Ridge regression

Omar Ghezzi[1] (matr. 984352),
project for the exam of Statistical methods for machine learning

Abstract

In this study, we analyze the foundational concepts of statistical learning, exemplifying our insights with a practical case: predicting the popularity of Spotify tracks using ridge regression on a dataset that consists of either purely continuous features (labeled as problem $c = 1$) or a mix of continuous, binary, ordinal, and categorical attributes (designated as problem $c = 2$). We detail our implementation of the closed-form solution for ridge regression and outline prominent techniques for hyperparameter tuning and optimal predictor risk estimation, such as the K-fold nested CV and the slightly less precise K-fold best CV. Our findings reveal that while ridge regression performs better on $c = 2$, a linear model tends to oversimplify the underlying relation between features and the target variable, leading to pronounced underfitting.

1 Introduction

In the following pages we present the principles of statistical learning and their application to the real-world task of predicting, using the ridge regression learning algorithm, the popularity of the music tracks included in the 'Spotify Tracks Dataset'. Section 2 lays out the foundational concepts of statistical learning. Section 3 addresses the practical challenges and implementation details of the project. Our findings and outcomes are presented in Section 4 The foundations of this work, especially the theoretical parts, are grounded in the lecture materials from the Statistical Methods for Machine Learning course, supplemented by the following references: (Shalev-Shwartz & Ben-David 2014), (Taboga 2021), (Halford 2018), (Cordone 2018) and (Wong 2023).

2 Theoretical Background

2.1 Statistical description of the learning problem

A statistical learning problem can be fully specified by the pair ($\mathcal{D}$, ℓ). Statistical learning aims to effectively model a deterministic approximation of the intrinsic relationship between the feature characterization of a phenomenon (statistically modeled as a d-dimensional random vector X of features) and its target description (statistically modeled as a random variable Y).

The population of interest (X, Y) is distributed according to a joint probability distribution $\mathcal{D} = P(\mathrm{X}, \mathrm{Y})$, which can be expressed, by marginalizing over the statistical characterization of the features, in terms of the true underlying feature-target relationship $P(\mathrm{Y}|\mathrm{X})$:

$$P(\mathrm{X}, \mathrm{Y}) = P(\mathrm{Y}|\mathrm{X})P(\mathrm{X})$$

A deterministic function h, called predictor, is considered a reliable proxy for $P(\mathrm{Y}|\mathrm{X})$ if it produces a random variable $h(\boldsymbol{X})$ that exhibits similar behavior to Y in expectation with respect to the draw of the

Corresponding author(s): University of Milan, omar.ghezzi@studenti.unimi.it

random example $(\boldsymbol{X}, Y) \sim \mathcal{D}$, which statistically represent a concrete realization of the phenomenon (X, Y), namely:

$$h \text{ is a good proxy for } P(\mathrm{Y}|\mathrm{X}) \quad \Leftrightarrow \quad \ell_{\mathcal{D}}(h) = \mathbb{E}_{(\boldsymbol{X},Y)\sim\mathcal{D}}\big[\ell(Y, h(\boldsymbol{X}))\big] \text{ is small.}$$

The **statistical risk** $\ell_{\mathcal{D}}(h)$ is the measure of the performance of a predictor h and it is computed as the expected **loss** on a random example $(\boldsymbol{X}, Y)$ drawn from $\mathcal{D}$. Typically, a learning algorithm A expresses a preference for a specific predictor, among the possible alternatives in a predetermined hypothesis class $\mathcal{H}$, by initially evaluating the candidates' performance on a sequence of random examples $(\boldsymbol{X}_1, Y_1), ..., (\boldsymbol{X}_m, Y_m)$. The risk of the predictor chosen by a A can be then assessed in expectation with respect to the random sampling of the statistical sample $(\boldsymbol{X}_1, Y_1), ..., (\boldsymbol{X}_m, Y_m)$ employed by A to build the predictor, namely[1]:

$$A\Big((\boldsymbol{X}_1, Y_1), ..., (\boldsymbol{X}_m, Y_m)\Big) \text{ is a good proxy for } P(\mathrm{Y}|\mathrm{X})$$

$$\Leftrightarrow$$

$$\mathbb{E}_{(\boldsymbol{X}_1,Y_1),...,(\boldsymbol{X}_m,Y_m)\sim\mathcal{D}}\Big[\ell_{\mathcal{D}}\big(A\big((\boldsymbol{X}_1, Y_1), ..., (\boldsymbol{X}_m, Y_m)\big)\big)\Big] \text{ is small.}$$

However, it is not possible to fully comprehend a phenomenon (Y, X) by statistically describing it through the probability distribution $\mathcal{D}$, as it would necessitate access to all its potential realizations. Conversely, the characteristics of the population of interest can be inferred from a limited number of observations or exemplifications of the phenomenon. In statistical learning, the assumption is made that the population being studied can be effectively represented or exemplified by a dataset consisting of m specific observations. These observations are considered representative of the reference population as they are mathematically modeled as independent random examples drawn from the joint probability distribution $\mathcal{D}$.

2.2 Elements of a learning problem

In this paragraph, we provide a rigorous definition of the elements comprising a learning problem, emphasizing the relationship between the abstract description of the problem and the practical methodologies used to extend the analysis from a limited set of realizations to the entire reference population:

Statistical sample A statistical sample is a sequence of m random variables or, equally, a random vector of m random examples.

$$(\boldsymbol{X}_1, Y_1), ..., (\boldsymbol{X}_m, Y_m)$$

we assume that random examples in a statistical sample are pairwise independent:

$$\forall i, j = 1...m \quad s.t. \quad i \neq j \quad (\boldsymbol{X}_i, Y_i) \perp (\boldsymbol{X}_j, Y_j)$$

Random example Following the same joint probability distribution that characterizes the original population (X, Y), a random example associates an event (i.e. the t-th occurrence of the phenomenon of interest) with a suitable tuple of values in $(\boldsymbol{x}_t, y_t) \in \mathcal{X} \times \mathcal{Y}$. Each random example is therefore a random variable that statistically models each of the available realizations of (X, Y), as it is assumed to be a random draw from its unknown joint probability distribution $\mathcal{D}$:

$$\forall t = 1, ..., m \;\; : \;\; (\boldsymbol{X}_t, Y_t) \sim \mathcal{D}$$

[1] By abuse of notation, we simplify the representation of the expected risk, and any other expectation taken w.r.t the draw of a random sample, by denoting a random sample $(\boldsymbol{X}_1, Y_1), ..., (\boldsymbol{X}_m, Y_m)$ as one of its possible concrete realizations S_m:

$$\mathbb{E}_{S_m\sim\mathcal{D}^m}\Big[\ell_{\mathcal{D}}\big(A(S_m)\big)\Big]$$

Example A example is a specific tuple of values which concretely realizes the t-th random example $(\boldsymbol{X}_t, Y_t)$.

$$(\boldsymbol{x}_t, y_t) \in \mathcal{X} \times \mathcal{Y}$$

Examples $(\boldsymbol{x}_t, y_t)$ are linked to (X, Y) by being modeled as random examples $(\boldsymbol{X}_t, Y_t)$. An example is composed by a **datapoint** $\boldsymbol{x}_t \in \mathcal{X}$ (vector of d values) and its corresponding **label** $y \in \mathcal{Y}$. The sets $\mathcal{X}$ and $\mathcal{Y}$ are respectively called **data domain** and **label space**, typically:

- $\mathcal{X} \equiv \mathbb{R}^d \ , \ \mathcal{Y} \equiv \mathbb{R}$ for **regression** problems.
- $\mathcal{X} \equiv \mathbb{R}^d \ , \ \mathcal{Y} \equiv \{-1, 1\}$ for **classification** problems.

Dataset A dataset is a sequence of m examples, i.e. it is a concrete realization of a statistical sample $(\boldsymbol{X}_1, Y_1), ..., (\boldsymbol{X}_m, Y_m)$.

$$S_m = \{(\boldsymbol{x}_t, y_t)\}_{t=1}^m = \{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_m, y_m)\}$$

Predictor A predictor $h \in \mathcal{H}$ is a function that maps datapoints to its corresponding prediction:

$$h : \mathcal{X} \to \mathcal{Y}$$

$$\boldsymbol{x} \mapsto \hat{y} = h(\boldsymbol{x})$$

the set of all possible predictors $\mathcal{H}$ is called **hypothesis class**. Building upon our previous discussion, the deterministic approach to learning rests on the assumption that the genuine relationship between features and target can be approximated by a predictor that maps data points to labels. A good predictor, when provided with a specific data point $\boldsymbol{x} \in \mathcal{X}$, generates a prediction $\hat{y} = h(\boldsymbol{x})$ that accurately approximates the true label y associated with $\boldsymbol{x}$. In this work we will focus on homogeneous linear predictors, a super-parametric hypothesis class described by a bounded number of parameters that equals the number d of features:

$$\mathcal{H} = \{h : h(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}, \boldsymbol{w} \in \mathbb{R}^d\}$$

Loss function A loss function is a mapping that quantifies, on a single example $(\boldsymbol{x}, y) \in \mathcal{X} \times \mathcal{Y}$, the capability of a predictor $h \in \mathcal{H}$ to approximate the actual features-target correspondence. This is done measuring the discrepancy between the predicted label and the correct label ($\ell(\hat{y}, y) = 0$ when $\hat{y} = y$, $\ell(\hat{y}, y) > 0$ otherwise)[2]:

$$\ell : (\mathcal{X} \times \mathcal{Y}) \times H \to [0, +\infty)$$

$$(\boldsymbol{x}, y), h(\boldsymbol{x}) \mapsto \ell(h(\boldsymbol{x}), y) = \ell(\hat{y}, y)$$

Regression problems are featured by the **square loss**:

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$

Learning algorithm A learning algorithm A is a function that goes from the set $\mathcal{S}$ of all possible realizations of a statistical sample $(\boldsymbol{X}_1, Y_1), ..., (\boldsymbol{X}_m, Y_m)$ to the hypothesis class $\mathcal{H}$, defining the image subspace $\mathcal{H}_m = \{h \in \mathcal{H} : \exists S_m \ , \ h = A(S_m)\}$:

$$A : \mathcal{S} \to \mathcal{H}$$

$$S_m \mapsto h_{S_m} = A(S_m) \in \mathcal{H}_m$$

it receives a training set S_m as input and produces in output a predictor $h_{S_m} = A(S_m)$ according to some prefixed choice criteria. In this work we will mainly focus on the **ERM** (empirical risk minimizer)

[2] Even if we wrote $[0, +\infty)$, bounded data typically implies bounded losses.

learning algorithm. ERM chooses $h_{S_m} = ERM(S_m)$ to be the predictor $h \in \mathcal{H}$ that minimizes the **training error** $\ell_{S_m}(h)$, i.e. the valued assumed by the **empirical risk**, namely the sample mean $\frac{1}{m}\sum_{t=1}^{m} \ell(h(\boldsymbol{X}_t), Y_t)$, for the specific realization S_m that is fed to $A = ERM$ as input[3]:

$$\ell_{S_m}(h) = \frac{1}{m}\sum_{t=1}^{m} \ell(h(\boldsymbol{x}_t), y_t) \qquad h_{S_m} = ERM(S_m) = \arg\min_{h\in\mathcal{H}} \ell_{S_m}(h)$$
$$\ell_{S_m}(h_{S_m}) \leq \min_{h\in\mathcal{H}} \ell_{S_m}(h)$$

ERM on the class of homogeneous linear predictors for a regression problem (namely, a linear regression problem) has a closed for solution:

$$\boldsymbol{w}_{S_m} = \arg\min_{\boldsymbol{w}\in\mathbb{R}^d} \sum_{t=1}^{m} (\boldsymbol{w}^T\boldsymbol{x} - y_t)^2 \tag{1}$$
$$= (S^T S)^{-1} S^T \boldsymbol{y} \tag{2}$$

where:

- $S^T = [\boldsymbol{x}_1, ..., \boldsymbol{x}_m]_{d\times m}$ is the transposed version of the **design matrix** S, composed by the m datapoints in the training set S_m.
- $S^T S = [\boldsymbol{x}_i \boldsymbol{x}_j]_{d\times d}$ is a $d \times d$ symmetric matrix.
- $y = [y_1, ..., y_m]^T$ is the vector composed by the m labels in S_m.

Some learning algorithms (e.g. Ridge regression, a more stable variant of linear regression: $\boldsymbol{w}_{S_m} = (\alpha\mathbf{I} + S^T S)^{-1} S^T \boldsymbol{y}$) are characterized by a certain number of **hyperparameters** $\theta \in \Theta$, i.e. non-trainable parameters whose values need to be chosen before the training phase begins ($\theta = (\alpha)$ for Ridge regression). In practice, in a preliminar phase called **hyperparameters tuning**, we will consider a family of learning algorithms parametrized by tuples of hyperparameters in a discrete and finite hyperparameter subspace Θ_0 of Θ:

$$A_{\Theta_0} = \{A_\theta : \theta \in \Theta_0\}$$

and we will choose the learning algorithm A_{θ^*} which produces a predictor $A_{\theta^*}(S_m)$ which has minimum statistical risk among all the avaliable alternatives $A_\theta(S_m)$, $\forall\theta \in \Theta_0$:

$$A_{\theta^*}(S_m) \;:\; \theta^* = \arg\min_{\theta\in\Theta_0} \ell_{\mathcal{D}}(A_\theta(S_m)) \tag{3}$$

2.3 Statistical risk and expected risk estimates

Test error Consider a statistical learning problem ($\mathcal{D}$, ℓ) represented by potentially different realizations $S_m = \{(\boldsymbol{x}_t, y_t)\}_{t=1}^m$ (training set) and $S'_n = \{(\boldsymbol{x}'_t, y'_t)\}_{t=1}^n$ (test set) of potentially different statistical samples $(\boldsymbol{X}_1, Y_1), ..., (\boldsymbol{X}_m, Y_m)$ and $(\boldsymbol{X}'_1, Y'_1), ..., (\boldsymbol{X}'_n, Y'_n)$, which are assumed to be both randomly drawn from the same fixed and unknown joint probability distribution $\mathcal{D}$. Let ℓ be a generic loss function used to evaluate the expected performance of a generic predictor $h \in \mathcal{H}$ w.r.t the draw of a random example $(\boldsymbol{X}, Y) \sim \mathcal{D}$, namely the statistical risk of h:

$$\ell_{\mathcal{D}}(h) = \mathbb{E}_{(\boldsymbol{X},Y)\sim\mathcal{D}}\big[\ell(Y, h(\boldsymbol{X}))\big]$$

Recalling the definition of empirical risk, consider the sample mean of the losses of h, when it is evaluated on the statistical sample $(\boldsymbol{X}'_1, Y'_1), ..., (\boldsymbol{X}'_n, Y'_n)$:

$$\frac{1}{n}\sum_{t=1}^{n} \ell\big(Y'_t, h(\boldsymbol{X}'_t)\big)$$

[3] In Appendix A we show the way ERM can be modelled as a decision-maker in a deterministic (single scenario), single decision-maker decision problem.

It can be shown that the sample mean of the losses of $h_{S_m} = A(S_m)$, evaluated on $(\boldsymbol{X}'_1, Y'_1), ..., (\boldsymbol{X}'_n, Y'_n) \sim \mathcal{D}$, is a good estimator for its statistical risk[4]. In other words, the expected sample mean of the losses of a predictor coincides with the expected loss of the same predictor evaluated on the random draw of $(\boldsymbol{X}, Y) \sim \mathcal{D}$:

$$\mathbb{E}_{S'_n \sim \mathcal{D}^n}\Big[\frac{1}{n}\sum_{t=1}^{n} \ell\big(Y'_t, h_{S_m}(\boldsymbol{X}'_t)\big)\Big] = \frac{1}{n}\sum_{t=1}^{n} \mathbb{E}_{S'_n \sim \mathcal{D}^n}\Big[\ell\big(Y'_t, h_{S_m}(\boldsymbol{X}'_t)\big)\Big] \tag{4}$$

$$= \frac{1}{n}\sum_{t=1}^{n} \mathbb{E}_{(Y'_t, \boldsymbol{X}'_t) \sim \mathcal{D}}\Big[\ell\big(Y'_t, h_{S_m}(\boldsymbol{X}'_t)\big)\Big] \tag{5}$$

$$= \frac{1}{n}\sum_{t=1}^{n} \mathbb{E}_{(\boldsymbol{X}, Y) \sim \mathcal{D}}\Big[\ell\big(Y, h_{S_m}(\boldsymbol{X})\big)\Big] \tag{6}$$

$$= \mathbb{E}_{(\boldsymbol{X}, Y) \sim \mathcal{D}}\Big[\ell\big(Y, h_{S_m}(\boldsymbol{X})\big)\Big] = \ell_{\mathcal{D}}(h_{S_m}) \tag{7}$$

As a consequence, the **test error** of the predictor produced by A on a fixed training set S_m, namely $\ell_{S'_n}(h_{S_m}) = \frac{1}{n}\sum_{t=1}^{n} \ell\big(y'_t, h_{S_m}(\boldsymbol{x}'_t)\big)$, becomes a good estimate for the statistical risk $\ell_{\mathcal{D}}(h_{S_m})$.

K-fold cross validation Following the same idea, the expected risk of $A\big((\boldsymbol{X}_1, Y_1), ..., (\boldsymbol{X}_m, Y_m)\big)$, taken with respect to the random draw of $(\boldsymbol{X}_1, Y_1), ..., (\boldsymbol{X}_m, Y_m)$, can be estimated using the so-called **K-fold cross-validation** estimate $\ell^{CV}_{S_m}(A(S_m))$.

1. Starting from a dataset S_r, K-fold CV partitions the data into K test sets (referred to as "folds"):

$$S_r = S_n^{(1)} \cup ... \cup S_n^{(K)} \qquad \text{each of size } n = \frac{r}{K}$$

2. K-fold CV then generates K training parts, each of size $m = \frac{r(K-1)}{K}$, by removing the i-th fold from the original dataset $S_m^{(-i)} = S_r \setminus S_n^{(i)}, \quad \forall i = 1, ..., K$.
3. K predictors $h_{S_m^{(-i)}}$ are trained on the corresponding training part $S_m^{(-i)}$ and then tested on the corresponding fold $S_n^{(i)}$.
4. The mean of the resulting K test errors is taken as the estimate $\ell^{CV}_{S_m}(A(S_m))$.

The pseudocode for K-fold CV is reported in ALG. 1.

Algorithm 1 K-fold CV

Require: $S_r,\ K,\ r$
Ensure: $\ell^{CV}_{S_m}(A(S_m)) \approxeq \mathbb{E}_{S_m \sim \mathcal{D}^m}\big[\ell_{\mathcal{D}}\big(A(S_m)\big)\big]$, $\ell^{CVT}_{S_m}(A(S_m)) \approxeq \mathbb{E}_{S_m \sim \mathcal{D}^m}\big[\ell_{S_m}\big(A(S_m)\big)\big]$

$S_r = S_n^{(1)} \cup ... \cup S_n^{(K)}$ {partition S_r into K folds ($n = \frac{r}{K}$)}
for $i = 1$ **to** K **do**
  $S_m^{(-i)} = S_r \setminus S_n^{(i)}$ {remove the i-th fold from S_r ($m = \frac{r(K-1)}{K}$)}
  $h_{S_m^{(-i)}} = A(S_m^{(-i)})$ {Train the i-th predictor on $S_m^{(-i)}$}
  $\ell_{S_m^{(-i)}}\big(h_{S_m^{(-i)}}\big) = \frac{1}{n}\sum_{(\boldsymbol{x},y) \in S_m^{(-i)}} \ell\big(y, h_{S_m^{(-i)}}(\boldsymbol{x})\big)$ {Compute the i-th training error}
  $\ell_{S_n^{(i)}}\big(h_{S_m^{(-i)}}\big) = \frac{1}{n}\sum_{(\boldsymbol{x},y) \in S_n^{(i)}} \ell\big(y, h_{S_m^{(-i)}}(\boldsymbol{x})\big)$ {Test the i-th predictor on $S_n^{(i)}$}
end for
$\ell^{CVT}_{S_m}(A(S_m)) = \frac{1}{K}\sum_{i=1}^{K} \ell_{S_m^{(-i)}}(h_{S_m^{(-i)}})$ {Compute the average training error}
$\ell^{CV}_{S_m}(A(S_m)) = \frac{1}{K}\sum_{i=1}^{K} \ell_{S_n^{(i)}}(h_{S_m^{(-i)}})$ {Compute the average test error}

[4] It's crucial to highlight that this results relies on the fact that the choice of the predictor h_{S_m} depends only of the information contained in the training set S_m and is not influenced in any way by the examples in the test set (otherwise, the steps starting from (4) would not be valid).

K-fold nested cross validation As previously mentioned, a learning algorithm may be characterized by a set of hyperparameters. In this case, the maximum we can achieve with K-fold CV is to estimate the expected risk of a predictor $A_\theta\big((\boldsymbol{X}_1, Y_1), ..., (\boldsymbol{X}_m, Y_m)\big)$ produced by an algorithm $A_\theta \in A_{\Theta_0}$, which is parameterized by a fixed tuple of hyperparameters $\theta \in \Theta_0$. K-fold CV does not implicitly allow us to tune the best hyperparameters θ^* and determine which algorithm A_{θ^*}, from the family A_{Θ_0}, will be used to select the predictor $A_{\theta^*}\big((\boldsymbol{X}_1, Y_1), ..., (\boldsymbol{X}_m, Y_m)\big)$ to be generate. The **K-fold nested cross-validation** technique incorporates hyperparameters tuning into the process of estimating the expected risk of a predictor. This allows us to estimate the expected risk, with respect to the random draw of $(\boldsymbol{X}_1, Y_1), ..., (\boldsymbol{X}_m, Y_m)$, of the optimally-tuned predictor produced by A_{θ^*}:

$$\ell^{NCV}_{S_m}(A_{\theta^*}(S_m)) \approxeq \mathbb{E}_{S_m \sim \mathcal{D}^m}\big[\ell_{\mathcal{D}}\big(A_{\theta^*}(S_m)\big)\big]$$

In practice, the **K-fold nested cross-validation** method closely mirrors the K-fold CV procedure, with a key distinction: before using each training portion $S_m^{(-i)}$ to train the K predictors (which are later evaluated on their corresponding $S_n^{(i)}$ folds), $S_m^{(-i)}$ is further divided into a smaller training subset, $S^{(-i)}_{\text{train}}$, and a **validation** or **development** set, $S^{(-i)}_{\text{dev}}$[5]:

$$\forall i = 1, ..., K, \quad S_m^{(-i)} = S^{(-i)}_{\text{train}} \cup S^{(-i)}_{\text{dev}} \qquad \text{with, } |S^{(-i)}_{\text{dev}}| = \frac{m}{5} \quad \text{and} \quad m = \frac{r(K-1)}{K}$$

The training portion $S^{(-i)}_{\text{train}}$ is provided as input to each algorithm in the family of learning algorithms A_{Θ_0}, while $S^{(-i)}_{\text{dev}}$ is used to evaluate each of the resulting predictors $h_\theta^{(-i)}$, as θ varies within Θ_0. From all the potential algorithms A_θ, the one that delivers the predictor with the lowest test error, as gauged on the validation set, is selected:

$$A_{\theta^*}(S^{(-i)}_{\text{train}}) \;\; : \;\; \theta^* = \arg\min_{\theta \in \Theta_0} \ell_{S^{(-i)}_{\text{dev}}}(h_\theta^{(-i)})$$

Algorithm 2 K-fold nested CV

Require: S_r, Θ_0, K, r
Ensure: $\ell^{NCV}_{S_m}(A(S_m)) \approxeq \mathbb{E}_{S_m \sim \mathcal{D}^m}\big[\min_{\theta\in\Theta_0} \ell_{\mathcal{D}}\big(A_\theta(S_m)\big)\big]$, $\ell^{NCVT}_{S_m}(A(S_m)) \approxeq \mathbb{E}_{S_m \sim \mathcal{D}^m}\big[\min_{\theta\in\Theta_0} \ell_{S_m}\big(A_\theta(S_m)\big)\big]$

$S_r = S_n^{(1)} \cup ... \cup S_n^{(K)}$ {partition S_r into K folds ($n = \frac{r}{K}$)}
for $i = 1$ **to** K **do**
  $S_m^{(-i)} = S_r \setminus S_n^{(i)}$ {Remove the i-th fold from S_r ($m = \frac{r(K-1)}{K}$)}
  $S_m^{(-i)} = S^{(-i)}_{\text{train}} \cup S^{(-i)}_{\text{dev}}$ {Split the i-th part into sub-trainig and validation sets ($|S^{(-i)}_{\text{dev}}| = \frac{m}{5}$)}
  for each $\theta \in \Theta_0$ **do**
    $h_\theta^{(-i)} = A_\theta(S^{(-i)}_{\text{train}})$ {Train $h_\theta^{(-i)}$ (run A_θ on $S^{(-i)}_{\text{train}}$)}
    $\ell_{S^{(-i)}_{\text{dev}}}\big(h_\theta^{(-i)}\big) = \frac{1}{|S^{(-i)}_{\text{dev}}|} \sum_{(\boldsymbol{x},y)\in S^{(-i)}_{\text{dev}}} \ell\big(y, h_\theta^{(-i)}(\boldsymbol{x})\big)$ {Test $h_\theta^{(-i)}$ on the validation set $S^{(-i)}_{\text{dev}}$}
  end for
  $\theta^*_{(-i)} = \arg\min_{\theta\in\Theta_0} \ell_{S^{(-i)}_{\text{dev}}}\big(h_\theta^{(-i)}\big)$ {Choose the best tuple of hyperparameter θ^*}
  $h_{\theta^*}^{(-i)} = A_{\theta^*}(S_m^{(-i)})$ {Train the i-th predictor on $S_m^{(-i)}$}
  $\ell_{S_m^{(-i)}}\big(h_{\theta^*}^{(-i)}\big) = \frac{1}{n} \sum_{(\boldsymbol{x},y)\in S_m^{(-i)}} \ell\big(y, h_{\theta^*}^{(-i)}(\boldsymbol{x})\big)$ {Compute the i-th training error}
  $\ell_{S_n^{(i)}}\big(h_{\theta^*}^{(-i)}\big) = \frac{1}{n} \sum_{(\boldsymbol{x},y)\in S_n^{(i)}} \ell\big(y, h_{\theta^*}^{(-i)}(\boldsymbol{x})\big)$ {Test the i-th predictor on $S_n^{(i)}$}
end for
$\ell^{NCVT}_{S_m}(A_{\theta^*}(S_m)) = \frac{1}{K} \sum_{i=1}^K \ell_{S_m^{(-i)}}(h_{\theta^*}^{(-i)})$ {Compute the average training error}
$\ell^{NCV}_{S_m}(A_{\theta^*}(S_m)) = \frac{1}{K} \sum_{i=1}^K \ell_{S_n^{(i)}}(h_{\theta^*}^{(-i)})$ {Compute the average test error}

Afterwards, the predictor $h_{\theta^*}^{(-i)}$, chosen for the i-th iteration, is retrained on the entire training subset $S_m^{(-i)}$ and tested on the corresponding fold. The average of the K test errors computed on these folds is taken as an estimation of the expected risk. The pseudocode for the K-fold nested CV is reported in ALG. 2.

[5] The size of the validation set is empirically set to $\frac{m}{5}$ (namely, 80% training subpart and 20% validation).

Best CV estimate There also exists a less computationally demanding but statistically less rigorous variant, which can be defined as the **best CV estimate**, and which proceeds in the following steps:

1. for each possible tuple $\theta \in \Theta_0$:
   (a) train K predictors $h_\theta^{(-i)} = A_\theta(S_m^{(-i)})$ and subsequently test them on the respective fold.
   (b) compute the average $\ell_{S_m}(A_\theta(S_m))$ of the K test errors calculated for the fixed tuple θ.
2. Choose, from the $|\Theta_0|$ possible alternatives, the tuple θ^* of hyperparameters that yields the smallest average test error. This leads to the predictor h_{θ^*}: $\ell_{S_m}(h_{\theta^*}) \leq \min_{\theta \in \Theta_0} \ell_{S_m}(A_\theta(S_m))$.

The pseudocode for the best CV estimate is shown in ALG. 3

Algorithm 3 Best CV estimate

Require: $S_r,\ \Theta_0,\ K,\ r$
Ensure: $\ell_{S_m}^{BCV}(A_{\theta^*}(S_m)) \approxeq \mathbb{E}_{S_m \sim \mathcal{D}^m}\big[\min_{\theta \in \Theta_0} \ell_{\mathcal{D}}\big(A_\theta(S_m)\big)\big]$, $\ell_{S_m}^{BCVT}(A_{\theta^*}(S_m)) \approxeq \mathbb{E}_{S_m \sim \mathcal{D}^m}\big[\min_{\theta \in \Theta_0} \ell_{S_m}\big(A_\theta(S_m)\big)\big]$

$S_r = S_n^{(1)} \cup ... \cup S_n^{(K)}$ {partition S_r into K folds ($n = \frac{r}{K}$)}
for $i = 1$ **to** K **do**
  $S_m^{(-i)} = S_r \setminus S_n^{(i)}$ {Remove the i-th fold from S_r ($m = \frac{r(K-1)}{K}$)}
end for
for each $\theta \in \Theta_0$ **do**
  for $i = 1$ **to** K **do**
    $h_\theta^{(-i)} = A_\theta(S_m^{(-i)})$ {Train $h_\theta^{(-i)}$ (run A_θ on $S_m^{(-i)}$)}
    $\ell_{S_m^{(-i)}}\big(h_\theta^{(-i)}\big) = \frac{1}{n} \sum_{(\boldsymbol{x},y) \in S_m^{(-i)}} \ell\big(y, h_\theta^{(-i)}(\boldsymbol{x})\big)$ {Compute the i-th training error}
    $\ell_{S_n^{(i)}}\big(h_\theta^{(-i)}\big) = \frac{1}{n} \sum_{(\boldsymbol{x},y) \in S_n^{(i)}} \ell\big(y, h_\theta^{(-i)}(\boldsymbol{x})\big)$ {Test $h_\theta^{(-i)}$ on the i-th fold set $S_n^{(i)}$}
  end for
  $\ell_{S_m}^{CVT}(A_\theta(S_m)) = \frac{1}{K} \sum_{i=1}^{K} \ell_{S_m^{(-i)}}(h_\theta^{(-i)})$ {Compute the average training error for fixed θ}
  $\ell_{S_m}^{CV}(A_\theta(S_m)) = \frac{1}{K} \sum_{i=1}^{K} \ell_{S_n^{(i)}}(h_\theta^{(-i)})$ {Compute the average test error for fixed θ}
end for
$\ell_{S_m}^{BCV}(A_{\theta^*}(S_m)) = \min_{\theta \in \Theta_0} \ell_{S_m}^{CV}(A_\theta(S_m))$ {Choose the best tuple θ^* and estimate the expected risk}
$\ell_{S_m}^{BCVT}(A_{\theta^*}(S_m)) = \ell_{S_m}^{CVT}(A_{\theta^*}(S_m))$ {Find the training error of $A_{\theta^*}(S_m)$}

2.4 Bias-variance and fitting-stability trade-offs

So far we have defined two ways to respectivelly estimate the statistical risk $\ell_{\mathcal{D}}(A(S_m))$ and the expected risk $\mathbb{E}_{S_m \sim \mathcal{D}^m}\big[\ell_{\mathcal{D}}\big(A(S_m)\big)\big]$ of the predictor $h_{S_m} = A(S_m)$ produced by the learning algorithm A on the training set S_m. These are the test error $\ell_{S'_n}(A(S_m))$ and the K-fold CV estimate $\ell_{S_m}^{CV}(A(S_m))$ (or K-fold nested CV estimate $\ell_{S_m}^{NCV}(A(S_m))$, to properly handle hyperparameters tuning).

To understand for which reasons, $A(S_m)$, for a concrete choice of m and A (which runs on $\mathcal{H}$, inducing $\mathcal{H}_m \subseteq \mathcal{H}$) can fail to learn, i.e. approximate the underlying relationship between the features characterization X and the target description Y of the phenomenon of interest (X, Y). This happens obviously when $\ell_{\mathcal{D}}(A(S_m))$ and $\mathbb{E}_{S_m \sim \mathcal{D}^m}\big[\ell_{\mathcal{D}}\big(A(S_m)\big)\big]$ are high, but why this happens?

We proceed by decomposing the statistical risk and the expected risk into the bias-variance decomposition and the fitting-stability trade-off:

Bias-variance trade-off:

$$\ell_{\mathcal{D}}(A(S_m)) = \ell_{\mathcal{D}}(A(S_m)) - \ell_{\mathcal{D}}(h^*) + \quad (8)$$
$$+\ \ell_{\mathcal{D}}(h^*) - \ell_{\mathcal{D}}(f^*) + \quad (9)$$
$$+\ \ell_{\mathcal{D}}(f^*) \quad (10)$$

Fitting-stability trade-off:

$$\mathbb{E}_{S_m \sim \mathcal{D}^m}\big[\ell_{\mathcal{D}}\big(A(S_m)\big)\big] = \quad (11)$$
$$= \mathbb{E}_{S_m \sim \mathcal{D}^m}\big[\ell_{S_m}\big(A(S_m)\big)\big] + \quad (12)$$
$$+\ \mathbb{E}_{S_m \sim \mathcal{D}^m}\big[\ell_{\mathcal{D}}\big(A(S_m)\big) - \ell_{S_m}\big(A(S_m)\big)\big] \quad (13)$$

where: $\ell_{\mathcal{D}}(h^*) \leq \min_{h \in \mathcal{H}_m} \ell_{\mathcal{D}}(h)$ is the statistical risk best predictor $h^* \in \mathcal{H}_m$ that A can compute in $\mathcal{H}$.

Considering the bias-variance decomposition, (8) is called **estimation error**, while (9) is the **approximation error** and (10) is the unavoidable Bayes risk. For what concerns the fitting-stability trade-off, we refer to (12) as the **fitting** term, which is strongly related to the approximation error (9), while (13) is called **stability** term and it is connected to estimation error (8).

We now express the estimation error in terms of the training error $\ell_{S_m}(A(S_m))$:

$$\ell_{\mathcal{D}}(A(S_m)) - \ell_{\mathcal{D}}(h^*) = \ell_{\mathcal{D}}(A(S_m)) - \ell_{S_m}(A(S_m)) + \ell_{S_m}(h^*) - \ell_{\mathcal{D}}(h^*)$$

We now define:

- **Overfitting** or **high variance - high estimation error**: given a learning problem specified by $(\mathcal{D}, \ell)$, along with concrete choices for m, S_m, $\mathcal{H}$, and A, the learning algorithm A is said to overfit if the training error $\ell_{S_m}(A(S_m))$ of $h_{S_m} = A(S_m)$ is small but fails to provide a reliable estimate for the statistical risk $\ell_{\mathcal{D}}(A(S_m))$, which is high and significantly different from the statistical risk $\ell_D(h^*)$ of the best learnable predictor h^* in $\mathcal{H}$. Since the statistical risk is appropriately estimated by the test error $\ell_{S'_n}(A(S_m))$, A overfits when $A(S_m)$ has small training error but high test error. Overfitting can be thought also in terms of stability. Practically, the learning algorithm A is said to be unstable when, even if it fits well all the training parts $S_m^{(-1)}, ..., S_m^{(-K)}$, the (high) K-fold CV estimate (or the K-fold nested CV estimate) is significantly different from the (small) average training error $\frac{1}{K}\sum_{i=1}^{K} \ell_{S_m^{(-i)}}(h_{S_m^{(-i)}})$ (which estimates the fitting term).
- **Underfitting** or **high bias - high approximation error**: given a learning problem specified by $(\mathcal{D}, \ell)$, along with concrete choices for m, S_m, $\mathcal{H}$, and A, the learning algorithm A is said to underfit if the training error $\ell_{S_m}(A(S_m))$, even though it is a reasonable estimate for the statistical risk $\ell_{\mathcal{D}}(A(S_m))$, which is also close to $\ell_D(h^*)$, is really far from being small (indicating that both $\ell_{\mathcal{D}}(A(S_m))$ and $\ell_D(h^*)$ are also high). In terms of stability, A underfits the K-fold CV estimate (or the K-fold nested CV estimate) is close to the average training error, but the latter is high.

2.5 Consistency

Regarding their capability to minimize the estimation error and the approximation error, learning algorithms are typically classified as:

- **Non-parametric learning algorithm** a learning algorithm A is said to be *non-parametric* if the best predictor that it can produce h^* is a good proxy of the Bayes optimal predictor f^*, as the sample size m grows significantly, i.e. if $\mathcal{H}_m$ is rich enough, in terms of space complexity[6] to grant $f^* \in \mathcal{H}_m$. A non-parametric algorithm tends to kill the approximation error as the sample size increases:

$$\lim_{m\to\infty} \min_{h\in\mathcal{H}_m} \ell_{\mathcal{D}}(h) = \ell_{\mathcal{D}}(f^*)$$

$$\text{approximation error} \to 0 \text{ as } m \to \infty: \qquad \lim_{m\to\infty} \ell_{\mathcal{D}}(h^*) - \ell_{\mathcal{D}}(f^*) = 0$$

- **Parametric learning algorithm** A learning algorithm A is referred to as *parametric* if it is expected to reduce the estimation error to zero as the sample size increases:

$$\lim_{m\to\infty} \mathbb{E}_{S_m\sim\mathcal{D}^m}\Big[\ell_{\mathcal{D}}(A(S_m))\Big] = \ell_{\mathcal{D}}(h^*)$$

$$\text{estimation error} \to 0 \text{ as } m \to \infty: \qquad \lim_{m\to\infty} \mathbb{E}_{S_m\sim\mathcal{D}^m}\Big[\ell_{\mathcal{D}}(A(S_m)) - \ell_{\mathcal{D}}(f^*)\Big] = 0$$

- **Consistent learinig algorithm**: A learning algorithm A is consistent w.r.t a loss function ℓ if, for any distribution $\mathcal{D}$, the statistical risk of the predictor generated by A on a random sample

[6] In loose terms and without delving into specifics, a measure that relies on the number of parameters defining a generic $h \in \mathcal{H}$

$(\boldsymbol{X}_1, Y_1), ..., (\boldsymbol{X}_m, Y_m) \sim \mathcal{D}$ converges, in expectation w.r.t the random draw of the statistical sample, to the Bayes risk $\ell_{\mathcal{D}}(f^*)$ for the learning problem $(\mathcal{D}, \ell)$, as the sample size increases:

$$\lim_{m\to\infty} \mathbb{E}_{S_m\sim\mathcal{D}^m}\Big[\ell_{\mathcal{D}}(A(S_m))\Big] = \ell_{\mathcal{D}}(f^*)$$

A non-parametric learning algorithm A that also kills the estimation error as the sample size goes to infinity is consistent. A learning algorithm is therefore consistent if its complexity (its size, in terms of its number of parameters) can grow unbounded as the sample size of S_m increases and, at the same time, ensuring that $\ell_{S_m}(A(S_m)) \approx \ell_{\mathcal{D}}(h^*)$ and $\ell_{\mathcal{D}}(h^*) \approx \ell_{\mathcal{D}}(f^*)$.

Linear regression, i.e the learning algorithm which selects $\boldsymbol{w}_{S_m} = (S^TS)^{-1}S^T\boldsymbol{y}$ as the produced predictor of the training set S_m, is a parametric learning algorithm. The $d \times d$ matrix S^TS is invertible if and only if $\det(S^TS) \neq 0$ which implies that all the rows or columns of S^TS must be linearly indipendent (none can be written as a linear combination of the others). Equally, recalling the spectral theorem, all the eigenvalues of S^TS must be strictly positive:

$$\det(S^TS) \neq 0 \Leftrightarrow \det(V^T\Sigma V) \neq 0 \Leftrightarrow \det(\Sigma) \neq 0 \Leftrightarrow \lambda_i \neq 0 \ \ \forall i = 1, ..., d$$

where we used the spectral theorem followed by the fact that V is orthonormal ($\det(V) = \det(V^T) = 1$) and by the fact that Σ is diagonal with $[\lambda_1, ..., \lambda_d]$ on the main diagonal.

However, if $\lambda_i \approx 0$, the ERM solution becomes unstable. To overcome this issue, we introduce the regualarization factor $\alpha > 0$, i.e. the hyperparameter ensuring that, even if the eigenvalues λ_i of S^TS are almost zero, the eigenvalues of $(\alpha\boldsymbol{I} + S^TS)$ are always strictly positive. The ridge regression can be expressed as:

$$\boldsymbol{w}_{S_m} = (\alpha\mathbf{I} + S^TS)^{-1}S^T\boldsymbol{y} \tag{14}$$

where $\boldsymbol{I}$ is the identity matrix. If $\alpha \to 0$ the Ridge solution reverts back to $\boldsymbol{w}_{S_m} = (S^TS)^{-1}S^T\boldsymbol{y}$ (possibly overfitting), if $\alpha \to \infty$ the solution becomes the vector with minimum norm in $\mathbb{R}^d$, i.e. the zero vector $\boldsymbol{w}_{S_m} = [0, ..., 0]_d^T$, leading to underfitting.

Ridge regression, by definition of parametric learning algorithm, can't be a consistent learning algorithm. However, learning a more complex non-linear predictor in the original space (which is linear in a expanded space) can be achieved through feature expansion.

In the following chapter we apply all concept exposed so far to a specific and concrete regression problem.

3 Methods

The objective of this project is to approximate the relationship between the popularity Y of a spotify audio track and its description provided by a random vector X of d features (the name of the artist, the genre, the loudness level ecc...). As the target variable is a continuous random variable, the learning problem under investigation can be framed as a linear regression problem:

$$\ell(\hat{y}, y) = (\hat{y} - y)^2, \quad \mathcal{X} = \mathbb{R}^d, \ \ \mathcal{Y} = \mathbb{R}$$

We consider the hypothesis class $\mathcal{H}$ of the homogeneous linear predictors.

At the outset, the chosen learning algorithm is the regularized empirical risk minimization (ERM) approach for linear regression problems. Specifically, the selected predictor is the one obtained through the closed-form solution of Ridge regression:

$$\boldsymbol{w}_{S_m} = (\alpha\mathbf{I} + S^TS)^{-1}S^T\boldsymbol{y}$$

The project is divided into two parts $c = 1, 2$, each considering two different ways to model the population (X, Y):

- $c = 1$ X is a random vector only composed by continuous features.

– $c = 2$ X is a sequence of continuous, binary, ordinal (qualitative) and categorical features.

Each c-th problem is represented by a dataset $S_r^{(c)}$, i.e. a preprocessed version of the "Spotify Tracks Dataset"[7], which originally comprises 114000 examples. The resulting pre-processed dataset $S_r^{(c)}$ is assumed to be a realization of a statistical sample $(\boldsymbol{X}_1, Y_1)^{(c)}, ..., (\boldsymbol{X}_r, Y_r)^{(c)}$ consisting of r independent and identically distributed random examples.

The ultimate goal is to estimate, running 5-fold nested cross-validation on $S_r^{(c)}$, the expected risk of the c-th predictor $w_{S_m}^*$ produced by the ridge regression learning algorithm.

3.1 Dataset and pre-processing

3.1.1 Continuous features ($c = 1$)

In the first part, each music track is represented by a random vector X consisting of $d^{(c=1)} = 10$ continuous features $\mathrm{X} = [\mathrm{X}_1, ..., \mathrm{X}_{10}]$. These features, along with the target Y to be predicted, are listed and described in TABLE 1. The table also includes information about the data domain $\mathcal{X}$ (broken down into its 10 components), the target space $\mathcal{Y}$ and the unit of measure.

(X, Y)	Continuous Features										
	$\mathrm{X}_1, \ldots, \mathrm{X}_{10}$, Y										
	$\mathbf{X}_1$	$\mathbf{X}_2$	$\mathbf{X}_3$	$\mathbf{X}_4$	$\mathbf{X}_5$	$\mathbf{X}_6$	$\mathbf{X}_7$	$\mathbf{X}_8$	$\mathbf{X}_9$	$\mathbf{X}_{10}$	**Y**
Name	duration_ms	danceability	energy	loudness	speechiness	acousticness	instrumentalness	liveness	valence	tempo	Popularity
$\mathcal{X}, \mathcal{Y}$	$\mathcal{X}_1$ =[0, 10^7]	[0, 1]	[0, 1]	[-50, 5]	[0, 1]	[0, 1]	[0, 1]	[0, 1]	[0, 1]	[0, 250]	[0, 100]
Units of measure	ms	/	/	dB	/	/	/	/	/	BPM	/

Table 1: Spotify Tracks Dataset with Continuous Features. **duration_ms**: track length in milliseconds. **danceability**: how suitable a track is for dancing (based on tempo, rhythm stability, beat strength, and overall regularity). **energy**: perceptual measure of intensity and activity. **loudness**: overall loudness (in decibels). **speechiness**: presence of spoken words. **acousticness**: whether the track is acoustic. **instrumentalness**: whether a track contains no vocals. **liveness**: presence of an audience in the recording. **valence**: musical positiveness conveyed by a track. **tempo**: overall estimated tempo of a track. **popularity**: popularity measure based on the total number of plays how recent those plays are.

Data Cleaning Operationally, the initial pre-processing phase involves:

- Loading the original dataset, identified in the local file system by the path './data/dataset.csv'.
- Retaining only the columns corresponding to the continuous features and the target (specifically, the less relevant columns 'Unnamed: 0' and 'track_id' are removed).
- Creating a random permutation, df_encoded, of the original dataset (ensuring the reproducibility of the shuffling by setting a specific random_state value).
- Removing redundant examples (different random examples realized by the same tuple of values) and examples that are equal in way they describe X but that are labelled differently. In total, 83479 examples are preserved.

The dataset $S_r^{(c=1)}$, resulting from this pre-processing, thus has a size of $r = 83479$. Hence, each training part $S_m^{(-i)}$ and each fold $S_n^{(i)}$ will have respective sizes of $m = 66783$ and $n = 16695$. In this section, we will refer to $S_r^{(c=1)}$ simply as S_r.

Pre-processing In Figure Fig.(1), histograms are provided for each continuous feature and the target, showing the absolute frequencies of their respective value ranges. In that figure, but also in the data domain specifications described in TABLE 1, it can be observed that the features 'duration_ms', 'loudness', and 'tempo', as well as the target 'popularity', have significantly different scales compared to the other features, which all assume values between 0 and 1. Such disparity in scales may adversely impact the performance of learning algorithms. To address this issue and ensure a fair comparison between features, it is generally essential to apply feature scaling techniques, such as **standardization**.

[7] Dataset available here for download.

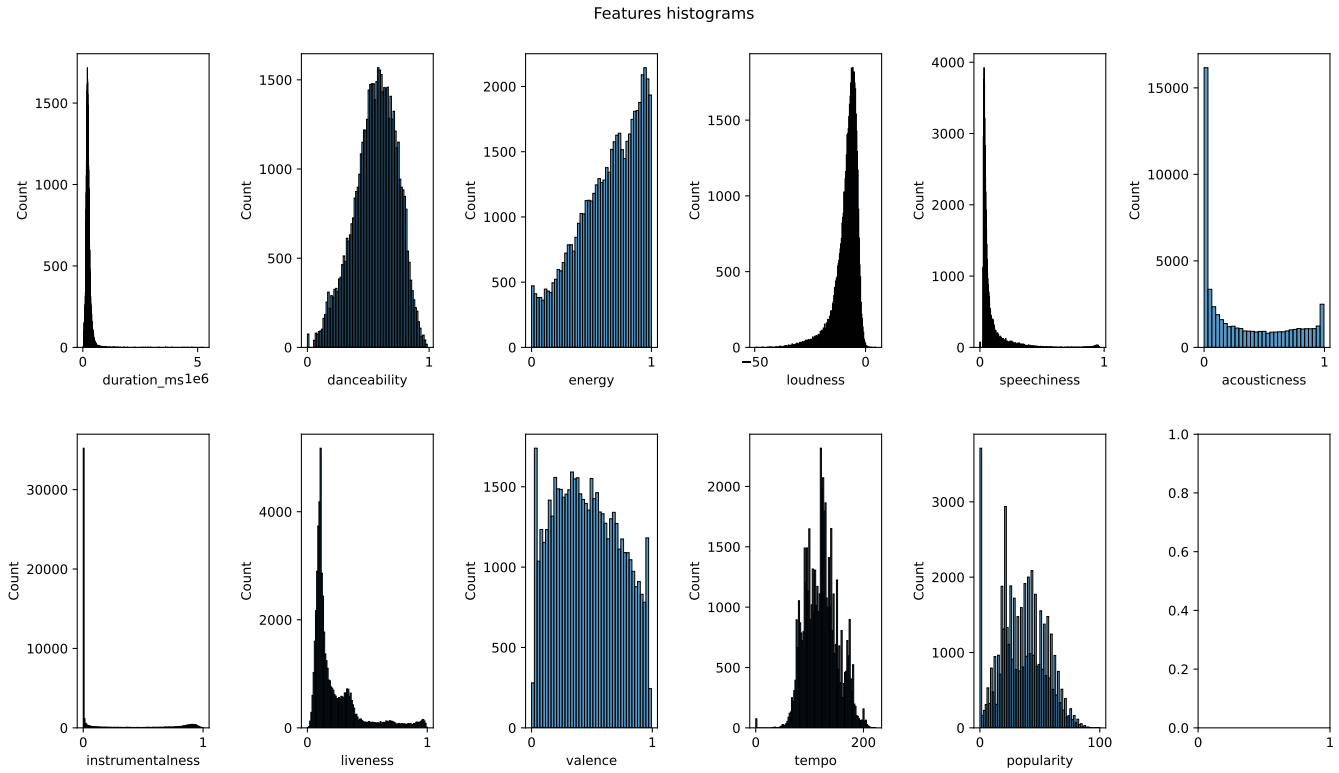


Figure 1: Continuous features histograms

Specifically, the predictors that ridge regression produces are not scale-invariant, unlike the closed-form solution of ERM for linear regression (Taboga 2021). Scale-invariancy refers to the following property: if we multiply the design matrix S by an invertible matrix D, the resulting predictor changes proportionally to D. Suppose we rescale, by D, the design matrix S, namely $Z = DS$:

$$\begin{aligned}
\hat{\boldsymbol{w}_{S_m}} &= \left(Z^T Z\right)^{-1} Z^T \boldsymbol{y} & \hat{\boldsymbol{w}_{S_m}} &= \left(Z^T Z + \alpha \mathbf{I}\right)^{-1} Z^T \boldsymbol{y} && (15)\\
&= \left(D^T S^T S D\right)^{-1} D^T S^T \boldsymbol{y} & &= \left(D^T S^T S D + \alpha \mathbf{I}\right)^{-1} D^T S^T \boldsymbol{y} && (16)\\
&= D^{-1}\left(S^T S\right)^{-1} \left(D^T\right)^{-1} D^T S^T \boldsymbol{y} & &= \left(D^T S^T S D + \alpha D^T (D^T)^{-1} D^{-1} D\right)^{-1} D^T S^T \boldsymbol{y} && (17)\\
&= D^{-1}\left(S^T S\right)^{-1} S^T \boldsymbol{y} & &= \left(D^T \left(S^T S + \alpha (D^T)^{-1} D^{-1}\right) D\right)^{-1} D^T S^T \boldsymbol{y} && (18)\\
& & &= D^{-1}\left(S^T S + \alpha (D^T)^{-1} D^{-1}\right)^{-1} \left(D^T\right)^{-1} D^T S^T \boldsymbol{y} && (19)\\
& & &= D^{-1}\left(S^T S + \alpha (D^T)^{-1} D^{-1}\right)^{-1} S^T \boldsymbol{y} && (20)
\end{aligned}$$

in the case of linear regresion (OLS) this proves that $\hat{\boldsymbol{w}_{S_m}} = D^{-1}\boldsymbol{w}_{S_m}$, i.e rescaling the design matrix by D produces a rescaled version of the solution, by D^{-1}. This is not true for the ridge regression solution until $D^T D = \mathbf{I}$, i.e. ridge estimator is scale-invariant if and only if $\alpha = 0$ or the scale matrix D is orthonormal. It is also important to note that the final line (12) in the proof suggests that the risk associated with $\hat{\boldsymbol{w}_{S_m}}$ (predictor computed with modified feature scales) may still be similar to the risk of $\boldsymbol{w}_{S_m}$ (predictor computed with original feature scales). This is due to the fact that the hyperparameter α, by absorbing the rescaling factor $(D^T)^{-1}D^{-1}$, allows the model to compensate for any changes in the feature scales introduced by D.

However, while scale invariance is not always satisfied for ridge regression, standardizing all the features and the target description of (X, Y) removes the influence of the original magnitudes on the coefficients of the ridge solution. By doing so, we ensure that the ridge estimates are not sensitive to

the initial scaling choices of the variables.

Standardization (also called **z-scores** data transformation) ensures that the each feature, represented by the corresponding column $\boldsymbol{x}_1^{(-i)}, \ldots, \boldsymbol{x}_d^{(-i)}$ in the design matrix $S^{(-i)}$ of the training part $S_m^{(-i)}$ (obtained removing the i-th fold from S_r), have zero mean and unit standard deviation:

$$\text{Given} \quad S_m^{(-i)} = [\boldsymbol{x}_1^{(-i)}, \ldots, \boldsymbol{x}_d^{(-i)}, \boldsymbol{y}^{(-i)}] = [S^{(-i)} | \boldsymbol{y}^{(-i)}]_{m \times d+1}, \quad \forall i = 1, ..., K, \quad :$$

$$\overline{\boldsymbol{x}}_j^{(-i)} = \frac{\boldsymbol{x}_j^{(-i)} - \mu_j^{(-i)}}{\sigma_j^{(-i)}} \quad \forall j = 1, ..., d \tag{21}$$

where:

- $\mu_j^{(-i)} = \frac{1}{m} \sum_{t=1}^{m} \boldsymbol{x}_{jt}^{(-i)}$, i.e. the average value for the j-th feature in the training set.
-

Also the target, namely the $m \times 1$ labels vector $\boldsymbol{y}^{(-i)}$ in $S_m^{(-i)}$, can be standardized by computing:

$$\overline{\boldsymbol{y}}^{(-i)} = \frac{\boldsymbol{y}^{(-i)} - \mu_{\boldsymbol{y}}^{(-i)}}{\sigma_{\boldsymbol{y}}^{(-i)}} \qquad \text{where: } \mu_{\boldsymbol{y}}^{(-i)} = \frac{1}{m} \sum_{t=1}^{m} y_t^{(-i)} \quad , \quad \sigma_{\boldsymbol{y}}^{(-i)} = \frac{1}{m} \sum_{t=1}^{m} (y_t^{(-i)} - \mu_{\boldsymbol{y}}^{(-i)}) \tag{22}$$

To standardize each of the i-th folds (i.e the testing part) we apply again (21) and (22), respectively on the columns of the design matrix $S^{(i)}$ and on the labels vector of $S_n^{(i)}$, but this time we use the means $\mu^{(-i)}$ and the standard deviations $\sigma^{(-i)}$ computed on the training data, namely:

$$\text{Given} \quad S_n^{(i)} = [\boldsymbol{x}_1^{(i)}, \ldots, \boldsymbol{x}_d^{(i)}, \boldsymbol{y}^{(i)}] = [S^{(i)} | \boldsymbol{y}^{(i)}]_{n \times d+1}, \quad \forall i = 1, ..., K, \quad :$$

$$\overline{\boldsymbol{x}}_j^{(i)} = \frac{\boldsymbol{x}_j^{(i)} - \mu_j^{(-i)}}{\sigma_j^{(-i)}} \quad , \quad \text{and} \quad \overline{\boldsymbol{y}}^{(i)} = \frac{\boldsymbol{y}^{(i)} - \mu_{\boldsymbol{y}}^{(-i)}}{\sigma_{\boldsymbol{y}}^{(-i)}} \forall j = 1, ..., d \tag{23}$$

The testing data are rescaled using the parameters computed on the training data. Allowing the learner to compute the rescaling parameters using the testing data would imply that the test set is made available to the learining algorithm, and this is statistically forbidden. In the nested 5-fold CV context we standardize $S_{\text{train}}^{(-i)}$ (using its relative mean and standard deviation to process the corresponding validation set $S_{\text{dev}}^{(-i)}$) and, separately, the full training part $S_m^{(-i)}$ and its fold $S_n^{(i)}$ in the same way we have previously described.

3.1.2 Binary, ordinal (qualitative) and categorical features ($c = 2$)

In the project's second phase, we describe each music track by an 18-feature random vector, $\mathrm{X} = [\mathrm{X}_1, ..., \mathrm{X}_{18}]$. It includes the original 10 continuous features plus eight new attributes: two binary, two ordinal, and four categorical. The 'popularity' target remains the same. Details for all attributes are given in TABLE 2.

X	Binary, ordinal (qualitative) and categorical features							
	$\mathrm{X}_1, \ldots, \mathrm{X}_5$							
	$\mathbf{X}_1$	$\mathbf{X}_2$	$\mathbf{X}_3$	$\mathbf{X}_4$	$\mathbf{X}_5$	$\mathbf{X}_6$	$\mathbf{X}_7$	$\mathbf{X}_8$
Name	key	time_signature	track_genre	explicit	mode	track_name	album_name	artists
$\mathcal{X}$	$\{-1, 0, 1, 2, ..., 11\}$	$\{3, 4, 5, 6, 7\}$	String	$\{T, F\}$	$\{0, 1\}$	String	String	String
# unique values	13	5	114	2	2	73609	46590	31438
Type	ordinal (qualitative)	ordinal (qualitative)	categorical	binary	binary	categorical	categorical	categorical

Table 2: Spotify Track Dataset (binary and categorical features). **key**: standard Pitch Class notation of the key (tonality) of the track (-1 if no key was detected), **time_signature**: beats per measure (3 = 3/4, 4 = 4/4, ..., 7 = 7/4), **track_genre**: string denoting the genre to which the track belongs.

Data cleaning Operativamente, la fase di pre-processing iniziale consiste in:

- The original dataset, *df*, is loaded into memory.
- The columns corresponding to continuous, binary, categorical features, and the target are preserved (specifically, the less relevant columns 'Unnamed: 0' and 'track_id' are removed).
- The presence of missing values for the 'key' variable is checked (examples that have a value of -1 for this feature)[8].
- The 'explicit' feature is converted from boolean to binary (i.e. its domain $\mathcal{X} \equiv T, F$ is transformed into $1, 0$).
- The 'artists' feature can contain multiple artist names in one string, separated by a semicolon. When this occurs for an example $(\boldsymbol{x}, y)$, we split it into a number q of new examples, where q corresponds to the count of distinct artists appearing in the string. Each new example alternately takes only one artist from the original string.
- Since the artist 'bada$$' has special characters in its name, these must be preceded by the appropriate escape character.
- A random permutation, 'df_encoded', of the original dataset is created.
- Redundant examples are removed, as well as the examples that have same characterization for X but are labeled differently ($r = 157193$ examples are preserved).

Therefore, the pre-processed dataset $S_r^{(c=2)}$ has $r = 157193$ examples. Each training subset $S_m^{(-i)}$ has $m = 125754$ and each test fold $S_n^{(i)}$ has $n = 31438$ entries. We will refer to the full dataset as S_r in this section. For what concerns the feature encoding procedure, 'key' and 'time_signature' are ordinal attributes that are already appropriately coded. However, the attributes 'track_genre', 'track_name', 'album_name', and 'artists' need to be properly encoded to represent their values as subsets of real numbers.

One-hot encoding Categorical attributes can be encoded using **one-hot encoding**. This technique converts the i-th categorical variable X_i into d_i binary features, where d_i equals the number of unique values that X_i assumes in S_r (i.e. the cardinality of its domain $|\mathcal{X}_i|$). For each datapoint $\boldsymbol{x} = [x_1, ..., x_{d^{(c=2)}}]$, the i-th feature, displaying value x_i, is therefore mapped to a binary vector $[x_{i1}, ..., x_{id_i}]$ that has 1 in the position x_{ij}, corresponding to the original value x_i assumed by the original i-th column of $\boldsymbol{x}$, and has 0 elsewhere. However, if the feature has an high number of unique values, one-hot encoding significantly increases the total number $d^{(c=2)}$ of features. In our case, using this encoding technique for 'artists', 'track_name' and 'album_name' would result in $d^{(c=2)} \approx 10^4$ features. However, it is manageable to encode 'track_genre' with this technique, leading to $d^{(c=2)} = 129$ features.

BaseN encoding BaseN Encoding encodes categorical variables into integers and then converts them to a base N code (Wong 2023) (if $N = 2$, it converts integers to binary code). Instead of adding a number of new features that equals the number of unique values $|\mathcal{X}_i|$ for the i-th feature, the baseN encoding creates only $\log_N |\mathcal{X}_i|$ new features. A potential problem with this approach arises when employing K-fold nested cross-validation to estimate risk. If the count of unique values found in the training set $S_{\text{train}}^{(-i)}$ differs from that in $S_m^{(-i)}$, the number of features in the two sets could diverge after encoding. As a result, the trained predictors could have different numbers of coefficients.

Hash encoding Hash encoding converts categorical variables into unique hash values using a hash function, allowing for a controlled number of new features. This method ensures that the training set $S_{\text{train}}^{(-i)}$ and $S_m^{(-i)}$ will have the same preset number of features. However, hash encoding is irreversible: original inputs can't be restored from hash values. Furthermore, if too few columns are generated, this could lead to information loss due to hash function collisions, where distinct inputs yield identical outputs.

[8] The dataset does not contain missing values for the 'key' feature.

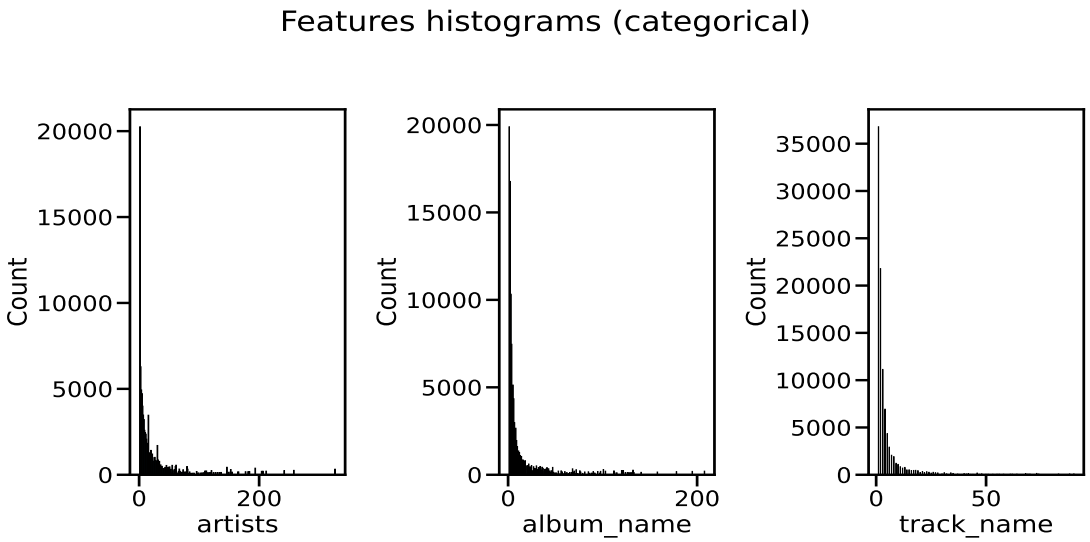


Figure 2: Count encoding of the categorical features 'artists', 'album_name', 'track_name'.

Count encoding Count encoding replaces x_i with the number of times that value occurs in the training set S_m as a value for the i-th feature. This type of encoding is useful when some values occur more often than others. However, if all values appear with the same frequency in the dataset, this encoding becomes ineffective.

Target encoding The **target encoding** involves replacing each value x_i with the average of the corresponding target values y_{jx_i} ($j = 1, ..., |y_{x_i}|$) in the training set S_m. For the test set S_n, the values are replaced with the averages calculated from the training set. This technique creates a new variable that is more correlated with the target, but it can lead to some issues:

- if $|y_{x_i}|$ is small, the target encoding tends to effectively replace the feature value x_i with the target itself. (i.e. this happens exactly when $|y_{x_i}| = 1$). This can undermine the learning process, as it's equivalent to providing the answer within the input data. As the feature takes on less unique values, the number of associated target values $|y_{x_i}|$ increases.
- If a feature has many unique values, there's a chance that only a portion of these values are seen during training. How can we encode those unseen values, since we must rely only on training-related information to encode the test data? This challenge can be addressed by either excluding test examples with feature values unseen during training (which actually is not a good practice) or setting a default value that fits the context of the target. For instance, if a musical genre, among the 114 expected genres, is not included as a 'track_genre' value in the training set S_m (with $m = 125754$), it may indicate that the genre is not very popular. Therefore, a possible default value, for the encoded version of 'track_genre', could be a multiple of the tenth percentile of the

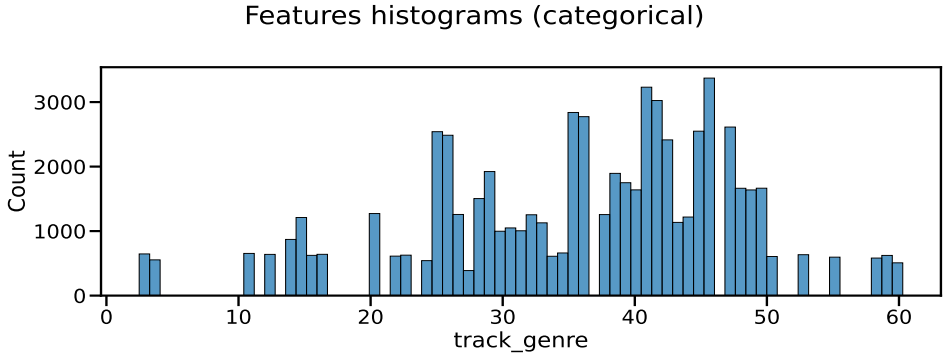


Figure 3: Target encoding of the categorical feature 'track_genre'.

averages $\frac{1}{|y_{x_i}|}\sum_{j=1}^{|y_{x_i}|} y_{jx_i}$ computed for each possible value $x_i \in \mathcal{X}_i$ observed in the training set.

We decided to compare different combinations of encoders, namely:

1. **target encoding** for 'track_genre', **count encoding** for 'artists', 'album_name', 'track_name'.
2. **one-hot encoding** for 'track_genre', **count encoding** for 'artists', 'album_name', 'track_name'.
3. **one-hot encoding** for 'track_genre', **hash encoding** for 'artists', 'album_name', 'track_name'.

3.2 Ridge regression implementation

The 'RidgeRegression' class implements the closed-form solution for ridge regression. It extends the 'BaseEstimator' and 'RegressorMixin' classes from the 'sklearn' library, allowing us to compare our K-fold nested cross-validation implementation with sklearn's ones (namely, using 'validation_curve' and 'GridSearchCV' on an instance of 'RidgeRegression').

The 'RidgeRegression' class is characterized by two attributes:

- **alpha**: regularization factor α.
- **w**: attribute that will store the predictor $\boldsymbol{w}_{S_m}$ trained on S_m.

The 'RidgeRegression' class includes three principal methods, adhering to the interface outlined by sklearn's 'BaseEstimator' and 'RegressorMixin': **fit**, **predict** and **score**.

Fit: The 'fit' method implements the training phase of the linear model. It receives the design matrix S and target vector $\boldsymbol{y}$ from the training set S_m (converting them from pandas dataframes to numpy arrays). It then appends a column of ones to S (allowing the homogeneous coordinates representation) and calculates the predictor $\boldsymbol{w}_{S_m}$ as:

```
self.w = np.dot(np.dot(np.linalg.inv(self.alpha*ID + np.dot(S.T, S)), S.T), ...
    y).reshape(-1)
```

Predict The 'Predict' method implements the prediction phase $\boldsymbol{w}_{S_m}^T \boldsymbol{x}$ given new unseen test instances $\boldsymbol{x}'$. It receives the dataframe S' of datapoints belonging to the test set S'_n. It converts S' to a numpy array expressed in homogeneous coordinates. Then, it computes and returns the vector of predicted labels $\hat{\boldsymbol{y}}' = S'\boldsymbol{w}_{S_m}$:

```
        return np.dot(S_prime, self.w)
```

Score 'Scores' assesses the performance of $\boldsymbol{w}_{S_m}$ on the test set S_n. Theoretically, it should measure the difference between the predicted label $\hat{y}'$ and the actual label y' using square loss, leading to the calculation of test error as the average loss. In practice, allowing performance comparisons across various feature scales, the R^2 score is often employed:

$$R^2_{S_n}(\boldsymbol{w}_{S_m}) = 1 - \; n\,\frac{\ell_{S_n}(\boldsymbol{w}_{S_m})}{\sum_{i=1}^{n}(y'_i - \overline{y}'_i)^2}$$

where $\overline{y}$ is the average true label. The method 'Scores' takes as input the test set S'_n, composed by S' and $\boldsymbol{y}'$, it also needs a 'method' specification, either 'mean_squared_error' or 'r2_score' from 'sklearn.metrics', to tell which evaluation metric will be used. It first predicts the labels calling the **predict** method, which generates the vector $\hat{\boldsymbol{y}}'$. Finally, depending on the specified evaluation metric, it calculates either $\ell_{S_n}(\boldsymbol{w}S_m)$ or $R^2_{S_n}(\boldsymbol{w}_{S_m})$[9].

[9] Indeed, both 'predict' and 'score' can also be used to calculate the predicted labels on the training set and the training error or accuracy $R^2_{S_m}(\boldsymbol{w}S_m)$ (in the previous explanation, simply replace S' with S, $\boldsymbol{y}'$ with $\boldsymbol{y}$, $\hat{\boldsymbol{y}}'$ with $\hat{\boldsymbol{y}}$, and

3.3 K-fold nested CV and best CV estimate implementation

The 'expectedRiskEstimate' class, implementing the K-fold nested cross-validation (ALG. 2) and best CV estimate (ALG. 3), has 20 attributes. Two of these, 'k' and 'model', are mandatory. Nine attributes can be left as default. The remaining eight are private, holding the validation procedure results.

- **k**: number of folds.
- **model**: family A_{Θ_0} of learning algorithms (choosing A between 'RidgeRegression', 'KernelRidgeRegression' and 'DCKRR') onto which the hyperparaameter tuning is performed, allowing the estimation of the expected risk of the best predictor $A_{\theta^*}(S_m^{-i})$.
- **scalers** (default=None): A dictionary where each key-value pair represents a tuple of features names and the corresponding transformation function. In our case, we'll use a dictionary with only one key, which includes all feature names (except for binary and the one-hot-encoded categorical ones, which don't need any scaling), and only one value, the 'StandardScaler()' function from 'sklearn.preprocessing'. If you want to standardize the target, just add the string 'Target' to the unique key of the dictionary.
- **encoders** (default=None): A dictionary where each key-value pair represents a tuple of categorical features names and the encoding method. If $c = 1$ encoders is left to default, otherwise we employ three alternative dictionaries, one for each encoding combination reported at the end of section 3.1.2.
- **target** (default=None): list containing the target name.
- **features** (default=None): list containing all the names of the features involved.
- **evaluation** (default=mean_squared_error): function used for calculating the test error (default) or the R squared score (if it receives r2_score as value).
- **dc_applied** (default=False): A boolean parameter to be set to True if the learning algorithm family implements a 'divide and conquer' heuristic to reduce the actual dimension of the reproducing kernel Hilbert space employed in the kernel ridge regression approach (in our case, it must be set to True if 'model' is DCKRR)[10].
- **sample_tollerance**: the maximum size of each subproblem when using a 'divide and conquer' approach. In other words, the maximum size each sub-sample of $S_m^{(-i)}$ or $S_{\text{train}}^{(-i)}$ can reach so that the respective kernel matrix can be loaded into the main memory.
- **verbose** (default=False): If set to True, it allows to view the printed output produced by each risk estimation method.
- **visualize** (default=False): If set to True, it allows to view plots. Set this to False if you want to execute the 'learning_curve_cv' function (otherwise too many plots will be generated).

The other fields are used for internal results or to store the outcomes of the estimation process. Among these, the 'risk_estimate' field contains either $\ell_{S_m}^{NCV}(A_{\theta^*}(S_m))$ or $\ell_{S_m}^{BCV}(A_{\theta^*}(S_m))$, depending on the chosen estimation method.

The 'expectedRiskEstimate' class has three main methods:

- **nestedKFoldCV**: implementa la K-fold nested cross validation (ALG.2). The procedure also includes the data pre-processing stage. For each single fold, it divides the data into sub-training $S_{\text{train}}^{(-i)}$, development $S_{\text{dev}}^{(-i)}$, training $S_m^{(-i)}$ and test sets $S_n^{(-i)}$, defining also their respective design matrices $S^{(-i)}$, $S^{(i)}$, $S^{(-i,\text{p})}$, $S^{(-i,\text{v})}$ and target vectors $\boldsymbol{y}^{(-i)}$, $\boldsymbol{y}^{(i)}$, $\boldsymbol{y}^{(-i,\text{p})}$, $\boldsymbol{y}^{(-i,\text{v})}$). It then applies the 'fit_transform' method of the current encoder function (value of a key-value pair in the 'encoders' dictionary) to the desired categorical features in $S_{\text{train}}^{(-i)}$ (whose names are in the related dictionary key) and the 'transform' method, of the same encoder, to the categorical features in the corresponding validation set $S_{\text{dev}}^{(-i)}$. It does the same for $S_m^{(-i)}$ and $S_n^{(-i)}$. Similar pre-processing is done for features that need to be standardized through the 'scalers' key-value pairs. If the 'divide and conquer' approach is enabled, it determines the number of subsamples into which $S_m^{(-i)}$ and $S_{\text{train}}^{(-i)}$ must be divided.

$\ell_{S_n}(\boldsymbol{w}S_m)$ with $\ell_{S_m}(\boldsymbol{w}_{S_m})$).

[10] All references to 'kernels' and 'reproducing kernel Hilbert spaces' will be comprehensively detailed in a separate report.

- **bestCV**: Implements the best CV estimate as described in algorithm ALG.3 (adding the same pre-processing described for **nestedKFoldCV**).
- **learningCurveCV**: Runs **nestedKFoldCV** or **bestCV** N times (specifically, $N = 25$ in our scenario), aiming to assess how the estimated expected accuracy evolves as the training set size m increases.

4 Results

4.1 Continuous features

In this part, we're looking at how well our model $A_{\theta^*}(S_m) = \boldsymbol{w}^*_{S_m}$ can match the real patterns in our dataset $S_r^{(c=1)}$. We use the estimates obtained through 5-fold nested cross-validation, as shown in table TABLE 3.

To start with, our model $\boldsymbol{w}^*_{S_m}$ notably underperforms, indicating that ridge regression highly underfits: the 'average training accuracy' and the 'average test accuracy' are both very small, indicating that a linear model is not complex enough to capture the underlying structure of the data ($\ell^{NCVT}_{S_m}(A_{\theta^*}(S_m))$ and $\ell^{NCV}_{S_m}(A_{\theta^*}(S_m))$ are both high). Figure FIG.(4) shows the accuracy of our optimal model $\boldsymbol{w}^*_{S_m}$ as a function of the hyperparameter α. The plot spans 150 evenly-spaced hyperparameter samples from $\alpha_1 = 0$ to $\alpha_{150} = 15000$. Interestingly, the peak value for the estimated expected accuracy is reached for $\alpha = 0$ (yielding a value of 0.052104), which closely aligns with the 'average training error' (0.052403). This means that the average training error $\ell^{NCVT}_{S_m}(A_{\theta^*}(S_m))$, even thought it is a reasonable estimate for the expected statistical risk of $A_{\theta^*}(S_m)$, is really far from being 0.

As mentioned in 2.5, ridge regression, like classical linear regression, is a (super)parametric algo-

Folds	hyper. opt.	Training (acc.)	Test (acc.)	Avg. Training (acc.)	Avg. Test (acc.)
Fold n.0	0.0	0.058069	0.032509		
Fold n.1	0.0	0.052153	0.050667		
Fold n.2	0.0	0.052359	0.051353	0.052403	0.052104
Fold n.3	0.0	0.049899	0.062133		
Fold n.4	0.0	0.049535	0.063855		

Table 3: 5-fold nested cross validation estimate with only continuous features (execution $c = 1$).

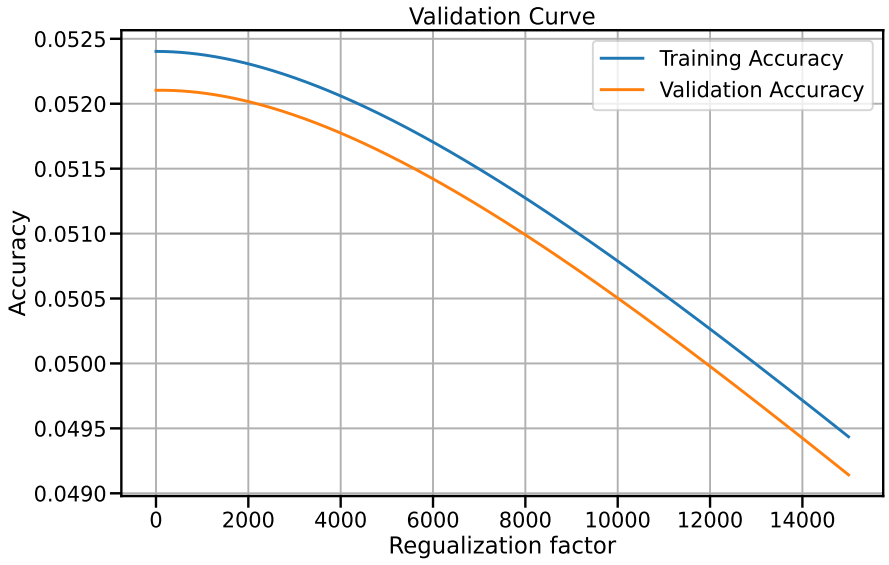
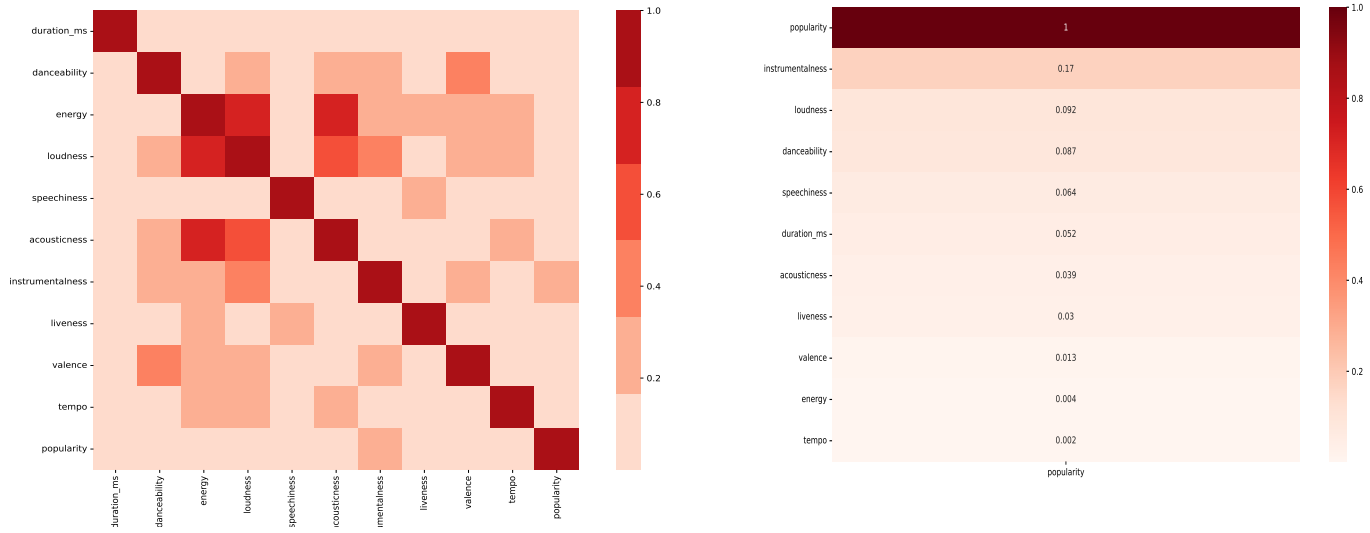


Figure 4: Validation curve ($c = 1$).



(a) Linear correlation ($c = 1$)

(b) Linear correlation, w.r.t. the target ($c = 1$)

Figure 5: Features-target Pearson correlation coefficient, computed on the dataset $S_r^{(c=1)}$.

rithm. This implies that if the phenomenon under investigation is complex and features a weak linear correlation with the target (either due to the intrinsic nature of the attributes or the choice of attributes), attempting to approximate the relationship between Y and X using a hyperplane in $d^{(c=1)}$ dimensions may not be a suitable approach. In FIG.(5a) we display the correlation table calculated on $S_r^{(c=1)}$, summarizing the linear correlation between all pairs of individual features, including the target. A focused view of the linear correlations between individual features and the target is provided in FIG. (5b). Initial analysis indicates a lack of substantial linear correlation among the continuous attributes describing X, with the 'instrumentalness' feature being a minor exception, exhibiting a Pearson correlation coefficient of $\rho = 0.017$.

The optimal predictor is shown in TABLE 4 and can be visualized, broken down into its $d^{(c=1)}$ components, in FIG. (6). In the mentioned figure, each subplot displays the predictor's intercept and i-th coefficient, revealing the relationships that $\boldsymbol{w}^*_{S_m}$ attempts to capture between each feature and the target. Underfitting is evident as all coefficients of $\boldsymbol{w}^*_{S_m}$ are close to zero, resulting in the components aligning with the horizontal axis almost in each subplot.

One might argue that the observed behavior is due to the ridge regression algorithm introducing too much bias (i.e. potentially an overly large α hyperparameter) to counterbalance its own instability. However, evidence suggests otherwise. When we consider a large training size of $r = 83475$ or $m = 66780$, we find that the optimal hyperparameter value is actually $\alpha^* = 0$. This suggests that even the basic linear regression model remains stable at this scale. As such, there's no necessity for regularization to enforce a simpler model, as the empirical risk minimization (ERM) on the linear predictor class naturally outputs a stable, optimal predictor.

In Figure FIG. (7a) we present a learning curve illustrating how the model's accuracy changes as the number of examples m in each i-th training part $S_m^{(-i)}$ increases. The curve results from 25 runs

Fold	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
Fold n.0	2.920478e-16	-0.049339	0.097607	-0.085117	0.054836	-0.085473	-0.052351	-0.189070	-0.007893	-0.095176	-0.001510
Fold n.1	-4.494533e-17	-0.046886	0.096548	-0.074812	0.054242	-0.085133	-0.042099	-0.176475	-0.003615	-0.096694	0.003057
Fold n.2	4.252668e-17	-0.044641	0.095973	-0.079710	0.061709	-0.085037	-0.038168	-0.176003	-0.001399	-0.099484	0.001774
Fold n.3	-1.365028e-16	-0.041419	0.097380	-0.067316	0.053855	-0.081911	-0.038092	-0.172998	-0.004976	-0.098849	-0.001382
Fold n.4	-2.090342e-16	-0.043602	0.091084	-0.068207	0.052873	-0.081386	-0.043672	-0.174029	-0.004304	-0.098355	0.002030

Table 4: Coefficients of the i-th optimal predictor $\boldsymbol{w}^*_{S_m^{(-i)}}$, tested on the i-th fold $S_n^{(i)}$ and trained on the i-th training part $S_m^{(-i)}$, through 5-fold nested cross validation ($c = 1$).

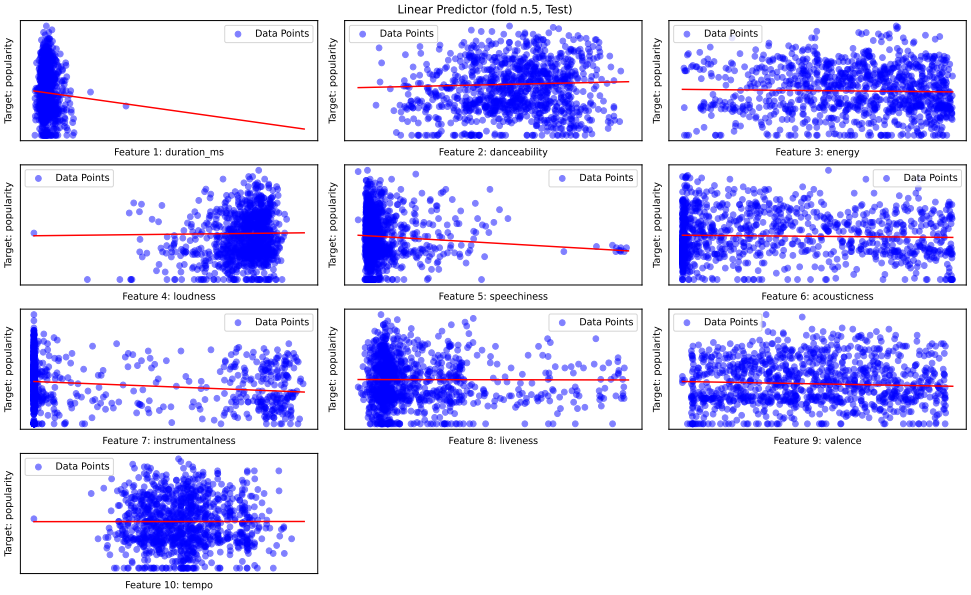
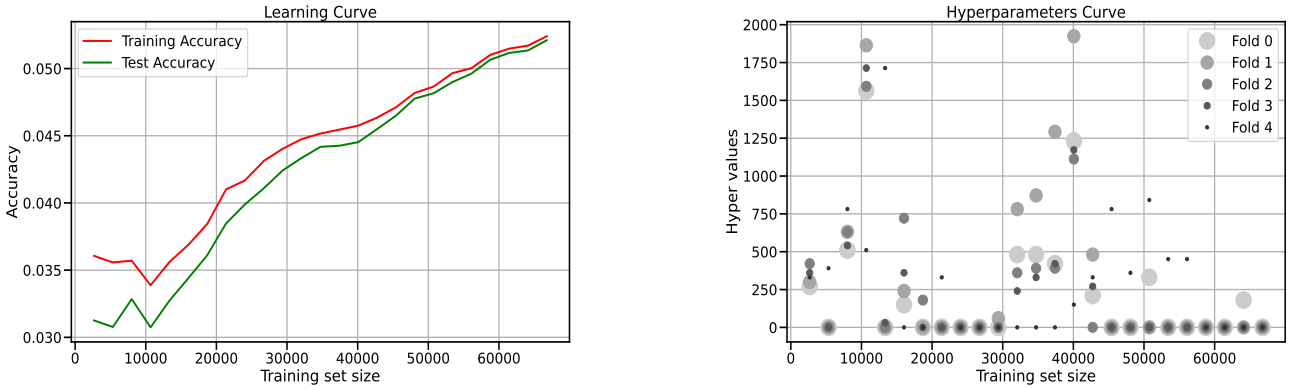


Figure 6: Optimal predictor for the 5-th fold ($c = 1$).



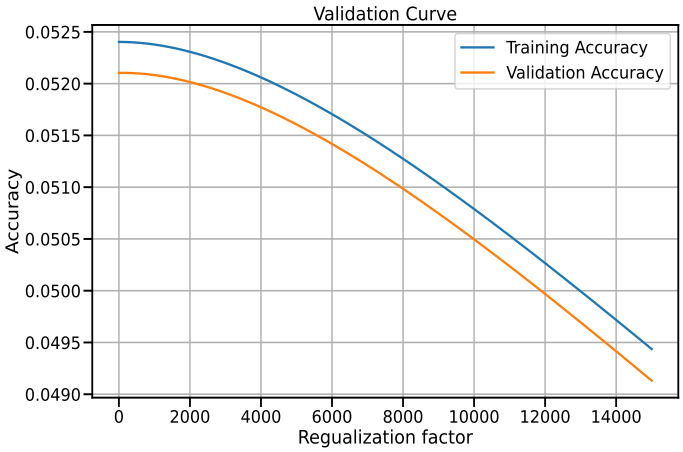
(a) Learning curve ($c = 1$)

(b) Hyperparameter plot ($c = 1$)

Figure 7: Learning accuracy and hyperparameter values, as the sample size m increases.

params	mean_test_score
'alpha': 0.0	0.052104
'alpha': 50.1672240	0.052104
'alpha': 100.3344481	0.052104

(a) GridSearchCV results



(b) validation_curve results

Figure 8: Rsults obtained using library implementation of 5-fold nested CV on $S_r^{(c)}$ with $c = 1$ (i.e. only continuous and standardized features) with a 150-points-evenly-sampled hyperparameter space $\Theta_0 = [\alpha_1, ..., \alpha_{150}]$, where $\alpha_1 = 0$ and $\alpha_{150} = 15000$.

of 5-fold nested cross-validation, gradually increasing m over 25 discrete values $(m_0, ..., m_{24})$ between

$m_0 = 2671$ and $m_{24} = 66780$. As the figure illustrates, the stability term 13 consistently shrinks as m expands, eventually reaching a point around 50000 samples where the average training accuracy becomes a reliable estimate for the expected accuracy. This trend is also apparent in FIG. (7b), where optimal hyperparameters for each i-th fold are plotted against their corresponding m value. For small training sizes, high hyperparameter values are preferred, but for larger sizes, they stabilize around $\alpha^* = 0$, independently of the specific fold.

To ensure that 'RidgeRegression and 'expectedRiskEstimate' perform in a compatible way w.r.t. the libraries implementation of the ridge regression and K-fold nested CV, in FIG.(8) we show the validation curve obtained running the sklearn implementation of K-fold nested CV, with A_{Θ_0} being the sklearn class for ridge regression (i.e. using the Ridge() object). We also run 'GridSearchCV' in the same setting, showing the estimated result in the table beside the plot. Comparing this results with the ones obtained using our implementations, we can see that there are no difference in the validation curves and in the estimates (despite our implementation is a little bit slower).

4.2 Continuous, ordinal (qualitative), binary and categorical features

In this section, we assess and discuss the performance of $A_{\theta^*}(S_m) = \boldsymbol{w}^*_{S_m}$ on the dataset $S_r^{(c=2)}$ using 5-fold nested cross-validation. Unlike the previous section, the phenomenon of interest is here represented by a combination of continuous, binary, ordinal (qualitative), and categorical features. In this analysis, we explore the impact of various encoding techniques for categorical attributes, as hinted in section 3.1.2.

Folds	hyper. opt.	Training (acc.)	Test (acc.)	Avg. Training (acc.)	Avg. Test (acc.)
Fold n.0	0.000000	0.399457	0.393754		
Fold n.1	0.000000	0.396725	0.397675		
Fold n.2	0.000000	0.397175	0.396536	0.397405	0.395789
Fold n.3	0.000000	0.399001	0.394533		
Fold n.4	402.684564	0.394665	0.396447		

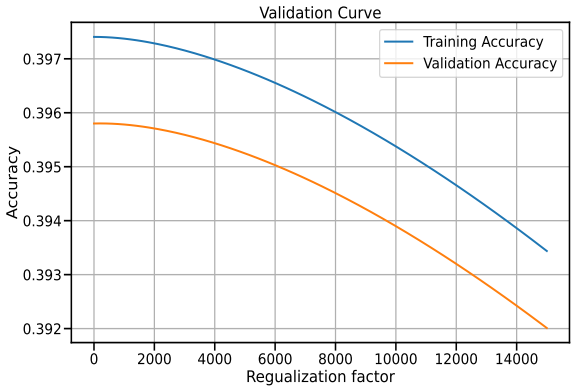
Table 5: 5-fold nested cross validation estimate with continuous, ordinal (qualitative), binary and categorical features (execution $c = 2$). Target encoding for 'track_genre' and count encoding for 'artists', 'track_name', 'album_name'.

Folds	hyper. opt.	Training (acc.)	Test (acc.)	Avg. Training (acc.)	Avg. Test (acc.)
Fold n.0	100.671141	0.453248	0.447921		
Fold n.1	100.671141	0.452226	0.452826		
Fold n.2	100.671141	0.453398	0.452717	0.452959	0.450925
Fold n.3	100.671141	0.455539	0.449489		
Fold n.4	100.671141	0.450384	0.451672		

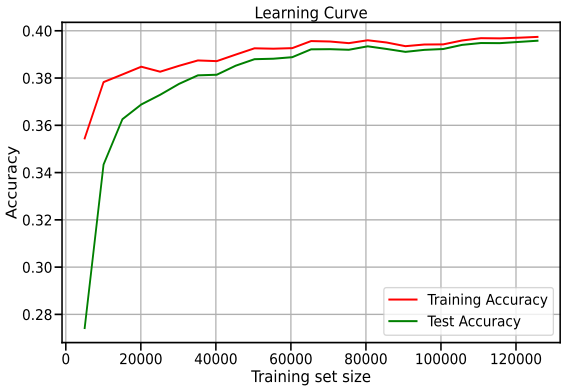
Table 6: 5-fold nested cross validation estimate with continuous, ordinal (qualitative), binary and categorical features (execution $c = 2$). One-hot encoding for 'track_genre' and count encoding for 'artists', 'track_name', 'album_name'.

Folds	hyper. opt.	Training (acc.)	Test (acc.)	Avg. Training (acc.)	Avg. Test (acc.)
Fold n.0	100.671141	0.266936	0.264152		
Fold n.1	100.671141	0.266055	0.266884		
Fold n.2	100.671141	0.266589	0.264821	0.266686	0.26471
Fold n.3	100.671141	0.266143	0.266763		
Fold n.4	100.671141	0.267708	0.260930		

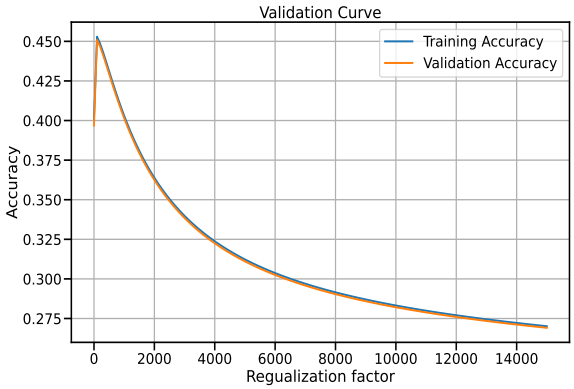
Table 7: 5-fold nested cross validation estimate with continuous, ordinal (qualitative), binary and categorical features (execution $c = 2$). One-hot encoding for 'track_genre' and hash encoding for 'artists', 'track_name', 'album_name'.



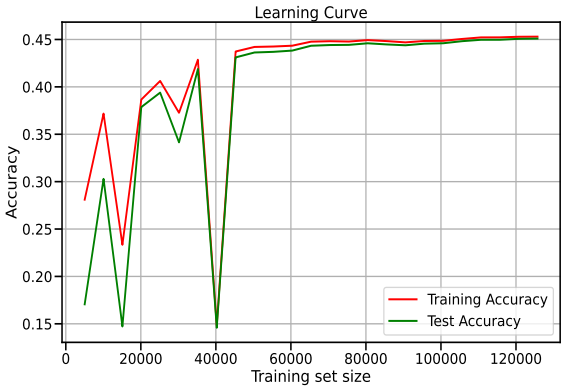
(a) Validation curve ($c = 2$)



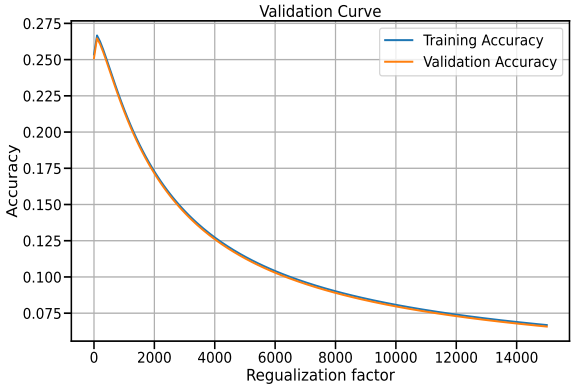
(b) Learning curve ($c = 2$)



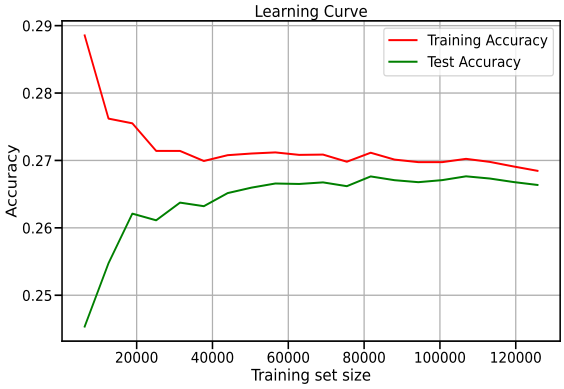
(c) Validation curve ($c = 2$)



(d) Learning curve ($c = 2$)



(e) Validation curve ($c = 2$)



(f) Learning curve ($c = 2$)

Figure 9: In FIG. (9a) and (9b): validation curve and learning curve using target encoding for 'track_genre' and count encoding for 'artists', 'track_name', 'album_name'. In FIG. (9c) and (9d): validation curve and learning curve using one-hot encoding for 'track_genre' and count encoding for 'artists', 'track_name', 'album_name'. In FIG. (9e) and (9f): validation curve and learning curve using one-hot encoding for 'track_genre' and hash encoding for 'artists', 'track_name', 'album_name' (50 newly added hash features).

We observed that integrating categorical features ('track_genre', 'artists', 'album_name', 'track_name') into our model consistently diminishes model bias, irrespective of the encoding approach employed. Unlike the almost null expected accuracy observed when addressing problem $c = 1$, here our average test accuracy approaches values close to 0.5, which is typical for a random classifier. Moreover, incorporating categorical features does not lead to overfitting, as the average training accuracy remains close to the estimated expected accuracy in all three encoding scenarios analyzed.

Focusing specifically on using one-hot encoding for the 'track_genre' attribute and count encoding for the remaining categorical ones, we find an optimal regularization factor of $\alpha^* = 100.671$. The optimal regularization factor $\alpha^* = 100.671$ remains the same across all validation folds, considering a hyperparameter space of 150 evenly distributed samples ranging from $\alpha_0 = 0$ to $\alpha_{150} = 15000$. These

results indicate that, with the given sample size of $r = 157193$ ($m = 125754$), classical linear regression yields a poorer predictor compared to ridge regression.

In a more detailed comparison, when we juxtapose Figure FIG.(9c) with its counterparts for different encoding strategies, it becomes evident that the pronounced increase in model bias at near-zero regularization factors (i.e., in the case of classical linear regression) is primarily due to the use of one-hot encoding for handling the 'track_genre' feature. One-hot encoding may introduce issues of linear dependence between the many newly added binary columns in both the training and testing parts. This could, in turn, cause the matrix S^TS to approach singularity. This issue isn't so much about the fact that $\boldsymbol{w}^*_{S_m}$, computed with $\alpha = 0$, can easily change if S_m is slightly perturbed. Rather, it relates to the fact that, whatever realization S_m or S'_n of statistical samples we take, the datapoints in these datasets, once one-hot encoded for their categorical attributes, will lie in a subspace of $R^{d^{(c=2)}}$ whose dimension is strictly lower than $d^{(c=2)}$. Despite these challenges, ridge regression, by design, proves to be robust against the effects of one-hot encoding (as illustrated, by the selecting $\alpha^{=}100.671$ across all folds)

Among the three encoding combinations tested, the most effective in reducing the bias of the linear predictor is the one that uses one-hot encoding for 'track_genre' and count encoding for the other categorical features. This approach yields an 'average test accuracy' of 0.450925 and an 'average training accuracy' of 0.452959. However, it is important to note that the model still falls short of providing a satisfactory approximation of $P(\mathrm{Y}|\mathrm{X})$. To address the identified problem, we can consider the following strategies:

- changing the hypothesis class $\mathcal{H}$. This could require using a distinct analytical expression for the optimal ERM solution or entirely switching to a different learning algorithm.
- redefining how we describe X. This might involve identifying features that we presume to be linearly correlated with the target Y.
- we could apply the 'kernel trick' to train a predictor in a Hilbert space using kernel ridge regression, which would effectively convert a linear learning algorithm into a non-linear one in the original space. This approach essentially transforms a parametric learning algorithm, already adept at minimizing estimation error, into a non-parametric one. In other words, as the sample size m tends to infinity, this algorithm becomes universally consistent (this holds true when the Gaussian kernel is employed to establish the inner product in the considered space).

Appendices

A Connections with decision models

Fixing ℓ, $A = \text{ERM}$ and S_m, we can consider the choice of the predictor $h_{S_m} \in \mathcal{H}_m$, performed by ERM, as a deterministic (single scenario $\omega \in \Omega$) single decision-maker decision problem (Cordone 2018):

$$\Big(\mathcal{H}_m,\ \Omega,\ F \subset \mathbb{R},\ \ell_{S_m}(h),\ \{\text{ERM}\},\ \preceq_{\text{ERM},S_m}\Big)$$

which is the problem of choosing an alternative $h_{S_m} \in \mathcal{H}_m$ whose impact $\ell_{S_m}(h_{S_m}) \in F \subset \mathbb{R}$ is preferred, by the single decision-maker ERM (whose rationality is represented by a weak-order preference relation $\preceq_{\text{ERM},S_m}$ over the impacts $\ell_{S_m}(h) \in F \subset \mathbb{R}$), to those of the other alternatives $h \in \mathcal{H}_m$. Specifically:

- **feasible reagion**: the image subspace $\mathcal{H}_m$, induced by A on the hypothesis class $\mathcal{H}$, is the set of all possible alternatives $h \in \mathcal{H}_m$ among which the decision maker $A = \text{ERM}$ should choose according to a preference relation $\preceq_{\text{ERM},S_m}$ expressed over the impacts $\ell_{S_m}(h) \in F \subset \mathbb{R}$.
- **(weak-order) preference relation**: a preference relation is a function $\preceq_{\text{ERM},S_m}$: ERM $\to 2^{F\times F}$ that associates to ERM a subset of ordered pairs of impacts $\ell_{S_m}(h) \preceq_{\text{ERM},S_m} \ell_{S_m}(h')$, $\forall h, h' \in \mathcal{H}_m$, meaning that ERM is more prone to choose h instead of h' as $\ell_{S_m}(h)$ is preferred to $\ell_{S_m}(h')$.

$\preceq_{\text{ERM},S_m}$ is 'weak-oreder' since it is *reflexive* (ERM is always able to compare an impact to itself), *complete* (ERM is always able to sort the impacts from the best one to the worst one in $\mathcal{H}_m$, though allowing ties) and *transitive*. ($\preceq_{\text{ERM},S_m}$ is not *antisymmetric* since it is not true that two impacts are indifferent only in the specific case in which they are perfectly identical).

- **(single-indicator) impact**: an impact $\ell_{S_m}(h)$ refers to the real-world consequences that a particular alternative $h \in \mathcal{H}_m$ will produce after being selected as the optimal solution by the decision-maker ERM among the available options. consists in the effect that a specific alternative $h \in \mathcal{H}_m$ will produce on the real world once it has been chosen as the optimal solution, by the decision-maker ERM, among the other possibilities. The impact domain, specified by the loss ℓ, is denoted as F. Given that, for ERM being the decision-maker, an impact is characterized by a single-indicator (i.e. it is scalar), the domain F is designated as a subset of $\mathbb{R}$.
- **set of decision-makers**: it is the set comprising whoever takes part in a decision. In our case (namely, $A = \text{ERM}$), there is only one entity that plays a role in the decision process, therefore the set $\{\text{ERM}\}$ of decision-makers has only one element.
- **scenarios**: factors that could influence the real-world consequences resulting from the selection of the specific alternative $h \in \mathcal{H}_m$ as the optimal solution $h_{S_m} \in \mathcal{H}_m$. Essentially, if $|\Omega| > 1$, impacts would additionally depend on $\omega \in \Omega$, rather than only on alternatives in $\mathcal{H}$. However, in our case we consider only the deterministic case in which $|\Omega| = 1$.

Weak-order preferences enable the arrangement of impacts along a line, as if each impact were associated to a cost (potentially with overlapping values). A **cost function** assigns a real value (the abovementioned cost) to every impact:

$$J : F \to \mathbb{R}$$

$$\forall h \in \mathcal{H}_m, \;\; \ell_{S_m}(h) \;\mapsto\; J(\ell_{S_m}(h))$$

This value $J(\ell_{S_m}(h))$ quantifies the decision-maker's perception of how less-favorable the real-world consequence of a particular alternative $h \in \mathcal{H}_m$ might be. A cost function is said to be **consistent** w.r.t the preference relation $\preceq_{\text{ERM},S_m}$ when:

$$\forall h, h' \in \mathcal{H}_m \,, \;\; \ell_{S_m}(h) \preceq_{\text{ERM},S_m} \ell_{S_m}(h') \;\Leftrightarrow\; J(\ell_{S_m}(h)) \leq J(\ell_{S_m}(h'))$$

reducing the whole decision process to selecting the alternative that minimizes the cost function:

$$h_{S_m} = \arg\min_{h \in \mathcal{H}_m} J(\ell_{S_m}(h))$$

When the impact is one-dimensional ($F \subset \mathbb{R}$), $J(\ell_{S_m}(h)) = \ell_{S_m}(h)$ is a consistent cost function w.r.t $\preceq_{\text{ERM},S_m}$, therefore the ERM decision process can be formulated as:

$$h_{S_m} = \arg\min_{h \in \mathcal{H}_m} \ell_{S_m}(h)$$

which is the minimization of the empirical risk, as stated in section 2.2.

5 Citations and references

References

Cordone, Roberto. 2018. 'decision methods and models' course lecture notes https://homes.di.unimi.it/cordone/courses/2022-mmd/DMM.pdf.

Halford, Max. 2018. Target encoding done the right way https://maxhalford.github.io/blog/target-encoding.

Shalev-Shwartz, Shai & Shai Ben-David. 2014. *Understanding machine learning: From theory to algorithms*. Cambridge university press.

Taboga, Marco. 2021. 'Ridge regression', Lectures on probability theory and mathematical statistics https://www.statlect.com/fundamentals-of-statistics/ridge-regression.

Wong, Kay Jan. 2023. Feature encoding techniques in machine learning with python implementation. *Towards Data Science* https://towardsdatascience.com/feature-encoding-techniques-in-machine-learning-with-python-implementation-dbf933e64aa.