Ridge Regression of Spotify popularity

Fumagalli Andrea

October 31, 2023

1 Introduction

This is a project developed for the Statistical Methods for Machine Learning course in the Computer Science Master Degree at Università degli Studi di Milano.

1.1 Objective

The request is to download the Spotify Tracks Dataset and perform Ridge Regression on it to predict the popularity of a track and then use 5-fold cross validation to compute the risk estimates of the model. The guidelines for the project ask to firstly use only the numerical features to train the model and only after to take in consideration also the categorical ones, so the development and also this paper follow this structure.

1.2 Dataset details

The data set contains 114k records organized in twenty columns:

- **track_id**
- **artists**
- **album_name**
- **track_name**
- **popularity**: The popularity of a track is a value between 0 and 100, with 100 being the most popular.
- **duration_ms**
- **explicit**: Whether or not the track has explicit lyrics (true = yes it does; false = no it does not OR unknown).
- **danceability**: how suitable a track is for dancing. A value of 0.0 is least danceable and 1.0 is most danceable.
- **energy**: Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity.
- **key**: The key the track is in. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on. If no key was detected, the value is -1.
- **loudness**: The overall loudness of a track in decibels (dB).
- **mode**: Mode indicates the modality (major or minor) of a track. Major is represented by 1 and minor is 0.
- **speechiness**: Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value.
- **acousticness**: A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
- **instrumentalness**: Predicts whether a track contains no vocals. The closer the instrumentalness value is to 1.0, the greater likelihood 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**: A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).

- **tempo**: The overall estimated tempo of a track in beats per minute (BPM).
- **time_signature**: An estimated time signature. It is a convention to specify how many beats are in each bar. The time signature ranges from 3 to 7 indicating time signatures of 3/4, to 7/4.
- **track_genre**

2 Theory

2.1 Ridge Regression

Ridge Regression is a regularized form of linear regression where the predictor is defined by

$$\omega_{S,\alpha} = \underset{\omega \in \mathbb{R}^d}{\operatorname{argmin}} ||S\omega - y||^2 + \alpha||\omega||^2$$

where $\alpha > 0$ is the regularization parameter used to control the bias of the algorithm. When $\alpha \to 0$ we have the standard linear regression, while when $\alpha \to \infty$ $\omega_{S,\alpha}$ become the zero vector. Using this definition the gradient vanishes for

$$\omega = \omega_{S,\alpha} = (\alpha I + S^\top S)^{-1} S^\top y$$

this closed form formula is what we are going to implement to obtain the regression.

2.2 Cross validation

Cross validation is a validation technique. It consists in partitioning the dataset S into K subsets $S_1, ..., S_k$, at each step one of these subsets S_i will be the test set while the others S_{-1} will be the training set. Using k-fold cross validation we can estimate the expected risk of a predictor with respect of the random draw of the training set. The estimate is obtained by computing the error on the testing part of each fold an then averaging these errors.

$$\mathbb{E}[l_D(A)] = \frac{1}{K} \sum_{i=1}^{K} \frac{K}{m} \sum_{(x,y) \in S_i}^{K} l_{S_i}(h_i)$$

3 Implementation

The project has been implemented in Phyton using Jupyter Notebook

3.1 Ridge Regression

To implement the Ridge Regression according to the theory discussed before we created two functions:

- **ridge_fit**

```
def ridge_fit(dataset, predictors_features, target_feature, alpha):
            S = dataset[predictors_features].copy()
            y = dataset[[target_feature]].copy()

            s_mean = S.mean()
            s_std = S.std()

            S = (S - s_mean) / s_std      #normalization
            X['intercept'] = 1
            X = X[['intercept'] + predictors]


            penalty = alpha * np.identity(S.shape[1])
            penalty[0][0] = 0
            w = np.linalg.inv(S.T @ X + penalty) @ S.T @ y
            return w, S_mean, S_std
```

This function gets as input a data set, a set of predictors, a target feature and the alpha parameter. Firstly it normalizes the data set keeping track of the data set mean and standard deviation(std), then it adds to it the intercept column of all 1. In the end it builds the predictor vector w using the formula $\omega_{S,\alpha} = (\alpha I + S^\top S)^{-1} S^\top y$ It outputs the predictor and also the data set's mean and std to use to normalize the future test set.

- **ridge_predict**

```
def ridge_predict(dataset, predictors_features, trainset_mean, trainset_std, w):
    testset = dataset[predictors_features]
    testset = (testset - trainset_mean) / trainset_std
    testset['intercept'] = 1
    testset = testset[['intercept'] + predictors_features]

    predictions =  testset @ w
    return predictions
```

This function get as inputs a data set, a set of predictors, the training set mean and std and the predictor vector. It normalizes the data set using the mean and the std given, it add the intercept column and then compute the predictions multiplying the data set for the predictor. The function outputs the array of predictions.

3.2 Cross validation

To partition the dataset we used the `KFolf` function from the `sklearn.model_selection` package and then we implemented the formula discussed before to estimate the risk of the model

```
from sklearn.model_selection import KFold
kf = KFold(n_splits = 5, shuffle = True)
loss = 0
for train_index, test_index in kf.split(dataset_numeric):
    train, test = (dataset_numeric.iloc[train_index]), (dataset_numeric.iloc[test_index])
    w, mean, std = ridge_fit(train, predictors, target, 180)
```

```
        prediction = ridge_predict(test, predictors, mean, std, w)
        loss_i = (5/dataset_numeric.shape[0]) *
            sum(pow((prediction.popularity - test.popularity),2))
        loss = loss + loss_i
    cv = 1/5 * loss
    print(pow(cv, 1/2))
```

We partition the dataset in 5 folds as requested. We used the quadratic loss as loss function so in the end we root the result.

4 Regression on numerical features

As requested we firstly performed the regression considering only the numerical features. These being *duration_ms, danceability, energy, loudness, speechiness, acousticness, instrumentalness, liveness, valence and tempo.*On top of these 10 we kept obviously the target feature (*popularity*) and also *track_id* to use during preprocessing as an index.

4.1 Preprocessing

Because the feature are all numeric we have no encoding to do. The only operation we had to do was due to the fact that some records were exactly identical except for the popularity score. This is something the model cannot learn so we decided to keep only the record with the higher popularity score.

4.2 Hyperparametrs tuning

The only hyperparameter to optimise in this case was alpha. To find the best value we used cross validation passing different values of alpha to the `ridge_fit` function. We repeated this process on different sets of values every time more and more little focusing on the subset that minimized the risk. Here the results:

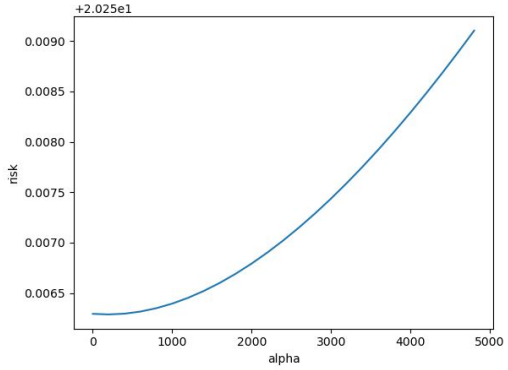

Figure 1: risk with alpha in range 1-5000

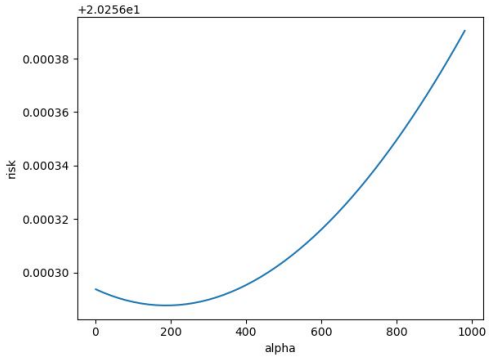

Figure 2: risk with alpha in range 1-1000

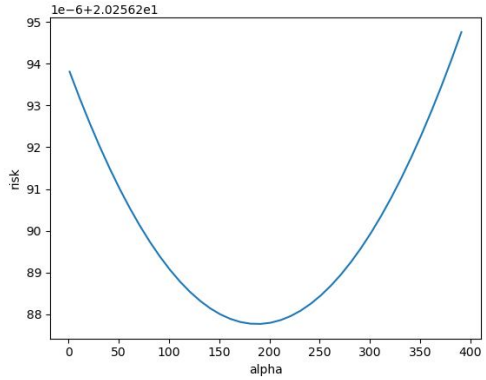

Figure 3: risk with alpha in range 1-400

As we can see see that the optimal value for alpha is 180. So we used this value to estimate the risk of our model.

4.3 Results

As result of the 5-fold cross validation, using the squared loss we got a rooted risk of 20.25566660656646. Given the fact that our target (popularity) has a range of 0 to 100 the result is not great, but neither atrocious. This tell us that using only numerical features we cannot predict accurately the popularity of a song, we can only say if a song is very popular moderately popular or not popular if the predicted popularity is respectively in range (100-70, 70-40, 40-0).

5 Regression on all features

5.1 Preprocessing

This time the preprocessing required some more work: we encoded the *key* and the *time_signature* features using the `OneHotEncoder` from the `sklearn.preproccesing` package.
For the *track_genre* feature we did something a little different, firstly we used the one hot encoder, but then due to the fact that the same song(same *track_id*) was present in the dataset more than once, one for every genre associated with it, we grouped the records for *track_id* and we summed the newly generated features so we unified every record for every genre of a song in one record.
Analyzing further the data we noticed that the same song (this time same *track_name* and *artists*) was present in the dataset multiple times with different *album_name* and *popularity*. This is probably due to the fact that a song has been published in some repacks or compilations, the problem is that this behavior is difficult to learn for our model because the popularity score was often very different (often the difference was bigger than 60 points) and also because the *album_name* feature that differentiate these records is a string so it is difficult to encode. So we decided to get rid of this feature and to keep only the records with the higher popularity score.
The last two categorical features to encode are *artists* and *track_name*. This features have a high cardinality so it is not possible to use one hot encoding. Instead we opted for target encoding which associate to every categorical value a measurement of the effect it might has on the target, usually a mean with some prior smoothing of the target feature of the records with that value. This solution is very powerful, but it has some drawbacks that we need to mention: target encoding works perfectly for our construed case, but if we consider a real life purpose of the model we are building that could be to classify new songs that don't belong to the database already, this means that if we encounter a new artist, not present in the database, we cannot do an exact prediction for it. On top of that to encode a new record we need to apply the encoding algorithm on the while dataset every time. An other important thing to notice is that if we have a feature with a cardinality very close to the number of records, applying target encoding on it means to build in practice a map 1:1 from that feature to the target making all other features and so the model pointless. In our dataset this is the exact case of the *track_name* feature (73k unique values on 81k final records) reason why we decided to exclude said feature from the model. Fore the encoding of the *artists* feature we used the `TargetEncoder` from the `category_encoders` package.

5.2 Hyperparametrs tuning

Using also the categorical feature we had two hyperparameters to tune: the alpha parameter and the smoothing parameter used in the target encoding of the *artists* feature. To optimize both of them we used cross validation to see how the estimate risk changed changing the value of these parameters. Here are the graphs for the smoothing parameter:

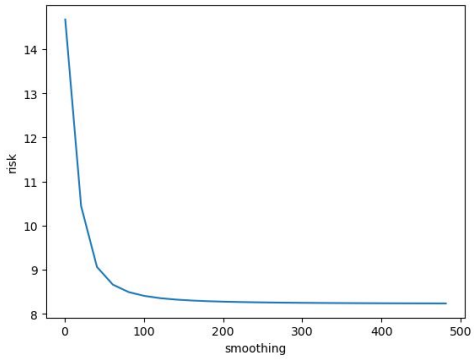

Figure 4: risk with smoothing in range 1-500

We can see that after 200 the risk stabilizes so we choose this value as our optimal value.

And here the graphs for the alpha parameter:

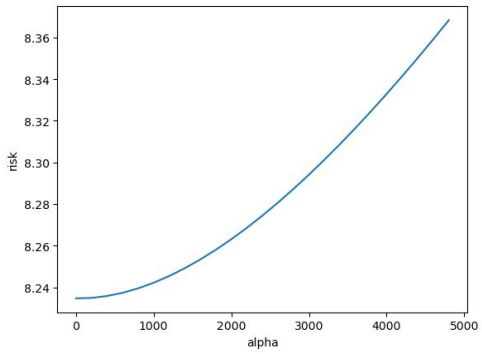

Figure 5: risk with alpha in range 1-5000

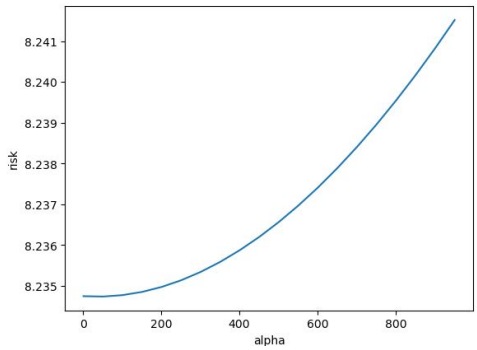

Figure 6: risk with alpha in range 1-1000

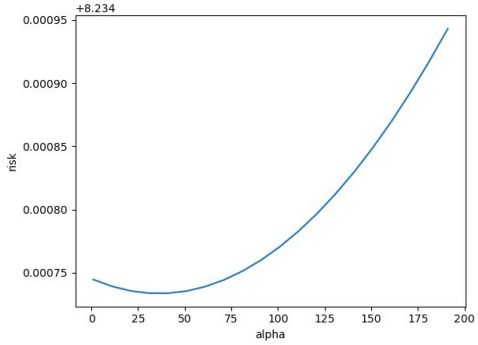

Figure 7: risk with alpha in range 1-200

For alpha the optimal value turns out to be 40.

5.3 Results

Using the optimal value previously obtained (smoothing = 180 and alpha = 40) and computing the 5-fold cross validation on the model we get an estimated risk of 8.23473366583011 which is not a bad result considering the range is 0-100. Using this model we can make a quite accurate prediction of the popularity of a song.