(Kernel) Ridge Regression

Pietro Manning

November 2023

1 Introduction

I downloaded the Spotify Tracks Dataset and conducted ridge regression to predict the popularity of tracks. The dataset comprises both numerical and categorical features. I followed these guidelines:

Initial Model Training with Numerical Features:

I started by training the model using only the numerical features present in the dataset. For this phase, I experimented with different training parameters to optimize the model's performance. Utilized ridge regression, implementing the code from scratch without relying on external libraries like Scikit-learn. Handling Categorical Features:

Recognizing the presence of categorical features, I implemented appropriate preprocessing techniques such as one-hot encoding. Integrated these categorical features with the numerical ones and trained the model. Similar to the first step, I explored various training parameters to fine-tune the model's performance. Experimentation and Cross-Validation:

In both cases, I experimented with different training parameters to find the optimal configuration for the ridge regression model. To assess the model's robustness and generalization, I employed 5-fold cross-validation. The 5-fold cross-validation helped in computing reliable risk estimates for the model. Performance Comparison and Discussion:

After training the model in both scenarios (numerical features only and combined numerical and categorical features), I thoroughly compared their performances. This comparison involved analyzing the model's accuracy, precision, recall, or any other relevant metrics. I discussed the strengths and weaknesses of each approach, considering factors like interpretability, computational efficiency, and predictive power. By following these steps and implementing ridge regression from scratch for the numerical features, I aimed to comprehensively evaluate and compare the model's performance under different conditions. I utilized both kernel ridge regression implemented from scratch and the ridge regression algorithm for predicting track popularity in the Spotify Tracks Dataset. Additionally, I employed the kernel ridge regression and ridge regression implementations from the Scikit-learn library to compare results. This allowed for a comprehensive assessment of the model's performance under various conditions and facilitated a comparison between the custom implementation and the widely-used Scikit-learn implementations.

2 Ridge Regression

Ridge Regression is a method of linear regression that incorporates a regularization term into the objective function. The goal of Ridge Regression is to minimize the following function:

$$J(\beta) = \sum_{i=1}^{n} (y_i - \mathbf{x}_i^T \beta)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{1}$$

where:

- n is the number of observations,
- $\mathbf{x}_i$ is the feature vector of the i-th observation,
- β is the vector of model coefficients,
- y_i is the output value of the i-th observation,

- p is the number of features,
- λ is the regularization parameter.

The regularization term $\lambda \sum_{j=1}^{p} \beta_j^2$ is added to penalize larger model coefficients. This helps prevent overfitting of the model to the training data.

The solution to Ridge Regression is given by:

$$\hat{\beta}_{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{2}$$

where:

- $\mathbf{X}$ is the feature matrix,
- $\mathbf{y}$ is the output vector,
- $\mathbf{I}$ is the identity matrix.

3 Ridge Regression Implementation in Python

After explaining the mathematical formulation of Ridge Regression, let's take a look at its implementation in Python:

```python
import numpy as np

class RidgeRegression:
    def __init__(self, alpha=1.0):
        self.alpha = alpha  # Regularization parameter

    def fit(self, X, y):
        # Add a bias column to the input X
        ones = np.ones((X.shape[0], 1))
        X = np.concatenate((ones, X), axis=1)
        # Calculate Ridge Regression coefficients
        n_features = X.shape[1]
        identity_matrix = np.eye(n_features)
        self.coefficients = np.linalg.inv(X.T.dot(X) +
        self.alpha * identity_matrix).dot(X.T).dot(y)

    def predict(self, X):
        # Add a bias column to the input X
        ones = np.ones((X.shape[0], 1))
        X = np.concatenate((ones, X), axis=1)
        # Calculate predictions
        return X.dot(self.coefficients)

    def get_params(self, deep=True):
        # Returns model parameters as a dictionary
        return {'alpha': self.alpha}
```

Listing 1: Ridge Regression Implementation

In the code above, we've implemented the Ridge Regression algorithm in Python using the NumPy library. The `fit` method computes the coefficients, the `predict` method generates predictions, and the `get_params` method returns the model parameters.

4 Kernel Ridge Regression

Kernel Ridge Regression is an extension of Ridge Regression that allows for non-linear relationships between the features and the target variable by leveraging the kernel trick. The objective function for Kernel Ridge Regression is defined as:

$$J(\alpha) = \|\mathbf{y} - \mathbf{K}\boldsymbol{\alpha}\|^2 + \lambda\boldsymbol{\alpha}^T\mathbf{K}\boldsymbol{\alpha} \tag{3}$$

where:

- $\mathbf{y}$ is the vector of target values,
- $\mathbf{K}$ is the kernel matrix,
- $\boldsymbol{\alpha}$ is the vector of dual coefficients,
- λ is the regularization parameter.

The kernel matrix $\mathbf{K}$ is computed based on the chosen kernel function, allowing the algorithm to capture complex relationships between the input features.

The solution to Kernel Ridge Regression is given by:

$$\mathbf{f}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}) \tag{4}$$

where:

- $\mathbf{f}(\mathbf{x})$ is the predicted output for input $\mathbf{x}$,
- α_i are the dual coefficients,
- $k(\mathbf{x}_i, \mathbf{x})$ is the kernel function between training point $\mathbf{x}_i$ and input $\mathbf{x}$.

Common kernel functions include the linear kernel, polynomial kernel, and radial basis function (RBF) kernel.

Kernel Ridge Regression is particularly useful when dealing with non-linear relationships in the data, allowing for more flexible modeling compared to traditional linear regression methods.

5 Kernel Gaussian Ridge Regression

Kernel Gaussian Ridge Regression is an extension of Ridge Regression that incorporates the kernel trick, allowing it to capture nonlinear relationships in the data. In this case, a Gaussian kernel is used to transform the input features.

5.1 Mathematical Formulation

The objective function of Kernel Gaussian Ridge Regression is defined as follows:

$$J(\boldsymbol{\alpha}) = \|\mathbf{y} - \mathbf{K}\boldsymbol{\alpha}\|^2 + \lambda \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} \tag{5}$$

where:

- $\mathbf{y}$ is the vector of target values,
- $\mathbf{K}$ is the kernel matrix,
- $\boldsymbol{\alpha}$ is the vector of dual coefficients,
- λ is the regularization parameter.

The Gaussian kernel is defined as:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right) \tag{6}$$

where σ is a parameter that controls the width of the Gaussian kernel.

5.2 Training

During the training phase, the kernel matrix $\mathbf{K}$ is computed using the Gaussian kernel for all pairs of input samples. The regularization term is added, and the dual coefficients $\boldsymbol{\alpha}$ are calculated by solving the linear system:

$$(\mathbf{K} + \lambda \mathbf{I})\boldsymbol{\alpha} = \mathbf{y} \tag{7}$$

5.3 Prediction

For making predictions on new input data, the Gaussian kernel is applied between the test data and the training data, and the predictions are calculated using the learned dual coefficients.

5.4 Model Parameters

The model parameters for Kernel Gaussian Ridge Regression include the regularization parameter λ and the width parameter σ of the Gaussian kernel.

6 Kernel Gaussian Ridge Regression Implementation in Python

After describing the mathematical background of Kernel Ridge Regression, let's explore its Python implementation:

```python
import numpy as np

class KernelRidgeRegression:
    def __init__(self, alpha=1.0, gamma=1.0):
        self.alpha = alpha
        self.gamma = gamma

    def gaussian_kernel(self, x1, x2):
        return np.exp(-self.gamma * np.linalg.norm(x1 - x2)**2)

    def fit(self, X, y):
        self.X = X
        self.y = y
        n_samples, n_features = X.shape

        # Calculate the Kernel matrix
        K = np.zeros((n_samples, n_samples))
        for i in range(n_samples):
            for j in range(n_samples):
                K[i, j] = self.gaussian_kernel(self.X.iloc[i], self.X.iloc[j])

        # Regularize the Kernel matrix
        K += self.alpha * np.identity(n_samples)

        # Solve for alpha coefficients
        self.alpha_coef = np.linalg.solve(K, y)

    def predict(self, X):
        n_samples, _ = X.shape
        y_pred = np.zeros(n_samples)

        # Calculate predictions using the learned alpha coefficients
        for i in range(n_samples):
            k_i = np.array([self.gaussian_kernel(self.X.iloc[i], x)
            for x in np.array(self.X)])

            y_pred[i] = np.dot(k_i, self.alpha_coef)

        return y_pred

    def get_params(self, deep=True):
        return {'alpha': self.alpha, 'gamma': self.gamma}
```

Listing 2: Kernel Ridge Regression Implementation

In the provided Python code, the 'fit' method calculates the Kernel matrix using the Gaussian kernel, regularizes it, and solves for the alpha coefficients. The 'predict' method uses these coefficients to make predictions.

7 Ridge Regression Script

This script implements Kernel Gaussian Ridge Regression to predict the popularity of songs from the Spotify dataset. The Gaussian kernel is applied to capture non-linear relationships in the data. The script follows these steps:

1. Reads the dataset from a CSV file.
2. Selects only numerical features for analysis.
3. Encodes categorical features using One-Hot Encoding.
4. Splits the dataset into training and testing sets.
5. Scales the numerical features using Min-Max scaling.
6. Fits a Kernel Gaussian Ridge Regression model to the training data.
7. Evaluates the model on the test set and prints Mean Squared Error (MSE) and R-squared (R2) scores.

8 One-Hot Encoder

The One-Hot Encoder is a technique used in machine learning to represent categorical variables as binary vectors. It is particularly useful when dealing with features that have discrete values and where the numerical order is not meaningful. The One-Hot Encoder converts categorical variables into binary vectors of 0s and 1s. For each category, it creates a new binary column, and the presence of a category is indicated by a 1 in the corresponding column. Consider a categorical feature "Color" with categories: Red, Green, and Blue. After applying One-Hot Encoding, the feature is transformed into three binary columns: "Color_Red," "Color_Green," and "Color_Blue." One-Hot Encoding is often applied before feeding data into machine learning algorithms that expect numerical input. It helps prevent the model from incorrectly assuming ordinal relationships between categories. [Implementation:] In Python, the `OneHotEncoder` class from the `sklearn.preprocessing` module is commonly used. It can be applied to specific columns of a DataFrame or to the entire dataset. [Considerations:] One should be mindful of the potential increase in dimensionality after One-Hot Encoding, especially in the presence of a large number of categories. This can affect model training time and memory requirements.

9 Code Explanation

Let's break down the code into sections:

9.1 Import Statements

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.kernel_ridge import KernelRidge
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score
```

These lines import necessary libraries, including pandas for data manipulation, scikit-learn for machine learning tools, and specific modules for preprocessing, cross-validation, and model evaluation.

9.2 Ridge Regression Model Initialization

```
from SpotifyPredictor import RidgeRegression

ridge_regression_model = RidgeRegression(alpha=.9)
```

This part imports a custom Ridge Regression model from the "SpotifyPredictor" module and initializes an instance of it with a regularization parameter (α) set to 0.9.

9.3 Data Reading and Preprocessing

```
df = pd.read_csv('C:/Users/pietr_3fbsul4/Desktop/MLproject/SpotifyDataSet.csv')
```

This line reads a CSV file containing Spotify dataset into a pandas DataFrame.

```
ndf = df.select_dtypes(include=['int64', 'float64'])
```

This line selects only the columns of numerical data (integer and floating-point types) from the DataFrame.

```
encoder = OneHotEncoder(sparse=False, drop='first')
time_signature_encoded = encoder.fit_transform(ndf[['time_signature']])
```

Here, the code uses `OneHotEncoder` to encode the `"time_signature"` column, converting categorical data into a binary matrix.

```
time_signature_encoded_DF = pd.DataFrame(time_signature_encoded, columns=
    encoder.get_feature_names_out(['time_signature']))
ndf = pd.concat([ndf, time_signature_encoded_DF], axis=1).drop('time_signature'
    , axis=1)
```

This code appends the encoded matrix back to the DataFrame and drops the original `"time_signature"` column.

```
X = ndf.drop("popularity", axis=1)
y = ndf["popularity"]
```

The features (X) and target variable (y) are defined, with "popularity" as the target variable.

9.4 Cross-Validation and Model Evaluation

```
kf = KFold(n_splits=5, shuffle=True)
```

A KFold cross-validator is created with 5 splits and shuffling of the data.

```
scores = cross_val_score(ridge_regression_model, X, y, cv=kf, scoring='r2')
```

Cross-validation scores for the custom Ridge Regression model are obtained using the KFold strategy and the R^2 scoring metric.

```
scores_sklearn = cross_val_score(model, X, y, cv=kf, scoring='r2') #scikit
    learn regression
```

Similarly, cross-validation scores for the scikit-learn Ridge Regression model are computed.

```
mean_score = scores.mean()
```

The mean cross-validation score for the custom Ridge Regression model is calculated.

```
print("Score:", mean_score)
print("scores_sklearn: ",scores_sklearn.mean())
```

as a result i obtain; Score: 0.025130803359362817 scores_sklearn: 0.0251027125644198

The model is not very effective at capturing and explaining the patterns or trends in the data. R-squared, also known as the coefficient of determination, is a measure of how well the independent variables in a regression model explain the variation in the dependent variable. It ranges from 0 percent to 100 percent, where a higher value indicates a better fit. A low R-squared value, such as 2 percent, suggests that the model does not provide a good representation of the relationships between the variables. a low R-squared does not necessarily mean that the model is useless. Depending on the context, a small R-squared may be acceptable if the goal is to simply understand the general trends in the data. However, for predictive purposes, a higher R-squared is generally desirable, as it indicates that a larger proportion of the variability in the dependent variable is explained by the model.

In Ridge regression, the introduction of a regularization term (controlled by the hyperparameter α, denoted as λ helps prevent overfitting by penalizing large coefficients of the features. As you increase the value of alpha, the penalty for large coefficients becomes stronger.

I got a slight improvement in R by raising the alpha level, thus the regularization is having a positive impact on the model's performance. This can happen for several reasons:

Preventing Overfitting: Ridge regression tends to be particularly useful when dealing with multicollinearity, where independent variables are highly correlated. By penalizing large coefficients, Ridge regression can prevent the model from fitting the noise in the data, leading to better generalization performance.

Stabilizing Coefficients: The regularization term helps to stabilize and shrink the coefficients of the features, reducing their sensitivity to individual data points. This can result in a more robust and generalizable model.

Feature Selection: While Ridge regression does not perform variable selection in the same way as Lasso regression, it can still effectively reduce the impact of less relevant features by shrinking their coefficients towards zero. This can lead to a more parsimonious model that focuses on the most important features.

10 Spotify Popularity Prediction Script 2

This script aims to predict the popularity of songs from the Spotify dataset using Ridge Regression and Kernel Ridge Regression models. Main Differences: Data Splitting:

Spotify Popularity Prediction Script 2: Splits the dataset into training and testing sets using `train_test_split`. This allows for explicit training of the model on one subset and testing on another. Scaling:

Spotify Popularity Prediction Script 2: Performs explicit feature scaling using MinMaxScaler before fitting the Ridge Regression model. This ensures that all features are on a similar scale, which can be important for some machine learning algorithms. Model Evaluation:

Spotify Popularity Prediction Script 2: Evaluates the Ridge Regression model on a separate test set using metrics such as mean squared error and R-squared. This provides insights into how well the model generalizes to new, unseen data. Purpose:

Spotify Popularity Prediction Script 2: Demonstrates a common train-test-split approach, where the dataset is divided into training and testing sets. This approach is useful for assessing how well the model performs on new, previously unseen data. Considerations: Choose this approach if your primary goal is to understand how well the model generalizes to new data. The train-test-split approach in Snippet 1 provides a clear separation between training and testing phases, allowing for a straightforward evaluation of the model's performance on unseen examples.

It follows these steps:

1. Reads the dataset from a CSV file.
2. Selects only numerical features for analysis.
3. Encodes categorical features using One-Hot Encoding.
4. Splits the dataset into training and testing sets.
5. Scales the numerical features using Min-Max scaling.
6. Fits a Ridge Regression model to the training data.
7. Evaluates the model on the test set and prints Mean Squared Error (MSE) and R-squared (R2) scores.

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score
from SpotifyPredictor import RidgeRegression

# Initialize models and scaler
ridge_regression_model = RidgeRegression(alpha=0.9)
model = Ridge(alpha=0.3)
min_max_scaler = MinMaxScaler()

```

```
# Step 1: Read the CSV
df = pd.read_csv('C:/Users/pietr_3fbsul4/Desktop/MLproject/SpotifyDataSet.csv')

# Step 1.5: Select only numerical features
ndf = df.select_dtypes(include=['int64', 'float64'])

# The time_signature feature is categorical due to its restricted values
encoder = OneHotEncoder(sparse=False, drop='first')
time_signature_encoded = encoder.fit_transform(ndf[['time_signature']])
time_signature_encoded_DF = pd.DataFrame(time_signature_encoded,
columns=encoder.get_feature_names_out(['time_signature']))
ndf = pd.concat([ndf, time_signature_encoded_DF], axis=1)
.drop('time_signature', axis=1)

# Prepare data for modeling
X = ndf.drop("popularity", axis=1)
y = ndf["popularity"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Scale the training data
X_train_scaled = min_max_scaler.fit_transform(X_train)
ridge_regression_model.fit(X_train_scaled, y_train)

# Scale the test data and make predictions
X_test_scaled = min_max_scaler.transform(X_test)
y_pred_scaled = ridge_regression_model.predict(X_test_scaled)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred_scaled)
r2 = r2_score(y_test, y_pred_scaled)

% Print the evaluation metrics
```

Listing 3: Spotify Popularity Prediction Script

In this way i can observe the coefficients self.coefficients:

$$\begin{bmatrix} 38.37007538 \\ 1.88165019 \\ -9.70288609 \\ 7.62188054 \\ -3.58002749 \\ -0.18198252 \\ 6.00094206 \\ -0.57831188 \\ -10.44990578 \\ -0.95214765 \\ -8.32439475 \\ 1.0935043 \\ -9.93885069 \\ 3.96081824 \\ -9.82796962 \\ -10.15179275 \\ -6.58090115 \\ -9.72399219 \end{bmatrix}$$

11 Kernel Ridge Regression Comparison Script

This script compares the performance of two Kernel Ridge Regression models, one implemented using scikit-learn and the other using a custom implementation. It follows these steps:

1. Reads the dataset from a CSV file and selects a subset.
2. Selects only numerical features for analysis.
3. Encodes categorical features using One-Hot Encoding.
4. Initializes Ridge Regression, scikit-learn Ridge, scikit-learn Kernel Ridge, and custom Kernel Ridge models.
5. Applies cross-validation using K-Fold with both scikit-learn and custom Kernel Ridge models.
6. Prints the mean R-squared (R2) scores for both models.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.kernel_ridge import KernelRidge
from sklearn.linear_model import Ridge
from SpotifyPredictor import RidgeRegression
from kernelridge import KernelRidgeRegression

# Initialize models and scaler
ridge_regression_model = RidgeRegression(alpha=0.9)
model = Ridge(alpha=0.3)
modelK = KernelRidge(alpha=0.3, kernel='rbf', gamma=1)
model_kernel_custom = KernelRidgeRegression(alpha=0.3, gamma=1)
min_max_scaler = MinMaxScaler()

# Read and preprocess the dataset
df = pd.read_csv('C:/Users/pietr_3fbsul4/Desktop/MLproject
/SpotifyDataSet.csv').drop('Unnamed: 0', axis=1)
df = df.iloc[0:200]
ndf = df.select_dtypes(include=['int64', 'float64'])

# Encode categorical features (time_signature)
encoder = OneHotEncoder(sparse=False, drop='first')
time_signature_encoded = encoder.fit_transform(ndf[['time_signature']])
time_signature_encoded_DF =

pd.DataFrame(time_signature_encoded, columns=encoder.get_feature_names_out(['
    time_signature']))
ndf = pd.concat([ndf, time_signature_encoded_DF], axis=1)
.drop('time_signature', axis=1)

# Prepare data for modeling
X = ndf.drop("popularity", axis=1)
y = ndf["popularity"]

# Initialize K-Fold for cross-validation
kf = KFold(n_splits=5, shuffle=True)

```

```
# Cross-validation with custom Kernel Ridge model
scores_custom = cross_val_score(model_kernel_custom, X, y, cv=kf, scoring='r2')

# Cross-validation with scikit-learn Kernel Ridge model
scores_sklearn = cross_val_score(modelK, X, y, cv=kf, scoring='r2')
mean_score_custom = scores_custom.mean()
mean_score_sklearn = scores_sklearn.mean()

# Print the mean R-squared (R2) scores
print("scikit-learn Kernel Ridge Model Score:", mean_score_sklearn)
print("Custom Kernel Ridge Model Score:", mean_score_custom)
```

Listing 4: Kernel Ridge Regression Comparison Script

As a resulte i obtained a negative R^2; scores sklearn: -2.6542282623845805, mean score gr custom: -2.0907751936566905 a negative R^2 indicates that the model is unable to explain the variability in the dependent variable significantly, and it likely fits worse than a baseline mean model. It may suggest that the independent variables in the model are not suitable for explaining the variability in the dependent variable.