Ridge Regression: Experimental Project

Edoardo Vergani
Università degli Studi di Milano

Abstract

Predicting the success of songs is essential in the ever-changing music industry. However, the fundamental question is: what factors contribute the most to a song's popularity? To answer this question, I investigated the Spotify tracks dataset, which contains audio characteristics and metadata for approximately 90,000 songs, using Ridge Regression.

1 Introduction

The digital age has brought about a significant transformation in the music industry, with streaming platforms revolutionising the way music is accessed and consumed. Understanding the dynamics that govern the popularity of tracks on platforms like Spotify has become a crucial endeavour for artists, record labels, and music enthusiasts alike. This experimental project explores music data analysis to predict track popularity using ridge regression and a dataset that includes numerical and categorical features.

Music has held a significant place in human culture throughout history, and this cultural connection continues today in the music industry's substantial $26.2 billion in annual global revenues by 2022. The United States, the world's largest music market, has 82.2 million of the world's 616.2 million paid subscribers to music streaming services.

Interestingly, a significant portion of the global recorded music industry's revenues can be traced back to specific sources. Some 47.3% of this revenue comes from subscription-based audio streams, 17.7% from ad-supported streams and 19.2% from physical music sales [1].
Notably, the lion's share of this revenue is generated by popular, mainstream songs.

Understanding the factors that contribute to a song's popularity is crucial for businesses closely connected to popular music, such as radio stations, record labels, and digital and physical music marketplaces. Accurately predicting song popularity is also essential for personalized music recommendations and addressing the challenge of predicting preferred songs within specific demographics.

In this context, the focus is on the concept of 'Hit Song Science', which was coined by Mike McCready and trademarked by Polyphonic HMI. The concept revolves around the idea of predicting a song's potential for success even before its official release.

The project consists of two primary phases. In the first phase, a ridge regression model was trained and evaluated using only numerical features. These numerical attributes encapsulate various aspects of a track's musical composition, offering insights into its danceability, energy, tempo, and more. The aim is to assess the predictive power of these numerical characteristics in gauging a track's popularity.

However, the popularity of music is not solely determined by its inherent attributes. It is also influenced by the genre under which it is categorized. Therefore, the second phase of my project involves incorporating categorical features through one-hot encoding.

During my investigation, I conducted meticulous experimentation, employing 5-fold cross-validation to fine-tune model hyperparameters, notably the regularization term (alpha). A battery of evaluation metrics, including Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2), is employed to gauge the model's precision and effectiveness.

2 Dataset and Features

The dataset used for this analysis is the Spotify Tracks Dataset by Maharshipandia [2], obtained from Kaggle. It includes audio features for tracks across 125 genres, such as key, year, and genre. The dataset consists of 114,000 rows and 21 columns. To ensure data quality, tracks without the considered features were removed, as well as those with duplicate track IDs. The variables included in the dataset are:

- **track_id** (*int*): the Spotify ID for the track
- **artists** (*string*): the artists' names who performed the track
- **album_name** (*string*): the album name in which the track appears
- **track_name** (*string*): name of the track
- **popularity** (*int*): the popularity of a track (0-100) is calculated by algorithm and is based, in the most part, on the total number of plays the track has had and how recent those plays are
- **duration_ms** (*int*): the track length in milliseconds
- **explicit** (*string*): whether or not the track has explicit lyrics
- **danceability** (*float*): danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity
- **energy** (*float*): energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity
- **key** (*int*): the key the track is in.
- **loudness** (*float*): the overall loudness of a track in decibels (dB)
- **mode** (*int*): mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0
- **speechiness** (*float*): speechiness detects the presence of spoken words in a track
- **acousticness** (*float*): a confidence measure from 0.0 to 1.0 of whether the track is acoustic

- **instrumentalness** (*float*): predicts whether a track contains no vocals
- **liveness** (*float*): detects the presence of an audience in the recording
- **valence** (*float*): a measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track.
- **tempo** (*float*): the overall estimated tempo of a track in beats per minute (BPM)
- **time_signature** (*int*): an estimated time signature. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).
- **track_genre** (*string*): the genre in which the track belongs

It is commonly understood that a track is more likely to be popular if it is created by an artist who is already popular. Therefore, I have introduced a new feature called 'artist_popularity', which calculates the average popularity of songs in the dataframe grouped by artist and can be used as an additional prediction feature.

I reviewed all songs with duplicated track IDs and kept only one per ID. The resulting dataframe contains 89,740 rows and 21 columns.

3 Methods

The aim of the project is to perform ridge regression to predict the popularity of tracks.

The model will be trained using only numerical features, and then categorical features will be handled by applying one-hot encoding. The resulting features will be used together with the numerical ones to train the model.

In both cases, 5-fold cross-validation will be used to compute risk estimates.

3.1 Ridge Regression

In the linear model, the estimate of β is obtained by solving the normal equations $X^T X\beta = X^T y$ [3].
The difficulty of solving this system of linear equations can be described by the *condition number*

$$k(X^T X) = \frac{d_{\max}}{d_{\min}}$$

which is the ratio between the largest and the smallest singular values of $X^T X$.
If this condition number is very large, the matrix is said to be ill-conditioned.

Ridge provides a remedy for an ill-conditioned $X^T X$ matrix.
If our $n \times p$ design matrix X has column rank less than p, then the usual least-squares regression equation is in trouble:

$$\hat{\beta} = (X^t X)^{-1} X^t y$$

What we do is add a *ridge* on the diagonal - $X^t X + \lambda I_p$ with $\lambda > 0$ - which takes the problem away:

$$\hat{\beta}_\lambda = (X^t X + \lambda I_p)^{-1} X^t y$$

which is the ridge regression solution proposed by Hoerl and Kennard (1970).

Ridge regression modifies the normal equations to

$$(X^T X + \lambda I_p)\beta = X^T y$$

The optimization problem that ridge is solving

$$\min_{\beta} ||y - X\beta||^2 + \lambda||\beta||^2$$

where $||\cdot||$ is the ℓ_2 Euclidean norm.

When including an intercept term, we usually leave this coefficient unpenalized, solving

$$\min_{\alpha,\beta} ||y - 1\alpha - X\beta||^2 + \lambda||\beta||^2$$

Ridge regression is not invariant under scale transformations of the variables, so it is standard practice to centre each column of X (hence making them orthogonal to the intercept term) and then scale them to have Euclidean norm $\sqrt{n}$.
It is straightforward to show that after this standardisation of X, $\hat{\alpha} = \bar{y}$ so we can also centre y and then remove α from our objective function.

Let $\tilde{y} = (y - 1\bar{y})$ and $\tilde{X} = (X - 1\bar{x}^t)\text{diag}(1/\text{s})$ be the centered y and standardized X, respectively, with:

- $\bar{y} = (1/n)\sum_{i=1}^{n} y_i$,
- $\bar{x} = (1/n)X'1$,
- $s = (s_1, \ldots, s_p)^t$ and $s_j^2 = (1/n)\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2$

Then, compute the scaled coefficients

$$\tilde{\beta}_\lambda = (\tilde{X}^t\tilde{X} + \lambda I_p)^{-1}\tilde{X}^t\tilde{y}$$

and transform back to unscaled coefficients

$$\hat{\beta}_\lambda = \text{diag}(1/\text{s})\tilde{\beta}_\lambda, \hat{\alpha} = \bar{y} - \hat{x}^t\hat{\beta}_\lambda$$

3.2 k-fold Cross Validation

For this task, to select λ from a set of candidate values, I will use k-fold cross-validation, specifically the 5-fold. In k-fold cross-validation, the original sample is randomly divided into k equal subsamples, often referred to as "folds". Of the k subsamples, a single subsample is retained as validation data to test the model, and the remaining k-1 subsamples are used as training data. The cross-validation process is then repeated k times, with each of the k subsamples used exactly once as the validation data. The k results can then be averaged to produce a single estimate. The advantage of this method over repeated random subsampling is that all observations are used for both training and validation, and each observation is used exactly once for validation.

4 The project

Let's start with the importing of the necessary libraries, including pandas for data manipulation and scikit-learn for machine learning tools.

```
import pandas as pd
import numpy as np
from numpy import mean
from numpy import absolute
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
```

The .csv file is imported and cleaned by removing nulls and duplicates from the dataset and unnecessary columns for analysis. Additionally, an average popularity variable is created as previously mentioned.

```
df = df.drop(['Unnamed: 0', 'album_name', 'track_name'], axis=1)
df=df.dropna().drop_duplicates(subset=['track_id']).set_index('track_id')
df['avg_artist_pop'] =
 ↪  df.groupby('artists')['popularity'].transform('mean')
```

4.1 Numerical variables

For the initial stage of the project, I partitioned the dataset into a training set and a test set. The test set comprises approximately 20% of data.

The selected predictors are the variables *"avg_artist_pop", "danceability", "tempo", "energy", "valence", "key", "loudness", "mode", "speechiness", "acousticness", "instrumentalness" and "liveness"*, while the target variable is obviously *"popularity"*.

The ridge_fit function takes a training dataset (train), a list of predictor variables (predictors), a target variable (target), and a regularization hyperparameter λ, which is referred to as alpha in the code.

The key steps involved in the implementation include:

1. Data preprocessing:
   - The target variable y is mean-centered to have a zero mean.
   - The predictor variables in X are standardized to have zero mean and unit variance.
2. Ridge Penalty Term:
   - A penalty term is introduced using the regularization hyperparameter alpha, multiplying it to a identity matrix.
3. Calculating Ridge Regression Coefficients:
   - The Ridge Regression coefficients B are computed using matrix operations: $\tilde{\beta}_\lambda = (\tilde{X}^t\tilde{X} + \lambda I_p)^{-1}\tilde{X}^t\tilde{y}$.
   - The coefficients are then transformed back to their original scale using a diagonal matrix based on the standard deviation of the predictor variables: $\hat{\beta}_\lambda = \text{diag}(1/s)\tilde{\beta}_\lambda$.

This function gives as results $\hat{\beta}$, that will be used as a parameter for the next function.

```
def ridge_fit(train, predictors, target, alpha):
    X = train[predictors].copy()
    y = train[[target]].copy()

    y_mean = y.mean()
    y = y - y_mean

    x_mean = X.mean()
    x_std = X.std()

    X = (X - x_mean)/x_std

    penalty = alpha * np.identity(X.shape[1])

    B = np.linalg.inv(X.T @ X + penalty) @X.T @ y
    diagonal_matrix = np.diag(1/(x_std))
    beta_hat = diagonal_matrix @ B
    beta_hat.index =  ["avg_artist_pop","danceability", "energy", "key",
    ↪  "loudness", "mode", "speechiness", "acousticness",
    ↪  "instrumentalness", "liveness",
            "valence", "tempo"]

    return beta_hat, x_mean, x_std
```

The second function built is ridge_predict, which is used to make predictions using a previously trained ridge regression model.
This function takes as input a test data set (test), a list of predictor variables (predictors), and the ridge regression coefficients ($\hat{\beta}$).
It then returns predictions based on the input test data, using matrix multiplication between test_X and beta_hat.

```
def ridge_predict(test, predictors, x_mean, x_std, beta_hat):
    test_X = test[predictors]

    predictions = test_X @ beta_hat
    return predictions
```

The last function creates a dictionary of λ values to compare within the model in order to chose the best one: the range goes from very small numbers to a value over a billion.

```
from sklearn.metrics import mean_absolute_error

errors_dict = {}

alphas = [20**i for i in range(-8, 8)]
```

Hyperparameter tuning is a crucial step in the development of predictive models, as it involves selecting the appropriate settings for parameters that are not learned from the

data but are essential for model performance.
It's now time to implement the 5-fold ridge regression on the model: the code iterates through a range of alpha values, performing 5-fold cross-validation.
Within each fold, a Ridge Regression model is trained on the training data (train_fold) and evaluated on the validation data (val_fold). In our case, the best performing one was $\lambda = 400$, with the following metrics:

```
Best Alpha (MAE): 400
Errors for Best Alpha (MAE): [5.251772384718319, 5.21887513187134, 5.2461
86689705342, 5.197186328559765, 5.306344167451345]

Best Alpha (MSE): 400
Errors for Best Alpha (MSE): [102.88583800890424, 103.86364030331202, 10
4.91788491986789, 105.35896367221521, 106.73641813625946]

Best Alpha (RMSE): 400
Errors for Best Alpha (RMSE): [10.143265648148246, 10.191351250119487, 1
0.242943176639606, 10.264451455007968, 10.33133186652425]

Best Alpha (R²): 400
Errors for Best Alpha (R²): [0.7586924199092713, 0.7535161684694536, 0.75
1277867148691, 0.7524150998800646, 0.7508326350458296]
```

Figure 1: Error metrics values for $\lambda = 400$

Let's also take a look at the average error metrics per each λ when performing the 5-fold cross-validation for comparison.

	Alpha	MAE	MSE	RMSE	R2
0	3.906250e-11	5.284530	104.814448	10.237701	0.753201
1	7.812500e-10	5.284530	104.814448	10.237701	0.753201
2	1.562500e-08	5.284530	104.814448	10.237701	0.753201
3	3.125000e-07	5.284530	104.814448	10.237701	0.753201
4	6.250000e-06	5.284530	104.814448	10.237701	0.753201
5	1.250000e-04	5.284530	104.814448	10.237701	0.753201
6	2.500000e-03	5.284529	104.814447	10.237701	0.753201
7	5.000000e-02	5.284520	104.814428	10.237700	0.753201
8	1.000000e+00	5.284339	104.814050	10.237682	0.753202
9	2.000000e+01	5.280766	104.806718	10.237323	0.753219
10	4.000000e+02	5.244073	104.752549	10.234669	0.753347
11	8.000000e+03	7.538760	130.076541	11.404908	0.693720
12	1.600000e+05	25.627249	903.154324	30.052478	-1.126692
13	3.200000e+06	32.692672	1479.537497	38.464661	-2.483920
14	6.400000e+07	33.195256	1525.918789	39.062913	-2.593135
15	1.280000e+09	33.220952	1528.310997	39.093520	-2.598767

Figure 2: Average metrics per each λ

I have tested a wide range of alpha values, ranging from very small values (e.g., 3.906250e-11) to very large values (e.g., 1.280000e+09).
The values of MAE, MSE, and RMSE appear to be relatively stable and don't change significantly across different alpha values until 8000.
As alpha increases, the model's complexity decreases.
Higher alpha values lead to more substantial regularization, which can shrink coefficients towards zero.

With a high alpha value, the estimated regression coefficients are null (approximately null).
Therefore, the estimated values of y are also null because the intercept is not present.
Consequently, the mean error is the difference between the mean of y and the mean of the predictions (MAE), which is 33 - 0 = 33.
With the highest penalty, all beta values are driven to 0.

Finally, it's time to use the best alpha value to retrain the ridge regression model on the entire training set and evaluate it on your test set to assess its generalization performance.

	Test Set MAE	Test Set MSE	Test Set RMSE	Test Set R^2
$\lambda = 400$	5.066	98.228	9.911	0.765

Figure 3: Model performance

This means that, on average, the predicted results are 5.066 different from the actual results. Considering that we are working on a 0-100 scale, I feel like this result can be considered quite good.

Let's take a look at the $\hat{\beta}$ coefficients for this model:

	popularity
avg_artist_pop	0.992083
danceability	0.887614
energy	-0.046691
key	0.002503
loudness	-0.039834
mode	-0.122663
speechiness	-0.647720
acousticness	-0.405512
instrumentalness	-0.579143
liveness	-0.376141
valence	-0.059867
tempo	-0.000135

Figure 4: $\hat{\beta}$ coefficients

It looks like the danceability of the song, the song's key and the artist popularity have an high positive impact in a song's popularity, while the other parameters contribute negatively to the popularity predicted score.

4.2 Categorical variables model

For the second part, I have incorporated categorical variables into the model. I used the pandas function 'get_dummies()' to perform one-hot encoding on the relevant categorical columns for our research, namely track genre and explicit. One-hot encoding is a technique that represents categorical data as binary vectors. It involves representing each category with a binary vector where only one bit is set to '1' to indicate the presence of that category, while all other bits are set to '0'. One-hot encoding is a technique used in machine learning models to represent categorical data. The pandas library provides a convenient function called 'get_dummies()' to perform one-hot encoding on categorical variables. The function takes a dataframe with categorical columns and generates new binary columns for each unique category in those columns. The original categorical column is then substituted by these binary columns, which indicate the presence or absence

of each category. This procedure converts categorical data into a numerical format that is appropriate for machine learning models. This function creates a column for each musical genre with a value of 1 if the song corresponds to the column title's genre and 0 if it does not.

I divided again the dataset into train and test set, the latter one containing around 20% of data.

As predictors, I chose the variables *"avg_artist_pop", "danceability", "tempo", "energy", "valence", "key", "loudness", "mode", "speechiness", "acousticness", "instrumentalness","liveness", and the newely added dummies "track_genre_..." and "explicit_False"/"explicit_True"*, while the target variable is obviously *"popularity"*.

The three functions I've used are the same as before, with the only difference in the predictors chosen into the function.

It's now time to implement again the 5-fold ridge regression on the model.

In our case, the best performing one was $\lambda = 7.8125e-10$, with the following metrics:

```
Best Alpha (MAE): 7.8125e-10
Errors for Best Alpha (MAE): [5.380497212175437, 5.286984868987192, 5.297
8951844930595, 5.34944542845412, 5.417509813352854]

Best Alpha (MSE): 7.8125e-10
Errors for Best Alpha (MSE): [102.88753217642503, 103.0928455899026, 104.
29996512685459, 105.67133183101949, 105.97175173605717]

Best Alpha (RMSE): 7.8125e-10
Errors for Best Alpha (RMSE): [10.143349159741325, 10.153464708655001, 1
0.212735438013391, 10.279656211713478, 10.29425819260704]

Best Alpha (R²): 7.8125e-10
Errors for Best Alpha (R²): [0.7586884464230013, 0.7553453787082804, 0.75
27427301600517, 0.7516810603954052, 0.7526176856905324]
```

Figure 5: Error metrics values for λ = 7.8125e-10

Let's also take a look at the average error metrics per each λ when performing the 5-fold cross-validation.

	Alpha	MAE	MSE	RMSE	R2
0	3.906250e-11	5.777822	106.585980	10.323974	0.749018
1	7.812500e-10	5.346467	104.384685	10.216693	0.754215
2	1.562500e-08	5.350976	104.415461	10.218190	0.754143
3	3.125000e-07	5.351023	104.415597	10.218196	0.754143
4	6.250000e-06	5.351043	104.415743	10.218204	0.754143
5	1.250000e-04	5.351043	104.415747	10.218204	0.754143
6	2.500000e-03	5.351044	104.415750	10.218204	0.754143
7	5.000000e-02	5.351052	104.415807	10.218207	0.754142
8	1.000000e+00	5.351222	104.416945	10.218262	0.754140
9	2.000000e+01	5.354686	104.440194	10.219399	0.754085
10	4.000000e+02	5.449648	105.089035	10.251085	0.752559
11	8.000000e+03	9.290449	158.843205	12.602991	0.626005
12	1.600000e+05	26.723292	939.076644	30.644307	-1.211263
13	3.200000e+06	32.751480	1479.083165	38.458755	-2.482850
14	6.400000e+07	33.198152	1525.877410	39.062383	-2.593037
15	1.280000e+09	33.221096	1528.308878	39.093493	-2.598762

Figure 6: Average metrics per each λ

It doesn't seem to change much from the non-categorical variables model, probably due to the fact that there are more than a hundred genres and therefore those are parameters than don't have a big impact on evaluating whether a song will be popular or not.

Finally, it's time to use the best alpha value to retrain the ridge regression model on the entire training set and evaluate it on your test set to assess its generalization performance.

	Test Set MAE	Test Set MSE	Test Set RMSE	Test Set R^2
$\lambda = 7.8125e\text{-}10$	5.172	97.697	9.884	0.766

Figure 7: Model performance

This means that, on average, the predicted results are 5.172 different from the actual results. Considering that we are working on a 0-100 scale, I feel like this result can be considered quite good, very similar to the other model.

Considering that the model without categories is more light to compute, it is wiser to use that instead of the second one, which is more expensive in computational terms with very similar results.

5 Conclusions

I have developed a Ridge model that can predict a song's success with reasonable accuracy, with an average difference of only 5 points from the actual success of the track.

However, it is important to note that there may be a musical component that is difficult to quantify, and the success of a song can sometimes be unpredictable and go beyond any measurable parameter. From an objective perspective, the results obtained by our two Ridge regression models are satisfactory, considering the size of the database. However, it cannot be compared to the entire discography available on Spotify.

Furthermore, the introduction of the artist's popularity parameter assumes that an emerging artist has little chance of their songs going viral. However, history has shown that this is not always the case.

References

[1] Abby McCain. *30+ 2023 Harmonious Music Industry Statistic*. URL: https://www.zippia.com/advice/music-industry-statistics/#:~:text=The%20music%20industry%20is%20worth,2021%20earnings%20of%20%2415%20billion.

[2] Maharshi Pandya. *Spotify Tracks Dataset*. 2022. DOI: 10.34740/KAGGLE/DSV/4372070. URL: https://www.kaggle.com/dsv/4372070.

[3] Aldo Solari. *Ridge Regression (Statistical Learning - CLAMSES, University of Milano - Bicocca)*. URL: https://aldosolari.github.io/SL/docs/SLIDES/Ridge.pdf.