read about, that consisted of 333 audio samples.

Feature Engineering

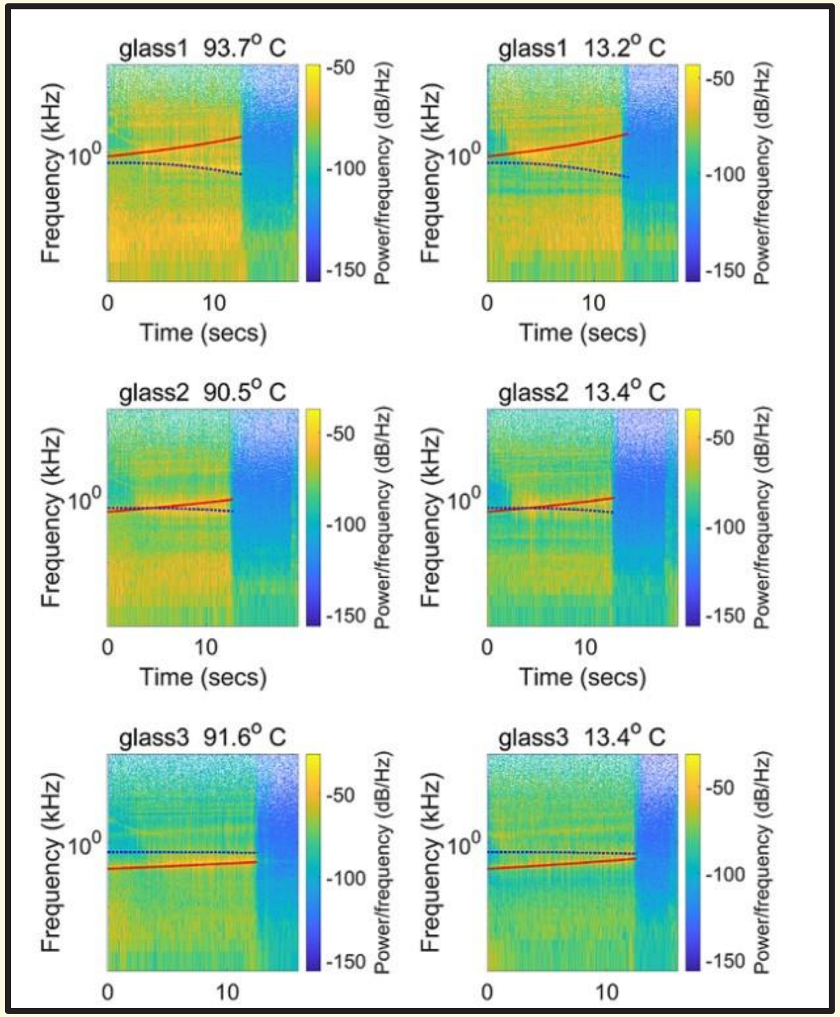


There clearly is an empirical evidence in discerning between hot and cold water: the viscosity and density of the liquid change with respect to the temperature, so it is expected to cast different sounds when poured. This is a good starting point, but at the same time it is not enough to explain acoustic phenomena.

Feature Engineering

Infact when comparing spectrograms of audio recording of liquid poured at the different temperature there is no apparent difference between the two scenarios.

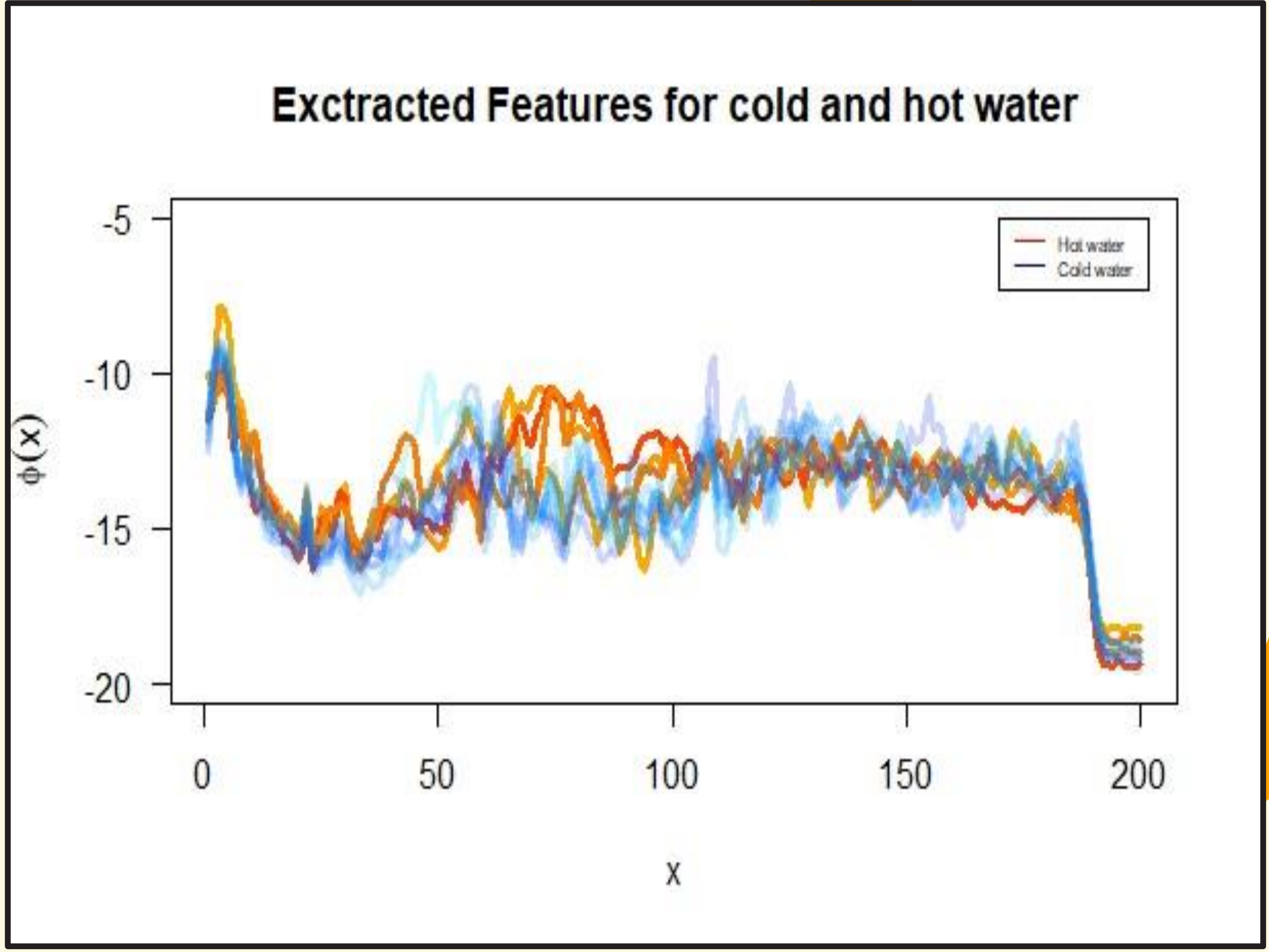
This meant to us that the interesting findings were not in the harmonic content per se but somewhere else.



Feature Engineering

We tried to model the perception of tone color with respect to temperature designing a customized feature extraction pipeline inspired by *Mel filters cepstrum coefficients* and the human hearing mechanism.

We did the following:
1. Filter the power spectrum with a Mel-Filter-bank to return an estimate of the perceived pitch;
2. Apply Weber-Fechner law of psychophysics that models the intensity of the perception as a *logarithm*.



```r
# Define a custom audio object given a .wav file path
get_audio_object <- function(audio_file_path){

  # Load audio file
  audio <- readWave(audio_file_path)

  # Get sample size
  n <- length(audio@left)

  # Get sampling rate
  sampling_rate <- audio@samp.rate

  # Get frequencies
  frequencies <- c(0:(n-1)) * (sampling_rate / n)
  frequencies <- frequencies[1:(n %/% 2)] # Halve since it is real valued

  # Get power spectrum
  power_spectrum <- Mod(fft(c(audio@left))^2) / n
  power_spectrum <- power_spectrum[1:(n %/% 2)] # Halve since it is real valued

  # Get label
  num_part <- unlist(strsplit(audio_file_path, "_"))[4]
  label <- as.numeric(substring(num_part, 1, nchar(num_part) - 4))

  # Get author
  author <- unlist(strsplit(audio_file_path, "_"))[3]

  return(list("Author" = author,
              "Recording" = audio@left,
              "Label" = label,
              "Frequencies" = frequencies,
              "Power_spectrum" = power_spectrum,
              "Sample_size" = n,
              "Sampling_rate" = sampling_rate))
}
```

```r
# Get all audio objects given a folder path
get_audio_objects <- function(directory_path){

  files <- list.files(path = directory_path) # Get audio file names
  n_iter <- length(files) # Get number of iterations

  # Set up a progress bar
  pb <- txtProgressBar(min = 0,      # Minimum value of the progress bar
                       max = n_iter, # Maximum value of the progress bar
                       style = 3,    # Progress bar style [1, 2, 3]
                       width = 50,   # Progress bar width
                       char = "=")   # Character used to create the bar

  i = 1 # Set iterator to update the progress bar

  audio_objects <- list()

  for(file_path in files){
    audio_objects <- append(audio_objects,
                        list(get_audio_object(paste(directory_path,
                                                    file_path,
                                                    sep="\\"))))

    setTxtProgressBar(pb, i) # Update progress bar
    i = i + 1
  }

  close(pb)
  return(audio_objects)
}
```

```r
# Compute MFCCs of an audio's power spectrum using mel-filterbank analysis
mel_filter_feature <- function(audio_obj, n = 200){

  # Retrieve the power spectrum
  power_spectrum <- audio_obj$Power_spectrum

  # Calculate the mel filterbank
  mel_filter_bank <- melfilterbank(f = audio_obj$Sampling_rate,
                                   wl = 2 * length(power_spectrum),
                                   m = n)

  mel_filter <- mel_filter_bank$amp # Extract mel filter amplitudes
  mel_freq <- 1000 * mel_filter_bank$central.freq # Extract central frequencies

  # Apply mel filterbank to the power spectrum and normalize
  mel_ps <- (c(power_spectrum %*% mel_filter)) / colSums(mel_filter)
  mel_ps <- log(mel_ps / sum(mel_ps * mel_freq) )

  return(list("Mel" = mel_ps,
              "Label" = audio_obj$Label,
              "Author"= audio_obj$Author))
}


# Apply mel_filter_feature to a list of audio objects
mel_filter_features <- function(audio_obj_list, n = 200){

  n_iter <- length(audio_obj_list) # Get number of iterations

  # Set up a progress bar
  pb <- txtProgressBar(min = 0,      # Minimum value of the progress bar
                       max = n_iter, # Maximum value of the progress bar
                       style = 3,    # Progress bar style [1, 2, 3]
                       width = 50,   # Progress bar width
                       char = "=")   # Character used to create the bar

  i = 1 # Set iterator to update the progress bar

  mel_features <- list()

  for(audio_object in audio_obj_list){
    mel_features <- append(mel_features,
                           list(mel_filter_feature(audio_object, n)))

    setTxtProgressBar(pb, i) # Update progress bar
    i = i + 1
  }

  close(pb)
  return(mel_features)
}
```

Statistical Framework

1. Define the functional setting
2. Rebuild distances
3. Rebrand nonparametric models
4. Validate hyperparameters
5. Deploy the model!

In almost any of our attempt we ended up working with functional covariates. Since we thought that a linear model wouldn't be powerful enough to recover the underlying richness of the problem, we had to inject non-linearity. The easiest ways to do so were basically two:

- **Implement the *continuously additive model* $Y_i = \alpha + \int f(X_i(t), t)dt + \varepsilon_i$ where the functional form has to be estimated through a splines expansion;**
- **Go for *nonparametric approaches* correctly adapted for functional covariates.**

We need to define some useful ways to approximate distance between functions. Assuming $x(t), y(t) \in L^2([0,1])$ **the distance is defined as consequence of the existence of a norm, so that we have** $||x(t) - y(t)||_{L_2} = \sqrt{\int (x(t) - y(t))^2 dt}$. **We could do two things:**

- **Approximate the integral through a finite sum over the data points;**
- **Approximate the distance through a basis expansion on an orthonormal basis and then leveraging Parseval's identity and bilinearity of inner product.**

In the second case, given an orhonormal basis $\{\phi_j\}_{j=1}^{\infty}$ what happens is the following:

$$\left|\left|x(t)-y(t)\right|\right|^2_{L_2}=\sum_{j=1}^{\infty}\left|\left\langle x(t)-y(t),\phi_j\right\rangle\right|^2=\sum_{j=1}^{\infty}\left|\left\langle x(t),\phi_j\right\rangle-\left\langle y(t),\phi_j\right\rangle\right|^2=\left|\left|\beta_{\infty}^{x}-\beta_{\infty}^{y}\right|\right|$$

So from the last equality descends an approximation on a finite orthonormal basis which we expand our functions on to gather the empirical Generalized Fourier Coefficients.

This two approximations for the distance between functions are useful to be plugged into two well known estimators for the nonparametric model $Y = m(x) + \epsilon, m(x) = E[Y|X = x]$**:**

- **The Nadaraya-Watson estimator** $\widehat{m}(x) = \frac{\sum_{j=1}^{N} Y_j K(h^{-1} d(X_j,x))}{\sum_{j=1}^{N} K(h^{-1} d(X_j,x))}$ **;**
- **The k-nearest-neighbours regression** $\widehat{m}(x) = \frac{1}{|K_{nn}|} \sum_{k \in K_{nn}} Y_k$

Failures and Successes

Before presenting our successful model it is important to quickly go through our many failures. We tried many approaches in solving the problem: we are quickly going to have a look to what happened when we tried to generalize non-parametric Nadaraya-Watson estimator to a multivariate case after a full MFCC feature extraction.

In case of an exponential kernel the kernel regression estimator can be rewritten as

$$\widehat{m}(x) = \frac{\sum_{j=1}^{N} Y_j K(\Omega^T D(x, X_j))}{\sum_{j=1}^{N} K(\Omega^T D(x, Xj))}$$

Being $x = (x_1, \dots, x_p)$ **a vector of (functional) covariates,** $\mathrm{D}(x, X_j) = \begin{bmatrix} d(x_1, X_{1j}) \\ \dots \\ d(x_p, X_{pj}) \end{bmatrix}$

the vector of component-wise distances and $\Omega = \begin{bmatrix} \omega_1 \\ \dots \\ \omega_p \end{bmatrix}$ **a vector of**

weights.

Given the LOOCV prediction for each data point we can consider the following optimization problem:

$$\begin{cases} \min\limits_{\omega_1,\dots,\omega_p} \sum_{i=1}^{N} (Y_i - \hat{Y}^{-i})^2 \\ \omega_i \geq 0 \,, i = 1, \dots, p \end{cases}$$

```python
# Non-parametric regression on a vectorial functional space

def K(t):
    return(0.5*np.exp(-0.5*(t**2)))

def L2(x1,x2):
    return np.linalg.norm(x1-x2)

def weightedCompWiseDist(X1,X2,omega):
    L = np.shape(X1)[0]
    D = np.zeros(L)

    for i in range(0,L):
        D[i] = L2(X1[i,:],X2[i,:])

    return np.sum(D*omega)

def KR_estimator(x,X,Y,omega):
    weights = np.ones(len(Y))
    for i in range(len(Y)):
        weights[i] = K(weightedCompWiseDist(x,X[:,:,i],omega))
    return np.sum(weights*Y)/np.sum(weights)

# Minimization problem objective function

def objective(params):
    omega = params[0:20]
    output = 0
    for i in range(len(y_true)):
        x = design_tensor[:,:,i]
        y = y_true[i]
        _X = design_tensor[:,:,[j for j in range(0,len(y_true)) if j != i]]
        _Y = y_true[[j for j in range(0,len(y_true)) if j != i]]
        pred = KR_estimator(x,_X,_Y,omega)
        output += (pred-y)**2
    return np.sqrt(output/len(y_true))
```

```python
minimize(objective, x0 = np.array([0.1 for j in range(20)]), options={'maxiter':50}, method='Nelder-Mead')
```

```
      message: Maximum number of iterations has been exceeded.
      success: False
       status: 2
          fun: 19.21410479191212
            x: [ 8.631e-02  1.077e-01 ...  9.985e-02  9.693e-02]
          nit: 50
         nfev: 74
final_simplex: (array([[ 8.631e-02,  1.077e-01, ...,  9.985e-02,
                         9.693e-02],
                       [ 8.222e-02,  1.042e-01, ...,  1.018e-01,
                         9.858e-02],
```

The result after all the implementation (full implementation of the pipeline is on the report) are quite weak. This might be due to several reasons:

- **The dataset needed more preprocessing;**
- **The eGFC have to be regularized;**
- **The optimizer lacks in precision because of the initializiation.**

The working model

The models

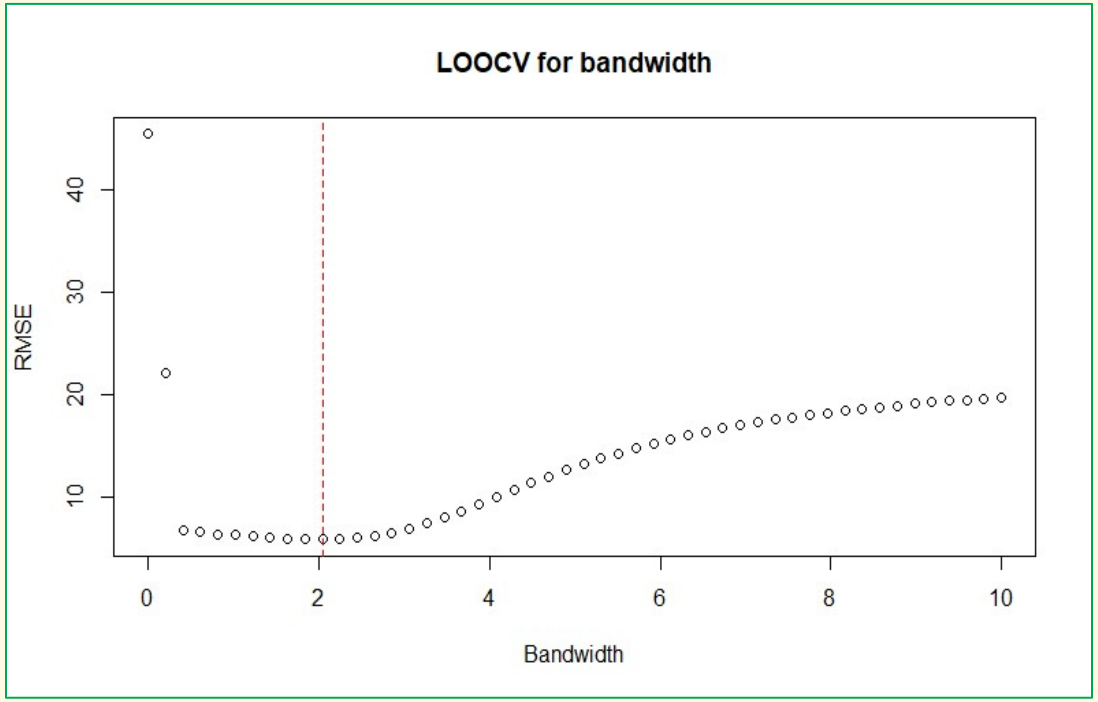
- **Nadaraya-Watston Kernel Regression**

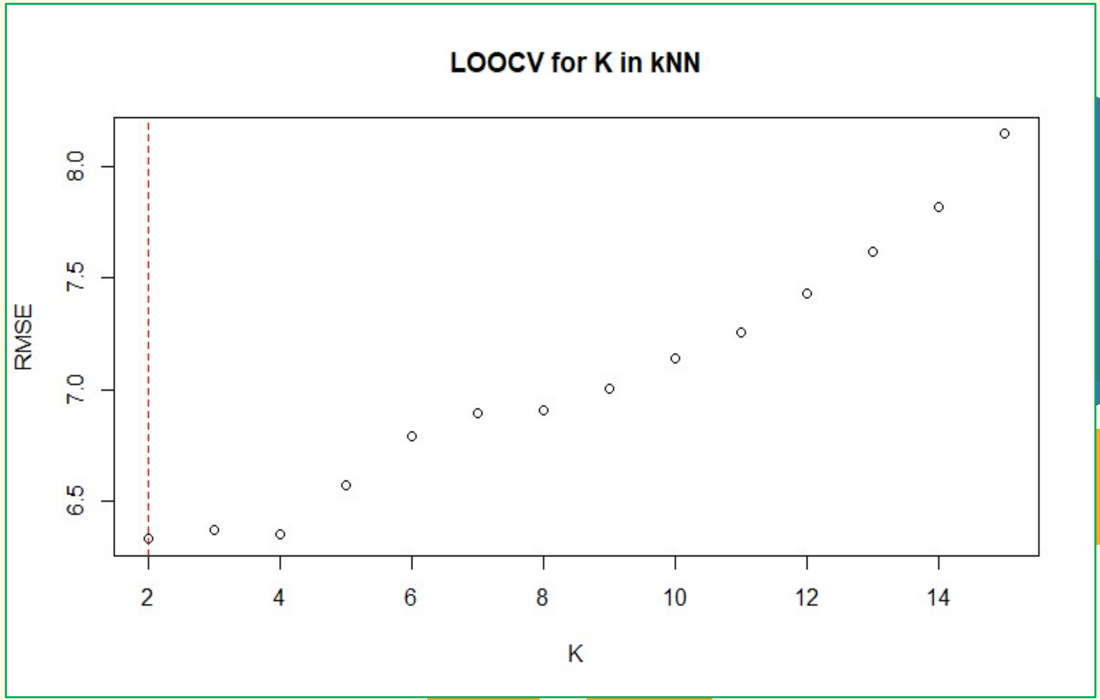
```r
KR_predict_audio ← function(train_set, new_data, h){

  # Set up a dataframe to save distances
  weights_df ← data.frame()

  # Calculate L2 distance between new_data and each element in train
  for(train_audio in train_set){
    weight ←  Kernel(L2(train_audio$Mel, new_data$Mel)/h)
    weights_df ← rbind(weights_df, c(weight, train_audio$Label))
  }

  names(weights_df) ← c("Weight", "Label")

  # Sort by weight
  weights_df ← weights_df[order(weights_df$Weight, decreasing = T), ]

  # Evaluate prediction
  tot_weight ← sum(weights_df$Weight)
  prediction ← sum(weights_df$Weight * weights_df$Label) / tot_weight

  if(is.na(prediction)) prediction ← 0

  return(prediction)
}


# Multiple prediction
KR_predict_set ← function(train_set, test_set, h){

  # Set up a dataframe to store predictions
  preds ← data.frame()

  # Predict for each element in the test_set
  for(audio in test_set){
    preds ← rbind(preds,
                  c(audio$Label, KR_predict_audio(train_set, audio, h)))
  }

  names(preds) ← c("Label", "Prediction") # Add column names
  preds ← preds[order(preds$Label), ] # Sort by Label

  return(preds)
}
```

- **KNN Regression**

```r
kNN_predict_audio ← function(train_set, new_data, K) {
  distances_matrix ← matrix(NA,nrow=length(train_set),ncol=2)
  for (i in 1:length(train_set)) {
    audio ← train_set[[i]]
    distances_matrix[i,1] ← L2(audio$Mel, new_data$Mel)
    distances_matrix[i,2] ← audio$Label
  }
  idxs ← order(distances_matrix[,1])[1:K]
  return(mean(distances_matrix[idxs,2]))
}

kNN_predict_set ← function(train_set, test_set, K) {

  # Set up a dataframe to store predictions
  preds ← data.frame()

  # Predict for each element in the test_set
  for(audio in test_set){
    preds ← rbind(preds,
                  c(audio$Label, kNN_predict_audio(train_set, audio, K)))
  }

  names(preds) ← c("Label", "Prediction") # Add column names
  preds ← preds[order(preds$Label), ] # Sort by Label

  return(preds)
}
```

LOOCV





- **h = 2.0408**
- **RMSE = 5.8674**

- **K = 2**
- **RMSE = 6.3303**

Interpretation

By looking at a sensitivitity metric, we notice that the model performs better at lower temperatures.

Visually we can see that the feature extracted for cold water are LESS dispersive for the central filters than the one extracted for hot water.

	Range	Sensitivity
r1	0-20	0.875
r2	20-40	0.7293
r3	40-60	0.3457
r4	60-80	0.3573
r5	80-100	NA

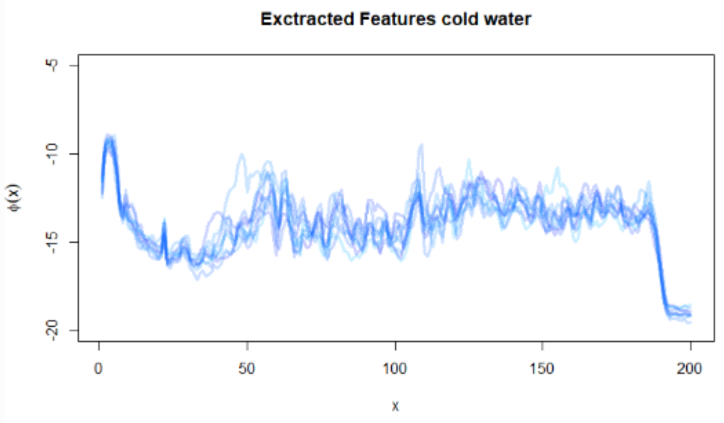


Fig. 9. Features extracte for water of temperature < 20

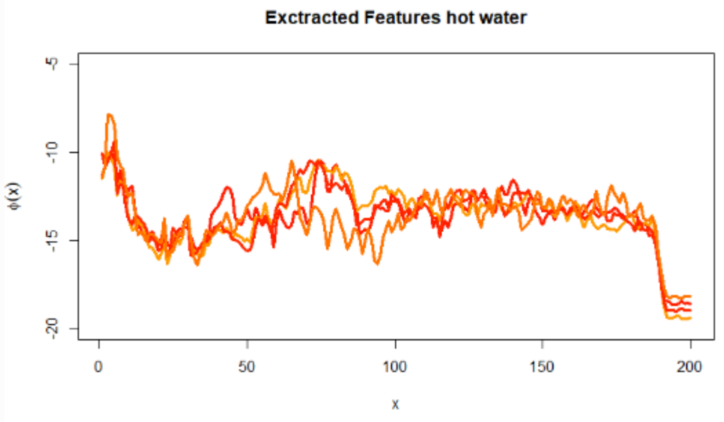


Fig. 10. Features extracte for water of temperature > 80

Thank you!