HOUSING PRICES
Ridge Regression from scratch
Statistical Methods for Machine Learning

Martina Viggiano (954603)

Università degli Studi di Milano - MSc Data Science and Economics

Abstract. In this paper we are going to present the project for Machine Learning module exam.
In particular, we will work on California Houses' prices dataset with the aim of implementing from scratch a ridge regression over the data and predict the label *median_house_value*, i.e. the house prices.

1 Theoretical Introduction

1.1 Linear Regression

Linear regression is the standard algorithm for regression that assumes a linear relationship between inputs and the target variable, where labels are real numbers and every predictor has a form $f : \mathbb{R}^d \to \mathbb{R}$. The function is composed of the inner product between the parameter that defines the predictor - the vector of real coefficients $\boldsymbol{w}$ - and the data point x:

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

Based on training set, we can learn linear predictors, by using the Empirical Risk Minimizer (ERM) algorithm: it minimizes the training error with respect to loss. In our case, the loss function we use is the squared loss.

$$\hat{\boldsymbol{w}} = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{t=1}^{m} (\boldsymbol{w}^T \mathbf{x}_t - y_t)^2 \tag{2}$$

To rewrite the formula in a more convenient form, we employ the vector v of predictions and the vector y of labels. The first one is composed by the predictions, that is $\boldsymbol{v} = (w^T x_1,..., w^T x_m)$ and the second is $\boldsymbol{y} = (y_1,...,y_m)$. Then, we can express the training error as the squared norm of the difference between the two vectors:

$$\sum_{t=1}^{m} (\boldsymbol{w}^T \mathbf{x}_t - y_t)^2 = \|\boldsymbol{v} \text{ - } \boldsymbol{y}\|^2 \tag{3}$$

Starting from the fact that $\boldsymbol{v} = \boldsymbol{S}\boldsymbol{w}$ where S is the matrix with dimension $m \times d$ where rows are data points, we can rewrite the formula as follows:

$$\hat{\boldsymbol{w}} = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \|\boldsymbol{Sw} \text{ - } \boldsymbol{y}\|^2 \tag{4}$$

Since the right hand side of the formula is a convex function of $\boldsymbol{w}$, we can compute the gradient of the squared difference with respect to $\boldsymbol{w}$. Then $\bigtriangledown\|\boldsymbol{Sw} \text{ - } \boldsymbol{y}\|^2 = 0$ for $\boldsymbol{w} = (S^TS)^{-1}S^Ty$ provided that S^TS is invertible; in particular, the data matrix is invertible in the case in which we a higher number of data points with respect to dimension. If this happens we have that $\hat{\boldsymbol{w}} = (S^TS)^{-1}S^Ty$.

1.2 Ridge Regression

A way to increase bias in the linear regression, making the model more stable, is to introduce a regularization in the ERM functional. This extension to linear regression invokes adding penalties to the loss function during training that encourages simpler models that have smaller coefficient values.

Ridge Regression is a popular type of regularized linear regression that includes a penalty. This has the effect of shrinking the coefficients for those input variables that do not contribute much to the prediction task.

As we just said, the equation of the ridge regression introduces a regularizer - which we will identify as α - multiplied to the squared norm of our vector learned $\boldsymbol{w}$: this is done to keep it the smallest possible.

$$\hat{\boldsymbol{w}} = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \|\boldsymbol{Sw} \text{ - } \boldsymbol{y}\|^2 + \alpha\|\boldsymbol{w}\|^2 \tag{5}$$

The regularization parameter $\alpha \geq 0$ when it is equal to 0, we go back to the linear regression solution, since we cancel off the additional part of the formula. On the other hand, with α increasing a lot, $\hat{\boldsymbol{w}}$ becomes a zero vector, since the right hand side of the formula will dominate.

Implementing again the derivation we did in the previous section, we obtain that:

$$\bigtriangledown(\|\boldsymbol{Sw} \text{ - } \boldsymbol{y}\|^2 + \alpha\|\boldsymbol{w}\|^2) = 2S^T(\boldsymbol{Sw} \text{ - } \boldsymbol{y}) + 2\alpha\boldsymbol{w} \tag{6}$$

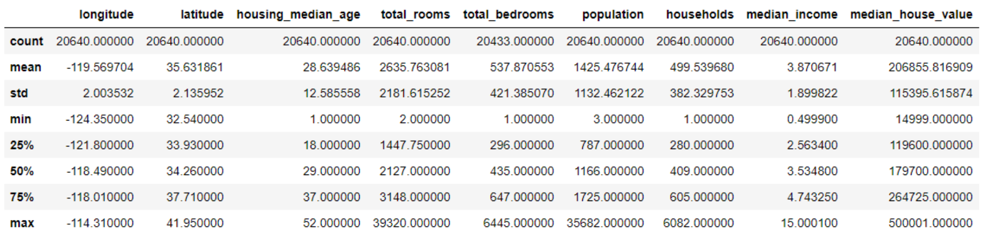
Then the gradient goes to zero for $\boldsymbol{w} = (\alpha I + S^TS)^{-1}S^Ty$ and we don't have to worry about invertibility, since the data matrix happens to be always invertible, because the sum of eigenvalues are always bigger than zero.

2 Introducing data

For this project we used the dataset *cal-housing*, which contains information about groups of houses. In particular, it consists of 20640 rows (observations) and 10 columns (9 variables and the target label).

The variables are as follows:

– longitude : numerical independent variable for longitude;
– latitude : numerical independent variable for latitude;
– housing_median_age : numerical independent variable representing age;
– total_rooms : numerical independent variable for amount of rooms per house;
– total_bedrooms : numerical independent variable for amount of bedrooms per house;
– population : numerical independent variable for population in the area;
– households : numerical independent variable for amount of households;
– median_income : numerical independent variable for income;
– median_house_value : numerical dependent variable for price;
– ocean_proximity : categorical independent variable for proximity with respect to ocean/sea;



	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

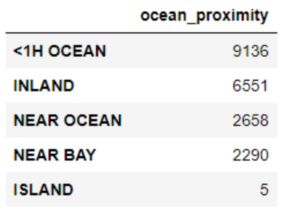
Fig. 1: Basic description of the original dataset.

Before moving to further explorative analysis, we checked for potential missing errors.

As a matter of fact, we found that the *total_bedrooms* variable contains 20433 instead of 20640 values, thus we have 207 missing observations. We can use different techniques to address this problem: the solution we choose was to replace missing values with the average value of the column itself.

Moreover, we may have two further issues with one of the variables of the dataset: the column *ocean_proximity* contains a categorical variable, which is not suitable for our project, and has unbalanced distribution of observations between categories.

As regards the latter problem, we noticed that the variable has different numerousness within the five classes: in particular, *ISLAND* category has only 5 observations, while the other four have more than 2 thousands rows each. For this reason we decided to remove the *ISLAND* class.



	ocean_proximity
<1H OCEAN	9136
INLAND	6551
NEAR OCEAN	2658
NEAR BAY	2290
ISLAND	5

Fig. 2: Distribution among classes in *ocean_proximity* variable.

Regarding the first issue - the variable being categorical - also in this case we may implement several techniques to solve the problem. One example can be to generate a dummy variable for every class, in order to cover all the categories: for each of those we would have value 0 for *false* or value 1 for *true*. However, in this case we chose to sort the values based on proximity and assign to each of them a numerical value, from 0 to 3. In particular, we assigned 0 to *NEAR BAY*, 1 to *NEAR OCEAN*, 2 to *<1H OCEAN* and 3 to *INLAND*, respectively from the nearest to the farthest from sea/ocean.

In this way we created a new ordinal numerical variable, which describes the level of proximity of the house with respect to bay, which is likely to have some impact on the price of the house.

2.1 Plotting variables

In this section we will display data in order to show the distribution of observations in our dataset.

First of all, we showed the relations existing between each of the variables, by deploying the *seaborn pairplot()* based on the correlation between them.

As we can see from Fig. 3, we have a sort of linear decreasing relation between *longitude* and *latitude*, and also a linear increasing relation within the variables *total_rooms*, *total_bedrooms*, *population* and *households*. About these last 4 variables, it is easily predictable they share some sort of correlation: a house with a higher number of rooms is likely to have also a higher number of bedrooms, while an area where the amount of households is higher may also imply a higher amount in population.

Later on we will examine better the correlation between them.

Then, we look at each of the variables separately. To do so, we plot one histogram for each of them (Fig. 4), displaying the frequency for every value of the variable and highlighting the average value drawing a vertical red line on the graph.

What we noticed was that two of those variables have a peculiarity in the distribution. The first one is *median_house_value* which has a peak at 50000 level, where have been summed all the observation equal to and above that value; this is also reasonable because the trend is clearly decreasing and there would be no point to have such a growth at that level.

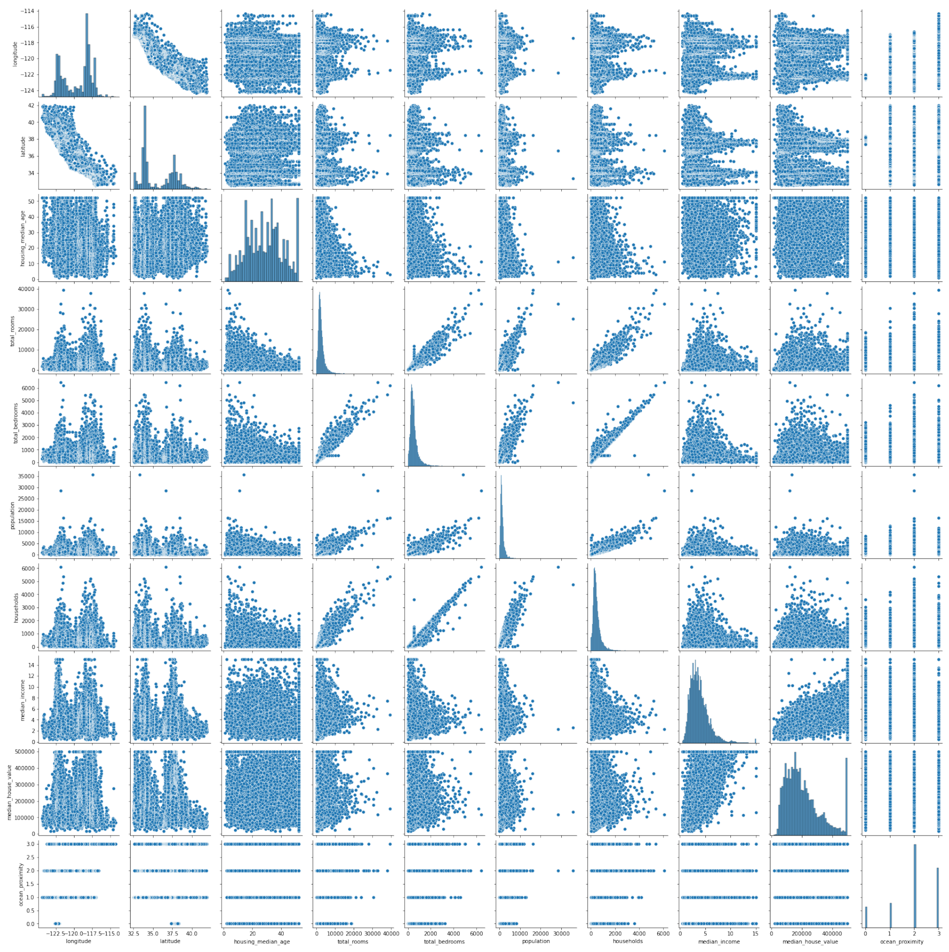
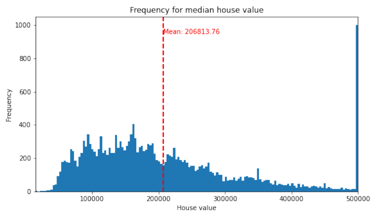
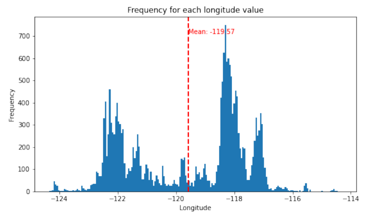


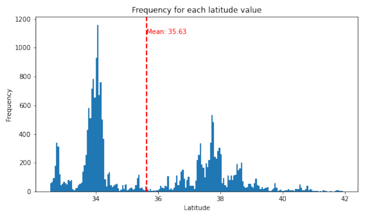
Fig. 3: Distribution and relations between variables.



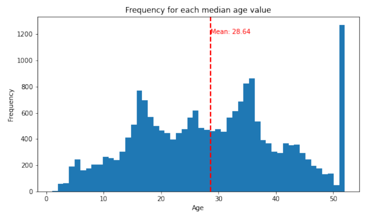
(a) Median house value (target label).



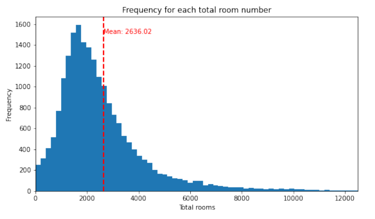
(b) Longitude.



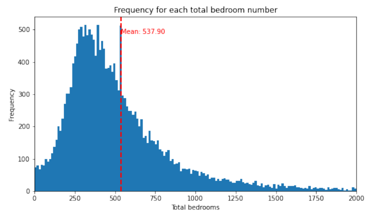
(c) Latitude.



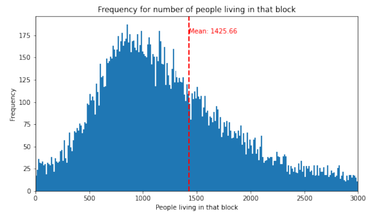
(d) Median age.



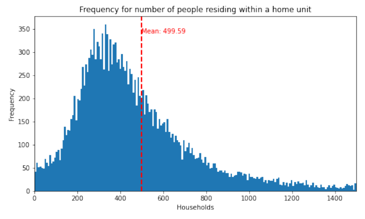
(e) Number of rooms.



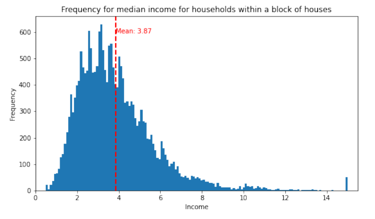
(f) Number of bedrooms.



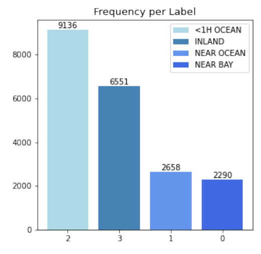
(g) Population.



(h) Number of households.



(i) Median income.



(j) Ocean proximity.

Fig. 4: Distribution of variables.

The same thing happens with the variable *housing_median_age*, in which the frequency bar steeply increases when it reaches the level 52.

To avoid distortions that may be produced by these outliers, we decided to remove all the observations corresponding to that values - 50000 for *median_house_value* and 52 for *housing_median_age*.

2.2 Correlation and Feature selection

Before moving to the regression part, we decided to study the feature importance and correlation between variables.

To perform the feature selection, we first scaled columns between 0 and 1 taking a random sample of only 2000 observations - to speed the computation - and then we modelled data by using the random forest method of *sklearn RandomForestRegressor()*. Then, with *feature_property* we retrieved the corresponding importance scores for each input feature and, based on the scores, we plotted a sorted bar plot, in which higher bars imply a higher "importance".

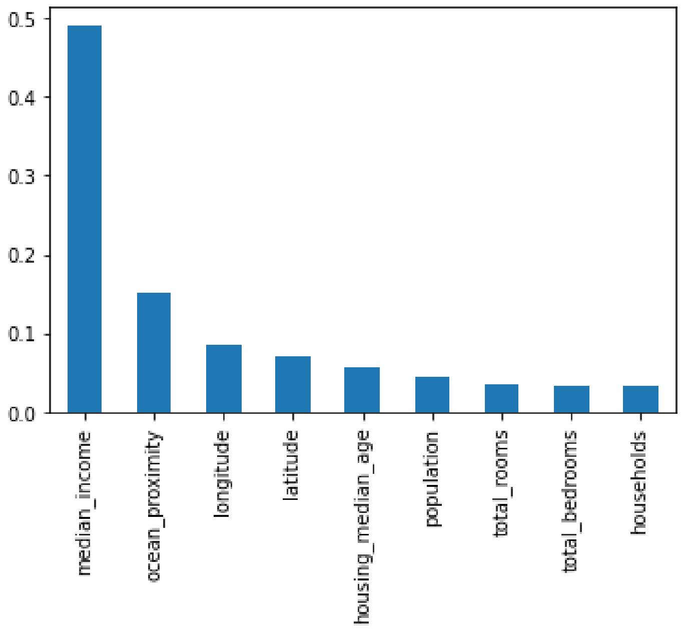
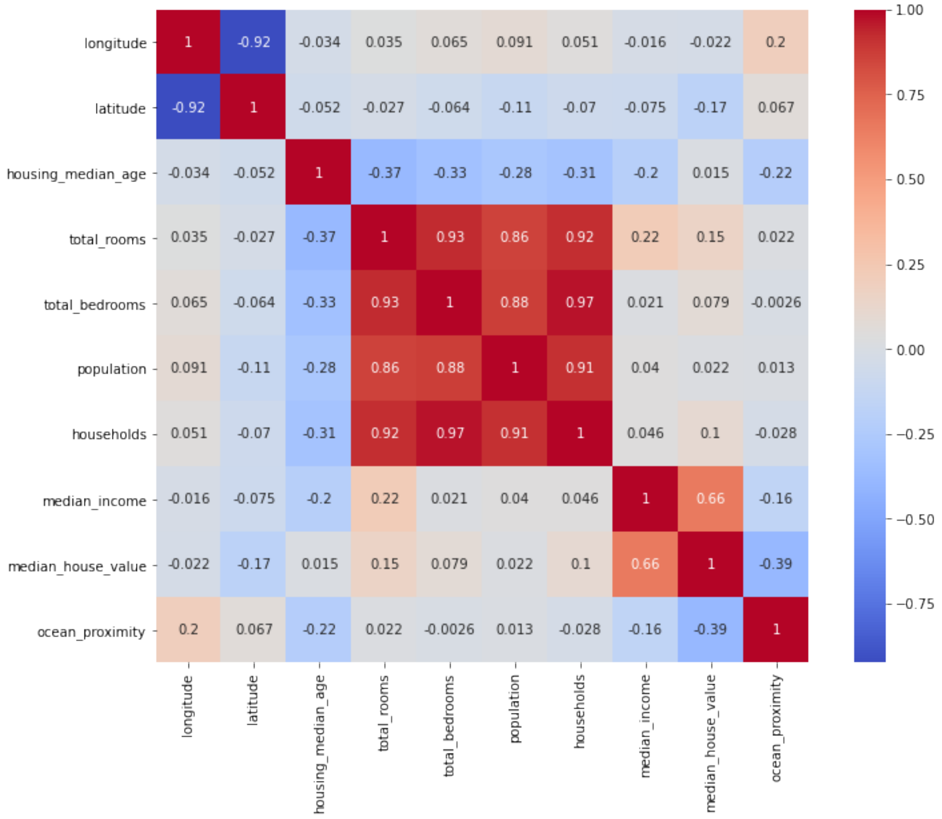


Fig. 5: Feature importance.

Looking at the graph above, we can see that the model assigned the higher score to *median_income* and the lower to *households*. Then, we looked for correlations between features. We computed the Pearson correlation coefficients of the sample with the method *corr()* of *pandas*: we plotted the correlation matrix, where in each cell we can see the correlation coefficient between two variables i and j:

$$\boldsymbol{r}_{i,j} = \frac{\sum_{t=1}^{m}(x_{i,t}-\mu_i)(x_{j,t}-\mu_j)}{\sqrt{\sum_{t=1}^{m}(x_{i,t}-\mu_i)^2}\sqrt{\sum_{t=1}^{m}(x_{j,t}-\mu_j)^2}} \tag{7}$$

As we noted in the section where we looked for linear relations, we can affirm that some variables are correlated one with the other. In particular, *longitude* and *latitude* are highly negatively correlated and there is a strong positive correlation between *total_rooms*, *total_bedrooms*, *population* and *households*.



	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
longitude	1	-0.92	-0.034	0.035	0.065	0.091	0.051	-0.016	-0.022	0.2
latitude	-0.92	1	-0.052	-0.027	-0.064	-0.11	-0.07	-0.075	-0.17	0.067
housing_median_age	-0.034	-0.052	1	-0.37	-0.33	-0.28	-0.31	-0.2	0.015	-0.22
total_rooms	0.035	-0.027	-0.37	1	0.93	0.86	0.92	0.22	0.15	0.022
total_bedrooms	0.065	-0.064	-0.33	0.93	1	0.88	0.97	0.021	0.079	-0.0026
population	0.091	-0.11	-0.28	0.86	0.88	1	0.91	0.04	0.022	0.013
households	0.051	-0.07	-0.31	0.92	0.97	0.91	1	0.046	0.1	-0.028
median_income	-0.016	-0.075	-0.2	0.22	0.021	0.04	0.046	1	0.66	-0.16
median_house_value	-0.022	-0.17	0.015	0.15	0.079	0.022	0.1	0.66	1	-0.39
ocean_proximity	0.2	0.067	-0.22	0.022	-0.0026	0.013	-0.028	-0.16	-0.39	1

Fig. 6: Correlation matrix.

Considering both the results above, we decided to remove from the dataset the variable *households*, because it has been associated to the lowest score in feature selection by the random forest model and it is correlated with other 3 variables of the dataset.

3 Ridge Regression from scratch

Before focusing on the implementation of the ridge regression from scratch, we decided to normalize our data since we found outliers and variety of numerousness between the variables. [5]

In this way every data point is transformed: we subtract the minimum value from each entry and then divide the result by the difference between the maximum value and the minimum value - i.e. the range.

After analyzing and cleaning data, we implemented the ridge regression from scratch.

We started from the equation we presented before - the one that retrieves the value of $\boldsymbol{w}$:

$$\boldsymbol{w} = (\alpha I + S^T S)^{-1} S^T y. \tag{8}$$

Looking at the formula (8), we see that we already have most of the components we need: α is arbitrary - so we can choose and set it - S consists of the matrix of data points, and y is the vector of the target label *median_house_value*.

However, to learn the vector $\boldsymbol{w}$ of coefficients, we still need to compute the identity matrix I: we defined it by using *numpy identity()* function on the dot product between S^T and S.

When we retrieved all the necessary components, we defined a *class* in which we computed $\boldsymbol{w}$ and specified which calculus the class *MyRidge* has to do in order to return the *mean squared error* and the vector v of predictions.

By default we set $\alpha = 1$, but then we also studied the changes of the loss with respect to different values of α, investigating the sensitivity of cross-validation risk estimate with respect to the choice of α.[1]

We assigned an interval of α values generating 10 candidates between 0.001 and 10 with and fitted a 5-fold cross validation, setting as *scoring* - that is to say the strategy to evaluate the performance of the cross-validated model - the *neg_mean_squared_error*. This metric measures the distance between the model and the data, returning the negated value of mean squared error regression loss, thus higher values are more desirable than lower values.

The result we obtained was that the best value of α is 0.001 and the best score retrieved by *GridSearchCV* is -0.017325, that is the average score of the best estimator in terms of minimum loss.

As we can see from Fig. 7, while on one hand the variance remains quite the same, bias increases for increasing values of α. In particular, we notice that we

[1] Part of the code used to display the following plots replicate some passages of second module of Coding for Data Science and Data Managements [1]. In particular reference is made to lesson 10.

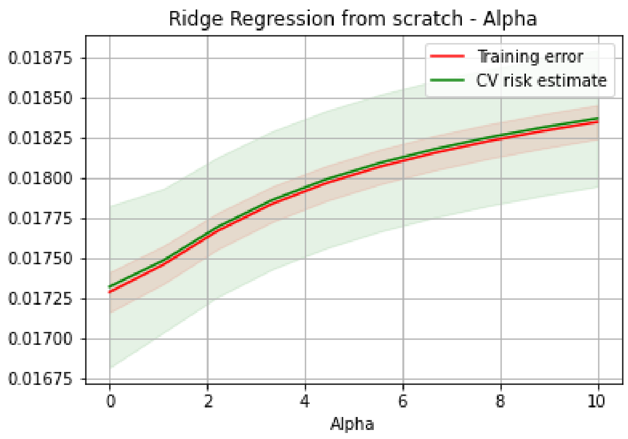


Fig. 7: Alpha values with respect to squared loss.

have quite steep curve, composed of both training error and cross-validation risk estimate curves moving in parallel.

After that, we compared our results with the *Ridge* model provided by *sklearn* setting both $\alpha = 1$: what be learned is that the mean squared error does not diverge much. The mean squared error we obtained in the predictor implemented from scratch is equal to 0.017404, while the one given by *sklearn* is 0.015404. The error of our model is higher, but pretty reasonable.

We also wanted to compare the two model on a plot: as displayed of Fig. 8, we plotted the learning curves of the *sklearn Ridge Regression()* and the model we deployed from scratch on a range from 4500 to 14855 of training size.

What we see is that for low size sets the model implemented from scratch seems to perform better, but after 5000 in training size the *sklearn* model is always superior in terms of CV risk estimate.

Then we repeated the analysis, but in this case with respect to *sklearn LinearRegression()*, instead of ridge. What we can observe from Fig. 9 is that the evolution is very similar to the plot before: for small values our ridge regression works better, but increasing the training size linear regression performs even better.

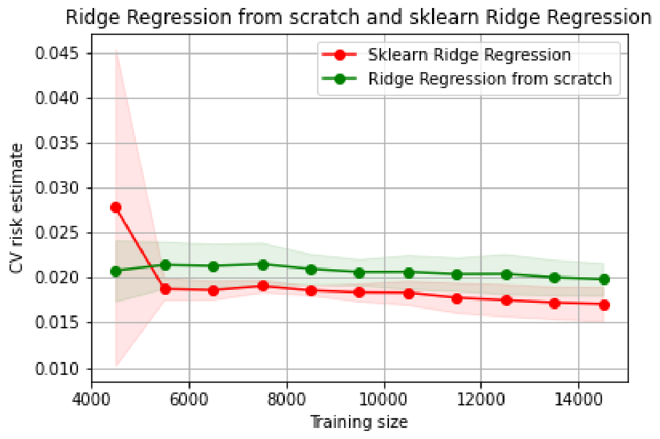


Fig. 8: Ridge Regression from scratch and sklearn Ridge Regression.

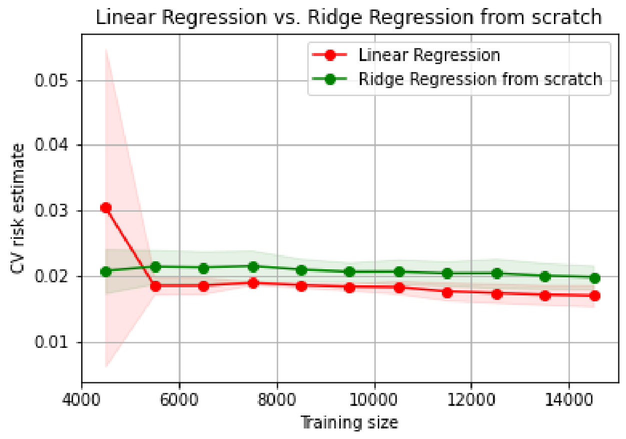


Fig. 9: Linear Regression and Ridge Regression from scratch.

4 PCA for improving the risk estimate

Due to the high correlations between variables we found in section 2.2, we can infer that our model may be quite unstable. The use of ridge regression instead of the simple linear model is already a good solution to handle the collinearity. Besides, we took off one highly correlated variable - *households* - and this should have already enhance the stability of the model.

However, with the purpose of improving the risk estimate, we exploit the Principal Component Analysis (PCA). It is based on linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space.

In particular, we have two error scores to monitor: one for the validation set and one for the training sets. Plotting the evolution of the two error scores as training size change, we obtain two curves: the learning curves. [4]

Thus, we displayed the learning curves describing the squared loss - also in this case we considered the *neg_mean_squared_error* as scoring - for training set values from 4500 to 14855.

Typically if we increase the training set size, the model starts having higher training error and lower validation error because it fits better the validation set having more data on which it can be estimated.

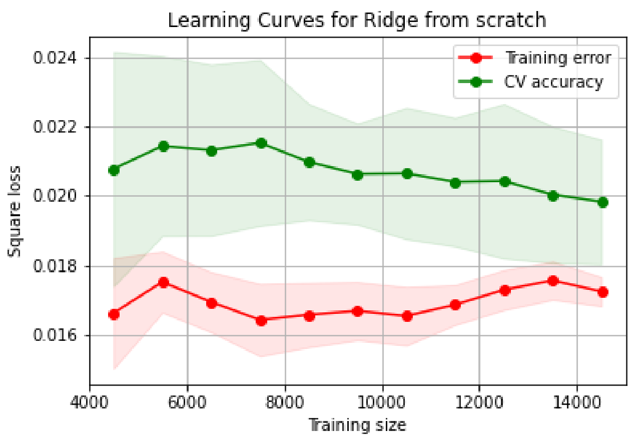
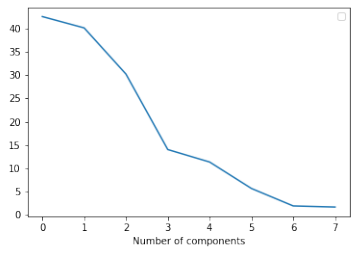
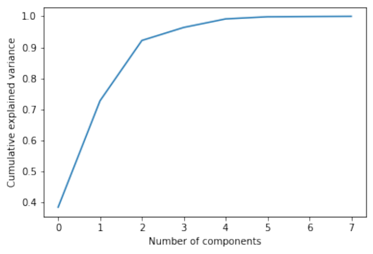


Fig. 10: Ridge Regression from scratch learning curves.

As you can see in Fig. 10, both the curves are not so linear and they do not seem to converge. This may suggest that adding more data may lead to a better model



(a) PCA Singular Values.



(b) PCA Cumulative Explained Variance.

Fig. 11: Choosing the number of components.

To increase the stability of regressors, we used PCA to reduce the number of features by keeping only the most informative ones and make the process more stable by increasing bias.

In order to get to know what are the most informative directions, we computed the *pca.singular_values_*, which correspond to the measures of how informative are the components. An alternative for choosing the number of components is to look at the cumulative explained variance ratio as a function of the number of components.

As we can see from Fig. 11 (a), after 6 principal components directions we do not get a lot of information from the following component.

Thus we projected the data points on the 6 principal components and look at the new learning curves.

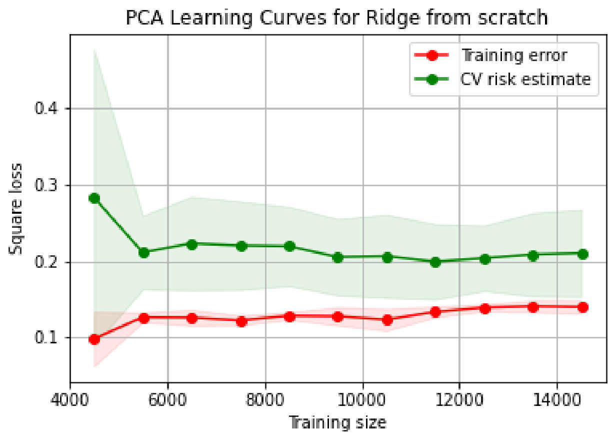


Fig. 12: PCA Learning Curves for Ridge from scratch.

The gap between the two lines in Fig. 12 seems to become more and more narrow as the training set size increases, than the two lines may converge by adding further training data. However, in terms of squared loss and variance, the deployment of PCA does not seem to be beneficial.

In fact, we obtained to increase the squared error values especially for cross validation curve worsening the overall error. This could be justified by the fact that in our project we do not have a high number of features, thus reducing dimensionality may make us lose valuable information.

5 Declaration

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.

References

1. Cesa-Bianchi N.: Coding for Data Science and Data Management lectures and notebooks. `https://github.com/nicolo63/CDS`.
2. Cesa-Bianchi N.: Statistical Methods for Machine Learning lectures. `http://cesa-bianchi.di.unimi.it/MSA/`
3. Aaron Blythe, Josh Janda, Jeanette Pettibone: Predicting Housing Prices - Data Analysis Project, 07/21/2019 `https://rstudio-pubs-static.s3.amazonaws.com/520912_bbd05dad7ffe4eb08ab51ee2c5bd90bb.html`.
4. Olteanu A., Tutorial: Learning Curves for Machine Learning in Python, Dataquest website, 01/03/2018 `https://www.dataquest.io/blog/learning-curves-machine-learning/`.
5. Compare the effect of different scalers on data with outliers, scikit learn web page `https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#sphx-glr-auto-examples-preprocessing-plot-all-scaling-py`.