Ridge regression for housing prices

Giovanni Laganà, matr. 928792
giovanni.lagana@studenti.unimi.it

F943X - Statistical Methods for Machine Learning
Università degli Studi di Milano

October 1, 2020

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.

1 Introduction

The aim of this project is to create a **ridge regression** model capable of inferring housing prices. In general, many software libraries have been developed and are available today to do this task but in this project the implementation has been carried out from scratch.
The first part of the document introduces the theoretical tools that have been used, secondly, the dataset is presented; afterwards, it is shown how it has been preprocessed, and how ridge regression has been implemented.
The second part of the report deals with the different techniques that have been used to tune the regularization parameter, like cross validation and nested cross validation.
Moreover, principal component analysis has been performed with the aim of trying to reduce the cross-validated risk estimate.
Finally, results are presented, including a comparison with *Scikit-learn* library.

2 Ridge regression: a supervised learning method

Ridge regression is a special case of linear regression and belongs to the family of supervised learning methods. We are, so, dealing with a numerical set of labels attached to data from the past and we want to predict a value which is as close as possible to the label of future unseen data.

Training, test and validation set Let us assume that we have dataset S organised as follows:

$$S = \{(\underset{\to}{x}_1, y_1), (\underset{\to}{x}_2, y_2), ..., (\underset{\to}{x}_m, y_m)\}$$

where $\underset{\to}{x}_i \in \mathbb{R}^d$ is the vector of features at i-th row of S and $y_i \in \mathbb{R}$ is the relative single label. Each row of the dataset represents an observation.
S is randomly partitioned into training, validation and test set, with the purpose of isolating training, tuning and final tests on unseen data.

Training set The training set is used to fit the model.

Validation set The validation set is used to evaluate the performances of the trained model on new data, it is typically used to tune parameters in order to get the best configuration for the model that minimizes the loss function.

Test set The test set aim is to test the predictor on unseen data such that it is possible to evaluate its capability of generalization.

It is really important to split these sets such that the predictor never sees test set, otherwise *data leakage* phenomenon occurs, which leads to overfitting. Typically, this happens because the predictor tries to automatically adapt itself on the data which it should not know and for which it is trained to work on.

Loss function It is a function $\ell(\hat{y}, y)$ that compares a predicted label $\hat{y}$ to the real one y.
Because of this, $\ell(\hat{y}, y)$ gives a quantifiable value of how much the prediction is close to the correct value.
The loss function is, typically, used to compute the training and the test errors.
There are plenties of loss functions, for this project we used the square loss[1]:

$$\ell(\hat{y}, y) = (\hat{y} - y)^2 \tag{1}$$

Statistical risk We assume that the dataset S is a random sample of the form $(X,\ Y)$ drawn from an unknown joint probability distribution $\mathcal{X}\times\mathcal{Y}$, we call this distribution D.
We have a learning task, described by $(D,\ \ell)$ where ℓ is the loss function that we use and h is a predictor such that $h : \mathcal{X} \to \mathcal{Y}$, then:

$$\ell_D(h) = E[\ell(Y, h(X))] \tag{2}$$

defined as the expected value of loss function, the statistical risk measures the performances of a predictor h with respect to a given learning task $(D,\ \ell)$; since D is unknown, we cannot compute exactly this value but we can estimate it through the test error ([4], [6]).
Test error is a sample mean of loss function and, thanks to the law of large numbers, we know that, with m increasing, sample mean tends to the expected value.

MSE We use *mean squared error*, an error function that, through square loss, computes training and test error, according to which set we use:

$$MSE(\underset{\to}{\hat{y}}, \underset{\to}{y}) = \frac{1}{m}\ell(\underset{\to}{\hat{y}}, \underset{\to}{y}) = \frac{1}{m}\left\|\underset{\to}{\hat{y}} - \underset{\to}{y}\right\|^2 = \frac{1}{m}\sum_{j=1}^{m}(\hat{y_j} - y_j)^2 \tag{3}$$

This is going to be our risk estimator.

[1]vectorial form: $\ell(\underset{\to}{\hat{y}}, \underset{\to}{y}) = \left\|\underset{\to}{\hat{y}} - \underset{\to}{y}\right\|^2 = \sum_{j=1}^{m}(\hat{y_j} - y_j)^2$ where $\underset{\to}{\hat{y}}$, $\underset{\to}{y}$ are vectors respectively containing m predictions and m real labels

Linear regression Linear regression is the case where predictors are linear functions $h : \mathbb{R}^d \to \mathbb{R}$ where d is the number of features in input. The goal of regression is to predict a quantitative value, instead of a fixed symbolic set of classes, like happens in categorization.
The prediction is carried out by learning a linear relationship between features and the label to predict[2]:

$$\hat{y}_i = \underset{\rightarrow}{w} \cdot \underset{\rightarrow}{x}_i \tag{4}$$

where $\hat{y}_i$ is the single prediction of the label at i-th row, computed through $\underset{\rightarrow}{w}$, a vector of weights, and $\underset{\rightarrow}{x}_i$, the vector of i-th observation.

ERM The *empirical risk minimization* is used to fit the model through a closed form that allows us to compute the weight vector that minimize ℓ on training set.
Let X be the *design matrix* of the form $[\underset{\rightarrow}{x}_1, \underset{\rightarrow}{x}_2, ..., \underset{\rightarrow}{x}_m]$, then:

$$\hat{\underset{\rightarrow}{w}} = \arg\min \ell(\hat{\underset{\rightarrow}{y}}, \underset{\rightarrow}{y}) = \arg\min \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \underset{\underset{\rightarrow}{w} \in \mathbb{R}^d}{\arg\min} \sum_{i=1}^{m} (\underset{\rightarrow}{w} \cdot \underset{\rightarrow}{x}_i - y_i)^2 = \underset{\underset{\rightarrow}{w} \in \mathbb{R}^d}{\arg\min} \left\| X\underset{\rightarrow}{w} - \underset{\rightarrow}{y} \right\|^2 \tag{5}$$

Since $\left\| X\underset{\rightarrow}{w} - \underset{\rightarrow}{y} \right\|^2$ is a convex function, we compute for which $\underset{\rightarrow}{w}$ its gradient is equal to $\underset{\rightarrow}{0}$:

$$\nabla \left\| X\underset{\rightarrow}{w} - \underset{\rightarrow}{y} \right\|^2 = 2X^T(X\underset{\rightarrow}{w} - \underset{\rightarrow}{y}) = \underset{\rightarrow}{0} \iff X^T X \underset{\rightarrow}{w} = X^T \underset{\rightarrow}{y} \tag{6}$$

if $X^T X$ is non singular (i.e. invertible) then:

$$\hat{\underset{\rightarrow}{w}} = (X^T X)^{-1} X^T \underset{\rightarrow}{y} \tag{7}$$

when $X^T X$ is nearly singular this vector can be very unstable when the dataset is perturbed (e.g. changing a few examples) ([3]). Since instability leads to a variance error increase and so overfitting, we need ridge regression.

Regularization with ridge regression In order to fix the instability issue of the vector, we introduce a regularization term:

$$\hat{\underset{\rightarrow}{w_\alpha}} = \underset{\underset{\rightarrow}{w} \in \mathbb{R}^d, \alpha > 0}{\arg\min} \left\| X\underset{\rightarrow}{w} - \underset{\rightarrow}{y} \right\|^2 + \alpha \left\| \underset{\rightarrow}{w} \right\|^2 \tag{8}$$

so if we now minimize it, we obtain:

$$\nabla \left\| X\underset{\rightarrow}{w} - \underset{\rightarrow}{y} \right\|^2 + \alpha \left\| \underset{\rightarrow}{w} \right\|^2 = 2(X^T X \underset{\rightarrow}{w} - X^T \underset{\rightarrow}{y}) + 2\alpha \underset{\rightarrow}{w} = \underset{\rightarrow}{0} \iff (X^T X + \alpha I)\underset{\rightarrow}{w} = X^T \underset{\rightarrow}{y} \tag{9}$$

such that the update will look like:

$$\hat{\underset{\rightarrow}{w}}_\alpha = (X^T X + \alpha I)^{-1} X^T \underset{\rightarrow}{y} \tag{10}$$

- when $\alpha \to 0$ we have (7)
- when $\alpha \to \infty$ the solution becomes the zero vector

The idea is that α is used to control the bias error of the algorithm to balance variance error and so prevent overfitting ([3], [7]).

[2] The intercept value is considered by adding a feature to the dataset with constant value 1.

3 Dataset

The dataset `cal-housing` contains information about 20640 houses in USA and it is composed by the following attributes:

- `longitude`: a measure of how far west a house is; a higher value is farther west
- `latitude`: a measure of how far north a house is; a higher value is farther north
- `housing_median_age`: median age of a house within a block; a lower number is a newer building
- `total_rooms`: total number of rooms within a block
- `total_bedrooms`: total number of bedrooms within a block
- `population`: total number of people residing within a block
- `households`: total number of households, a group of people residing within a home unit, for a block
- `median_income`: median income for households within a block of houses (measured in tens of thousands of US Dollars)
- `median_house_value`: median house value for households within a block (measured in US Dollars)
- `ocean_proximity`: location of the house w.r.t ocean/sea

Since we want to infer housing prices, we will consider `median_house_value` as label.

Data visualization In Figure 1 we can see the head of the dataset showing the first 10 rows.

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY
5	-122.25	37.85	52.0	919.0	213.0	413.0	193.0	4.0368	269700.0	NEAR BAY
6	-122.25	37.84	52.0	2535.0	489.0	1094.0	514.0	3.6591	299200.0	NEAR BAY
7	-122.25	37.84	52.0	3104.0	687.0	1157.0	647.0	3.1200	241400.0	NEAR BAY
8	-122.26	37.84	42.0	2555.0	665.0	1206.0	595.0	2.0804	226700.0	NEAR BAY
9	-122.25	37.84	52.0	3549.0	707.0	1551.0	714.0	3.6912	261100.0	NEAR BAY

Figure 1: Dataset

The main benefit of this step is that we have quickly realised the presence of a categorical column: `ocean_proximity`; recalling the definition of regression, we must provide to the model an input of numerical values ($\mathbb{R}^d$), therefore, we will handle this issue.

Outliers detection The graphic tool we have chosen in order to detect outliers at first sight is the boxplot: in its definition, boxplot represents outliers as dots all those values x that are

$$(x > q_3 + 1.5 \times iqr) \vee (x < q_1 - 1.5 \times iqr) \tag{11}$$

where q_3 and q_1 are respectively the 3^{rd} and the 1^{st} quartiles, whereas iqr indicates the interquartile range.

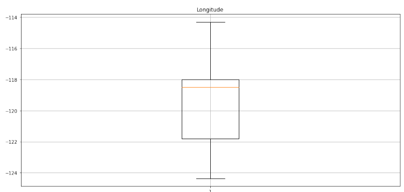

Figure 2: Longitude

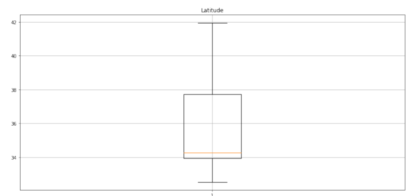

Figure 3: Latitude

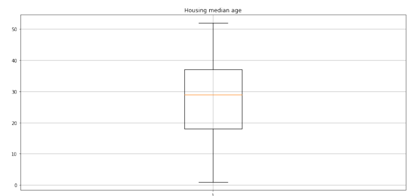

Figure 4: Median age

For the first three attributes (Figures 2, 3, 4), we do not see any outlier.

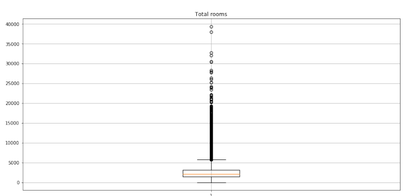

Figure 5: Total rooms

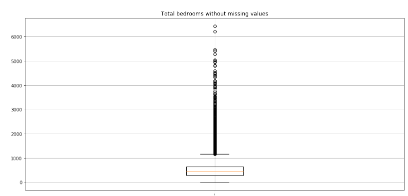

Figure 6: Total bedrooms

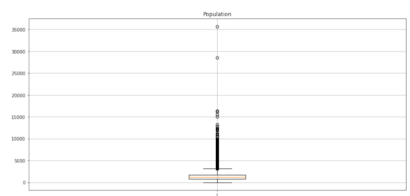

Figure 7: Population

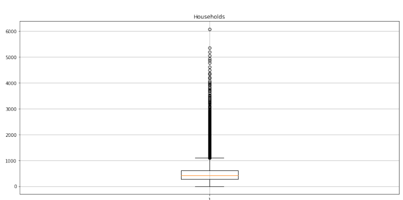

Figure 8: Households

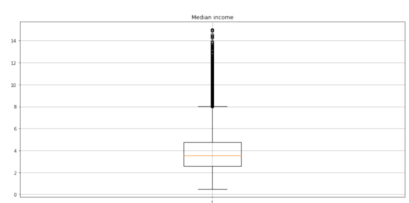

Figure 9: Median income

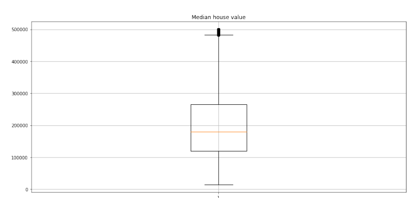

Figure 10: Median value

On the contrary, we notice outliers in the last six attributes, respectively `total_rooms, total_bedrooms, population, households, median_income` and `median_house_value` (Figures 5, 6, 7, 8, 9, 10).

4 Preprocessing

Once we detected anomalies, the next step is doing preprocessing.
Before applying the pipeline, we checked the presence of duplicates in the dataset, but luckily, there were not any. The preprocessing pipeline is composed by:

- handling categorical values
- outliers removal

- handling missing values
- scaling
- adding intercept
- shuffling

Handling categorical values We have decided to go ahead with one-hot encoding: a technique that replaces categorical values with truth values by adding as many columns as many categories and setting 1 if that value occurs, 0 otherwise.

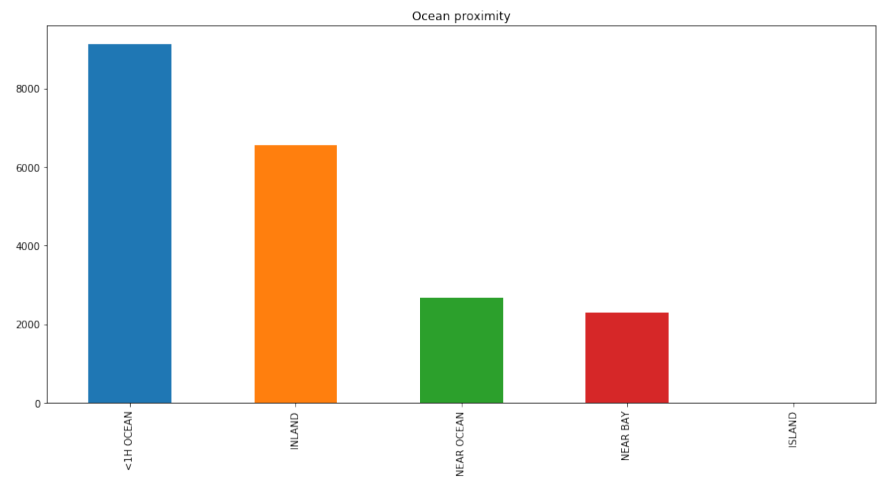

Figure 11: `ocean_proximity` distribution

housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	0	0	0	1	0
21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	0	0	0	1	0
52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	0	0	0	1	0
52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	0	0	0	1	0
52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	0	0	0	1	0
52.0	919.0	213.0	413.0	193.0	4.0368	269700.0	0	0	0	1	0
52.0	2535.0	489.0	1094.0	514.0	3.6591	299200.0	0	0	0	1	0
52.0	3104.0	687.0	1157.0	647.0	3.1200	241400.0	0	0	0	1	0
42.0	2555.0	665.0	1206.0	595.0	2.0804	226700.0	0	0	0	1	0
52.0	3549.0	707.0	1551.0	714.0	3.6912	261100.0	0	0	0	1	0

Figure 12: Dataset after one-hot encoding

We have applied this encoding method (Figure 12) for these reasons:

1. the number of possible values of `ocean_proximity` is small (see Figure 11)
2. an alternative label encoding forces us to choose discrete values according to some criterion (e.g. from the furthest to the closest w.r.t. the ocean) which could influence the results
3. the definition of the possible categories are in some cases vague and subjective for the dataset annotator (e.g. let us take `<1H OCEAN`, `NEAR OCEAN` and `NEAR BAY`: we do not know which category really indicates a closer location to the ocean)

the drawback is that linear models suffer immensely from the curse of dimensionality so adding features could be risky but, since we are talking about 4 more features (5 categories minus the drop of the original column) we think that this is the best trade off.

Outliers removal Outliers are values that show huge variance with respect to the rest of the distribution, this means that

- they could be due to an error
- they could be due to a legitimate property but they could mislead the regressor in catching rare dependencies between features and labels

in both cases leaving them would not be an acceptable solution, since their presence may impact on the final behavior of our regressor, for this reason we decide to remove them. Outliers are contained in `total_rooms, total_bedrooms, population, households, median_income` and `median_house_value`. We had confirmed their presence through boxplots, so we compute quartiles and interquartile range for the involved features so that we can use (11) to filter outliers out.

total_rooms	total_bedrooms	population	households	median_income	median_house_value
6.23 %	6.16 %	5.79 %	5.91 %	3.30 %	5.19 %

Table 1: Percentage of outliers

We notice from Table 1 that the percentage of outliers varies from 3 to 6 percent with respect to the whole dataset, which is a non-negligible fraction.
Moreover, they are contemporary present in multiple columns at the same rows: `median_house_value` and `median_income` for example, or the same rows even include missing values as well.
In total, 3925 rows with at least one outlier have been filtered out, that is to say the 19.02 % of data[3].

Missing values Another anomaly which has been detected in this dataset is missing values, in particular they are contained only in `total_bedrooms` column, they actually represent the 0.97 % of the total dataset (a sample is taken in Figure 13).

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	<1H OCEAN	INLAND
290	-122.16	37.77	47.0	1256.0	NaN	570.0	218.0	4.3750	161900.0	0	0
341	-122.17	37.75	38.0	992.0	NaN	732.0	259.0	1.6196	85100.0	0	0
563	-122.24	37.75	45.0	891.0	NaN	384.0	146.0	4.9489	247100.0	0	0
696	-122.10	37.69	41.0	746.0	NaN	387.0	161.0	3.9063	178400.0	0	0
738	-122.14	37.67	37.0	3342.0	NaN	1635.0	557.0	4.7933	186900.0	0	0
1097	-121.77	39.66	20.0	3759.0	NaN	1705.0	600.0	4.7120	158600.0	0	1
1456	-121.98	37.96	22.0	2987.0	NaN	1420.0	540.0	3.6500	204100.0	0	1
1493	-122.01	37.94	23.0	3741.0	NaN	1339.0	499.0	6.7061	322300.0	0	0
1606	-122.08	37.88	26.0	2947.0	NaN	825.0	626.0	2.9330	85000.0	0	0
2028	-119.75	36.71	38.0	1481.0	NaN	1543.0	372.0	1.4577	49800.0	0	1

Figure 13: Missing values

[3]Note that this percentage is different than the sum of percentages in Table 1, which is 32.59 %, because many rows with multiple outliers and/or missing values have been found

In order to handle missing values, we propose two solutions:

1. a reduced dataset without missing values
2. a dataset of same size but with replaced missing values

Our goal is to compare the final results to see actually how much the two strategies differ.

Replacing missing values We have used the stochastic regression imputation method: in general, `total_bedrooms` must be a subset of `total_rooms` so we can infer the value of the latter from the former; the correlation coefficient between these two features is 0.89, confirming that there is a positive linear relationship.
Let us consider μ, σ respectively as mean and standard deviation of a distribution; we compute the distribution of fractions of total bedrooms with respect to the number of total rooms for the two vectors and then we compute their μ and σ. Our aim is to find a suitable range that describes how many bedrooms there are, on average, with respect to the number of rooms.
Since $\mu = 0.21$ and $\sigma = 0.05$, we can define a range $[\mu - \sigma, \mu + \sigma] = [0.16, 0.26]$ from which we can uniformly extract a number r_i at random $\forall i = 1...g$ where g is the number of missing values; afterwards, we replace the i-th missing value with $total\_rooms_i \cdot r_i$.
In order to keep consistency, we have to round this value to an integer value, since we are dealing with bedrooms and they belong to a discrete set[4].

Scaling A general practice in machine learning is to scale features to a common range such that the application of some techniques can perform better (e.g. PCA), otherwise the different scale of each feature could impact on results. This approach is called scaling and, more specifically, normalization when the range is [0, 1].
Moreover, scaling is usually done only on features and not on the target variable, this would allow us to keep trace of information about the error expressed in dollars. However, since there is a relevant difference in scale between the target variable and the rest of features, we have decided to scale it as well, since it gets difficult to see how MSE changes every time[5].
We have used two different techniques: *min-max* and *zscore*:

Min-max Basically, it compresses values in the range $[0, 1]$.

$$\frac{value - min}{max - min}$$

We apply this formula for each row of each feature, for which we compute its minimum and maximum (Figure 14).

Zscore It centers the involved distribution in the origin of the Cartesian plane, let us denote μ, σ as mean and standard deviation of a fixed feature, the range will be approximately $[-3\sigma, +3\sigma]$:

$$\frac{value - \mu}{\sigma}$$

again, we apply it to each feature of the dataset (Figure 15).

[4]The order of operations in the pipeline is crucial: if we replace missing values and then remove outliers we risk to use outliers and create more of them; on the other hand, if we remove outliers and then replace missing values we have to be careful about the code type of missing values: in most of programming languages their presence influence operations of filtering like the one of outliers removal. Our solution was to temporary convert their type to an impossible integer (a negative number of bedrooms cannot exists), safely remove outliers, and then replace missing values without outliers influence.

[5]We must recall that MSE is a relative measure that is scale-dependent: by scaling the target variable, we have closer MSEs that are easily comparable otherwise big scale pushes MSE to diverge to $+\infty$.

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	<1H OCEAN	INLAND	ISLAN
0.213996	0.564293	1.000000	0.170526	0.089202	0.151095	0.150418	0.899633	0.721533	0.0	0.0	0
0.212982	0.564293	1.000000	0.132653	0.124413	0.171070	0.189415	0.684719	0.698417	0.0	0.0	0
0.212982	0.564293	1.000000	0.201923	0.159624	0.173325	0.226555	0.445496	0.700343	0.0	0.0	0
0.212982	0.564293	1.000000	0.062991	0.107199	0.124356	0.165274	0.470871	0.545164	0.0	0.0	0
0.212982	0.563231	1.000000	0.380102	0.323161	0.343750	0.463324	0.420587	0.608306	0.0	0.0	0
0.212982	0.563231	1.000000	0.491758	0.478091	0.364046	0.586815	0.348816	0.484590	0.0	0.0	0
0.211968	0.563231	0.803922	0.384027	0.460876	0.379832	0.538533	0.210414	0.453126	0.0	0.0	0
0.212982	0.563231	1.000000	0.579082	0.493740	0.490979	0.649025	0.424861	0.526756	0.0	0.0	0
0.211968	0.564293	1.000000	0.314757	0.280125	0.284472	0.359331	0.359880	0.570420	0.0	0.0	0
0.211968	0.564293	1.000000	0.570055	0.528951	0.475838	0.667595	0.368853	0.485446	0.0	0.0	0

Figure 14: After min-max normalization

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	<1H OCEAN	INLAND
2	-1.319182	0.999125	1.869305	-0.752144	-1.305628	-1.305431	-1.324942	2.547547	1.743141	-0.882688	-0.69612
3	-1.324179	0.999125	1.869305	-0.938804	-1.092339	-1.197896	-1.109645	1.419003	1.627918	-0.882688	-0.69612
4	-1.324179	0.999125	1.869305	-0.597401	-0.879051	-1.185755	-0.904601	0.162805	1.637520	-0.882688	-0.69612
5	-1.324179	0.999125	1.869305	-1.282142	-1.196614	-1.449390	-1.242924	0.296052	0.864035	-0.882688	-0.69612
6	-1.324179	0.994511	1.869305	0.280771	0.111556	-0.268235	0.402558	0.032005	1.178764	-0.882688	-0.69612
7	-1.324179	0.994511	1.869305	0.831078	1.050025	-0.158966	1.084330	-0.344875	0.562110	-0.882688	-0.69612
8	-1.329175	0.994511	1.044328	0.300114	0.945751	-0.073978	0.817773	-1.071651	0.405279	-0.882688	-0.69612
9	-1.324179	0.994511	1.869305	1.261459	1.144820	0.524404	1.427780	0.054446	0.772284	-0.882688	-0.69612
10	-1.329175	0.999125	1.869305	-0.041290	-0.149130	-0.587373	-0.171567	-0.286780	0.989927	-0.882688	-0.69612
11	-1.329175	0.999125	1.869305	1.216971	1.358108	0.442886	1.530302	-0.239662	0.566377	-0.882688	-0.69612

Figure 15: After zscore normalization

Adding intercept As mentioned before, we add a feature of constant value 1 to the datasets with the aim of including the intercept.

median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN	intercept
1.000000	0.170526	0.089202	0.151095	0.150418	0.899633	0.721533	0.0	0.0	0.0	1.0	0.0	1
1.000000	0.132653	0.124413	0.171070	0.189415	0.684719	0.698417	0.0	0.0	0.0	1.0	0.0	1
1.000000	0.201923	0.159624	0.173325	0.226555	0.445496	0.700343	0.0	0.0	0.0	1.0	0.0	1
1.000000	0.062991	0.107199	0.124356	0.165274	0.470871	0.545164	0.0	0.0	0.0	1.0	0.0	1
1.000000	0.380102	0.323161	0.343750	0.463324	0.420587	0.608306	0.0	0.0	0.0	1.0	0.0	1
1.000000	0.491758	0.478091	0.364046	0.586815	0.348816	0.484590	0.0	0.0	0.0	1.0	0.0	1
0.803922	0.384027	0.460876	0.379832	0.538533	0.210414	0.453126	0.0	0.0	0.0	1.0	0.0	1
1.000000	0.579082	0.493740	0.490979	0.649025	0.424861	0.526756	0.0	0.0	0.0	1.0	0.0	1
1.000000	0.314757	0.280125	0.284472	0.359331	0.359880	0.570420	0.0	0.0	0.0	1.0	0.0	1
1.000000	0.570055	0.528951	0.475838	0.667595	0.368853	0.485446	0.0	0.0	0.0	1.0	0.0	1

Figure 16: `REP_MINMAX` with intercept

Shuffling Sometimes, datasets can show some order in the data, this obviously affects the results. Since the split of training and test set can be unlucky, they could be not enough representative of possible values.
We have noticed some kind of geographical order in this dataset, it can be seen in `ocean_proximity`: rows tend to have some locality principle, hence, there are a lot of sequences of rows with the same value (see Figure 17); in order to fix this issue we shuffle the four datasets.

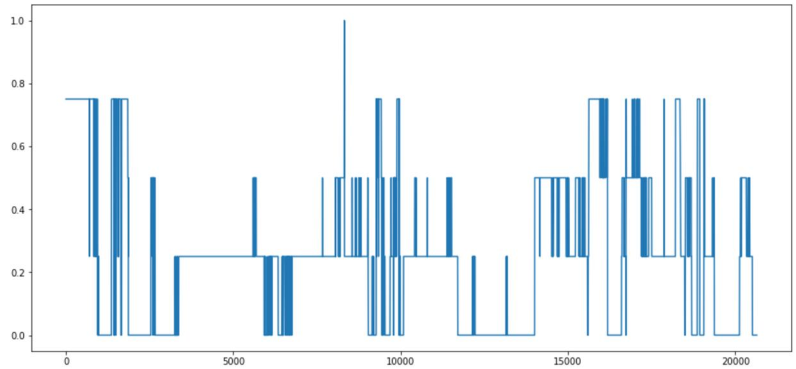

Figure 17: label-encoded `ocean_proximity` distribution

Any multicollinearity? We know that linear models suffer also from multicollinearity problem: the presence of multiple highly-correlated features could turn out to be redundant and also counterproductive. For this reason, they are typically detected and removed, but this is not always the case ([2]).

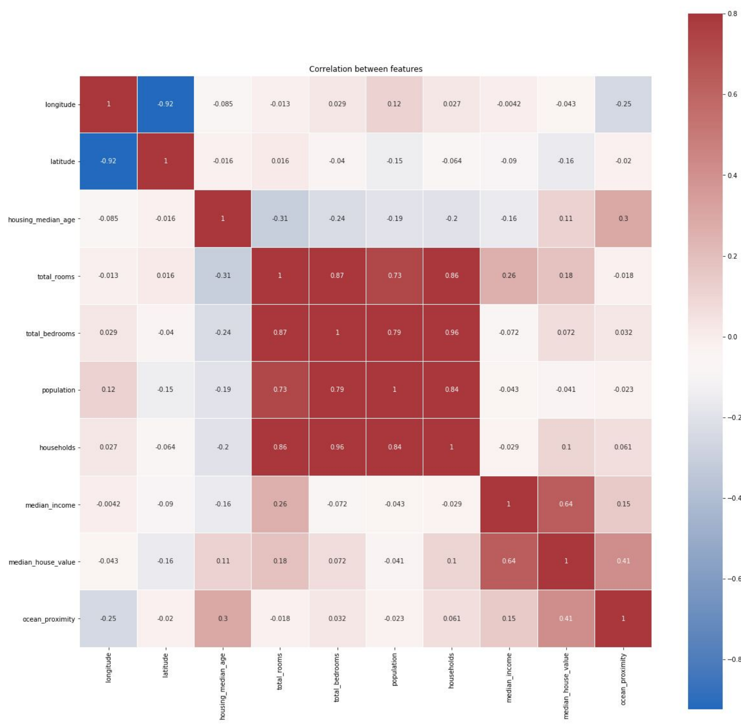

Figure 18: Correlation matrix between features

In Figure 18 we see that, actually, `latitude`, `longitude`, `total_rooms`, `total_bedrooms` and `population` are highly correlated features but in this case it is better to keep them for three reasons:

1. results get worse with their removal

2. we had used the correlation between `total_rooms`; and `total_bedrooms` to impute missing values
3. the regularization parameter should fix multicollinearity problems ([1])

At the end of preprocessing pipeline, we obtain four datasets:

1. `REP_MINMAX`: replaced missing values and min-max normalization (16715 rows)
2. `RED_MINMAX`: reduced missing values and min-max normalization (16552)
3. `REP_ZSCORE`: replaced missing values and scaled with zscore (16715 rows)
4. `RED_ZSCORE`: reduced missing values and scaled with zscore (16552)

5 Experiments

After preprocessing, the part of experiments is discussed in this section.

Model The model has been implemented with the following structure:

- a `fit` procedure (see Algorithm 1)
- a `predict` procedure (see Algorithm 2)

Algorithm 1 `fit` procedure

Procedure fit(X, $\vec{y}$, α)
Input: X is the design matrix, $\vec{y}$ is the vector of labels, α is the regularization parameter
Output: weight vector $\vec{w}$

1: $\vec{w} = (X^T X + \alpha I)^{-1} X^T \vec{y}$
2: **return** $\vec{w}$

Algorithm 2 `predict` procedure

Procedure predict(X, $\vec{w}$)
Input: X is the design matrix, $\vec{w}$ is the weight vector
Output: predictions vector $\vec{\hat{y}}$

1: $\vec{\hat{y}} = X\vec{w}$
2: **return** $\vec{\hat{y}}$

Tuning hyperparameters Tuning hyperparameters means finding the best configuration for those variable whose value control the learning process, in this case α.
In order to find the best α of the model we have run multiple executions of the algorithm with different values of α (Algorithm 3).

Algorithm 3 `training`, `validation` and `test error` computation

training, validation, test = split(dataset)
for α in $\underset{\rightarrow}{\alpha}$ **do**
    $\underset{\rightarrow}{w}$ = fit(training.X, training.$\underset{\rightarrow}{y}$, α)
    $\underset{\rightarrow}{\hat{y}}$ = predict(training.X, $\underset{\rightarrow}{w}$)
    training_error = MSE($\underset{\rightarrow}{\hat{y}}$, training.$\underset{\rightarrow}{y}$)
    $\underset{\rightarrow}{\hat{y}}$ = predict(validation.X, $\underset{\rightarrow}{w}$)
    validation_error = MSE($\underset{\rightarrow}{\hat{y}}$, validation.$\underset{\rightarrow}{y}$)
    $\underset{\rightarrow}{\hat{y}}$ = predict(test.X, $\underset{\rightarrow}{w}$)
    test_error = MSE($\underset{\rightarrow}{\hat{y}}$, test.$\underset{\rightarrow}{y}$)
end for

At this stage, we have randomly partitioned the four datasets in training, validation and test set with a proportion of 60-20-20, whereas during the next paragraphs the cross-validation technique has been used, hence, no validation set was needed.
We have created a vector of 5000 values from 0.1 to 500 and set it to $\underset{\rightarrow}{\alpha}$.

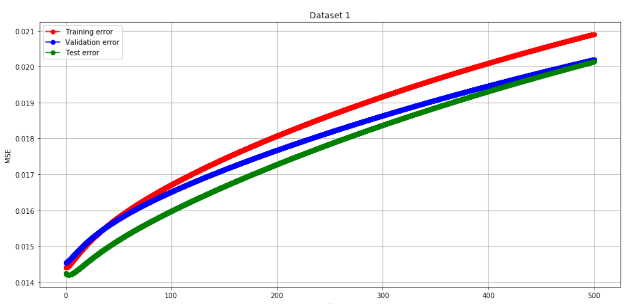

Figure 19: `REP_MINMAX`

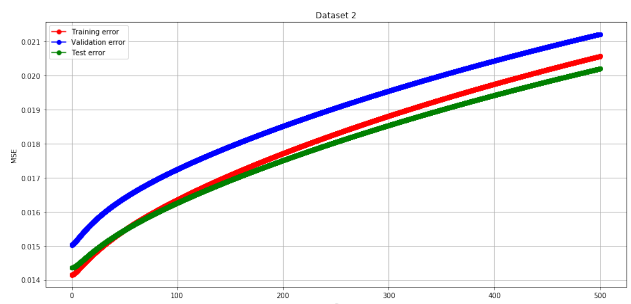

Figure 20: `RED_MINMAX`

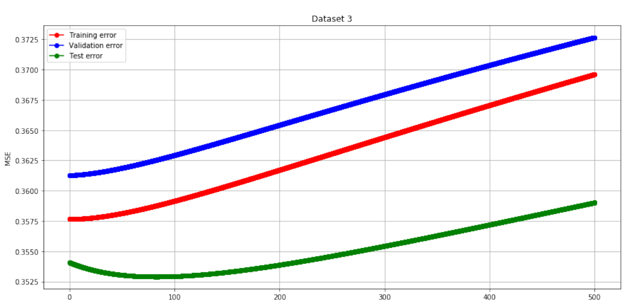

Figure 21: `REP_ZSCORE`

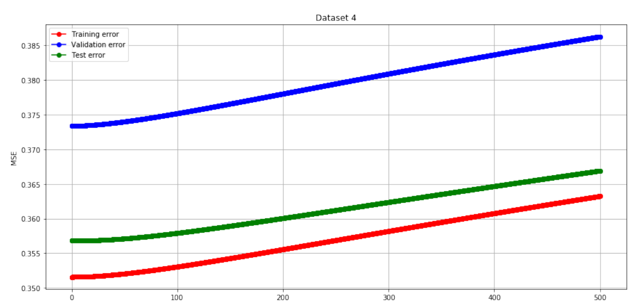

Figure 22: `RED_ZSCORE`

In Figures 19, 20, 21, 22 the execution of Algorithm 3 on four datasets is shown.
We quickly notice that in min-max normalized datasets, errors are quite close to each other, wheres in zscore, they are more distinguished.
In general, training error tends to be lower than validation and test error, but in Figure 21 is shown a particular case: test error is lower than training error, this suggests that it can happen that test set is composed by many more ”easier” cases than the training set.
We see that smaller alphas are suitable for the first two datasets, but seems interesting to investigate

on larger values for the last twos.
For this reason we will continue using this range of values for the next experiments.

K-fold cross validation When we refer to a learning algorithm $A(S)$ with S training set, we are implicitly meaning a family $A_\theta(S)$ of learning algorithms with $\theta \in \Theta$, the set of all possible hyperparameter values (in this case α).
Let us fix α and let ℓ_D be the statistical risk, we want to compute $E[\ell_D(A)]$ to understand the goodness of A.
We can do it through K-fold cross validation, a partioning method that splits the dataset S in K folds $S_1, S_2, ..., S_K$ such that $\forall k = 1...K$ we consider $S^{(k)} \equiv S \setminus S_k$ as training part and S_k as testing part. Given that m is the size of S, K the number of folds, S_k the k-th fold and h_k the linear predictor returned by the training performed by A over S_k, then the scaled test error (in our case MSE) of each fold is

$$\hat{\ell}_{S_k}(h_k) = \frac{K}{m} \sum_{(x,y) \in S_k} \ell(h_k(x), y) \tag{12}$$

(12) estimates the statistical risk of the predictor h_k output by $A(S^{(k)})$ and it is done through MSE over S_k; we compute the mean to estimate the expected risk of A

$$E[\ell_D(A)] = \frac{1}{K} \sum_{k=1}^{K} \hat{\ell_{S_k}}(h_k) \tag{13}$$

we have done this in Algorithm 4, note that for a complete analysis, we have computed both training and test errors: In general, K is chosen freely, but is typically recommended to be equal to 5 or 10

Algorithm 4 `K-fold cross validation` procedure

Procedure cv(S, K, α)
Input: S is the dataset, K the number of folds, α the regularization parameter
Output: training errors and test errors

$S_1, S_2, ..., S_K$ = random_partitioning(S, K)
for S_i in $S_1, S_2, ..., S_K$ **do**
  test = S_i
  train = $S \setminus S_i$
  $\underset{\to}{w}$ = fit(train.X, train.$\underset{\to}{y}$, α)
  $\underset{\to}{\hat{y}}$ = predict(train.X, $\underset{\to}{w}$)
  training_error = MSE($\underset{\to}{\hat{y}}$, train.$\underset{\to}{y}$)
  $\underset{\to}{\hat{y}}$ = predict(test.X, $\underset{\to}{w}$)
  test_error = MSE($\underset{\to}{\hat{y}}$, test.$\underset{\to}{y}$)
  training_errors.add(training_error)
  test_errors.add(test_error)
end for
return training_errors, test_errors

([8]). We report in Table 2 the cross-validated risk estimate with $K = 5$ and $\alpha = 0.1$ relative to the four datasets (Algorithm 5).

Algorithm 5 `cross-validated risk estimate` computation

1: $\hat{r_1}$ = mean(cv(`REP_MINMAX`, 5, 0.1).test_errors)
2: $\hat{r_2}$ = mean(cv(`RED_MINMAX`, 5, 0.1).test_errors)
3: $\hat{r_3}$ = mean(cv(`REP_ZSCORE`, 5, 0.1).test_errors)
4: $\hat{r_4}$ = mean(cv(`RED_ZSCORE`, 5, 0.1).test_errors)

$\hat{r_1}$	$\hat{r_2}$	$\hat{r_3}$	$\hat{r_4}$
0.0144	0.0144	0.3584	0.3577

Table 2: Cross-validated risk estimate

Dependence of the cross-validated risk estimate on the parameter alpha With the aim of studying the dependence between them we use again the vector $\underset{\rightarrow}{\alpha}$ containing 5000 values from 0.1 to 500 and we use the procedure in Algorithm 4 to compute the cross-validated risk estimate for all α in $\underset{\rightarrow}{\alpha}$.
We do this for the four datasets, and we plot both training and test error:

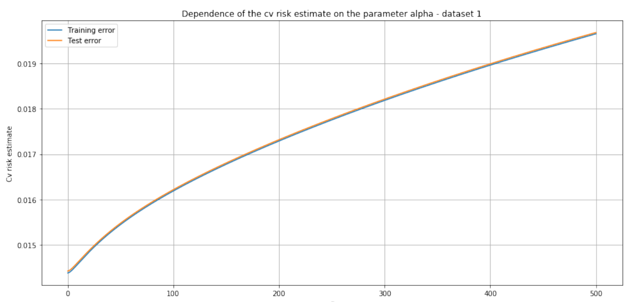

Figure 23: `REP_MINMAX`

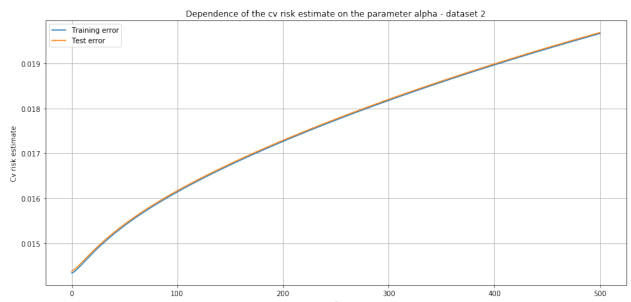

Figure 24: `RED_MINMAX`

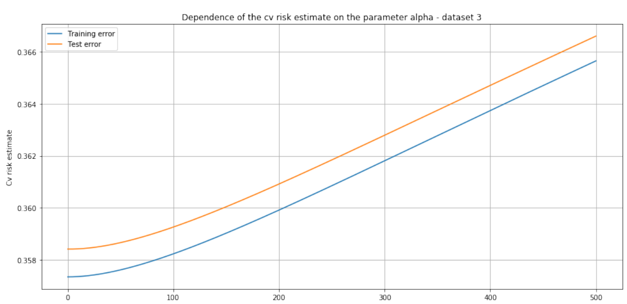

Figure 25: `REP_ZSCORE`

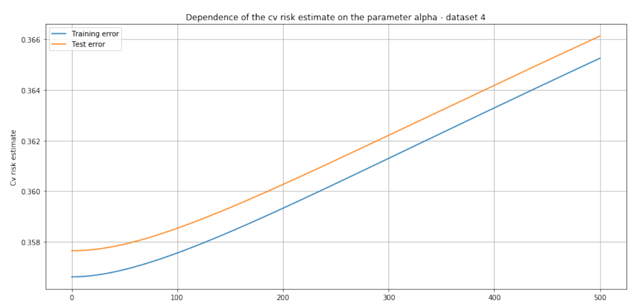

Figure 26: `RED_ZSCORE`

We clearly see that the cross-validated risk estimate increases with α and also decreases with α decreasing. It means that there is a strong dependence of positive linear relationship between them, which is highly confirmed by a correlation coefficient of 0.99.
Concerning training and test errors, since the range of α is big, they are very close, especially in the dataset that has been normalized with min-max (see Figures 23, 24). On the other hand, zscore dataset (Figures 25, 26) shows a better distinction of the two errors because the different range

obtained with this scaling allows MSE to be dilated.
Finally, we confirm the trend of last two datasets of less rapid growth of risk estimate.

Using PCA to reduce risk estimate Principal component analysis has been used with the goal of trying to reduce risk estimate.
PCA is a dimensionality-reduction method which is used to reduce the dimensionality of large datasets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.
Reducing the number of variables of a dataset naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity, because smaller datasets are easier to explore, visualize and make analyzing data much easier and faster for machine learning algorithms without extraneous variables to process ([5]).
We have tried all possible reductions from 1 to 14 components but they all have worsened results, in Table 3 is shown PCA decomposition with 5 components reduction and $\alpha = 0.1$ computation of cross-validated risk estimate.

$\hat{r_1}\_PCA$	$\hat{r_2}\_PCA$	$\hat{r_3}\_PCA$	$\hat{r_4}\_PCA$
0.1556	0.1556	0.4563	0.4508

Table 3: Cross-validated risk estimate with PCA

Nested cross validation It adds an inner layer of cross validation to the one we had previously defined in order to try a range of values for hyperparameter α and find the best one (Algorithm 6)
We define the vector $\underset{\rightarrow}{\alpha}$ containing 2500 values from 0.01 to 25, $K = 5$, $L = 4$; we use the procedure defined in Algorithm 6 to compute the nested cross-validated risk estimate (Algorithm 7).

we represent in Table 4 final results.

$nest\_\hat{r_1}$	$\hat{\alpha_1}$	nest_$\hat{r_2}$	$\hat{\alpha_2}$	$nest\_\hat{r_3}$	$\hat{\alpha_3}$	$nest\_\hat{r_4}$	$\hat{\alpha_4}$
0.0144	0.01	0.0144	0.61	0.3584	0.01	0.3577	13.13

Table 4: Nested cross-validated risk estimate and best alphas

6 Results and comments

In order to compare results, we must define how small low MSE must be to be considered sufficiently "low": with min-max values have been shrinked between 0 and 1, this implies that loss can be at most 1.
On the other hand, zscore lets loss vary over a wider range, so in this case ℓ could reach, in the worst case, approximately $6\hat{\sigma}$, which is equal[6] to 6.
Consequently, if we rescale the risk of `REP_ZSCORE` and `RED_ZSCORE` by dividing with 6, we get 0.0597 and 0.0596 which are much more similar to the 0.0144 obtained by the first two datasets.
In general, the best performance is apparently achieved by the dataset normalized with min-max.

[6]This value has been computed by calculating the mean of standard deviation of each feature $\hat{\sigma}$, we have obtained $\hat{\sigma} = 1$, therefore $6\hat{\sigma} = 6$

Algorithm 6 `nested cross validation` procedure

Procedure nest_cv(S, K, L, $\vec{\alpha}$)
Input: S is the dataset, K folds, L subfolds, $\vec{\alpha}$ the vector of α values to try
Output: training errors, test errors and the best α

$S_1, S_2, ..., S_K$ = partitioning(S, K)
for S_i in $S_1, S_2, ..., S_K$ **do**
  test = S_i
  train = $S \setminus S_i$
  $S_1, S_2, ..., S_L$ = partitioning(test, L)
  for S_j in $S_1, S_2, ..., S_L$ **do**
    in_test = S_j
    in_train = train $\setminus S_j$
    for α in $\vec{\alpha}$ **do**
      $\vec{w}^*$ = fit(in_train.X, in_train.$\vec{y}$, α)
      $\vec{\hat{y}}^*$ = predict(in_train.X, $\vec{w}^*$)
      in_training_error = MSE($\vec{\hat{y}}^*$, in_train.$\vec{y}$)
      $\vec{\hat{y}}^*$ = predict(in_test.X, $\vec{w}^*$)
      in_test_error = MSE($\vec{\hat{y}}^*$, in_test.$\vec{y}$)
      in_training_errors.add(in_training_error)
      in_test_errors.add(in_test_error)
    end for
    best_in_result = min(in_test_errors)
    $\hat{\alpha}^* = \arg\min_{\alpha \in \vec{\alpha}}$(in_test_errors)
    if best_in_result < best_result **then**
      best_result = best_in_result
      $\hat{\alpha} = \hat{\alpha}^*$
    end if
  end for
  $\vec{w}$ = fit(train.X, train.$\vec{y}$, $\hat{\alpha}$)
  $\vec{\hat{y}}$ = predict(train.X, $\vec{w}$)
  training_error = MSE($\vec{\hat{y}}$, train.$\vec{y}$)
  $\vec{\hat{y}}$ = predict(test.X, $\vec{w}$)
  test_error = MSE($\vec{\hat{y}}$, test.$\vec{y}$)
  training_errors.add(training_error)
  test_errors.add(test_error)
end for
return training_errors, test_errors, $\hat{\alpha}$

Algorithm 7 `nested cross-validated risk estimate` and `best alpha` computation

result = nest_cv(`score_REP_MINMAX`, 5, 4, $\underrightarrow{\alpha}$)
$nest\_\hat{r_1}$ = mean(result.test_errors)
$\hat{\alpha_1}$ = result.$\hat{\alpha}$
result = nest_cv(`score_RED_MINMAX`, 5, 4, $\underrightarrow{\alpha}$)
$nest\_\hat{r_2}$ = mean(result.test_errors)
$\hat{\alpha_2}$ = result.$\hat{\alpha}$
result = nest_cv(`score_REP_ZSCORE`, 5, 4, $\underrightarrow{\alpha}$)
$nest\_\hat{r_3}$ = mean(result.test_errors)
$\hat{\alpha_3}$ = result.$\hat{\alpha}$
result = nest_cv(`score_RED_ZSCORE`, 5, 4, $\underrightarrow{\alpha}$)
$nest\_\hat{r_4}$ = mean(result.test_errors)
$\hat{\alpha_4}$ = result.$\hat{\alpha}$

Concerning the choice of removing or replacing missing values, there is no evident difference: probably, the impact of these values is so low because their fraction with respect to the whole dataset is small.
In this context, only in zscore datasets there is a slightly visible difference, but this is still due to the different scale that it has on MSE, in this case the reduced dataset performs a little better.
PCA has not improved the risk estimate in any case, on the contrary, it has worsened results. Finally, nested cross validation has confirmed that the best performances are in general achieved with α going towards 0, except `RED_ZSCORE`, whose best value is 13.13.

7 External libraries

Scikit-learn is the classical example of software library which has been developed with many useful functions, including ridge regression implementation and cross validation.
We compute the cross-validated risk estimate through this library to compare results (Algorithm 8):

Algorithm 8 `cross validation` with *Scikit-learn* library

r_1^* = mean(`sklearnCV(REP_MINMAX, sklearnRIDGE(0.1))`)
r_2^* = mean(`sklearnCV(RED_MINMAX, sklearnRIDGE(0.1))`)
r_3^* = mean(`sklearnCV(REP_ZSCORE, sklearnRIDGE(0.1))`)
r_4^* = mean(`sklearnCV(RED_ZSCORE, sklearnRIDGE(0.1))`)

r_1^*	r_2^*	r_3^*	r_4^*
0.0144	0.0144	0.3584	0.3577

Table 5: Scikit-learn cross-validated risk estimate

As we can see in Table 5 we obtain the same results, exploiting the consistency of our work.

8 Conclusions

We have used a dataset of houses in USA to build, train and test a model in order to infer housing prices.
During the experiments we have showed how the best value for α tends to be close to 0, this means that the amount of regularization needed to reduce the risk of overfitting is pretty small, meaning that there is little instability.
The only exception is represented by `RED_ZSCORE`, whose best α is 13.13, this may be a symptom of more instability when scaling with zscore and removing missing values.

9 Future works

Obviously, this project can be extended and deepened, trying other approaches like data augmentation, different imputation methods, analysing if there are optimal values for k such that better k-fold cross validation is performed or, since no particular regularization here was needed, simply creating a linear regression model to see how different results are.
We put these points as possible guidelines for future works about this learning task.

References

[1] M Abdalla and Khaled IA Almghari. "Remedy of multicollinearity using Ridge regression". In: *Journal of Al Azhar University-Gaza (Natural Sciences)* 13 (2011), pp. 119–134.

[2] Paul Allison. *When can you safely ignore multicollinearity?* URL: `https://statisticalhorizons.com/multicollinearity`. Sept. 2012.

[3] Nicolò Cesa-Bianchi. *Statistical methods for machine learning, linear prediction.* URL: `http://cesa-bianchi.di.unimi.it/MSA/Notes/linear.pdf`. Apr. 2020.

[4] Nicolò Cesa-Bianchi. *Statistical methods for machine learning, statistical risk.* URL: `http://cesa-bianchi.di.unimi.it/MSA/Notes/statRisk.pdf`. Mar. 2020.

[5] Zakaria Jaadi. *A step by step explanation of principal component analysis.* URL: `https://builtin.com/data-science/step-step-explanation-principal-component-analysis`. Sept. 2019.

[6] Barnabás Póczos. *Advanced introduction to machine learning CMU-10715, risk minimization.* URL: `https://www.cs.cmu.edu/~epxing/Class/10715/lectures/RiskMin.pdf`. Nov. 2015.

[7] Wessel N van Wieringen. "Lecture notes on ridge regression". In: *arXiv preprint arXiv:1509.09169* (2015).

[8] Ian H Witten and Eibe Frank. "Data mining: practical machine learning tools and techniques with Java implementations". In: *Acm Sigmod Record* 31.1 (2002), pp. 149–150.