PROJECT REPORT Milestone I

Linear regression :

For linear regression, we began our tests without giving any specific attention to the data (if not for the creation of a validation set which is done for every method). We tested the method for every *lambda* between **0** and **100** and got a small variation in the results as it went from *MSE* = **0.0510** (for *lmda* = **0**) to *MSE* = **0.0514** (for *lmda* = **100**), with the minimum being slightly below **0.0510** (for lmda = **20**).
Realizing that we couldn't improve the MSE by modifying the lamda, we decided to try adding a bias term to the data by adding an argument - -bias_term in the main. The impact of the bias term was immediate as for the same *lamda's* than above, the MSE is divided by **10** (see Fig.1). We then decided to test more specific values of *lmda* around *lmda* = **1** as it was the best value on the interval ***[0,100]***. With the more precise test of *lmda* (see Fig.2), we found that the best loss was achieved for *lmda* = **1.2**.

Best results (*lmda* = 1.2):
Test loss = 0.0046
Fit time : 0.00018692016601562 seconds
Predict time : 0.0000030994415 seconds

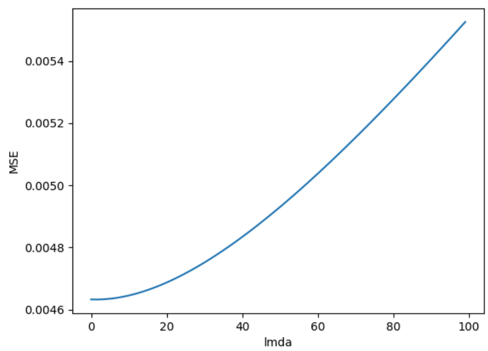


*Fig.1 : MSE as a function of the lambda parameter (lmda∈**[0,100]**)*

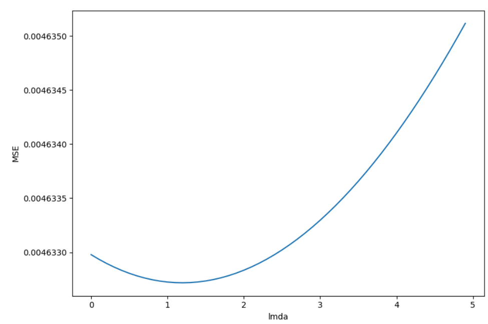


*Fig.2 : MSE as a function of the lambda parameter (lmda∈**[0,5]**)*

Logistic regression :

Similarly to the previous method, we started the test of logistic regression without processing the data. We tried for 5 different max_iters from **100** to **300** and for learning rates between **0.00001** and **0.1**. This gave us **77.982%** of accuracy at best, for *lr* = **0.001**. We were stuck to this percentage, no matter what precision we had on the learning rates we tried. Hence, we decided again to add a bias term with the argument - -bias_term.
With the data biased, we finally got a better accuracy (see Fig.3). We could then conduct more precise tests around *lr* = **0.001**, which led us to the conclusion that the best accuracy occurs when *max_iters* = **100** and *lr* = **0.0045** (see Fig.4).

Best results (*lr* = 0.0045, max_iters = 100):
Test set: accuracy = 87.156%
Fit time : 0.063515901565551 76 seconds
Predict time : 0.0000572204589 seconds

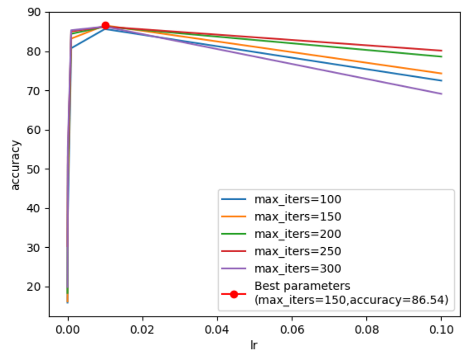


*Fig.3 : accuracy as a function of the learning rate (lr∈**[0.00001,0.1]**) for different max_iters*

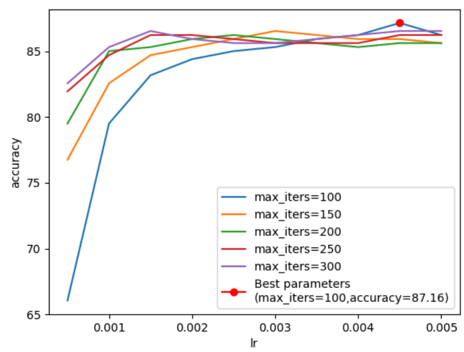


*Fig.4 : accuracy as a function of the learning rate(lr∈**[0.0005,0.005]**) for different max_iters*

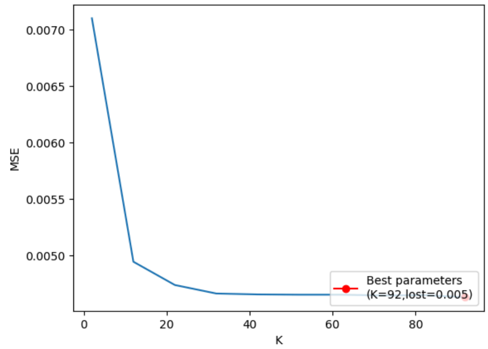
Najmeddine Abbassi (341889) Ghalia Bennani (344710) Lucas Simonnet (345619)

<u>KNN :</u>
In this part, we calculate the k-nearest neighbors by first computing the Euclidean distance between a given example and all the training points. We opt for the Euclidean distance seen in class, due to its simplicity and effectiveness, but alternatives like the chi-square distance could have also been utilized. After computing distances, we identify the k nearest elements. *K* serves as a hyperparameter, allowing us to specify the number of neighbors to consider. Increasing the value of *K* can result in more generalized patterns. However, it's important to note that excessively large values of *K* may lead to over-smoothing and potentially poorer performance on unseen data.

That is why when first testing values of *K*, for KNN as a Regression method, we decided to choose the interval ***[1,10]***.
Surprisingly, it was a decreasing function and the best *K* parameter returned was **10**. We then decided to extend the interval to ***[1,20]*** and then ***[1,30]*** but it always seemed like as *K* grows, the loss was decreasing. Finally, we decided to test on a very large interval to be fixed about our hypothesis and we choose to test K on the interval ***[1,100]***. The results are clear, the loss is indeed decreasing as *K* grows but converges to **0.0046** for *K*>=**30** (see Fig.5).

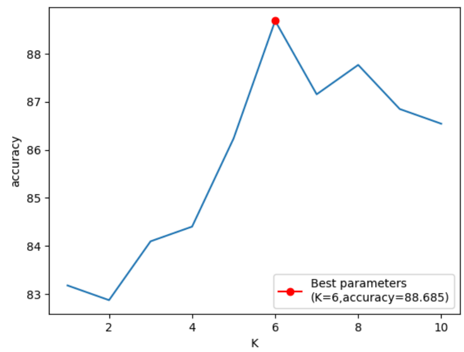
For KNN as a Classification method, we again started by testing for a small interval of *K* (see Fig.6). For this task, the results fitted our expectations as for *K*=**1**, the method was clearly overfitting (Train accuracy almost equal to **100%** but just **83%** for test data) and the best *K* parameter (*K*=**6**) gave us almost equivalent accuracy on both training and test data (test accuracy was **88.685%).**

KNN Regression best results:
Test loss = 0.0055
Fit time : 0.45464015007019043 seconds
Predict time : 0.052410125732421875 seconds



*<u>Fig.5 :</u> MSE as a function of the K parameter (K∈**[1,100]**)*

KNN Classification best results:
Test set: accuracy = 86.239%
Fit time : 0.4482560157775879 seconds
Predict time : 0.050034999847412 seconds



*<u>Fig.6 :</u> accuracy as a function of K parameter (K∈**[1,10]**)*