Project 3: Ridge Regression

Team RosAlene: Rosato Sara, Khachmadi Arsalene

Ridge regression is one of several regularized linear models and it serves as a model tuning technique. Regularization is the process of penalizing coefficients of variables either by removing them and or reduce their impact. The amount of the penalty can be fine-tuned using λ, the regularization parameter, which controls the strength of the regularization. Considering bias-variance trade-off, the choice of a correct λ for ridge regression is a crucial step. A very high value for the regularization term could lead to a model with high bias and low variance, because a large λ will shrink the coefficients of the model towards 0, which would be more stable but underfitting. Otherwise, with a very low value of λ, the model will have low bias and high variance, become more complex, and be more incline to overfitting.
The value of λ is passed to our model and the `init` function in our model initializes it and the weights.
We perform a ridge regression on the features X and the target variable y by using the function `ridge_fit`. There, we set the first element of the identity matrix to 0 in order to avoid penalizing the y intercept. The function returns the weights, given by: $\hat{w} = (X^T X + \lambda I)^{-1} X^T y$
To make predictions for new data points, we implemented the `ridge_predict` function, at first checking if the lengths are correct for doing matrix multiplication and then performing a multiplication between the input matrix X and the weights computed.
To test our model, we used the Olympics 100m dataset, considering as predictor the "year" and target the "time". At first, we split our dataset in train (80%) and test (20%), without shuffle to have in the test set only the data of the last years. After that, we centered and normalized our data to provide the non-relevance of the intercept effect (bias), adding it to the data through `polynomial features`.
To preserve the temporal order of the data points, the hyperparameter-tuning for λ is done by using Time Series Split cross-validation (already implemented in scikit-learn), where we split into k subsets, each one used as a validation set, while the remaining folds are used for training. The idea for time series splits is to divide the training set into two folds at each iteration on the condition that the validation set is always ahead of the training set (ref. [1]). For each value of λ, we train our Ridge model using the train data and predict on the validation, then we compute the mean absolute error and the mean squared error between the predicted values and the actual ones. After exploiting all the folds and λ values, we computed the average error for each λ across the subset and then selected the one corresponding to the lowest average error. Since this is done both for mean absolute error and mean squared error, we select the best λ among them considering the lower one, hence obtaining $\lambda = 1$.
With that value, we trained again our model and we predicted the result using the test set, previously unseen by the model. The values obtained are: $bias = -0.411$, $weight = 10.432$, $mse = 0.038$. The result is shown in the plot in Fig. 1.
We then compared our results with the scikit-learn implementation of Ridge, obtaining the same values.
In the end, we added a comparison with Lasso model, and we saw that it doesn't work well. This happens because for Lasso the value $\lambda = 1$ is too high, and this strong penalty leads to too

[1] https://medium.com/@soumyachess1496/cross-validation-in-time-series-566ae4981ce4

much shrinkage of the coefficients, causing underfitting. We saw, in fact, that with a smaller λ (e.g. $\lambda = 10^{-4}$) the Lasso model performs better, obtaining the same results in terms of bias and weights as our model.

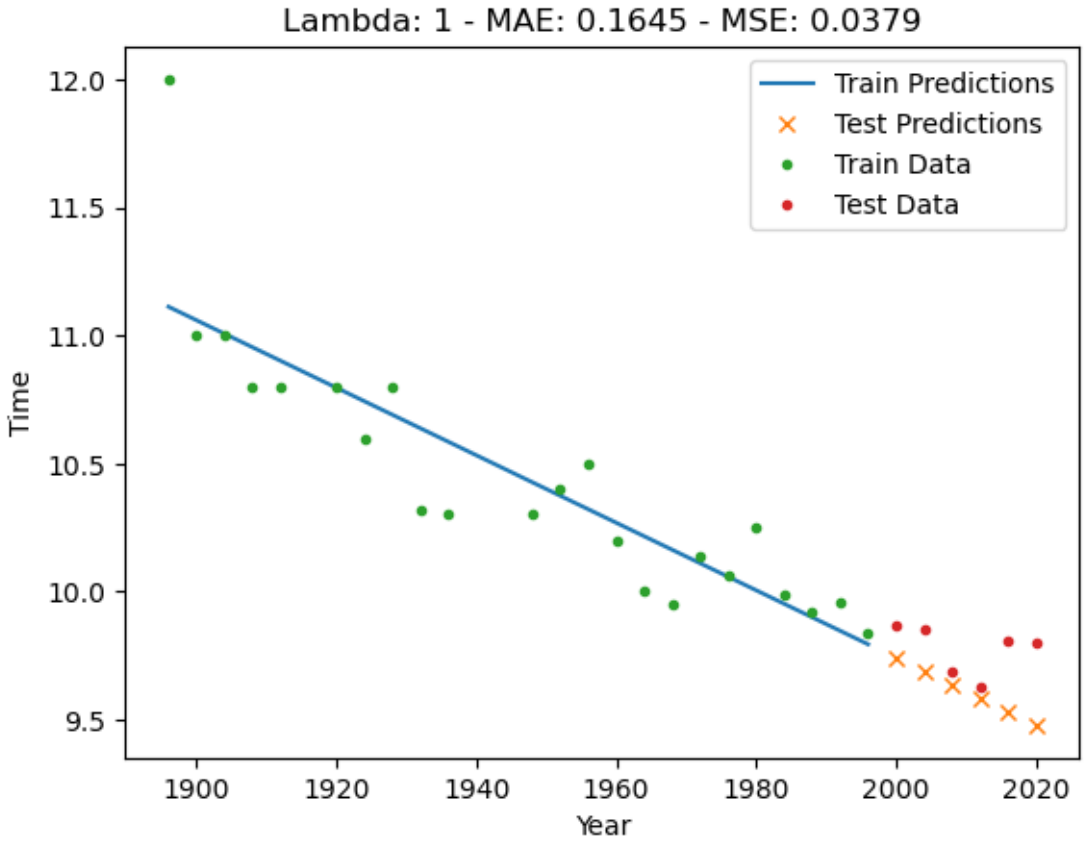

Figure 1: Plot