Predicting Spotify Track Popularity Using Ridge Regression

Statistical Methods for Machine Learning

Marwan Agourram (967349)

Accademic Year 2022-2023

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

Abstract

The goal of this report is to develop a predictive system for estimating the popularity of songs from the 'Spotify Track Dataset'. The predictive model is based on Ridge Regression, which has been implemented from scratch and finally compared with the result obtained with the Ridge Model from `Scikit-Learn package`. Furthermore, the risk estimates have been computed using 5-fold cross-validation.

Contents

List of Figures

1 Introduction

Spotify, a Sweden-based company, is one of the most important streaming services that is being utilized by the majority of the population across the globe. The simplicity of the user interface mixed with an enormous music catalogue allowed the company to increase its market share. Starting from its foundation in 2006 until more recent days, the company has always found ways to offer an excellent service to its customers. Moreover, in the last decade, the company has implemented several approaches, most of which are based on machine learning techniques, in order to increase the capability to provide a better experience to its subscribers and therefore, became the leading provider in the market.
Most of these machine learning techniques have been implemented on several features that the company itself generates about its platform reaching almost every aspect of the user experience, concentrating mostly on the musical domain of the service. In fact, all the songs that are included in Spotify have been analysed in order to extract additional informations about the song itself: these aspects range from the length of the song to the ability to induce the listener to dance while listening. An important aspect of the Spotify platform is the one regarding the 'track popularity' of a song, a numerical indicator computed by their own algorithm that represents the total number of plays the track has had and how recent those plays are.

In this report I'll focus on developing a predictive model from scratch that aims at predicting in the most accurate way the 'popularity' of a song. For this purpose, a sample of the original Spotify dataset (retrieved before 2022) has been extracted and obtained from Kaggle. Then, on this sample, several data processing techniques have been applied in order to handle both categorical and numerical features and therefore obtain a dataset that could be used to perform ridge regression. Initially, I performed the ridge regression model only on the numerical features and, after that, on all features that were available in the dataset, obtaining two different results that allow the comparison and the reasoning on whether the inclusion of the categorical features helps increasing the predictability of the model. Finally, 5-fold cross-validation is applied on both the models in order to evaluate the estimation capability of the models on previously unseen data.

2 Feature Description and Data Preprocessing

'Spotify Track Dataset', which is available in .csv format on the Kaggle website, is a sample obtained by collecting data via Spotify's Web API. It is composed by 114,000 tracks and 21 features, both numerical (15 and one boolean feature) and categorical (5). The categorical features resembles characteristics related to the `artist`, the `album_name`, the `track_name`, the `track_genre` (in the dataset there are 114 different genres) and, most importantly, the `track_id`, which is the unique identifier associated to each song in the dataset. On the other hand, the numerical features represents proper aspects of each track such as: `duration_ms`, indicated in milliseconds; `danceability`, which indicates how suitable a track is for dancing; `energy`, which indicates the intensity and activity of the song; `key`, that represents the key in which the song is recorded (0 being C and up to 11 being B#); `loudness`, which is the overall loudness of the track (indicated in decibels); `mode`, that represents the scale of the song (major or minor scale); `speechiness`, that detects the presence of spoken words in the track; `acousticness`, that indicates if the track is acoustic; `instrumentalness`, that measures if the track is more instrumental; `liveness`, which represents the presence of live audiences in the song; `valence`, which describes the perceived positivity conveyed by the song; `tempo`, which indicates the Beats Per Minute (BPM) of the song; `time_signature`, that specifies the amount of beats per bar of the song; and `explicit`, which is a boolean feature that indicates whether or not a song contains explicit words.

The dataset contains just one row that present three `NaN` values related to categorical features (`artist_name`, `track_name` and `album_name`). For simplicity and since the `NaN` values are not related to numerical features, the row has been deleted obtaining a dataset with 113,999 tracks. Moreover, different features are removed since they do not provide any additional information for the purpose. In particular, the `Unnamed: 0` column is removed since it contains only the count of the rows. Then, other features such as the `track_name`, the `artist` and the `album_name` have been removed during the feature encoding phase: considering just the `artists` feature, there are exactly $31,437$ unique artists. For instance, if we consider to implement one-hot encoding technique on this these features, we would end up with a dataset that shows an enormous increase in the cardinality that would not be matched by an increase in the predictability of the model.

Another important aspect that concerns the dataset is the one related to the `track_id` feature. As I said before, this feature represents the unique identifier associated with each track. However, considering the unique values for these feature, we end up with 89,740 unique track id's. This is due to the fact that Spotify assigns to the same track different musical genres (generating duplicates across the dataset). In fact, if we consider this duplicate songs we see that all of them has the same values for all of the attributes except for the track genre. In order to deal with this phenomenon we could have simply deleted this duplicates. However, this is not the chosen approach for this report since removing duplicate features without a methodological approach may lead to information loss. Therefore, one-hot-encoding have been implemented on the

`track_genre` features computed on a new different dataset (which has only `track_id` and `track_genre` features)[1] , and then joined the resulting encoded dataset with the original one. Therefore, we reduce the size of our dataset (from 113,999 to 89,740), but we also increase the cardinality of the same (from the initial 21 features to 145).

Finally, concentrating on each singular feature we notice that different numerical features have different distributions. By inspecting the dataset we can see how most of the numerical features takes values between $[0, 1]$. However, other features such as `tempo`, `loudness`, and `duration` (which have been transformed from the original `duration_ms` dividing by 1000, obtaining therefore the length of the song in seconds) takes continuous values that do not range in $[0, 1]$ and, most importantly, shows the presence of outliers[2]. These features in particular and all of the other ones (including the ones obtained from the encoding process) have been `min-max` normalized, therefore obtaining a dataset containing features for which values range between $[0, 1]$.
Finally `explicit`, which is the only boolean feature in the dataset, is trasformed in `int64` type and then also normalized.

After all the processing, I created two different dataset that will later on be used to perform the two objective tasks, that is, `numb_data` which contains only numerical features (shape $89,740 \times 11$), and `total_data` that contains all the features, included the one-hot-encoded ones (shape $89,740 \times 145$).

[1]Initially, target encoding was implemented with respect to `track_genre` feature, that is, each genre was encoded by considering the mean value of `popularity` grouped by genre. The obtained result was one numerical feature that was representing the average popularity associated with each genre. Compared with the final chosen solution, it is obvious that this technique was better in terms of the impact on the cardinality of the dataset. However, while developing the ridge regression model and in particular when setting the penalization coefficient $\alpha = 0$, several errors were faced. Moreover, the ridge regression model computed on `categorical_data` resulted in a singular matrix. This may be due to linearity dependency that have been introduced by encoding `track_genre` as the popularity mean. Therefore, it becomes impossible to uniquely determine the coefficients of the $(S^\top S)$ matrix.

[2]These outliers refers in particular to `loudness, speechiness, instrumentalness, liveness, tempo and duration`. As a rule of thumb, it is generally suggested to avoid considering outliers while performing analysis and, in particular, when performing regression tasks. On the other hand, these features are related to aspects of the song that may actually affect the popularity of the same (e.g., a listener may prefer shorter songs compared to 15 minutes tracks). Therefore, for the purpose, the outliers are kept in the dataset.

3 Ridge Regression

Linear regression is a statistical tool for modelling the relationship between some explanatory variables and some real valued outcome. The relationships are modelled using linear predictor functions whose unknown parameters are estimated from the data.
Formally, given a dataset $\boldsymbol{X} = \mathbb{R}^d$ and a label set $\boldsymbol{Y} \in \mathbb{R}$, a linear predictor is a function $h : \mathbb{R}^d \to \mathbb{R}$ such that $h(\boldsymbol{x}) = (\boldsymbol{w}^\top \boldsymbol{x})$, where $\boldsymbol{w} \in \mathbb{R}^d$ is the vector of real coefficients.
Given a training set $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\} \in \mathbb{R}^d \times \mathbb{R}$, the linear regression predictor is the empirical risk minimizer (ERM) with respect to the square loss function:

$$\boldsymbol{w}_S = \underset{\boldsymbol{w} \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{t=1}^{m} (\boldsymbol{w}^\top x_t - y_t)^2 = \|\boldsymbol{v} - \boldsymbol{y}\|^2,$$

where $\boldsymbol{v} = (\boldsymbol{w}^\top x_1, \ldots, \boldsymbol{w}^\top x_m)$ and $\boldsymbol{y} = (y_1, \ldots, y_m)$ are both column vectors. Let S be the design matrix (shape $m \times d$) such that $S^\top = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m]$, and since $\boldsymbol{v} = S\boldsymbol{w}$, we may write the ERM function as:

$$\boldsymbol{w}_S = \underset{\boldsymbol{w} \in \mathbb{R}^d}{\operatorname{argmin}} \|S\boldsymbol{w} - \boldsymbol{y}\|^2 = \underset{\boldsymbol{w} \in \mathbb{R}^d}{\operatorname{argmin}} F(\boldsymbol{w}).$$

Recalling that $F(\boldsymbol{w})$ is a convex function that is minimized at $\nabla(F(\boldsymbol{w})) = 0$, we aim at finding the solution for $\boldsymbol{w}$. After some matrix calculus and provided that $(S^\top S)$ is non-singular, we obtain that the ERM with respect to the square loss is:

$$\boldsymbol{w}_S = (S^\top S)^{-1} S^\top \boldsymbol{y}.$$

Two of the key assumptions in the linear regression model concern multicollinearity between independent variables and the singularity of the product $(S^\top S)$. When one of these two assumptions fails, it is preferable to introduce a regularization term in the objective function to prevent overfitting. In particular, there are two ways to introduce regularization: L1 regularization, also known as Lasso regularization, which shrinks all the coefficients towards 0; and L2 regularization, known as Ridge regularization, which penalizes all the coefficients, except for the intercept, by shrinking them towards a small value different from 0. For the purpose of this report, the latter technique has been implemented and performed from scratch.

Ridge regression is a regularized version of linear regression that introduces a regularization parameter commonly known as L2-norm. Formally, when $(S^\top S)$ is nearly-singular (i.e., ill-conditioned matrix), the coefficient vector $\boldsymbol{w}_S$ is highly sensitive to perturbations in the training set which may result in an increase in the variance error. The regularization parameter aims at increasing the bias error while reducing the variances' one, lowering the overall risk of the model.
Formally, the regularized form is known as Ridge Regression and it is defined by adding a penalty term which is based on the L2-norm of the coefficients:

$$\boldsymbol{w}_{S,\alpha} = \|S\boldsymbol{v} - \boldsymbol{y}\|^2 - \alpha\|\boldsymbol{w}\|^2 = F(\boldsymbol{w}_{S,\alpha}),$$

where $\alpha \in \mathbb{R}^+$ is the regularization parameter and $\|\boldsymbol{w}\|^2$ is the squared norm of the coefficient vector. Moreover, by controlling the value of α we can control the bias of the algorithm. Finally, when $\alpha = 0$ we recover the standard linear regression previously defined.
As before, $F(\boldsymbol{w}_{S,\alpha})$ is a convex function for which its minimizer satisfies $\nabla(F(\boldsymbol{w}_{S,\alpha})) = 0$. Therefore, by solving for the gradient we obtain that the objective function in the ridge regression model takes form:

$$\boldsymbol{w}_{S,\alpha} = (\alpha I + S^\top S)^{-1} S^\top \boldsymbol{y},$$

where I is the identity matrix. Moreover, we can see how the assumption about the singularity of the product $(S^\top S)$ becomes irrelevant: considering the eigenvalues for $S^\top S$ as $\lambda_1 \geq \ldots \lambda_d \geq 0$, we can see that the eigenvalues for $(\alpha I + S^\top S)$ becomes $\alpha + \lambda_1 \geq \ldots \alpha + \lambda_d \geq 0$. Therefore, $(\alpha I + S^\top S)$ is invertible for all $\alpha > 0$, a condition that is always satisfied.

4 Experimental Setup

In Section 2 I explained all of the data processing steps that we're performed on the original dataset in order to obtain `numb_data` and `total_data`. As the last step before implementing ridge regression, a `train-validation-test` split is performed on both datasets, randomizing the splitting process and selecting a 70% of the dataset to be included in the training set, 15% to be included in the validation set and the remaining 15% to be included in the test set[3]. For the purpose of this report, `popularity` have been chosen as the target variable `y` in the splitting process, excluding it from the training set.

After the data-processing phase, the Ridge Regression algorithm has been introduced as a `class` object by the name `RidgeRegression()` which contains all of the callable functions that will be used throughout the report. In particular, this function is obtained by the algorithm shown in Algorithm 1, reported in the Appendix.
Finally, the penalty α is introduced as an equally distanced set of 100 values, ranging from 10^{-3} to 10^{7}. This set of values of α will be used for all the computations, except for the part in which we will implement the linear regression model (by setting $\alpha = 0$).

4.1 Ridge Regression on Numerical Features

The `RidgeRegression()` object is implemented on the numerical features and the MSE computed both on the training and testing part, appending the results to two different lists. This same process is also performed by implementing the `Ridge()` function from the `sklearn.linear_model` package, in order to obtain results computed on the same data and therefore, allowing to compare also the performance of the scratch model with respect to the original one.
The scratch model has `mean(MSE)=0.056311` on the train data and `mean(MSE)=0.056928` on the test data. As we can notice from Figure 1, both the MSE curves plotted with respect to the regularization term follows the same behaviour. In particular, the MSE is below the average in both cases for approximately $\alpha < 10^5$, and then grows exponentially for very high α values. On the other hand, we can see how the Scikit ridge model behave similarly to the scratch model for $\alpha < 10^3$, and then grows moderately for higher values of α. Moreover the Scikit ridge regression model achieves an `mean(MSE)=0.041248` on the train data, while for the test data `mean(MSE)=0.042043`. In both cases, the obtained results are definitely lower than the ones achieved with the scratch model.

[3]Dataset containing numerical features has the following shape: `X_train`: $(62817, 10)$, `X_val`: $(13461, 10)$, `X_test`: $(13462, 10)$, `y_train`: $(62817,)$, `y_val`: $(13461,)$, `y_test`: $(13462,)$.
Dataset containing categorical features has the following shape: `X_train`: $(62817, 144)$, `X_val`: $(13461, 144)$, `X_test`: $(13462, 144)$, `y_train`: $(62817,)$, `y_val`: $(13461,)$, `y_test`: $(13462,)$.

4.2 Ridge Regression on Categorical Features

The same process previously described is also applied to `total_data`, that is, the dataset containing all of the features, even those that we're encoded. Starting from the scratch model, on the train data we have that `mean(MSE)=0.046554`, while on the test set `mean(MSE)=0.046554`. On the other hand, the Scikit model register `mean(MSE)=0.033376` for train set, while `mean(MSE) = 0.034049` for the test set. Furthermore, we can see in Figure 2 how for the scratch model the MSE is way below the average both for train and test part for regularization value approximately $\alpha < 10^5$, which then is followed by an increase. In particular, we can see how for values $10^2 < \alpha < 10^5$ the increase in the MSE is more contained. On the other hand, for the Scikit model shows a more homogeneous behaviour: compared to the results obtained on `numb_data`, MSE of train and test data do not exhibit an evident distance, leading to the conclusion that the Ridge model performed on the whole dataset is actually more accurate. This conclusion is obviously based on the result of the MSE, which is way more smaller than the one obtained in the previous sub-section.

4.3 Linear Regression on Numerical and Categorical Features

When the penalty term $\alpha = 0$, we reconstruct the linear regression setting. In order to compare the performance achieved by the scratch model, comparison is made with respect to the model implemented using `Scikit_module` and the standard `linear_regression` one.
When applied on `numb_data`, the three models achieve exactly the same results. In fact, for all of them, the `mean(MSE)` on the training set is `0.040829`, while the `mean(MSE)` on the test set is `0.041646`, which corresponds to the values obtained while implementing ridge regression with $\alpha = 10^{-3}$. See Figure 3 for the graphical representation.

Once again, when we perform linear regression on `total_data`, we notice that the three models achieve approximately the same results (differences starting from the sixth decimal digits, with the scratch model achieving the lowest errors). However, in all of the cases we notice that the errors are smaller than the ones obtained with the numerical dataset: the `mean(MSE)` on the training set is `0.027201`, while the `mean(MSE)` on the test set is `0.027841`. We can further explain this increase in the performance by the dataset itself, since these results are obtained while considering all of the features which again turns out to be more explainable compared to only the numerical ones. See Figure 4 for the graphical representation.

4.4 Cross-validation

Cross-validation is used to assess the performance of predictive models that we just observed. In particular, it is applied on the scratch model and on the comparison model, which implements the `RidgeCV()` module that is appropriately designed for this purpose.
For the scratch model, cross-validation have been performed from scratch following

Algorithm 2, which also implements hyper-parameter tuning in order to select the best value for the penalty term α.
When implemented on the scratch model, 5 cross-validation selects $\alpha = 0.001$ when we applied both on `numb_data` and `total_data`.
Similarly, when we use the Scikit model, 5 cross validation returns as best value for the penalty term for both `numb_data` and `total_data` hte value $\alpha = 0.001$.
This suggests that with the current setting, we do not need to penalize heavily the coefficients.

5 Conclusions

This report allows us to perform ridge regression in two different ways, that is, implementing the model from scratch and using the Scikit package. In the initial stage of the experiment, we saw that the scratch model is able to perform well in terms of metrics results, although it is less efficient compared to the Scikit model.
When cross-validation is applied, we notice that the estimates for penalty terms turns out to be exactly the same for both the scratch and the Scikit model. Finally, we dived also into linear regression in order to comprehend an important behaviour related to ridge regression.
The whole report relied heavily on the data processing stage: the approach that has been applied for this report is finalized to maintain the highest amount of information that comes originally with the dataset. Moreover, the results confirm that when we consider the whole dataset (the one with the encoded features and the numerical ones), we obtain better performances compared to when the model is trained only on numerical features. This suggests that features such as `track_genre` are very important when it comes to predict track popularity.

6 Appendices: Code

Algorithm 1 Ridge Regression from Scratch

Initialization: Initialize instance variables for α, coefficients, and mean squared error (MSE).

Fit(*input*: Data matrix $\boldsymbol{X}$ and target values $\boldsymbol{y}$):

- Augment the input data $\boldsymbol{X}$ by adding a column of ones for the intercept term.
- Calculate the coefficients using the closed-form solution for Ridge Regression:
  - Calculate $A = (\boldsymbol{X}^\top \boldsymbol{X} + \alpha I)$.
  - Calculate $b = (\boldsymbol{X}^\top \boldsymbol{y})$.
  - Solve the linear system, obtaining the coefficients values $\boldsymbol{w}_{\boldsymbol{X},\alpha} = A^{-1}b$.

Predict(*input*: Data matrix $\boldsymbol{X}$):

- Augment the input data $\boldsymbol{X}$ by adding a column of ones for the intercept term.
- Compute predictions for target value by multiplying the augmented data with coefficients.

Calculate MSE(*input*: Data matrix $\boldsymbol{X}$ and target values $\boldsymbol{y}$):

- Implement the **Predict($\boldsymbol{X}$)** method obtaining the predicted value for the target feature $\boldsymbol{y}$.
- Calculate the MSE between the feature $\boldsymbol{y}$ and the predicted one.
- **Output**: MSE.

Calculate MAE(*input*: Data matrix $\boldsymbol{X}$ and target values $\boldsymbol{y}$):

- Implement the **Predict($\boldsymbol{X}$)** method obtaining the predicted value for the target feature $\boldsymbol{y}$.
- Calculate the MAE between the feature $\boldsymbol{y}$ and the predicted one.
- **Output**: MAE.

Calculate RMSE(*input*: Data matrix $\boldsymbol{X}$ and target values $\boldsymbol{y}$):

- Implement the **Predict($\boldsymbol{X}$)** method obtaining the predicted value for the target feature $\boldsymbol{y}$.
- Calculate the MSE between the feature $\boldsymbol{y}$ and the predicted one, and the compute RMSE as the square root of MSE.
- **Output**: RMSE.

Algorithm 2 Cross Validation from Scratch

Initialization: Shuffle dataset index. Create empty lists for storing the k-folds and the various metrics.

Input: Dataset, list of α penalization terms, number of folds k:

- Compute the size of each fold by dividing the number of rows in the dataset by the number of folds k.
- For $i = 1, \ldots, k$:
  - Compute start and end index.
  - Create fold using the computed indexes.
  - Assign fold to the list of folders.
- Compute train and test set: in the former we drop the target column, while in the latter we keep just the target column.
- Create lists for the metrics that will be used temporary during the iteration.
- Set variable `best_alpha=0`, that will be replaced by the optimal value of the penalty term. Set the upper bound ($+\infty$) for the MSE that will be used for comparison.
- For all the values of α in the list:
  - Train and fit the ridge regression model.
  - Compute the MSE on the test set.
  - If MSE computed on test set is lower than the upper bound for MSE, assign the corresponding value of α as `best_alpha`.
- Run ridge regression using the optimal value of α and compute all the metrics.
- Append the result of the metrics to the corresponding lists.

7 Appendices: Figures

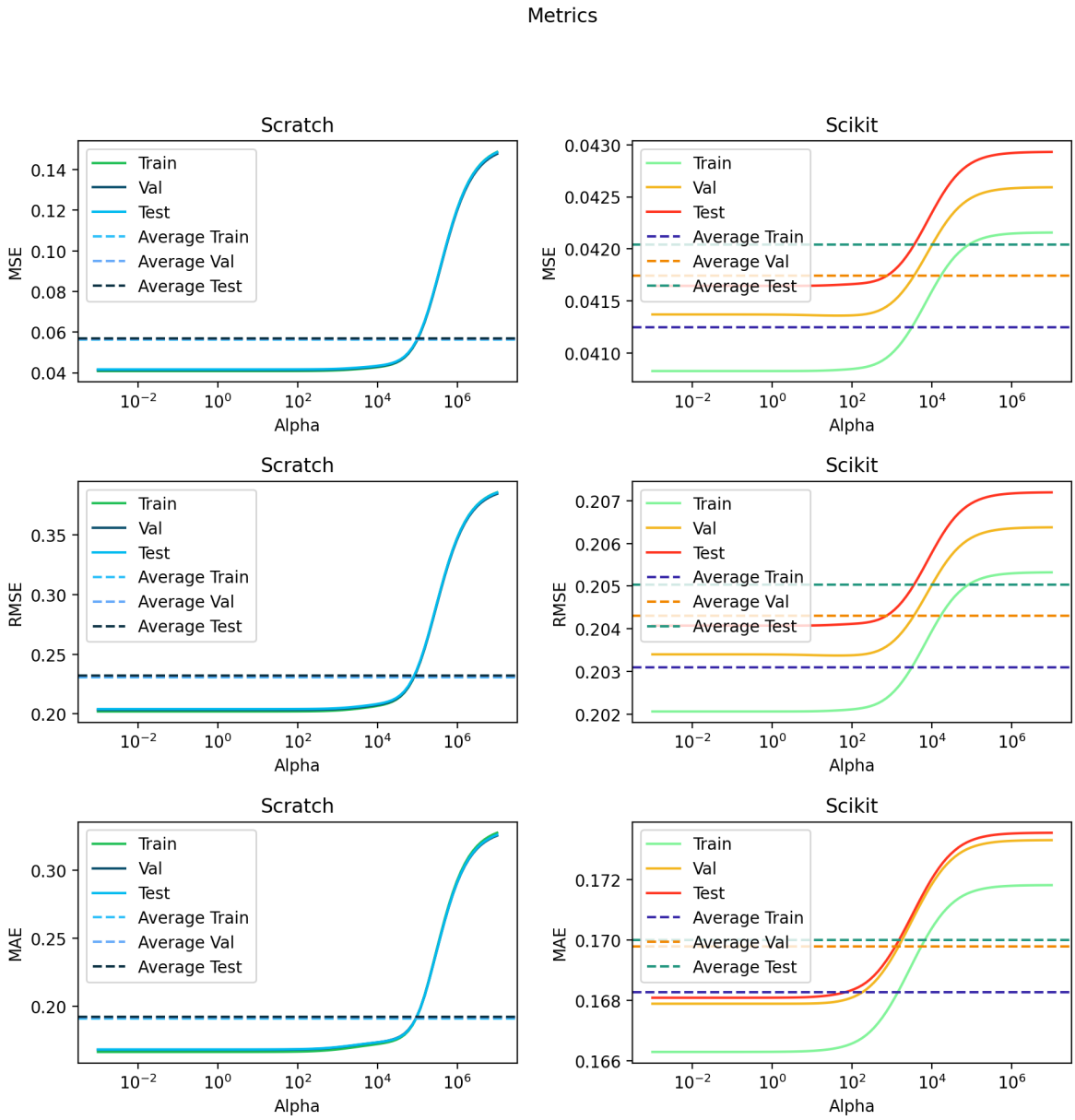

Figure 1: Comparison between the scratch model (left) and the Scikit-Learn ridge regression model (right) applied on `numb_data`: on the x-axis the regularization term α in log-scale, while on the y-axis the MSE (top), the RMSE (middle), and the MAE (bottom)

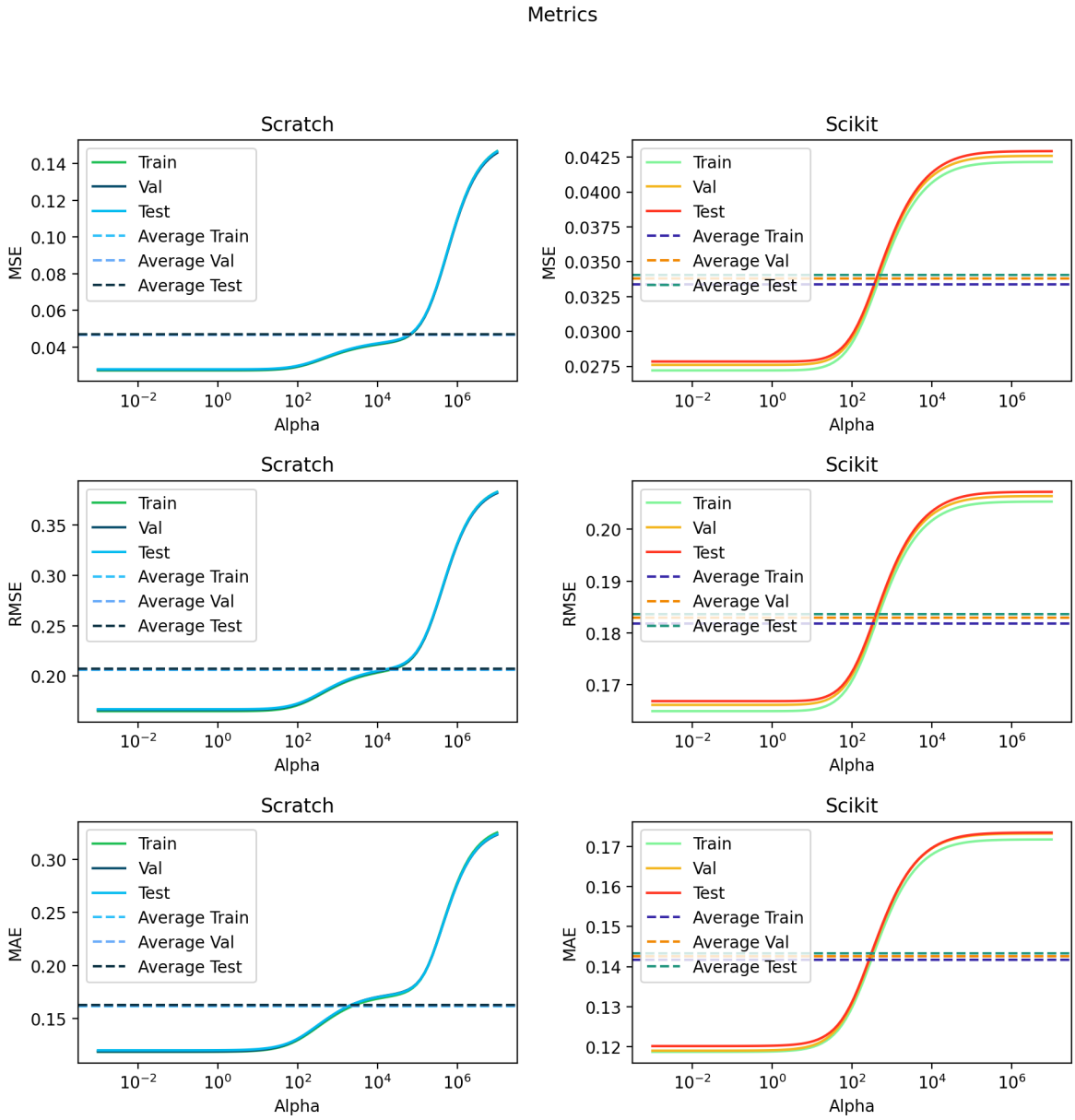

Figure 2: Comparison between the scratch model (left) and the Scikit-Learn ridge regression model (right) applied on `total_data`: on the x-axis the regularization term α in log-scale, while on the y-axis the MSE (top), the RMSE (middle), and the MAE (bottom).

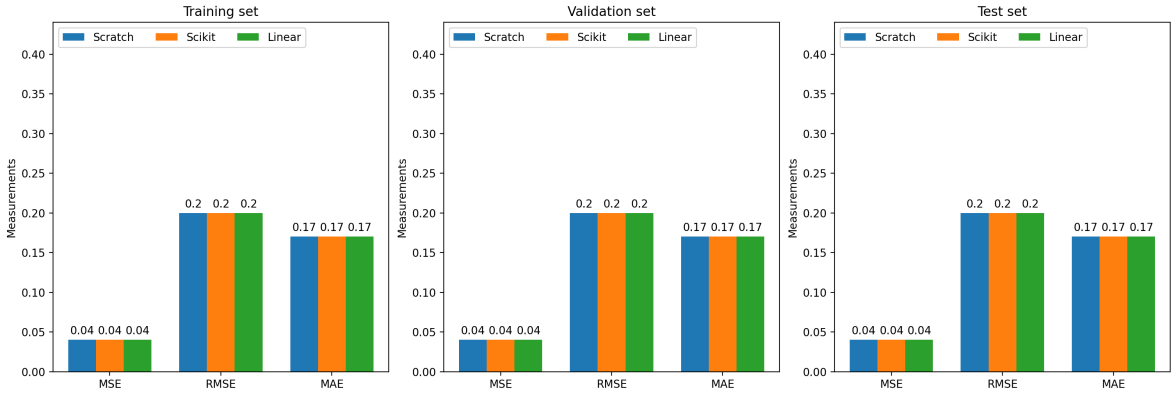

Figure 3: Comparison between the scratch model (blue), Scikit-Learn regression model (orange), and the linear regression model (green) applied on `numb_data`: on the x-axis the measurements, while on the y-axis the achieved results. From left to right: training set, validation set, test set.

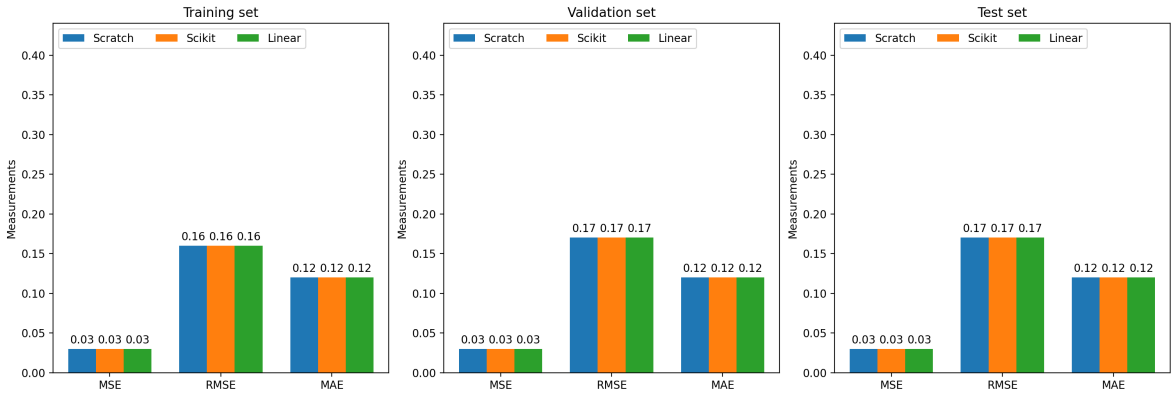

Figure 4: Comparison between the scratch model (blue), Scikit-Learn regression model (orange), and the linear regression model (green) applied on `total_data`: on the x-axis the measurements, while on the y-axis the achieved results. From left to right: training set, validation set, test set.

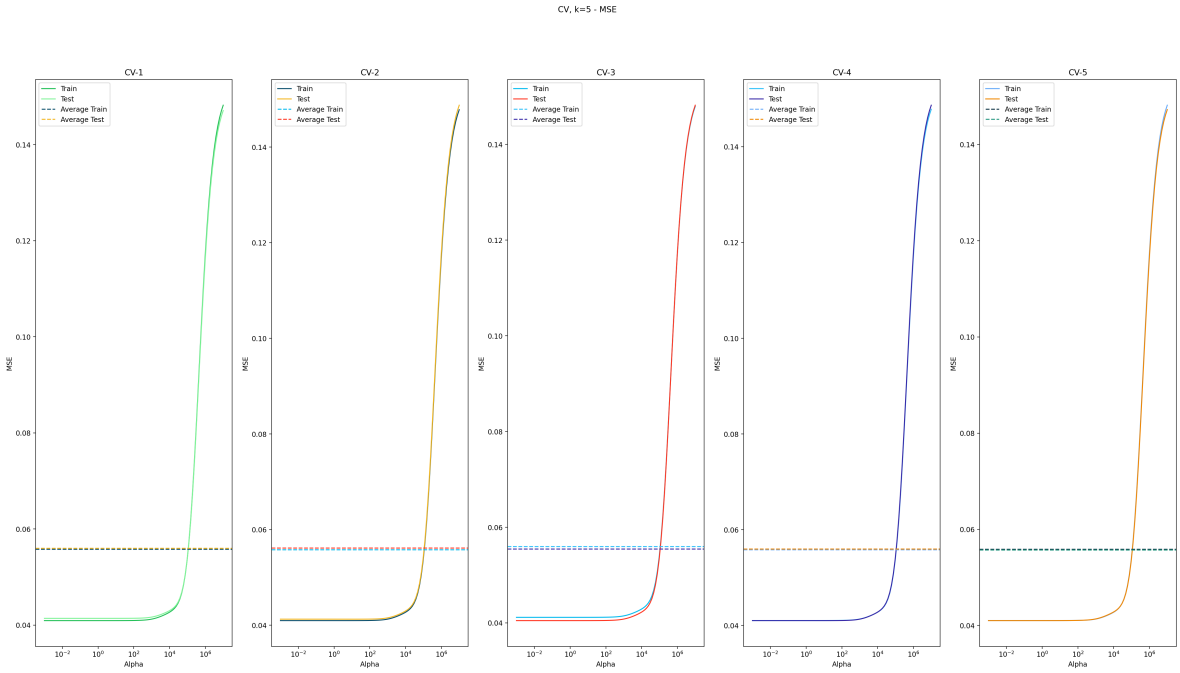

Figure 5: 5 cross validation on the numerical data using the scratch model.

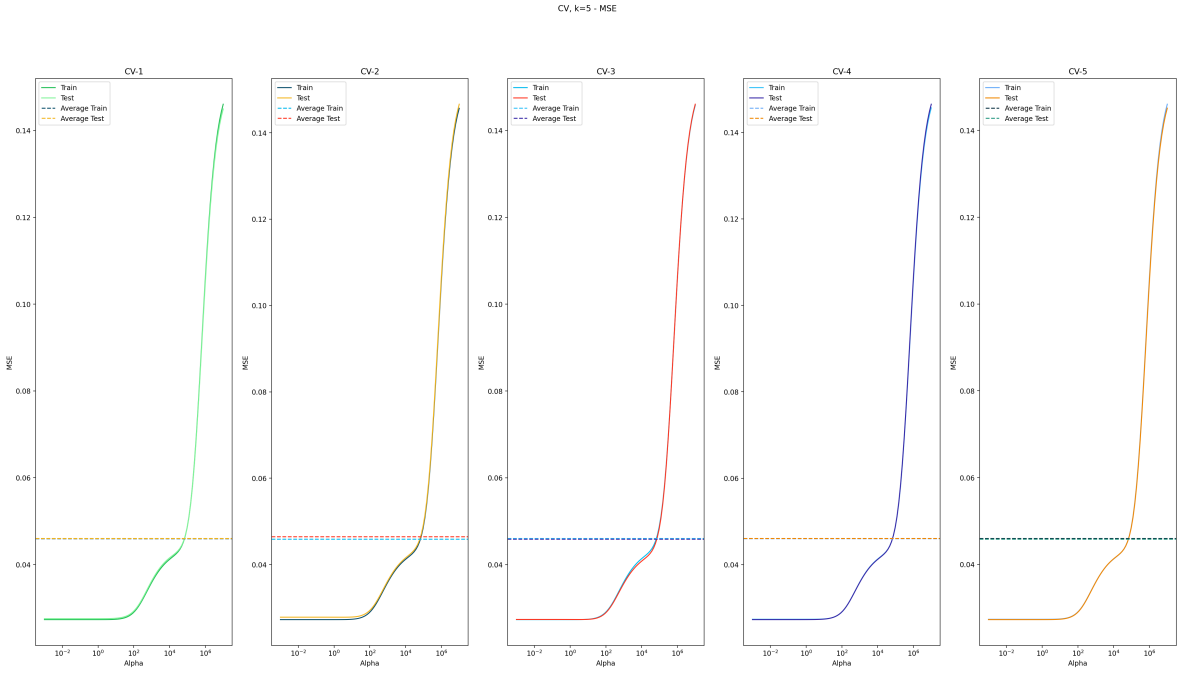

Figure 6: 5 cross validation on the categorical data using the scratch model.

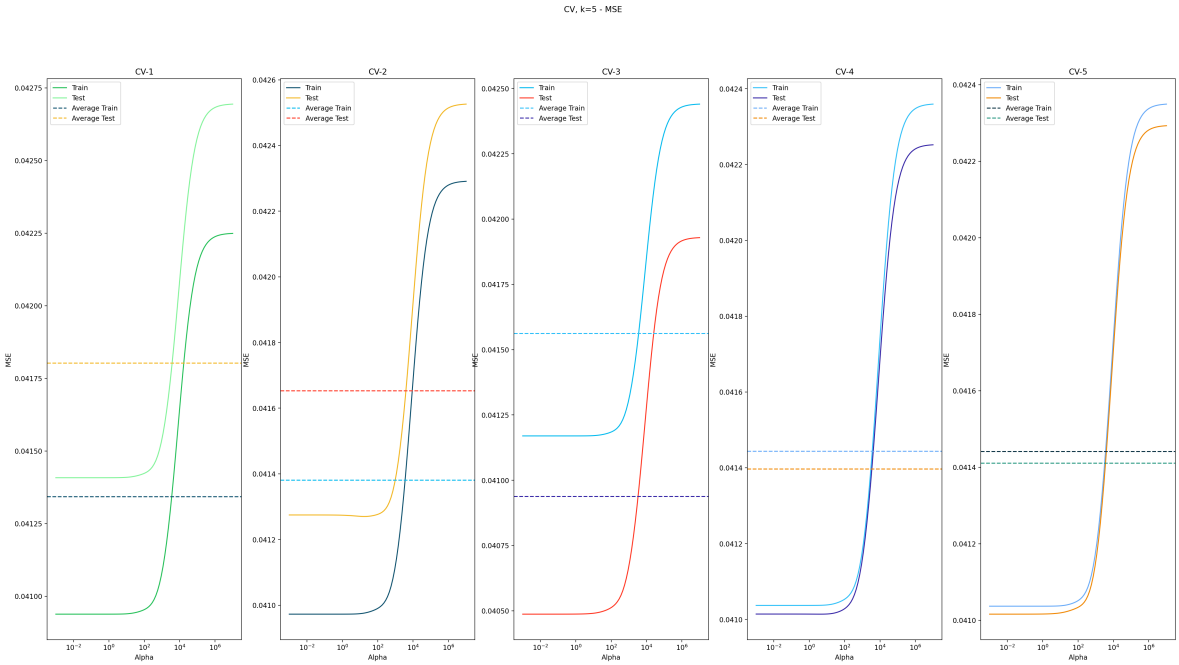

Figure 7: 5 cross validation on the numerical data using the Scikit model.

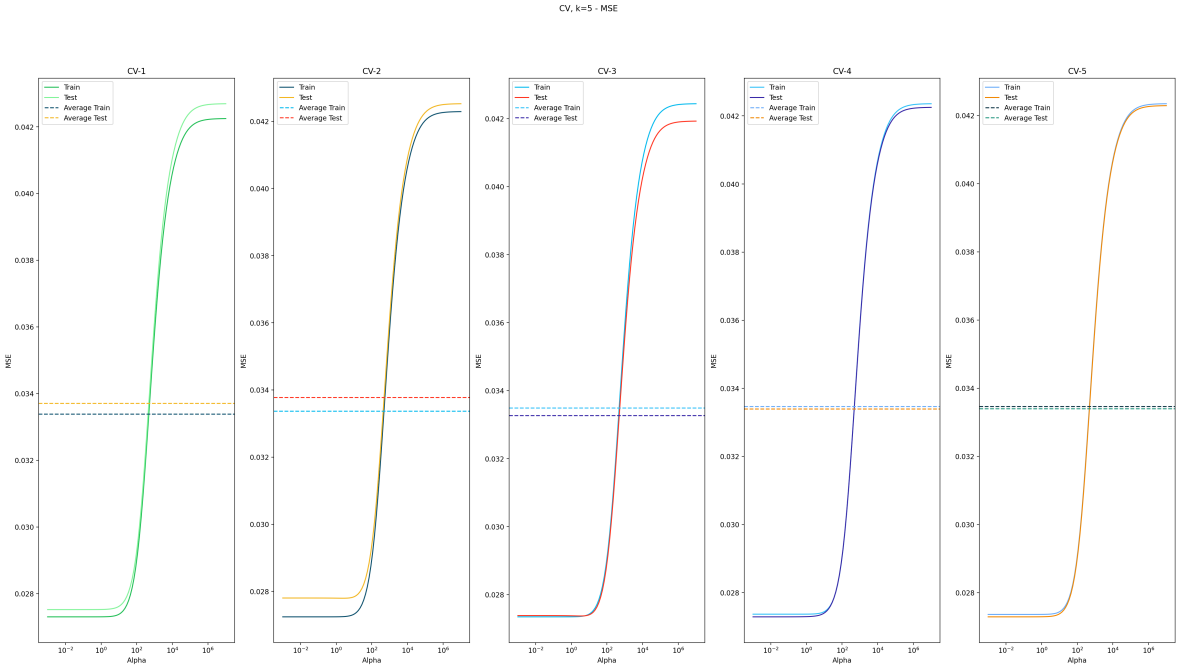

Figure 8: 5 cross validation on the categorical data using the Scikit model.

8 References

Spotify Tracks Dataset, *Kaggle, MAHARSHIPANDYA*
Linear predictors, *Cesa-Bianchi, Nicolo'*
Hyperparameter Tuning and Risk Estimates, *Cesa-Bianchi, Nicolo'*
Ridge Package, *Scikit Learn*
How to Evaluate Your Machine Learning Models with Python Code!, *Datatron Blog*
One Hot Encoder, *Scikit Learn*
Ridge Regression, *Wikipedia*
Spotify, *Wikipedia*
How to Code Ridge Regression from Scratch, *Towards Data Science, Jake Miller Brooks*
Target-encoding Categorical Variables *Towards Data Science, Vinícius Trevisan*
K-Fold Cross Validation Example Using Sklearn Python, *Towards Data Science, Cory Maklin*
Ridge and Lasso Regression in Python — Complete Tutorial (Updated 2023), *Analytics Vidhya, Aarshay Jain*