Spotify's Tracks Popularity Prediction With Ridge Regression

Luca Annese
Matriculation number: 18805A

1 Introduction

The report describes my experimental project for the course in Statistical Methods for Machine Learning. The chosen project requires implementing the Ridge Regression algorithm from scratch, without the help of outer libraries, to predict the popularity of musical tracks from the music application Spotify, using the data set available from Kaggle.com.

In particular, the project requires implementing the Ridge Regression algorithm and then testing its performance on the data set both including categorical labels as well excluding them, thus using only numerical features. Moreover, to validate the produced model, alias to compute the risk estimates and find the best hyperparameter, 5-fold Cross Validation is performed in both cases. When predicting with categorical features included, the project asks to choose the correct encoding method.

The code is all written in Python using Jupiter Notebook. The employed libraries for computation and visualization are: numpy, pyplot, seaborn, pandas.

2 The Ridge Regression algorithm

The Ridge Regression algorithm tries to solve the limitations of a linear model, by introducing a regularization term in the cost function preventing the coefficients of the predictor taking too large values. In classic linear regression the predictor is a linear function $h : \mathbb{R} \to \mathbb{R}$ parameterized by a vector in $\mathbb{R}^d$ of real coefficients that is:

$$h(x) = w^T x \tag{1}$$

Given a training set $(x_1, y_1) \dots (x_m y_m)$, the linear regression predictor w is Empirical Risk Minimization (ERM) with respect to the square loss:

$$w = \sum_{i=1}^{m} (w^T x_i - y_i)^2 \tag{2}$$

If we call S the mxn design matrix whose rows are the samples and the columns detect the features, we can explicit the cost function as follows:

$$F(w) = ||Sw - y||^2 \tag{3}$$

Since the function is a convex function, the minimum exists and satisfies the condition $\nabla F(w) = 0$, and can be directly computed as:

$$w = (S^T S)^{-1} S^T y \tag{4}$$

This solution is only available if the quantity $(S^T S)$ is invertible. In Ridge Regression the cost function is modified to stabilize this quantity and to penalize larger coefficients of w, as follows:

$$||Sw - y||^2 \alpha ||w||^2 \tag{5}$$

where $\alpha > 0$ is the regularization term. When $\alpha \to 0$ we recover the standard regression solution, instead, if $\alpha \to$ inf the solution becomes the zero vector. This way, we inject the solution with bias, thus we increase the approximation error, while lowering the variance. In fact, in the latter case the solution has zero variance, and the regression will yield the same solution, the zero vector, for any given data set. If we, as before, compute the minimum of the regularized cost function, by nullifying its gradient, we obtain the the following solution:

$$w = (S^T S - \alpha I)^{-1} S^T y \tag{6}$$

2.1 Python implementation

My implementation of the algorithm, in particular considers the closed formula, consisting of the *RidgeRegression* class. The class has three attributes: the penalization term *alpha*, given as input in the initialization, the *intercept* and the *coefficients* of the solution.

The class includes the *fit(X, y)* method which given a training set X and the corresponding y vector of target values, computes the predictor with (6). Before the computation I add the *intercept* column to the training set, initialized to 1. At the end, this column will contain the value of the intercept, or the bias. Then, I compute the penalty matrix whose value in position [0][0] is set to 0, to avoid penalizing the bias term. After the closed formula, I design the *weights* attribute to be more coherent with the other data set employed, which all are pandas DataFrames. The (1) is encoded by the *predict* method, whose input is a data set to which is added an *intercept* column, if not already present. The accuracy of the model is computed with via the methods *r2Score(self, target, predicted)* and *mseScore(self, target, predicted)*. The first method compute the $R^2 score$, which determines the proportion of variance in the dependent variable that can be explained by the independent variable. In other words, R^2 shows how well the data fit the regression model. The other measure, the *root mean square error*, shows how far predictions fall from measured true values using Euclidean distance.

```
class RidgeRegression:

    alpha = None
    intercept = None
    weights = []

    def __init__(self, alpha):
        self.alpha = alpha

    def fit(self, X, y):
        if "intercept" not in X:
            X.insert(0,"intercept",1, True)

        I = np.eye(X.shape[1])
        penaltyMat = self.alpha * I
        penaltyMat[0][0] = 0

        w = (np.linalg.inv(X.T @ X + penaltyMat) @ X.T) @ y

        w.index = list(X.columns)
        self.weights = w
        self.intercept = w.loc["intercept"]

    def predict(self, X):
        if "intercept" not in X:
            X.insert(0,"intercept",1, True)

        predictions = X @ self.weights
        return predictions

    def r2Score(self, target, predicted):
        return r2_score(target, predicted)

    def mseScore(self, target, predicted):
        return mean_squared_error(target, predicted,squared=False)
```

3 The data set

The employed data set is freely available from Kaggle.com. This data set contains 114k Spotify tracks, each described by a set of numerical features such as:

- **popularity** of the track, defined as an integer from 0 to 100, with 100 being the most popular. This feature is the target value.
- **duration_ms**: Track length in milliseconds.
- **explicit**: *True* when the track's lyrics is explicit, otherwise *False*.
- **danceability** describes how suitable a track is for dancing on a combination of musical elements. The value ranges from 0.0 to 1.0, 1.0 being the highest danceability.
- **energy**: Measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity.
- **key** of the track. If no key was detected, the value is -1.
- **loudness**: The overall loudness of a track in decibels (dB).
- **mode** indicates the modality (major or minor) of a track. Major is represented by 1 and minor is 0.
- **speechiness** detects the presence of spoken words in a track. Value is 1.0 for more speech-like content, and decreases to 0.0 for music or non-speech-like tracks.
- **acousticness** ranges from 0.0 to 1.0, being 1.0 if the track is acoustic.
- **instrumentalness** predicts whether a track contains no vocals. The closer the instrumentalness value is to 1.0, the greater the likelihood that the track contains no vocal content.
- **liveness** detects the presence of an audience in the recording. A value above 0.8 provides strong likelihood that the track is live.
- **valence** is a measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track.
- **tempo** of a track in beats per minute (BPM).
- **time_signature** specifies how many beats there are in each bar. The time signature ranges from 3 to 7 indicating time signatures of 3/4, to 7/4.

And a set of categorical features as well:

- **track_id** given by Spotify.
- **track_genre**: The musical genre of the track.
- **artists** that perform the track. If multiple, all the names are present.
- **album_name** in which the track appears.
- **track_name**: The title of the track.

3.1 Preprocessing

From the data set I can immediately drop the two columns: *track_id* and *Unnamed: 0*, since these two are indexing columns, without any relevant information for regression. To select which features to use in the model, I opted to observe the correlation coefficients between the target value, *popularity*, and the other columns. In doing so, I obtain the following coefficients:

Features	Correlation with popularity
duration_ms	0.008371
explicit	0.031007
danceability	0.019201
energy	0.010136
key	-0.006132
loudness	0.061858
mode	-0.009105
speechiness	-0.052703
acousticness	-0.029847
instrumentalness	-0.096086
liveness	0.007376
valence	-0.053841
tempo	0.023219
time_signature	0.02971

Table 1: Correlations coefficients of numerical values with respect to the target variable.

Notice that the numerical features all have low correlation value with respect to the popularity. This makes sense, since these acoustic features are not usually what people look for in choosing a song. I decided not to consider any feature whose absolute value of the correlation was ≤ 0.001: *duration_ms*, *key*, *mode* and *liveness* were dropped due to this decision. To measure the correlation of the categorical features, I had to perform an encoding. The encoding technique I chose is Target Encoding (TE), because more classic encoding methods like One-Hot-Encoding (OHE) were not suitable for this data set due to the high cardinality of unique values, which would have caused adding a lot of new columns for these values. On the other hand, TE substitutes the categorical columns with an encoded one, so the number of features does not change. The encoding is performed as follows: for each category, compute the mean of the target variable for all observations in that category and blend it with the mean of the target variable. In particular, I opted for the Leave-One-Out (LOO) version, to handle the amount of unique categories (Table 2).

Features	Uniques
artists	31438
album_name	46590
track_name	73609

Table 2: Number of unique values in the categorical features.

LOO works the same as TE, but excludes the current observation from the computation. This way I avoid overfitting, since by including the current observation every unique category would have a different and very specific encoded value, resulting in low generalization. The encoding is performed with the *category_encoders* library and the *LeaveOneOutEncoder* class. To study the correlation between the categorical features and the target variable *popularity*, I applied LOO encoding, and, thus, obtained the following heatmap:

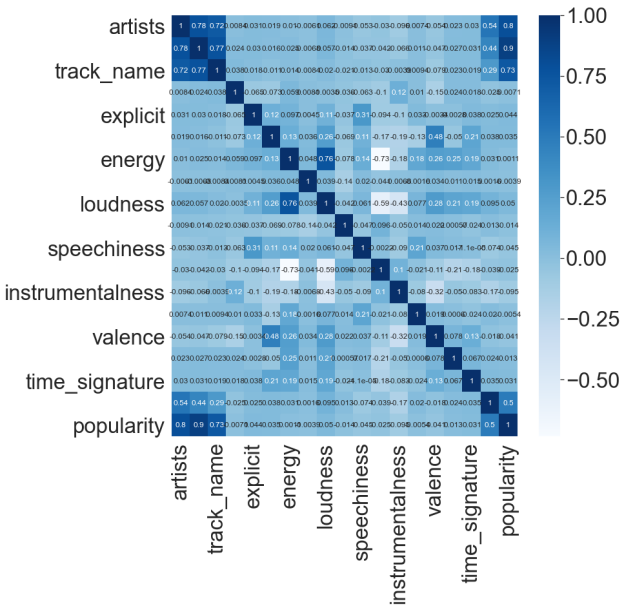

Figure 1: Cross-correlation heatmap of all features.

As you can see in Figure 1, *artists*, *album_name* and *track_name* show the highest correlation between each other, as well as with *popularity*. This result makes sense because when a new song is released the most determining factors of whether it will be popular, are the following three: it is clear that people tend to listen to songs popular artists more from. So if an artist who already has successfully released a song, makes a new song, it is probably going to be successful as well. Moreover, if a new song gets released with the same *track_name* of a popular song, its popularity is boosted. However, sharing the same *album_name* is not enough to gain popularity. Therefore, I decided to drop the *album_name* column, even though, due to its high correlation (0.9) with the target variable, this is expected to lead to worse results training-wise. Dropping this column allows the model to be more general and avoid creating wrong biases while training. During the training, I also performed a Z-score normalization of the feature variables: subtract from each column its mean, and divide it by the standard deviation. The normalization on the validation set was done with the mean and standard deviation values from the the training part. Even the encoder used for the validation set was fitted with the values from the training set, to avoid leakage of the target variable into the features variables.

4 Regression with categorical features

Firstly, I run the regression model defined in Section 2 considering both numerical and categorical features. To do so, I defined the *data* data set according to the above listed conditions: by dropping the *album_name* and the other poorly correlated features. Then I implemented the code for *5-fold Cross Correlation* and repeated it fifty times, by varying the Ridge Regression parameter α in the range from 0 to 5000, with a stepsize of 100. The following code shows my implementation:

```
target = "popularity" #target feature
encoder = ce.LeaveOneOutEncoder()
catCol = ["artists", "track_name", "explicit", "track_genre"]

k = 5
k_Fold=KFold(n_splits=k,shuffle=True)

MSEs = []
R2s = []

for alpha in np.arange(0, 5000, 100):

    valR2Scores = []
    valMSE = []
    w = RidgeRegression(alpha)


    for trainI, testI in  k_Fold.split(data):
        #Split i-th fold into training and validation set
        train = data.loc[trainI]
        test = data.loc[testI]
        #Separate target variable from feature variables
        train_X = train.loc[:, train.columns != target]
        train_y = train[target].astype(np.int64)
        val_X = test.loc[:, test.columns != target]
        val_y = test[target].astype(np.int64)

        #Encoding of categorical features
        enc = encoder.fit(train_X, train_y)
        train_X = enc.transform(train_X)
        val_X = enc.transform(val_X)

        #Normalize: mean=0, std=1
        mean_X = train_X.mean()
        std_X = train_X.std()

        train_X = (train_X - mean_X) / std_X
        val_X = (val_X - mean_X) / std_X

        train_X = train_X.astype(np.int64)
        val_X = val_X.astype(np.int64)
        train_y = train_y.astype(np.int64)
        val_y = val_y.astype(np.int64)
        #Compute predicto
        w.fit(train_X, train_y)

        #Evaluate fold performance
        predictions = w.predict(val_X)

        rescaledValMSE = w.mseScore(val_y,predictions)
        rescaledValR2Score = w.r2Score(val_y,predictions)
        valMSE.append(rescaledValMSE)
```

```
        valR2Scores.append(rescaledValR2Score)

    estimMSE = sum(valMSE) / k
    estimR2Score =sum(valR2Scores) / k
    MSEs.append(estimMSE)
    R2s.append(estimR2Score)
```

In doing so I obtained the following results:

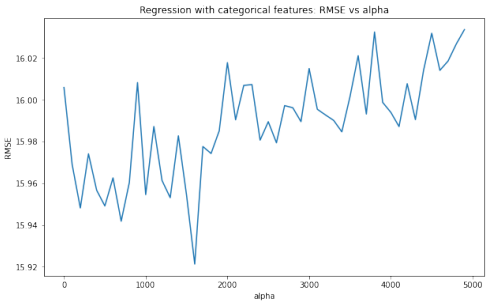

Figure 2: RMSE vs alpha

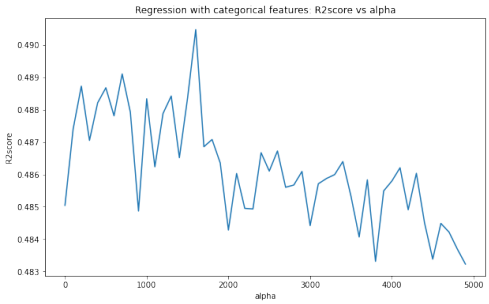

Figure 3: R2score vs alpha

The above graphs show what I suspected before: considering the categorical features in the model yield for better results. This result is foreseeable since the correlations of these features with the target variable are the highest among the other features, so, employing them, gives more predictive power to the model. In this case, the best value of the parameter α, thus the one that minimizes the *RMSE* and maximizes the *R2score* (in the considered range of values), is $\alpha = 1600$, that yields $RMSE = 15.92131$ and $R2score = 0.49047$

5 Regression with only numerical features

Now, I run the same model of the preceding section considering only numerical features. To do so, I follow the same methodology as before, without the encoding lines: I run a *5-fold Cross Correlation* for each value in the same range, and compute its performance. The following code shows my implementation:

```
k = 5
k_Fold=KFold(n_splits=k,shuffle=True)

MSEsnum = []
R2snum = []

for alpha in np.arange(0, 5000, 100):

    valR2Scores = []
    valMSE = []
    w = RidgeRegression(alpha)
    for trainI, testI in k_Fold.split(data):
        #Split i-th fold into training and validation set
        train = data.T[trainI].T
        test = data.T[testI].T
        #Separate target variable from feature variables
        train_X = train.loc[:, train.columns != target]
        train_y = train[target].astype(np.int64)
        val_X = test.loc[:, test.columns != target]
        val_y = test[target].astype(np.int64)

        #Normalize: mean=0, std=1
        mean_X = train_X.mean()
        std_X = train_X.std()

        train_X = np.divide(np.add(train_X, -mean_X), std_X)
        val_X = np.divide(np.add(val_X, -mean_X), std_X)

        train_X = train_X.astype(np.int64)
        val_X = val_X.astype(np.int64)

        #Compute predictor
        w.fit(train_X, train_y)

        #Evaluate fold performance
        predictions = w.predict(val_X)

        rescaledValMSE = w.mseScore(val_y,predictions)
        rescaledValR2Score = w.r2Score(val_y,predictions)
        valMSE.append(rescaledValMSE)
        valR2Scores.append(rescaledValR2Score)


    estimMSE = sum(valMSE) / k
    estimR2Score =sum(valR2Scores) / k
    MSEsnum.append(estimMSE)
    R2snum.append(estimR2Score)
```

By doing so, I obtained the following results:

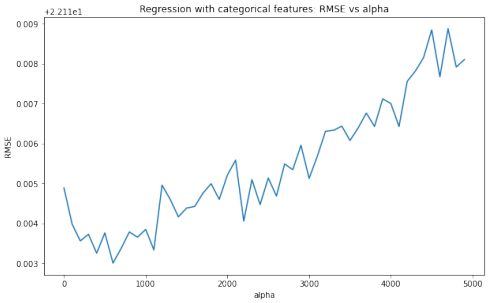

Figure 4: RMSE vs alpha

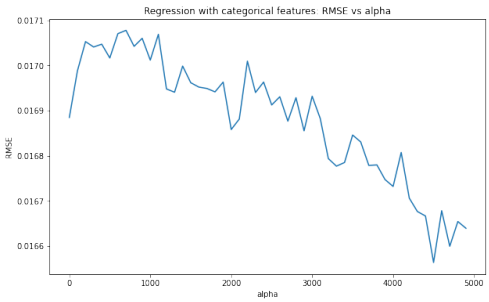

Figure 5: R2score vs alpha

Observe that the best value for α, thus, the one that is minimizing the $RMSE$ and maximizing the $R2score$, is about $\alpha = 800$, which yields $RMSE = 22.11337$ and $R2score = 0.01707$. A closer look displays that the best value of α lies in the range from 700 to 800, but, due to the fact that the differences lie in the range of 10^{-4}, I concluded that there is no effective difference in choosing 700 over 800. These results show what I discussed before, that the numerical features, with respect to their correlation with the target variable, are not good predictors for the popularity of a track. In fact, a $R2score$ so low, shows that the model is not capable of explaining the variability of the target variable.

6 Conclusion

In conclusion, the results show that when training with numerical features only, I obtained that the best value for α lies in the range between 700 to 800. I decided that more investigations were not needed because of the poor performances, with respect to the $RMSE = 22.11337$ and $R2score = 0.01707$, reflect what the correlation analysis foreshadowed: numerical features were not able to explain the variability of the target variable, in particular the *R2* value shows that the model explain approximately 2% of the variance of the dependent variable. On the other hand when employing categorical features in the training, the model performed better, as the correlation study highlighted. In this case the best value for α was 1600, yielding $RMSE = 15.92131$ and $R2score = 0.49047$. The much higher value of *R2* shows that, this time, the model is able to explain half of the variability of the target variable.

Disclaimer

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.