Statistical Learning

Due whenever you want before June 5th

Homework-02

(A) Data and Goal

What and where

For this homework you will work with a reduced version of the dataset I prepared for the 2020 edition of the SL-Hacka. It is already divided in train and test and can be downloaded from here. Although not huge, if you are working with `R` I strongly suggest to have a look at the `data.table` package and in particular to the function `fread()` to import the data.

The Task: Tempo Estimation

If you ever tried to play around with vinyl records and classic heavy-duty turntables, or even with lighter, shiny and more contemporary DJ-gears, the bare minimum you had to learn was how to sync the beat of two songs and come up with a smooth, silky transition between them: master that and you're halfway closer to success, as simple as that!

The musical aspects of tempo, beat, and rhythm play a fundamental role for the understanding of and the interaction with music. It is the beat, the steady pulse that drives music forward and provides the temporal framework of a piece of music. Intuitively, the beat can be described as a sequence of perceived pulses that are regularly spaced in time and correspond to the pulse a human taps along when listening to the music. The term tempo then refers to the rate of the pulse. Musical pulses typically go along with note onsets or percussive events. Locating such events within a given signal constitutes a fundamental task, which is often referred to as onset detection.

While many different **tempo estimation** techniques have been proposed, some comparative studies suggest that there has been a relatively small improvement in the state of the art recently. Current approaches for tempo estimation focus on the analysis of mainstream popular music with clear and stable rhythm and percussion instruments, which facilitates this task. These approaches mainly consider the periodicity of intensity descriptors to locate the beats, and then to estimate the tempo. Nevertheless, they usually fail when they are analyzing other music genres like classical music, because this type of music often exhibits tempo variations.

Contrary to many existing systems, which typically first identify beats and then derive a tempo, here you will try to estimate the tempo directly and blindly (read "ignorantly") from generic spectral signatures extracted from 8 seconds long song-snippets

Evaluation

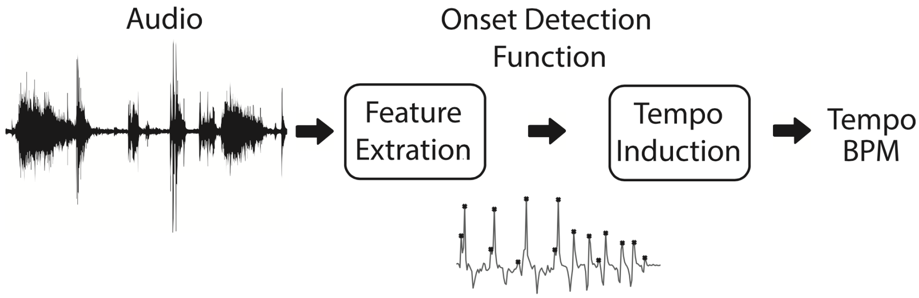
Tempo estimation can be casted as a **regression** or as a **multi-class classification** problem. Although you can "internally" work with the latter, to make things simple and familiar your submission will be scored via **RMSE**, as in a regression problem. This choice is clearly a suboptimal in this context since, as you may imagine, many musical factors can easily affect your prediction.

For the sake of completeness I will mention the other commonly used metrics:

- `Accuracy1`: it measures if an estimated tempo is within +/- 4% of the ground truth.
- `Accuracy2`: the same as `Accuracy1` but considering octave errors (multiple of the ground truth) as correct.

Data Description

The general scheme for automatic tempo estimation methods is depicted below. It represents the main steps to estimate the tempo of an audio signal.



This scheme includes a **feature list creation block** and a **tempo induction block**. You'll build the latter, whereas, regarding the former, I give you (mainly) spectral features extracted from 8 seconds long song-snippets. I will not bother you with the details, but I'll provide here a sketch of what's going on in the first box.

In the feature creation step (that I performed for you) we transform the audio waveform into a series of features representing (hopefully) predominant rhythmic information. Often, given a music recording, a short-time Fourier transform (STFT) is used to obtain a spectrogram. One simple, yet important step, which is often applied in the processing of music signals, is referred to as *logarithmic compression* and it consists in applying a logarithm to the magnitude *spectrogram* or *energy* of the signal. Such a compression step not only accounts for the logarithmic sensation of human sound intensity, but also balances out the dynamic range of the signal (for more details, see MEL-frequency cepstrum).

As you will see, the train dataset has 7042 columns/features (a bit too-much, don't you think?) that can be broken down in the following way:

1. From 1 to $171 \times 40 = 6840$: Mel-frequency cepstral coefficients of an 8 secs song-snippet sampled at 11025 (windows size = 1024 samples, overlap = 50%, number of mel-bands = 40). The result is a matrix with 171 rows (time instants) and 40 cols (MEL-frequencies) that has been column-vectorized (i.e. by stacking its 171-long columns one after the other).
2. From 6841 to 7011: Tracking of the dominant frequency (the frequency of highest amplitude) over 171 time instants.
3. From 7012 to 7019: 8 summary statistics extracted from the STFT (`time.P1` = the time initial percentile, `time.M` = the time median, `time.P2` = the time terminal percentile, `time.IPR` = the time interpercentile range, `freq.P1` = the frequency initial percentile, `freq.M` = the frequency median, `freq.P2` = the frequency terminal percentile, `freq.IPR` = the frequency interpercentile range).
4. From 7020 to 7033: 14 statistical properties of a frequency spectrum (`mean` = mean frequency, `sd` = standard deviation of the mean, `sem` = standard error of the mean, `median` = median frequency, `mode` = modal frequency, `Q25` = first quartile, `Q75` = third quartile, `IQR` = interquartile range, `cent`= centroid, `skewness` = skewness, `kurtosis` = kurtosis, `sfm` = spectral flatness measure, `sh` = spectral entropy, `prec.x` = frequency precision of the spectrum).
5. From 7034 to 7039: 6 additional statistical properties of the signal (e.g. `roughness`).
6. From 7040 to 7042: 3 annotations: `id`, `genre` and `tempo`, the target variable (not available in `test`)

⇝ Your job (A) ⇜

To achieve a `RMSE` of 8.66 (in test), the 2020-Champs essentially worked on a two steps procedure with a nonlinear dimensionality reduction followed by a SVM regression.

1. First of all, randomly pick $m = 10$ observations from the `training set` and put them aside, that is, don't use them in training/validation ⇝ you will use them in Part (B) of this homework.
2. Try to reproduce the 2020-winning solution with the tools you know.
3. I expect you to upload on Moodle a well commented working code that covers the entire pipeline with all the due explanations behind your choices: from data loading/pre-processing and feature engineering/dim-reduction to model fit and prediction on `test`.
4. You will also upload a `.csv` file containing the prediction on `test` that I will then score (RMSE).

BONUS: if you want, you may try to improve on their result by using any other method, but keep in mind that they only had 48 hours!

(B) Predicting with Confidence

Introduction

Back to Juan! Here's a recap: in statistics we usually provide confidence sets in addition to point estimates, is there a similar notion for **predictions**? The answer is yes: we provide *prediction sets* or *set-valued predictions*. Given data $\mathsf{D}_n = \{(\boldsymbol{X}_1, Y_1), \ldots, (\boldsymbol{X}_n, Y_n)\}$ we construct a set-valued function C_n, depending on D_n such that $\Pr\left(Y_{n+1} \in C_n(\boldsymbol{X}_{n+1})\right) \geqslant 1 - \alpha$.

The approach Juan described in his Anarchy-Friday is **conformal prediction**. The idea is due to Vovk, Gammerman and Shafer (2005). The statistical theory was developed in Lei, Robins and Wasserman (2013), Lei and Wasserman (2014), Lei, G'Sell, Rinaldo, Tibshirani and Wasserman (2017), Sadinle, Lei and Wasserman (2018). More recent variations on this idea can be found in Angelopoulos et al. (2021), Bates et al. (2021) and Romano, Sesia and Candès (2020).

To summarize the algorithmic side of *conformal prediction*, let's start from the **unsupervised case**: we observe $\mathsf{D}_n = \{Y_1, \ldots, Y_n\}$ and we want to predict Y_{n+1}. The basic algorithm is as follows:

Naive Algorithm

1. Define a permutation invariant *residual function* (or *non-conformity score*) $R = \phi(z, \mathsf{D}_{n+1})$ where $\mathsf{D}_{n+1} = \{Y_1, \ldots, Y_n, y\}$ is *any* **augmented dataset** of size $(n+1)$.
2. For each y we pick, do the following:
   - Set $Y_{n+1} = y$ and form the corresponding augmented dataset $\mathsf{D}_{n+1} = \{Y_1, \ldots, Y_n, Y_{n+1}\}$
   - Let $R_i = \phi(Y_i, \mathsf{D}_{n+1})$ for $i \in \{1, \ldots, n+1\}$
   - Test the hypothesis $H_0 : Y_{n+1} = y$ by computing the p-value $\pi(y) = \frac{1}{n+1}\sum_{i=1}^{n+1} \mathbb{1}(R_i \geqslant R_{n+1})$
3. Invert the test to get the confidence set $C_n = \{y : \pi(y) \geqslant \alpha\}$ for a pre-specified level $\alpha \in (0, 1)$.

Remarks

- It can be shown that $\Pr\left(Y_{n+1} \in C_n\right) \geqslant 1 - \alpha$ and this result is distribution-free and holds for all finite samples. The coverage validity does **not** depend on the choice of residual. But a poor choice can lead to large prediction sets.
- This algorithm requires to test $H_0 : Y_{n+1} = y$ for every y. In practice, we only consider a **grid** of y, but this can be slow. **Split conformal** is much faster but can result in larger prediction sets (and it depends on the split). The steps are:
  1. Randomly split D_n in two equal-sized subsets $\mathsf{D}^{(1)}$ and $\mathsf{D}^{(2)}$
  2. Compute the residuals on $\mathsf{D}^{(1)}$: $R_i = \phi(Y_i, \mathsf{D}^{(1)})$ for $Y_i \in \mathsf{D}^{(2)}$
  3. Let q be the $1 - \alpha$ quantile of the residuals $\{R_i\}_i$
  4. Return $C_n = \left\{y : \phi(y, \mathsf{D}^{(1)}) \leqslant q\right\}$, in other words, keep only the candidate predictions that are "conformal enough".

Regression

The extension to regression is straightforward. All the relevant information and algorithms can be found in Section 2.1 and 2.2 of Lei et al. (2017) and I strongly suggest you to read them. The following is their **Algorithm 2** at page 10:

Split Conformal Prediction for Regression

- *Input*: Data $\mathsf{D}_n = \left\{(\boldsymbol{X}_1, Y_1), \ldots, (\boldsymbol{X}_n, Y_n)\right\}$, miscoverage level $\alpha \in (0, 1)$, a generic regression algorithm $\mathcal{A}$
- *Output*: Prediction band over $\boldsymbol{x} \in \mathbb{R}^p$

1. Randomly split D_n in two equal-sized subsets $\mathsf{D}^{(1)}$ and $\mathsf{D}^{(2)}$
2. Train on $\mathsf{D}^{(1)}$ to get: $\widehat{f} = \mathcal{A}\left(\mathsf{D}^{(1)}\right)$
3. Predict and evaluate the residuals on $\mathsf{D}^{(2)}$: $R_i = \left|Y_i - \widehat{f}(\boldsymbol{X}_i)\right|$ for all $(\boldsymbol{X}_i, Y_i) \in \mathsf{D}^{(2)}$
4. Let $d =$ the $k^{\texttt{th}}$ smallest value of $\{R_i\}_i$ where $k = \lceil (n/2+1)(1-\alpha) \rceil$ and $\lceil \cdot \rceil$ denotes the ceiling function.
5. For any $\boldsymbol{x} \in \mathbb{R}^p$ you choose, return $C_{\texttt{split}}(\boldsymbol{x}) = \left[\widehat{f}(\boldsymbol{x}) - d, \widehat{f}(\boldsymbol{x}) + d\right]$

Also in this case it can be shown that, if $(\boldsymbol{X}_{n+1}, Y_{n+1})$ is a new datapoint, then $\Pr\left(Y_{n+1} \in C_{\texttt{split}}(\boldsymbol{X}_{n+1})\right) \geqslant 1 - \alpha$

⇝ Your job (B) ⇜

Starting from the (best) model used in Part (A), implement the *Split Conformal Prediction for Regression* algorithm.

1. Apply CP to the $m = 10$ observations you set aside in Part (A) from the `training set` and check if your intervals cover the actual response. Provide a suitable visualization of the results and comment.
2. Then, randomly pick $m = 100$ observations from the `test set` and build their predictive sets. Provide a suitable visualization of the results and comment.