Ridge Regression

Mazza, Lorenzo

1 Introduction

In this project I have faced the implementation of a ridge regression model over a dataset formed by different feature of songs from Spotify, with the aim to predict how much popular a song will be. First I've observed the dataset to fix some "irregularity" of some features and unexpected repetition of some data's songs. Next the ridge regression model has been executed only over the numerical feature, except for some features `(time_signature, key, mode)` that has numerical value but they are meant more in a discrete way, so I considered them like categorical variables. Then I've applied the model over both numerical and categorical features with the last ones encoded with the one-hot-encoder. These models are "tested" on a validation set, over different regularization coefficient, to identify the best value for the best prediction with the help of the MSE value and R^2. At the end, this analysis has been executed with a 5-fold cross-validation. After all I have made some consideration about the success or failing of these predictions.

2 Theoretical prelude

2.1 Regression Models

Linear Regression The linear regression model is a model used to predict a target value for a observation with the values of other variables assuming there's a linear dependence between the target variable and the others. Assuming I have n feature for each observation to predict the target value, as a result of this prediction model we'll have a predictor $h(\cdot)$ in this form:

$$h(x) = \sum_{j=1}^{d} w_j x_j = w^T \cdot x$$

where $x = (x_1, ..., x_d)$ is the variable vector of the observation and $w = (w_1, ..., w_d)$ are real the coefficient related to each feature. The w_j values are calculated minimizing the sum of square of the errors ($e_i = h(x_i) - y_i$ with x_i the i-th observation's features and y_i the i-th target value) over a set of observation called train set. Defining the design matrix of dimension $m \times d$ $S = [x_1, ..., x_m]^T$ and assuming that is invertible I can calculate the explicit values for the coefficient w_j in this way:

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

Ridge Regression After this short explanation of the linear regression model I can identify a problem that could occur and arouse numerical issue: what happen if the design matrix S isn't invertible or it's nearly singular? In this case we'll have some perturbation that could increase the estimation error. In order to stabilize the predictor's computation I can add to the design a regularization parameter α.
So instead of minimizing $\sum_{i=1}^{m} (h(x_i) - y_i)^2$ I look for the w that minimizes this value:
$\sum_{i=1}^{m} \left((h(x_i) - y_i)^2 + \alpha w_i^2 \right) = \|Sw - y\|^2 + \alpha \|w\|^2$
Hence, given a regularization parameter, I obtain from the design matrix S a matrix $Z = S^T S + \alpha I$ that is definite positive, so invertible.

In this way (and with an α enough big) I can calculate explicitly the ridge regression coefficient without worries:

$$w_{ridge} = (S^T S + \alpha I)^{-1} S^T y$$

2.2 One-Hot Encoding

I have to consider the implementation of a technique to deal with the categorical variables, for example representing them as binary vector or matrix, so I've considered the One-Hot Encoding.
One-Hot Encoding is a method that turns categorical variables into numerical ones. It created a new binary variable for each value of the categorical one, these new feature will be equal to `1` if the categorical variable of the observation has the value of the feature, instead will be `0`. This method doesn't preserve any ordinal information.

2.3 K-fold Cross Validation

Dividing the dataset in two arbitrary part there could be some asymmetry in this two set that could undermine the the model's result, therefore k-fold cross validation can solve this problem. This method divides in k sets the data, then the model is trained on a train set form by k-1 folds and tested on the excluded fold, so this is iterated changing the fold for the test set. In this way I obtain k different model and choose the best one.

2.4 Regression scores

Coefficient of determination : R^2 is a coefficient that assumes value from 0 to 1, to be defined I have to introduce before two quantities: the residual sum of squares $SS_{res} = \sum_i (\hat{y_i} - y_i)^2$ and the total sum of squares $SS_{tot} = \sum_i (\bar{y_i} - y_i)^2$. So R^2 can be defined as follows: $R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$ It indicates a good model if it's near 1 and a bad model if it's too low.

Mean Squared Error : the MSE measure how much the prediction is far from the real outcome computing the mean of the square difference between prediction and observed value. It's defined as follows: $MSE = \frac{1}{n} \sum_i^n (\hat{y_i} - y_i)^2$.

3 Spotify Dataset

The dataset is formed by 114.000 observation (or songs) associated with 20 feature and a variable identified as target variable (`popularity`). Observing the data I notice that some songs are repeated more times with the same value for all feature except one: `track_genre`, that's because a song can be identified with more than one genre. This repetition (more than the 20% of the observation are a repetition) could distort the prediction result, so later on will be discuss how to deal with this problem and if actually it is a problem. There are different types of feature, so I show below a brief description:

Literal Feature

- `track_id` : a string of characters that identify a song, useful in case of multiple observation for the same song
- `artists`: the song's artist, not so easy to use in a regression model
- `track_name`: the song's track name, as above not so easy to use in a regression model
- `album_name`: the song's album name, as above not so easy to use in a regression model
- `track_genre`: the song's genre, in this dataset there are a lot of different values, but it could be worth to give a chance

From this first feature I can see that maybe only the `track_genre` can be used to produce a prediction model.

Numerical Feature

- `duration_ms` : the song's duration in milliseconds
- `tempo` : bpm values (beats per minutes)
- `loudness`: is measured in decibel, it goes from some negative values (like -49) to some positive value (like 10), maybe the author of this dataset put some reference value to 0
- `danceability`: measures if the track ha dancing purpose, value from 0 (not danceable) to 1 (definitely danceable)
- `energy`: measures the intensity's track, value from 0 (not intense) to 1 (definitely intense)
- `speechiness`: measures how much spoken words are present,value from 0 (speechless) to 1 (only speeches)
- `acousticness`: measures if track is acoustic, value from 0 (not acoustic) to 1 (definitely acoustic)
- `instrumentalness`: measures if there's no vocals, if the track it's only instrumental value from 0 (not only instrumental) to 1 (definitely instrumental)
- `liveness`: measures if there's an audience, value from 0 (without audience noises) to 1 (with a lot audience noises)
- `valence`: measures the positiveness, value from 0 (song with sad vibes) to 1 (song with joyful vibes)
- `explicit`: binary variable, 1 if the song is explicit, 0 instead
- `mode`: binary variable, 1 if the song is in major scale, 0 for minor scale
- `key` : the music key of the song, (categorical variable)

- time_signature: estimated time signature (value of 3 means time signature of 3/4)

Now I can see that there are different "type" of numerical features, most of the has value in $[0,1]$, loudness assume also negative values or duration_ms, tempo assume values greater than 1, these will be shifted and normalized. Some are binary (explicit, mode), other are integer (key, time_signature) so I'll deal them as categorical variables.

4 Code Explanation

4.1 Ridge Regression

At the begging is implemented in an external class called RidgeRegression to fit and perform the ridge regression model. In the code below is reported the ridge regression implementation. The RidgeRegression class is defined by two "element", the regularization parameter and the weight coefficient (betas) result of the fit method.
To obtain the weight coefficient is applied the explicit formula $w_{ridge} = (S^T S + \alpha I)^{-1} S^T y$ enunciated before in the theory prelude. Before that the design matrix X is enlarge with a column of 1 in order to obtain an intercept value for the regression line.
Then is written the predict method in order to calculate the target value's prediction from a given X matrix of data where perform the model obtained in precedence that call this method.
At the end are implemented a get_params method that returns the regularization parameter used, this is requested by the cross_val_score function operate.

```
### RIDGE
import numpy as np

class RidgeRegression:
   def __init__(self,reg_param = 1.0):
      self.reg_param = reg_param
      self.betas = None
      self.bias = None

   def fit(self, X, y, sample_weights=None):
      n,d = X.shape
      ones = np. ones ((n , 1))
      X = np. concatenate (( ones , X), axis =1)
      self.betas = np.zeros(d+1)
      I = np.eye(d+1)
      S = np.dot(X.T,X)
      self.betas = np.linalg.inv(S + self.reg_param * I).dot(X.T).dot(y)
      return self.betas

   def predict(self, X):
      n,d = X.shape
      ones = np. ones ((n , 1))
      X = np. concatenate (( ones , X), axis =1)
      return np.dot(X, self.betas)

   def get_params(self, deep=True):
      return {'reg_param': self.reg_param}

   def get_coeff(self, deep=True):
      return self.betas
```

Code 1: Ridge Regression

4.2 Cleaning Data

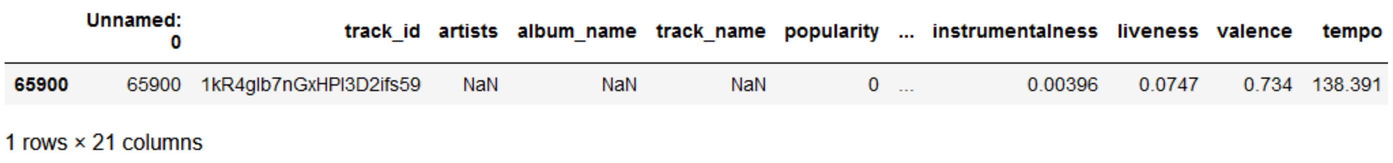
Like said in the previous section,there's some feature that aren't normalized (`duration, loudness, tempo`) and one (`loudness`) has also some negative values. So I shifted this one and normalized it and the others that have maximum bigger than 1.

```
# SHIFTING loudness
min_value = dataset['loudness'].min()
dataset['loudness'] = dataset['loudness'] - min_value

# NORMALIZING loudness + tempo + duration_ms (there are outliers...)
max_value_shifted = dataset['loudness'].max()
dataset['loudness'] = dataset['loudness'] / max_value_shifted
max_value_shifted = dataset['duration_ms'].max()
dataset['duration_ms'] = dataset['duration_ms'] / max_value_shifted
max_value_shifted = dataset['tempo'].max()
dataset['tempo'] = dataset['tempo'] / max_value_shifted
```

Code 2: Data cleaning: Shifting, normalizing

I've controlled also the presence of `Nan` values finding some only in a row, only for `artists, track_name, album_name`, these feature aren't used for the prediction purpose so I can ignore them.

```
print (dataset.isna().sum().sort_values(ascending = False))
dataset[dataset.isna().any(axis=1)]

artists       1
album_name    1
track_name    1
Unnamed: 0    0
..
explicit      0
duration_ms   0
popularity    0
track_genre   0
Length: 21, dtype: int64
```



	Unnamed: 0	track_id	artists	album_name	track_name	popularity	...	instrumentalness	liveness	valence	tempo
65900	65900	1kR4glb7nGxHPl3D2ifs59	NaN	NaN	NaN	0	...	0.00396	0.0747	0.734	138.391

1 rows × 21 columns

Figure 1: Row with NaN values.

4.3 Encoding Categorical features

After the data cleaning on the numerical features, I have to set the data for the analysis with also the categorical value. As introduce in the theory prelude, I have chosen to implement the one hot encoding technique, for this purpose it's imported from the `scikit-learn` library the `OneHotEncoder` function. The drop flag is set to `'first'`, in this way the first value of the encoded data won't be represented in the new binary feature in order to obtain regression coefficient more pertinent to the real effect of the encoded variable.

```
## ONE-HOT ENCODER - track_genre
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(sparse = False, drop='first')

encoded_genre = encoder.fit_transform(dataset[['track_genre']])
encoded_genre_df = pd.DataFrame(encoded_genre,
    columns=encoder.get_feature_names_out(['track_genre']))

## ENCODED DATA (track_genre) - NO AGGREGATION
dataset_encoded = pd.concat([dataset.drop('track_genre', axis=1), encoded_genre_df], axis=1)
genre_df = dataset_encoded.select_dtypes(include=['int', 'float','uint8']).drop(['time_signature',
    'key', 'popularity', 'Unnamed: 0'], axis = 1)
```

Code 3: One-Hote Encoding on `track_genre`

The categorical variables considered are: `track_genre, key, time_signature`. Given that `track_genre` has 114 different values, I've considered his encoding separately from the other two.
Furthermore I've looked to `time_signature values`, because it should assume values from 3 to 7, according to the data source and also to the feature meaning, but I have values from 0 to 5. If 0 could mean that the track isn't a song but something only spoken, I haven't found a reasonable meaning for the value 1 (time signature of 1/4). So I've decide also for a case where I've encoded only the key feature.

4.4 Aggregation

Now I considered the "`track_genre` problem", mentioned before, with the aim to have a single observation for each song identified by the `track_id`.
In the analysis cases where the `track_genre` isn't considered, I have only to aggregate the value to the first time the song is met, because all other values are the same. So I called the `pandas.DataFrame.agg` method with argument `'first'` to the dataset grouped by the `track_id`.

```
## AGGREGATION
df_aggr_no_genre = dataset.groupby('track_id').agg('first').sort_values(by='Unnamed: 0')
```

Code 4: Data aggregation ignoring `track_genre`

Instead the case where I considered also the `track_genre` encoded has to be treated with a little addition. Considering that for every duplicate observation the `track_genre` is different I have to preserve the information from all the different observation from the same song. To do so I have implemented a vector as an aggregation function that associate for the columns obtained from the `track_genre` encoding (the only columns of type `uint8`) the command `'max'`, in this way I obtain 1 if the song has the genre in one of its observation. For example the song with `track_id` 6S3JlDAGk3uu3NtZbPnuhS has 9 different observation with 9 different genres, so with this aggregation, after the encoding, I have obtained a dataset with a single observation that catch each genre taking the maximum value, i.e. 1. So, as before, this "function" is given as argument to the `pandas.DataFrame.agg` method applied to the dataset encoded grouped by the `track_id`.

```
#### OHE - AGGREGATION with genre 0-1
genre_columns = dataset_encoded.select_dtypes(include=['uint8']).columns.to_list()
other_columns = dataset_encoded.select_dtypes(include=['int',
    'float','object','bool']).columns.to_list()

## AGGREGATION FUNCTIONS
aggregation_functions_g = {}
for col in dataset_encoded.columns:
   if col in other_columns:
      aggregation_functions_g[col] = 'first'
   elif col in genre_columns:
      aggregation_functions_g[col] = 'max'

genre_df_aggr =
    dataset_encoded.groupby('track_id').agg(aggregation_functions_g).sort_values(by='Unnamed: 0')
```

Code 5: Data aggregation with `track_genre`

4.5 Model perform

Firstly are considered the numerical values (except for `key`, `time_signature`, as said before) fixing the regularization parameter to an arbitrary value and then taking multiple values to see when the model perform well and also if it's possible to perform linear regression.
As mentioned in advanced, the cases I've analysed are:

1. Numerical data only - not aggregated
2. Numerical data only - aggregated
3. Numerical data and `track_genre` encoded - not aggregated
4. Numerical data and `track_genre` encoded - aggregated
5. Numerical data and `key,time_signature` encoded - not aggregated
6. Numerical data and `key,time_signature` encoded - aggregated
7. Numerical data and `key` encoded - not aggregated
8. Numerical data and `key` encoded - aggregated

At the beginning the model is performed on the original dataset and also on the aggregated one, to see which one is preferable to work with. The code below perform the ridge regression model on the numerical case first with original data and then with the data aggregated, that's replicated in the same way for next cases, with data encoded and/or aggregated as below.

```
## NUMERICAL FEATURES ONLY
# NUMERIC - no aggr
numeric_df = dataset.select_dtypes(include=['int', 'float']).drop(['time_signature', 'key',
    'Unnamed: 0'], axis = 1)

## 1st dataset - num not aggregated
X = numeric_df.drop('popularity',axis = 1).values
y = numeric_df['popularity'].values
# SPLIT the dataset into train and test sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
```

```
## 1st REGRESSION
model = RidgeRegression(reg_coeff=1.5)
betas_1 = model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Code 6: Numerical features

```
## NUMERICAL FEATURES ONLY - AGGREGATED CASE
numeric_df_aggr = df_aggr_no_genre.select_dtypes(include=['int',
    'float']).drop(['time_signature', 'key', 'Unnamed: 0'], axis = 1)

Xa = numeric_df_aggr.values # Convert selected columns to array
ya = merged_DF['popularity'].values # Convert the target column to array

# SPLIT the dataset into train and test sets (75% train, 25% test)
X_train_a, X_test_a, y_train_a, y_test_a = train_test_split(Xa, ya, test_size=0.20,
    random_state=42)

## 2nd REGRESSION
model = RidgeRegression(reg_coeff=1.5)
betas_a = model.fit(X_train_a, y_train_a)
y_pred_a = model.predict(X_test_a)
```

Code 7: Numerical features, data aggregated

4.6 Cross Validation

At the end is implemented a 5-fold cross validation with the help of `cross_validate` function from the `scikit-learn` library. To analyse the performance of the models are calculated `r2_score`,`neg_mean_square_error` I've reported below the code only for the first case, it's implemented in the same way for the others.

```
from sklearn.model_selection import cross_validate

model = RidgeRegression(reg_param=1.5)
scores = ['r2' , 'neg_mean_squared_error']

cv_acc = cross_validate(model, X, y, cv=5, scoring=scores)
```

Code 8: 5-fold cross validation on original numerical data

In the next section I've illustrated the models' performance based on the MSE and R^2.

5 Models performance

To analyse the performance of the regression model, I've calculated the R^2 and the MSE with the initial regression (train/test 70/30) and the 5-fold cross validation, both with $\alpha = 1.5$. Printing the R^2 and MSE result,I can see that in every case there's a better performance using the data aggregated, so looking only this one, all model (with exception of data with `track_genre` encoded) has similar performance, with $MSE \approx 406$, $R^2 \approx 0.032$ from the initial regression and a better MSE from the best regression of the 5-fold with $MSE \approx 361$, $R^2 \approx 0.035$.
However these performance aren't so good as R^2 is close to 0.
Instead, looking the result with `track_genre` encoded in the dataset, I've obtained good result from the initial regression with $MSE \approx 284$, $R^2 \approx 0.32$, while, surprisingly, from the 5-fold cross validation I've a worse result.
Even tough there's a performance's improvement in this case respect to the previous cases, the R^2 is still closer to 0 than 1

```
r_squared = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
print('[1] r2 Max: ',cv_acc['test_r2'].max(), '\t mse Min:
    ',-cv_acc['test_neg_mean_squared_error'].max())
print('[1] r2 Min: ',cv_acc['test_r2'].min(), '\t msa Min:
    ',-cv_acc['test_neg_mean_squared_error'].min())
print('[1] r2 Mean: ',cv_acc['test_r2'].mean(),'\t mse Mean:
    ',-cv_acc['test_neg_mean_squared_error'].mean())
print('R^2 1:\t    ',r_squared,'\t  MSE  : ',mse)
```

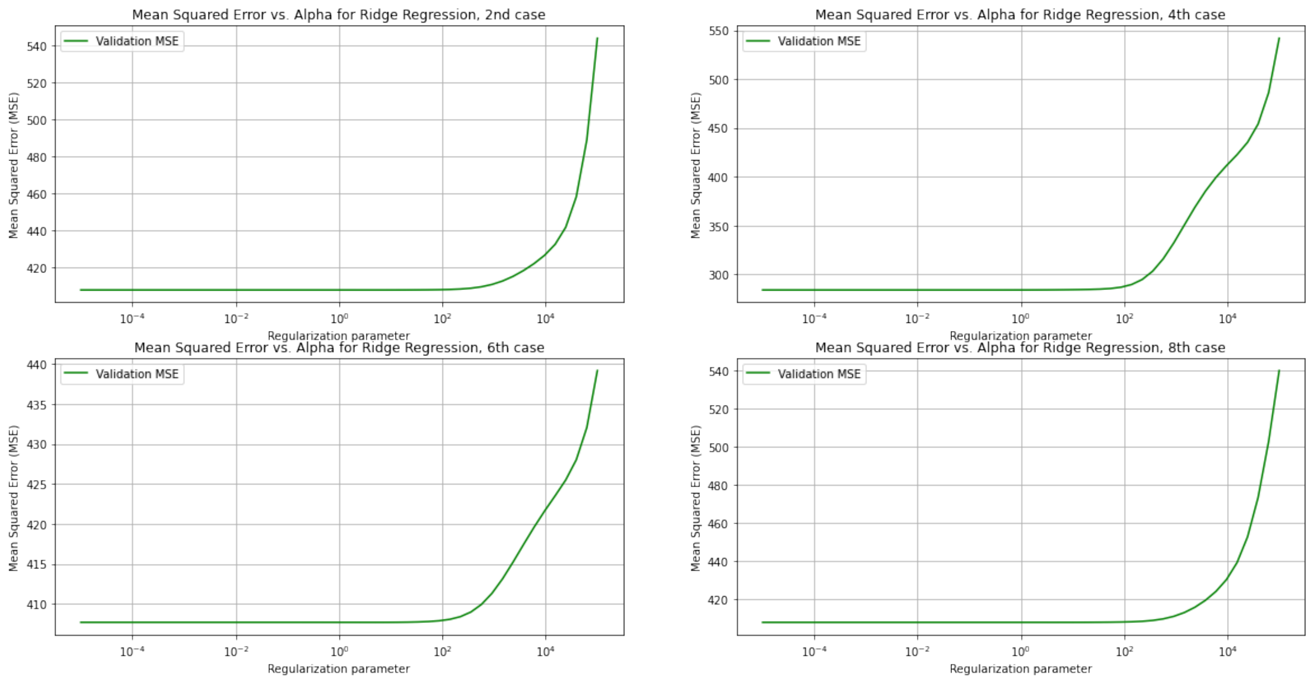
Code 9: Prints of R^2 and MSE

```
[1] r2 Max:   0.034220639926223595  mse Min:  433.89318544598825
[1] r2 Min:  -0.04853735794663705   mse Min:  587.8168766542688
[1] r2 Mean: -0.014924847946155052  mse Mean: 497.18971938355253
R^2 1:        0.02369507670110793     MSE  :  483.09981632856284
------------------------------------------------------------------
[2] r2 Max:   0.030127891626122727  mse Min:  360.42794591599625
[2] r2 Min:  -0.05488111644595883   mse Min:  494.99028964139654
[2] r2 Mean: -0.01524068342213376   mse Mean: 423.37648718000935
R^2 2:        0.03277967414778016     MSE  :  406.983102391591
------------------------------------------------------------------
[3] r2 Max:   0.003644453434414552  mse Min:  454.98875443926545
[3] r2 Min:  -0.24621190374164348   mse Min:  698.6345153608387
[3] r2 Mean: -0.08516133563156687   mse Mean: 533.5008876427408
R^2 3:        0.25884499752898293     MSE  :  366.74182114630963
------------------------------------------------------------------
[4] r2 Max:   0.01600315253119511   mse Min:  399.0114607765572
[4] r2 Min:  -0.3471668339381191   mse Min:  558.7968523912663
[4] r2 Mean: -0.12836662169987167   mse Mean: 469.2514636881733
R^2 4:        0.32387472519014293     MSE  :  284.49729042349105
------------------------------------------------------------------
[5] r2 Max:   0.03454335381309215   mse Min:  433.7482006159642
[5] r2 Min:  -0.04584023107105906   mse Min:  586.3048497494312
[5] r2 Mean: -0.013339149764893189  mse Mean: 496.39127065422383
R^2 5:        0.02626784488893097     MSE  :  481.8267470145276
------------------------------------------------------------------
[6] r2 Max:   0.030503546615719412  mse Min:  361.58419119397445
[6] r2 Min:  -0.05826515290467249   mse Min:  494.79623833533805
[6] r2 Mean: -0.015624280890445296  mse Mean: 423.47489481958536
```

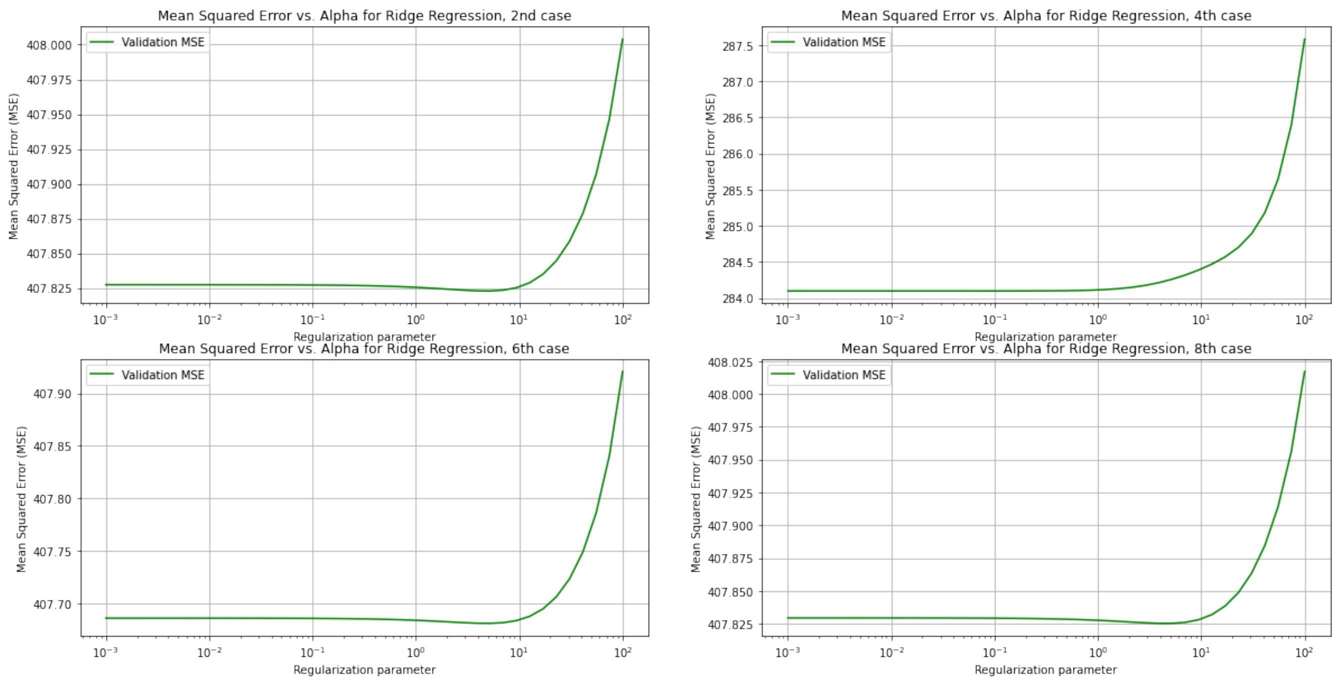
```
R^2 6:        0.03300374351020874     MSE :  406.8888193809525
-----------------------------------------------------------------
[7] r2 Max:   0.032952745150083396  mse Min:  434.4628092399652
[7] r2 Min:  -0.04777803717860296   mse Min:  587.3911964828741
[7] r2 Mean: -0.014831993388521213  mse Mean: 497.112017963805
R^2 7:        0.024317717382372872  MSE : 482.79171832395434
-----------------------------------------------------------------
[8] r2 Max:   0.02992115472460699   mse Min:  360.4546616019766
[8] r2 Min:  -0.05495930647803848   mse Min:  494.99396222868376
[8] r2 Mean: -0.015299860290828304  mse Mean: 423.40007531825677
R^2 8:        0.03277743935646982       MSE  :  406.9840427381453
-----------------------------------------------------------------
```

I've also analysed the models behaviour for different values of the regularization parameter to see which one could give the best performance. To choose a parameter is applied a new division in the test set, obtaining a validation set where testing and choosing the parameter, now the train/validation/test set division is with proportion 70/15/15.
To understand the performances I've calculated the MSE on the validation set for each parameter and for each (aggregated) case. At first, to have a wider overview, I choose 50 values from 10^{-5} to 10^{5} and then 50 values from 10^{-3} to 10^{2}, all spaced in a logarithmic way. I've noticed that for 2nd, 6th, 8th case there's a slight improvement between 1 and 10 and an increasing of the MSE after $\alpha = 10$. Looking in $\alpha \in [1, 10]$ I see that for these cases the optimal parameter should be something around 5. Instead in the 4th case the MSE increases after 2, I've tried to look until 10^{-1} and I observed a minimum between 10^{-2} and 10^{-1}.
But the MSE isn't decreasing so much, so I would say that a good regularization parameter should be pick smaller than 10 for 2nd, 6th, 8th case and smaller than 0.1 for the 4th case, to avoid an MSE explosion with big α.
At the end I've tried to perform the ridge regression with parameter equal to zero, i.e. a linear regression, to observe if there's some problem for the matrix inversion in the formula used to obtained the regression coefficient.
While in 2nd, 6th, 8th case the I've obtain a prediction model, for the 4th case there's some problem for the matrix inversion, so in this case it is necessary the implementation of a regression with some regularization parameter like ridge regression.

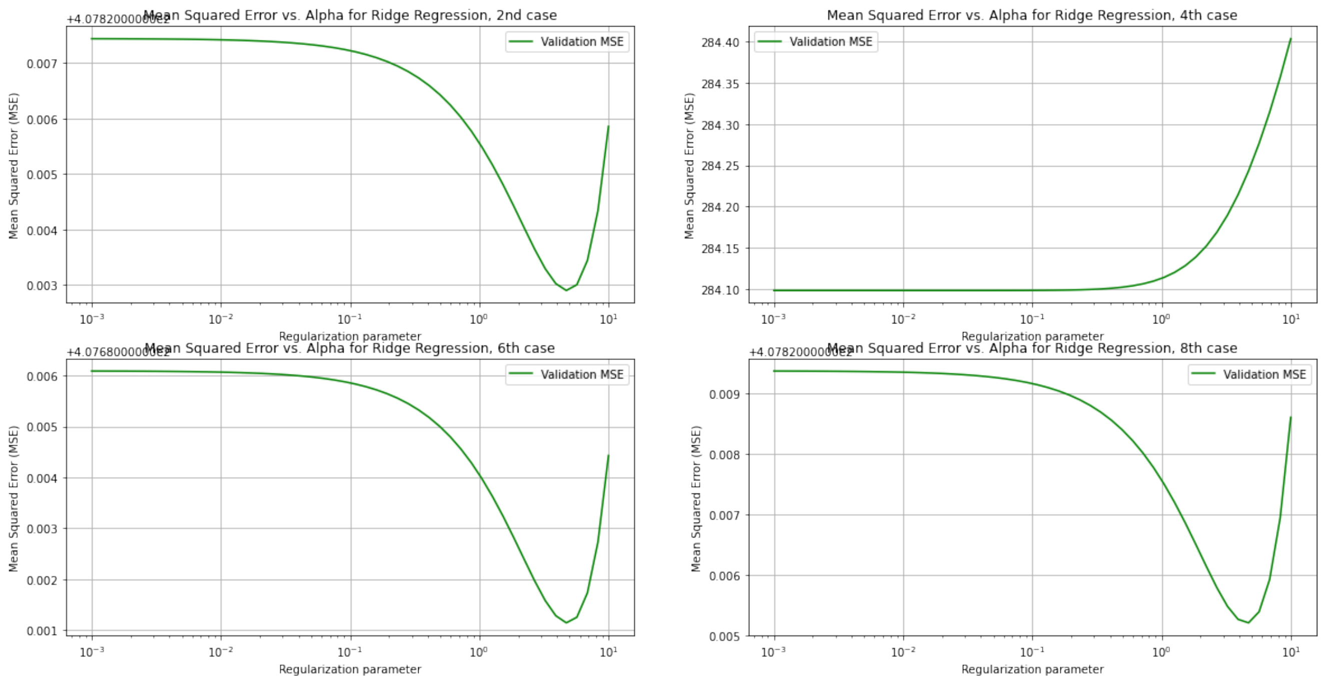
Mean Squared Error graphs in dependence of regularization parameter



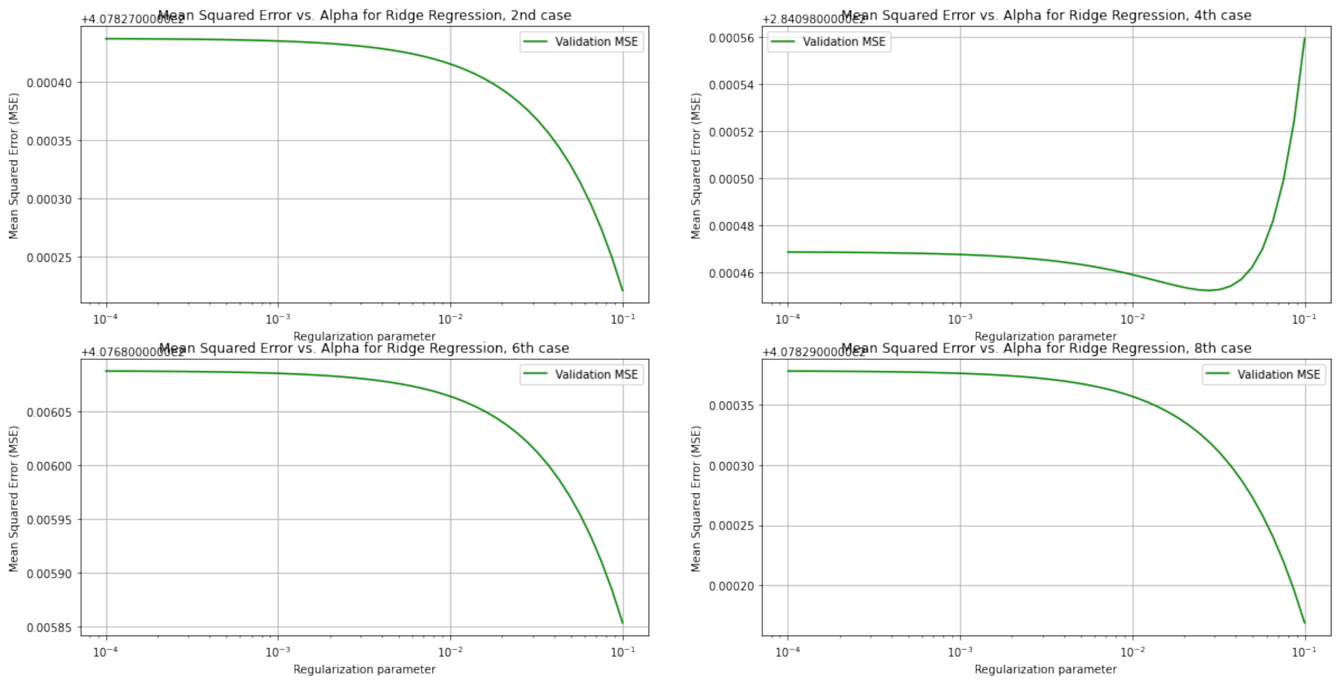
$\alpha \in [0.00001, 100\ 000]$



$\alpha \in [0.001, 100]$



$\alpha \in [0.001, 10]$



$\alpha \in [0.0001, 0.1]$