Project Report

Regression on diamonds prices

Emma Covili

January 10, 2022

Contents

1 Introduction

Diamonds are precious stones consisting of a clear and colourless crystalline form of pure carbon. They are the hardest gemstones known to man and can be scratched only by other diamonds.
Diamonds are formed deep within the Earth about 100 miles or so below the surface in the upper mantle. Obviously, the temperature of this part of the Earth is very high. Plus, there is a lot of pressure, the weight of the overlying rock bearing down. The combination of high temperature and high pressure is what's necessary to grow diamond crystals in the Earth.
Diamonds are rare because of the incredibly powerful forces needed to create them and therefore they are considered to be very costly.

The goal of this project is to do a regression on diamond prices. To do that we will present the dataset we will use, and then present the methods used and their results.

2 Dataset presentation

I chose a dataset from Kaggle about diamonds, as it seemed a rather exotic subject. It is composed of 6,000 points and 7 features which are the carat weight, the cut, the color, the clarity, the symmetry and the report.

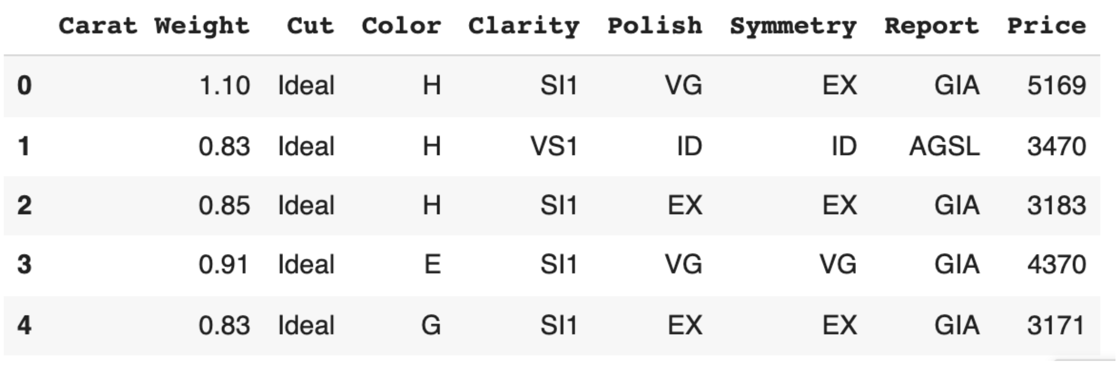
- Carat Weight: the carat weight of the diamond in grams;
- Cut: the cut quality of the diamond, `Fair`, `Good`, `Very Good`, `Ideal` and `Signature-Ideal`;
- Color: the color of the diamond, with `D` the best and `J` the worst;
- Clarity: the absence of inclusions and blemishes (small imperfections in the gem), from best to worst, `FL`, `IF`, `VVS1`, `VVS2`, `VS1`, `VS2`, `SI1`, `SI2`, `I1`, `I2`, `I3`.
- Polish: the polish of the diamond, `Good`, `Very Good`, `Ideal` and `Excellent`;
- Symmetry: the symmetry of the diamond, `Good`, `Very Good`, `Ideal` and `Excellent`;
- Report: the agency that graded the diamond, `GIA`, `AGSL`;
- Price: the price of the diamond, in US dollars.

We have qualitative features (categorical) such as `Cut`, `Color`, `Clarity`, `Polish`, `Symmetry` and `Report`; and some quantitative features (numerical) such as `Carat Weight` and `Price`. The target variable is `Price`. An overview of the dataset is available here 1.

Traditionally, a diamond's price is estimated with the "4C" rule:

- Carat weight, the heavier the diamond, the more expensive;
- Cut quality, the better the cut, the more brilliant the diamond will appear, and so the more expensive it will be;
- Clarity of the diamond, the brighter, the more expensive the diamond;
- Color, the lesser, the better and so the more expensive.

So we are going to focus on these criteria and we expect them to have an impact on the price. Here are some graphics to illustrate those points.



	Carat Weight	Cut	Color	Clarity	Polish	Symmetry	Report	Price
0	1.10	Ideal	H	SI1	VG	EX	GIA	5169
1	0.83	Ideal	H	VS1	ID	ID	AGSL	3470
2	0.85	Ideal	H	SI1	EX	EX	GIA	3183
3	0.91	Ideal	E	SI1	VG	VG	GIA	4370
4	0.83	Ideal	G	SI1	EX	EX	GIA	3171

Figure 1: Head of the dataset

First, we can see on figure 2, that the fisrt of the 4C rules tends to be verified. As the weight increases, the price does as well.

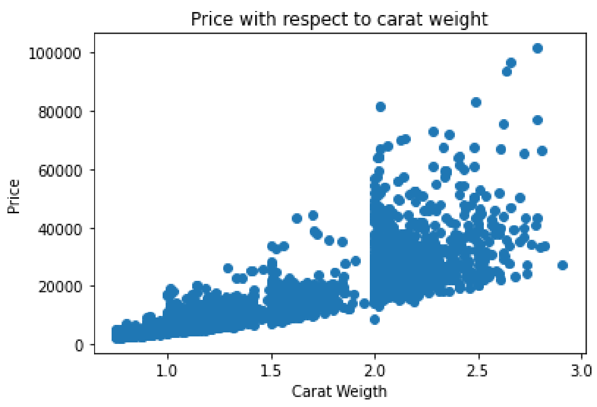


Figure 2: Prices with respect to carat weights

Here are the graphs for prices with respect to clarity, color and symmetry. We can see that the higher the quality of either the clarity, the color or the symmetry, the higher the prices.

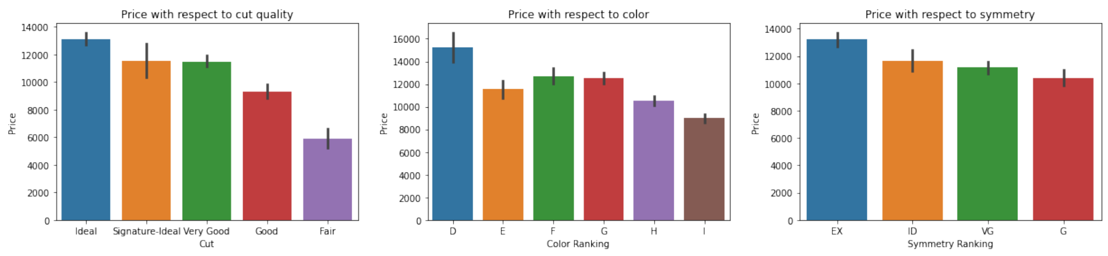


Figure 3: Price with respect to clarity, color and symmetry

On the heatmap correlation graph 4, we can see that the prices are highly correlated with the carat weight and the clarity, as the 4C rule predicted. We can also see that the cut is relevant even if it less correlated than the other two. However, the color does not seem correlated at all with prices.
We can also see the correlation between polish and symmetry.

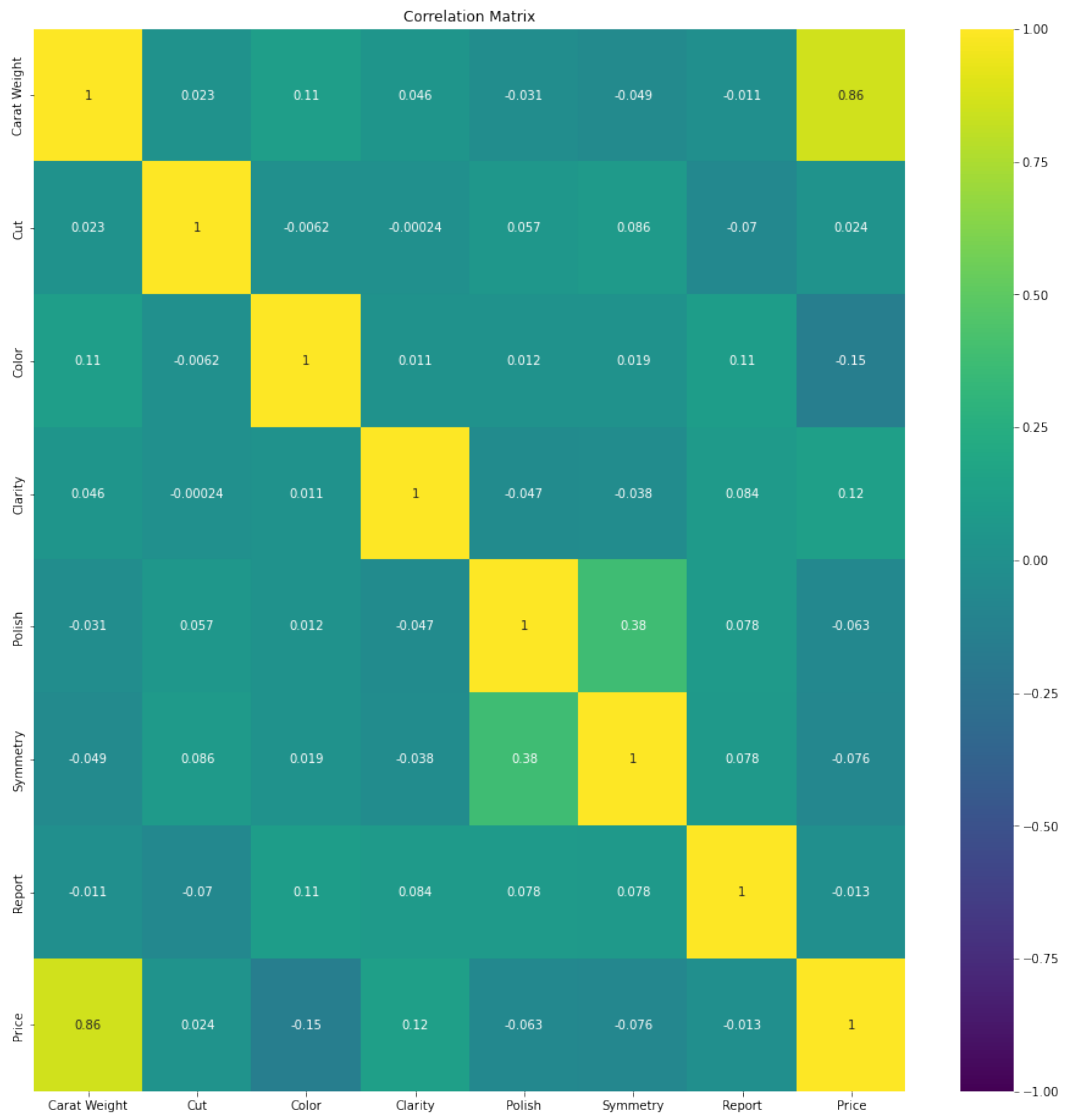


Figure 4: Heatmap correlation graph

3 Methods & Results

We want to minimize the function

$$\min_w f(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(x_i^\top w, y_i) + \lambda R(w)$$

where $\ell(x_i^\top w, y) = \frac{1}{2}\|x_i^\top w - y\|^2$ the least square regression, with $\|.\|$ the euclidean norm and

$$R(w) = \begin{cases} \frac{1}{2}\|w\|^2 & \text{for the Ridge regularization} \\ \frac{1}{2}\|w\|_1 & \text{for the Lasso regularization} \\ 0 & \text{otherwise} \end{cases}$$

First, I implemented a general class called `Init` with parameters `X` for the data matrix, `y`

for the target vector, `w` for the weight vector, `n` and `m` the dimensions of the data matrix and `hist` a dictionary to store the values of the losses of the train and test data during the learning process. Plus, the class has different methods such as `f`, `f_i`, `grad`, `grad_i`, `lipgrad`, `predict`, `loss` and `score` that I will use in the next implementations.
The `loss` method is the Root Mean Squarred Error (RMSE) and the `score` one is the R2-score.

3.1 Implementations

All the next implementations are based on the `Init` class, they all possess a `fit` method which is specific to each one of them.

3.2 Batch Gradient Descent

For the batch gradient descent the problem is:

$$\frac{1}{n}\sum_{i=1}^{n}\ell(x_i^\top w, y_i)$$

where $\ell(x_i^\top w, y) = \frac{1}{2}\|x_i^\top w - y\|^2$.
So, the problem becomes:

$$\min_w \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}\|x_i^\top w - y_i\|^2 = \min_w \frac{1}{2n}\|Xw - y\|^2$$

The gradient of $f(w) = \frac{1}{2n}\|Xw - y\|^2$ is $\nabla f = \frac{1}{n}X^\top(Xw - y)$.

In figure 5, we can observe the learning curves of the losses for the train and test sets. The final score of the method is 71.9% for the test set. With the sklearn library, the score we have is 76.43% for the test set which is quite similar to what I got with my own implementation. The score of the sklearn built-in function for gradient descent is the same as the one I use: the R2-score, which offers a safe comparison.

With some hyper-parameter tuning, I found the best τ, my learning rate parameter, for my model : 0.05.

For higher values of τ, the score decreases drastically. Indeed, as we saw in class, a large value of τ can cause the algorithm not to converge. For instance, I tried with $\tau = 0.06$ and I got a score of -351400085.74%. For very small values of τ, the algorithm tends to converge but it is very slow. Just like what we saw in class: a small learning rate will make small steps to reach minimum and it would need more than 250 epochs to reach it.

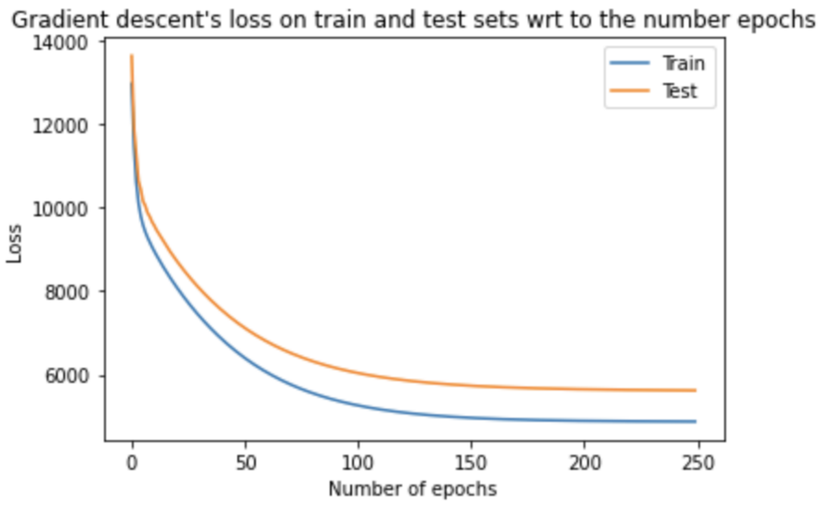


Figure 5

3.3 Accelerated Gradient Descent: Nesterov

For the Nesterov's accelerated gradient descent, I am going to use the Nesterov algorithm for convex functions since

$$f(w) = \frac{1}{2n}\|Xw - y\|^2$$

is convex.
The iteration step is :
Let $w_0 \in \mathbb{R}^d$, and $w_{-1} = w_0$
$\forall i \geq 0$, $w_{i+1} = w_i - \tau_i \nabla f(w_i + \beta_i(w_i - w_{i-1})) + \beta_i(w_i - w_{i-1})$ where $\beta_i = \frac{t_i - 1}{t_{i+1}}$ and $t_{i+1} = \frac{1}{2}(1 + \sqrt{1 + it_i^2})$ and $t_0 = 0$.

In figure 6, we can observe the learning curves of the losses for the train and test sets. The final score of the method is 71.7% for the test set, which is a bit lower than for the batch gradient descent.

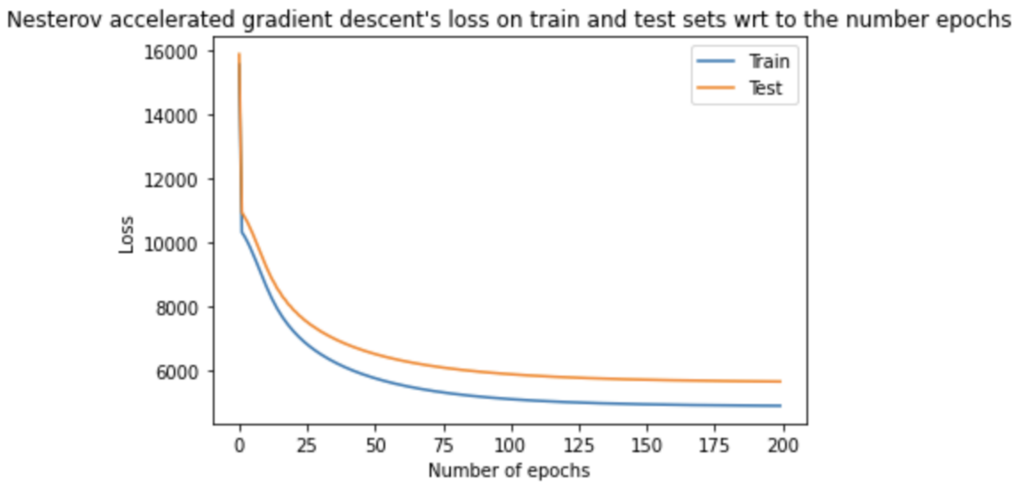


Figure 6

With some hyper-parameter tuning, I found the best τ, my learning rate parameter, for my model : 0.03.

3.4 Stochastic Gradient Descent

For the stochastic gradient descent the problem is:

$$\min_w f(w) \equiv \frac{1}{n}\sum_{i=1}^{n} f_i(w) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}(x_i w - y_i)^2$$

The algorithm reads

$$w_{k+1} = w_k - \tau_k \nabla f_{i_k}(w_k)$$

where at each iteration i_k is drawn in $\{1, \ldots, n\}$ uniformly at random.

In my implementation of the stochastic gradient descent, I added a `batch` and a `behavior` parameters to test different settings. The `behavior` parameter deals with the learning rate behavior, either `behavior` $= 0$ and the learning rate is constant, $\tau = \frac{1}{L}$ where L is the lipschitz constant, or `behavior`> 0 and the learning rate is decreasing in $\frac{1}{(k+1)^t}$ where k is the iteration and t the value of `behavior`.

In figure 7, we can observe the learning curves of the losses for the train and test sets. The final score of the method is 72.1% for the test set, which is better than gradient descent as expected. With the sklearn library, the score we have is 74.4% for the test set which is quite similar to what I got with my own implementation.

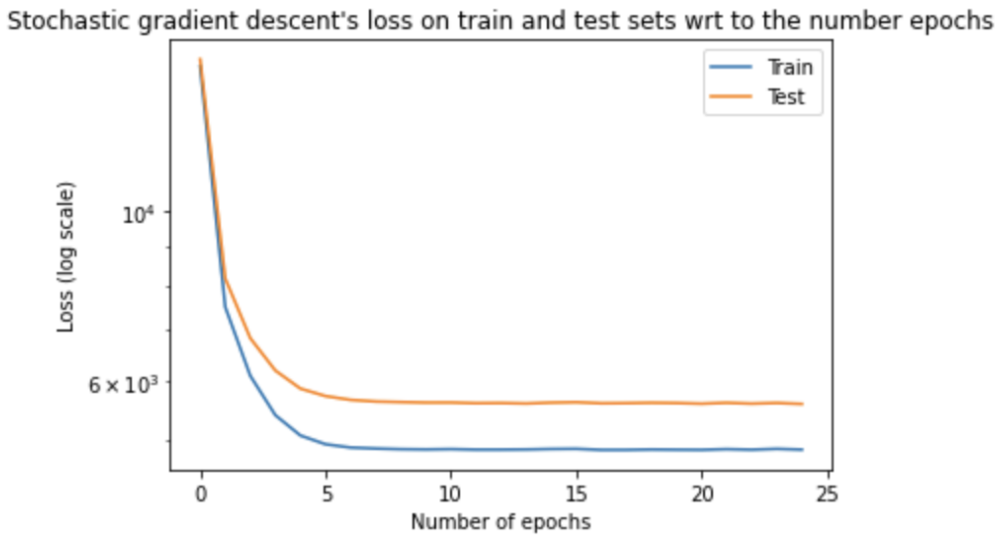


Figure 7

With some hyper-parameter tuning, I found the best τ, `behavior` and `batch size` for my model are 0.01, 0 and 1.

I ran some tests to compare and study gradient descent and stochastic gradient methods. For this part, I only used my implementation of stochastic gradient and I modified the batch size and the number of iterations to run either gradient descent (batch size $= n$ and nb_iter $= nb\_epochs$) or stochastic gradient (batch size $= 1$ and nb_iter $= nb\_epochs * n$).

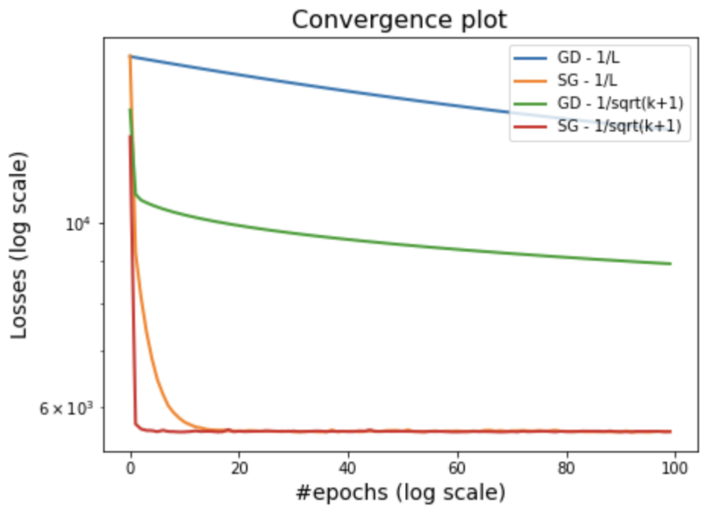
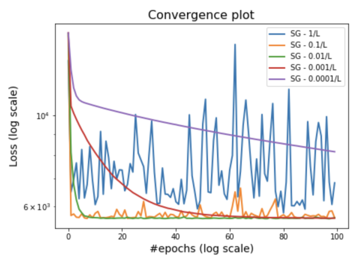
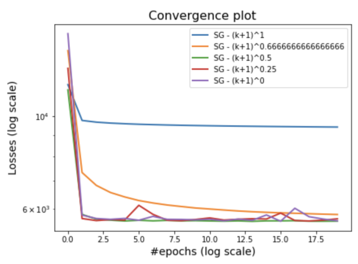


Figure 8: Gradient descent vs Stochastic gradient

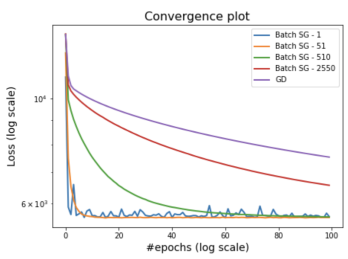
In figure 8, the red curve (SG with decreasing stepsize) illustrates a commonly observed behavior of SG, characterized by rapid progress during the first iterations, followed by an "oscillating phase" indicating that the method stalls and is taking small steps that marginally improve or worsen the objective value. We can also see that both stochastic gradient curves (red and orange), with constant or decreasing step size, converge faster than both the gradient descent curves.



(a) Constant stepsize



(b) Decreasing stepsize



(c) Batch size

Figure 9: Stochastic gradient convergence plots

In figure 9, the graphs from left to right are the comparison of variants of SG with different constant stepsizes and identical batch size, the comparison of variants of SG with different decreasing stepsizes and identical batch size and the comparison of variants of SG with different batch size with the same step size policy.

In figure 9a, we see that using a very small stepsize will lead to slow convergence (as illustrated by the violet curve $\frac{0.0001}{L}$). We expect this slow convergence behavior to be followed by an oscillatory phase, which we can readily see with other stepsize choices. The green and orange curves ($\frac{0.01}{L}$ and $\frac{0.1}{L}$, respectively) illustrate the trade-off between having a small stepsize, that guarantees convergence to a small neighborhood of the optimum, and having a large stepsize, that leads to faster convergence (but to a large neighborhood). The blue curve oscillates very much, which indicates that the stepsize is too large.

In figure 9b, the observations are similar, but we expect a rapidly decreasing stepsize sequence to lead to a sublinear convergence rate, without any oscillation around a certain value. This appears to be the case for the choices $\alpha_k = \frac{1}{k+1}$ and $\alpha_k = \frac{1}{(k+1)^{2/3}}$. On the other hand, the lowest losses (in 20 epochs) are reached by the green red and purple curves, despite an oscillatory trend.

And finally, in figure 9c, for different batch sizes: as in the graph in which we compared GD and SG, SG converges faster than GD (blue vs violet curves). Big batch sizes leads to slower convergence whereas small ones converge really fast (blue and orange curves). The orange curve illustrates the mini-batch SG setting and converges nicely. The green curve, which is for batch size 510, converges nicely, faster than higher batch size or GD but slower than mini-batch and SG.

3.5 Regularization

3.5.1 Ridge

For the Ridge regularization problem, we want to solve

$$\min_w f(w) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}\|x_i^\top w - y\|^2 + \lambda \frac{1}{2}\|w\|^2$$

which is equivalent to

$$\min_w \frac{1}{2}\|Xw - y\|^2 + \lambda \frac{1}{2}\|w\|^2$$

where $\lambda > 0$ is the regularization parameter. If $\lambda = 0$, we find ourselves in the Ordinary Least Squares (OLS) situation.

The solution is given using the following equivalent formula:

$$w = (X^\top X + \lambda Id_m)^{-1} X^\top y,$$

$$w = X^\top (XX^\top + \lambda Id_n)^{-1} y,$$

When `m` < `n` (which is the case here), the first formula should be prefered.

The score obtained by this method is of 72% for the test set. The score obtained by the sklearn built-in functions is 76% which quite similar to my results.

With some hyper-parameter tuning, I found the best λ for my model is 0.0001 and even 0, meaning the regularization is not needed with these data.

3.5.2 Lasso - with ISTA algorithm

The Lasso regularization problem is defined as

$$\min_w \frac{1}{2}\|Xw - y\|_2^2 + \lambda\|w\|_1$$

where $\lambda > 0$ is the regularization parameter.

The ISTA performs first a gradient step (forward) of the smooth part of the functional, $\frac{1}{2}\|Xw - y\|^2$, and then a proximal step (backward) step which account for the ℓ_1 penalty and induce sparsity. This proximal step is the soft-thresholding operator :

$$\mathcal{S}_\lambda(x) = sign(x)(|x| - \lambda)_+.$$

where $(|x| - \lambda)_+ = \max(|x| - \lambda, 0) = ReLU(|x| - \lambda)$.

The ISTA algorithm reads

$$w_{k+1} = \mathcal{S}_{\lambda\tau}(w_k - \tau X^\top(Xw_k - y)).$$

In figure 10, we can observe the learning curves of the losses for the train and test sets. The final score of the method is 71.6% for the test set.

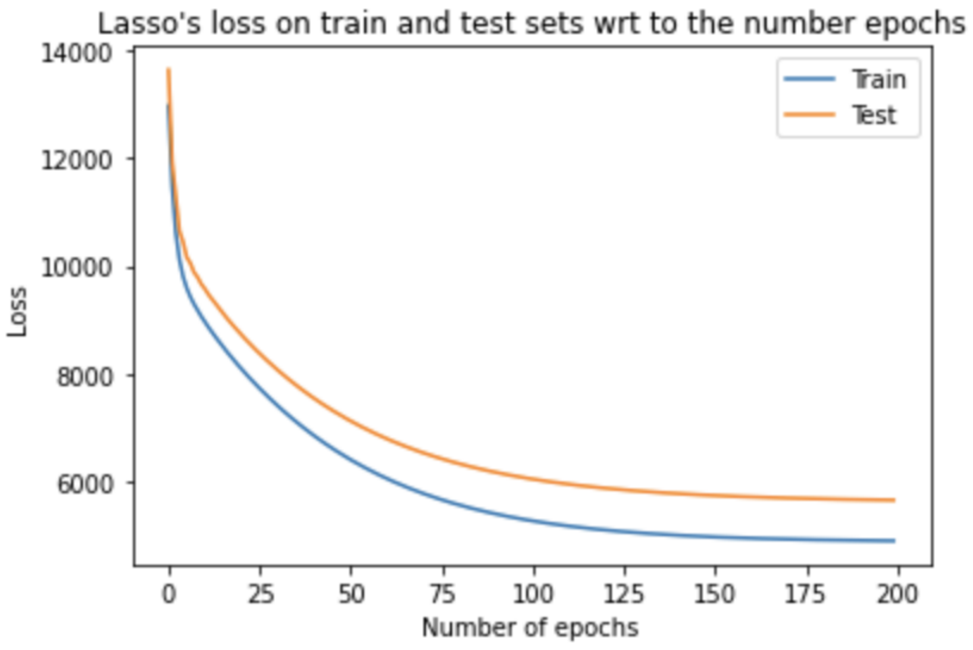


Figure 10

With some hyper-parameter tuning, I found the best τ for my model is 0.05 with $\lambda = 0.0001$. For the same lambda, and $\tau = 0.07$, the score decreases drastically which indicates that the learning rate is too high.

4 Conclusion

To conclude, the method that got the best results is the stochastic gradient one. The regularizations were not crucial in these settings. I think that an other approach like random forest would have given higher results.