Advanced Machine Learning Project 1

Tymoteusz Kwieciński Mateusz Nizwantowski Kinga Frańczak

summer semester 2024

Contents

1 Introduction

In the following project our goal is to implement and thoroughly investigate the performance of 3 iterative optimization algorithms: SGD, ADAM, and IWLS. Within the project we evaluated the logistic regression model in comparison to the models with different architecture.

2 Implementation

Our implementation of the logistic regression model with different optimization algorithm was written in `Python` and consists of one class `LogisticRegressor` along with several functions that perform the optimization. The implementation is available on our repository As the loss function we have took the standard log-likelihood loss, therefore the process of fitting the logistic regression model consisted of finding the minimum of the given loss function.

We considered and implemented the following optimization methods:

1. **Stochastic Gradient Descent** - before each epoch the dataset is being randomly shuffled; gradient is then calculated for each observation individually and each observation updates the weights individually (the online approach) [1]

2. **minibatch gradient descent** - the idea is similar to the stochastic gradient descent, but the dataset is being divided to small batches and in every epoch; the gradient is averaged for observations withing the single minibatch [1]

3. **ADAM** - another modification of gradient descent; utilizes concept of momentum and decaying gradients [2]

4. **Newton–Raphson** - this method tries to find a point where gradients of the minimized function equals to 0. It creates approximations of the objective function using Taylor expansions.

5. **IWLS** - modification of Newton-Raphson method suited to the logistic regression problem. [4]

But according to the project requirements we have only compared performance of the SGD, ADAM and IWLS methods.

Our optimization functions are implemented in an universal way and except for the IWLS algorithm, which is suited for logistic regression problem with log-likelihood loss, could be used for finding the minimum of wider variety of functions.

IWLS and Newton methods were additionally regularized. These methods require finding an inversion of matrices, therefore to avoid calculating the inversion of a singular hessian, small regularization were added to the hessian matrix. In other words, we calculated the inversion of the following matrix H_{reg}:

$$H_{reg} = H + \lambda \cdot I_{pxp}$$

where p equals number of features and H is the calculated hessian matrix. λ is a small hyperparameter that we set to 10^{-6}.

In IWLS we also denoted that sometimes matrix $W = \text{diag}(P(1-P))$, where $P = \sigma(X\beta)$ can be singular in case the predicted vector of probabilities P has some values that equals to 0 or 1. Therefore we *clipped* its values so the values of the vector P are within the interval $[10^{-6}; 1 - 10^{-6}]$.

2.1 Interactions

Our logistic regression model have also the option to include, except for the original data features, the interactions between them. Our implementation does not change the original data, but creates a copy of the passed dataframe and adds the artificial features to the new dataframe. The larger input data with interactions is later passed to standard implementation of logistic regression.

3 Data used for comparison

As required, we have collected 9 datasets suitable for binary classification tasks. Full description and sources of the datasets can be found here, we used different sources when it comes to data. Some of datasets are originally designed for binary classification problems, in others we merged certain classes to obtain binary classification problem from multi-class problem.

We have prepared 3 small datasets that consist of less than 10 variables and 6 larger ones, which consists of more than 10 columns.

3.1 Real data preprocessing

In order to prepare our raw datasets downloaded from web to our problem, we needed to perform some preprocessing. Firstly, since logistic regression operates on numerical features only, we removed the columns that contained any other types of data. Later on, we investigated the collinearity of the features and removed columns that had absolute value of the correlation coefficient higher than 0.8. Additionally we removed the columns where there were more than 10% missing values and, where this condition was not met we replaced NaN with mean.

3.2 Artificial data

To better understand if the gradient calculation works and to visually asses its performance, we have tested our models on the artificially created dataset. Firstly we have sampled $n = 1000$ observation the from the two-dimensional standard normal distribution and denoted the data as X. Then, with $\beta := [1, 5, 5, 1]^T$ obtained the target y values generating $n = 1000$ observations from the binominal distribution with probability

$$p = \sigma\left(X_{mod}\beta\right)$$

Where X_{mod} is a matrix X where first column is an intercept term and last column is multiplication of columns of X.
The visualization of the generated dataset is shown on the 1

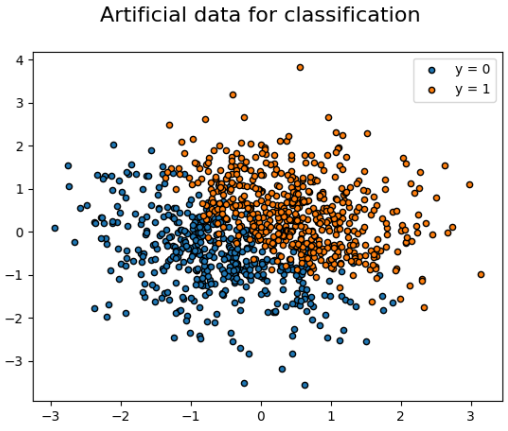

Figure 1: visualization of artificial dataset

To asses the way of calculating gradients of our loss function, we have created a contour plot 2 showing the minus log-likelihood for different values of coefficient β used in regression model.

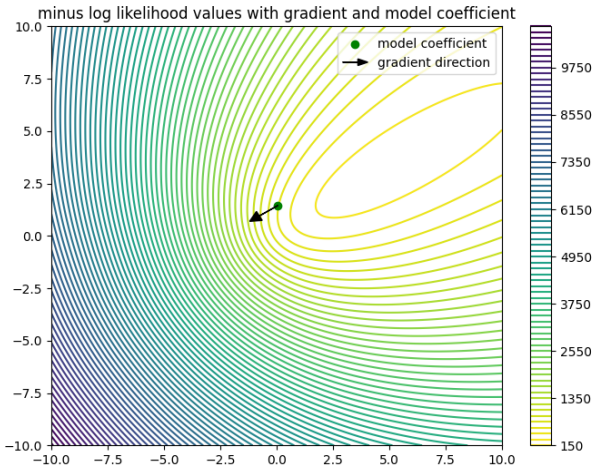

Figure 2: Contour plot of log-likelihood function with initial value of model coefficient and direction of gradient

4 Evaluation of performance

4.1 Description of methodology

Our main metric using which we performed the comparison between the different models and methods of classification were **balanced accuracy**. We compared performance using 10 different train-test splits, number of rows used for evaluating the models performance equaled to the 20% of the total number of observations in the given dataset.

For our implementation of the logistic regression we were also evaluating the values of loss function (log-likelihood) in each epoch during training, on the train data. Additionally we measure how much time algorithm is running.

4.2 Stopping rule

To make the comparison fair, in our performance analysis we have used the same stopping rule for all algorithms and datasets. To be exact, we used two different stopping rules:

1. limit of the L_∞ norm of the gradient: $\max_i |\Delta f(x)_i| < 10^{-6}$
2. time limit - training time was limited to 1 minute

During each epoch, we calculated the gradient of the objective (minus log-likelihood) function. We halted the further training if the maximum of the absolute values of the gradient was larger than 10^{-6}, which is essentially equivalent to finding the local minimum in case the function is differentiable. [3].

The second stopping condition was more down-to-earth one. Since our goal was to evaluate performance on plenty datasets and using enough repetitions to eliminate uncertainties due to randomness, we have decided that if an algorithm does not converge within 1 minute, the training should be stopped. All of the methods were trained using at most 500 epochs. If algorithm reached 500 epoch we halted further execution.

4.3 Different classification methods

We have also compared few different classification methods that were available in `scikit-learn` library. We evaluated the performance of QDA, LDA, Decision Trees and Random Forests.

4.4 Hyperparameters

For ADAM algorithm we used the default hyperparameters proposed by the authors of this method [2]. Newton and IWLS methods in our current implementation do not require any hyperparameters. However, we could not find any realible source proposing the default hyperparameters for SGD nor mini-batch SGD. We evaluated the performance of logistic regression model using learning rate set to 0.001 on dataset *diabetes*. The results of this evaluation can be seen on 3

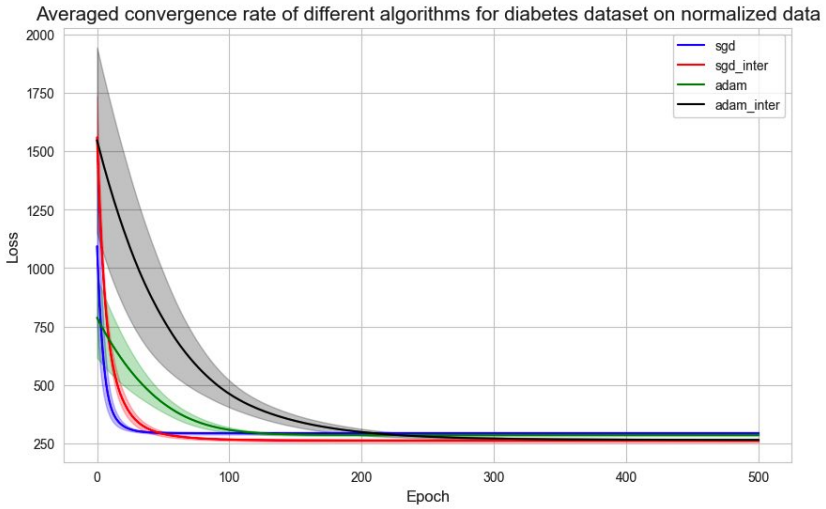

Figure 3: Comparison of convergence rate with the same learning rate for ADAM and SGD, considering logistic regression model with interactions added or without. We averaged the loss values for each epoch using the 5 different train-test splits.

Happy with the result, we set the default learning rate to 0.001 for SGD. Using smaller learning rate is a safer choice, since usually the algorithm with too small learning rate will simply converge slower. In case the learning rate is too big, the algorithm may not converge at all, oscillating around the solution. [1].

5 Results

5.1 Convergence analysis

Out of three algorithms SGD is the fastest one to converge, then ADAM and IWLS at the end. Unfortunately our implementation of IWLS is either hit or miss, and sometimes it does not converge. The most resilient to not optimal parameters, turns out to be ADAM. We did stumble for a runs of SGD where learning rate we used as a default was to high and algorithm had problems with convergence. Those problems disappeared when learning rate was fine-tuned manually, usually it was more than order of magnitude lower than default value. Given this results we thing it is worth to consider compromising, faster convergence, in order to obtain more reliable algorithm, that works for more tricky problems. Furthermore when we look at convergence rate of SGD, we can see it is one of the steepest line, this only solidifies our beliefs that learning rate for SGD should be decreased.

We did not need to tinker with other default parameters in ADAM such as *momentum_decay* or *squared_gradient_decay* in order to achieve good results. This fact confirms our perception of ADAM as one of the most versatile optimization algorithms.

What is interesting is the fact that even though, in some cases we did not converge at all (especially with IWLS), balanced accuracy was still high and comparable with algorithms that converged. Further testing is required to explain this behavior.

While convergence for ADAM and SGD is monotonic and smooth, that is not necessary true for IWLS even after the addition of regularization. On top of that we had trouble with computing solution using IWLS for the biggest dataset, while it managed to fit in our RAM, calculating Hessian base on it was impossible and that took more than 20GB of RAM, after that point kernel of our notebook was crashing. In order to obtain some results we limited datasets to using only 12k observations, less than 8% of our original data. Even then IWLS managed to calculate only one iteration during one minute. To summaries this approach gave us medicare results.

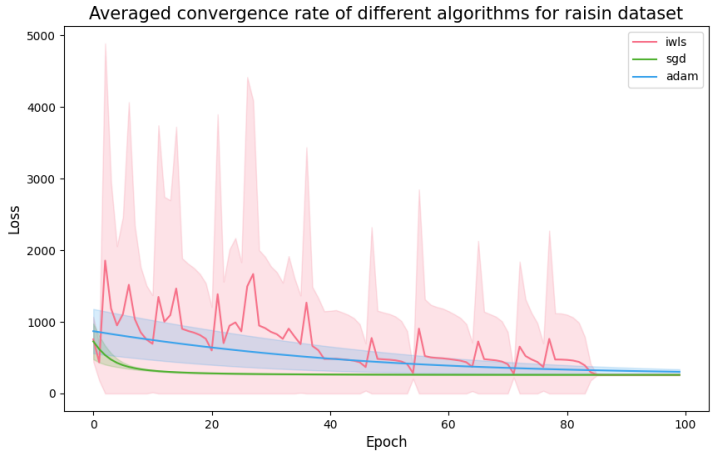

Figure 4: Rate of convergence for different algorithms, that shows weird IWLS behaviors

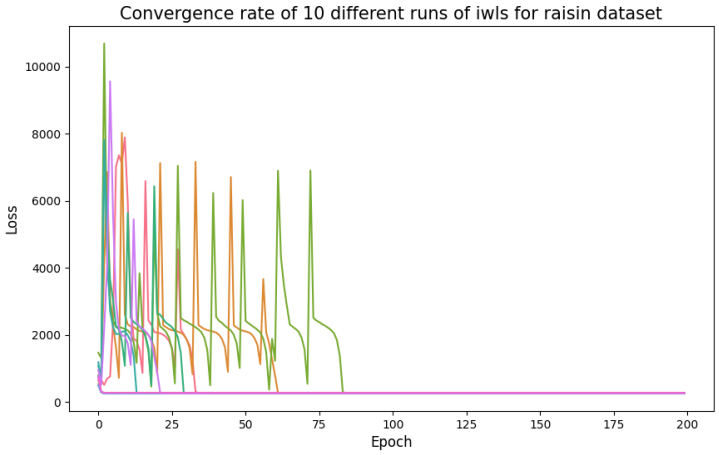

Figure 5: Rate of convergence for individual runs, some of them are converging while others oscillate

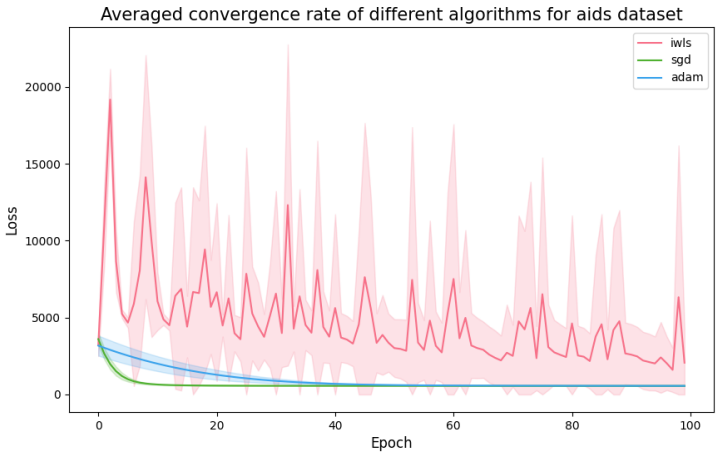

Figure 6: Rate of convergence for algorithms, IWLS has a hard time with converging

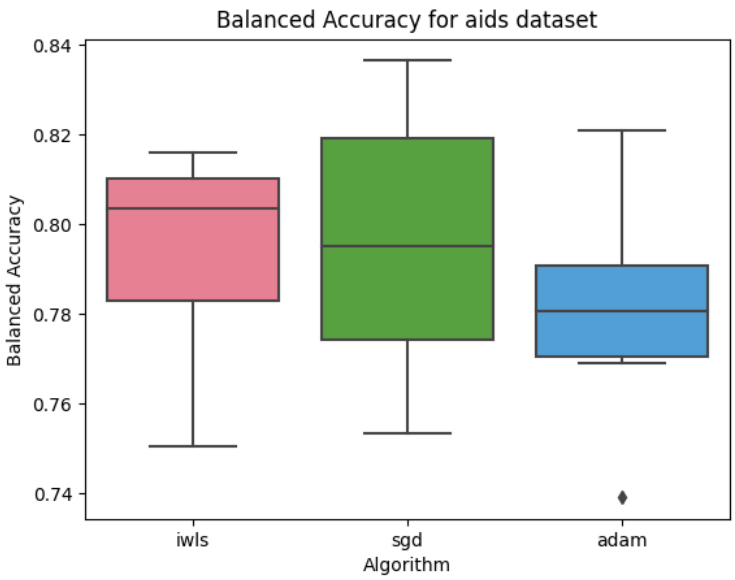

Figure 7: Even though we can see in the previous figure that IWLS has trouble with converging, balance accuracy is comparable with other algorithms.

The plots were build on 10 different train-test splits, we picked the most interesting examples and visualizations for other datasets can be found here

5.2 Comparison of classification performance

This group of datasets turned out to be especially diverse. It consists of data where achieving balance accuracy over 0.8 is not a special accomplishment, and the one where balance accuracy at 0.6 is the best any algorithm has got. On this data IWLS performed poorly. It is caused by the fact that on most of larger datasets algorithm performed only one iteration during one minute period. So we can not say about any type of convergence. Our implementation is not scalable enough to handle such high volume of data. On top of that the box for the box-plot of IWLS is the widest and usually centered at around 0.5 or slightly above. It is caused by randomized nature of initialization of parameters. It is like a coin flip, and it is hard to draw any conclusion from it. The ADAM and SGD are usually somewhere in the middle. There are algorithms that perform better than those two on a given dataset, but also there are methods that have worse balance accuracy.

From our testing we can conclude that on those datasets QDA performs the best especial on hard dataset (where best balanced accuracy is below 0.7), but solutions provided by QDA have the biggest variance. It is not as good when we are dealing with simple data. In those cases random forest tends to outperform other algorithms. Despite different train-test split SGD and ADAM are estimating consistently parameters that result in very similar balanced accuracy.

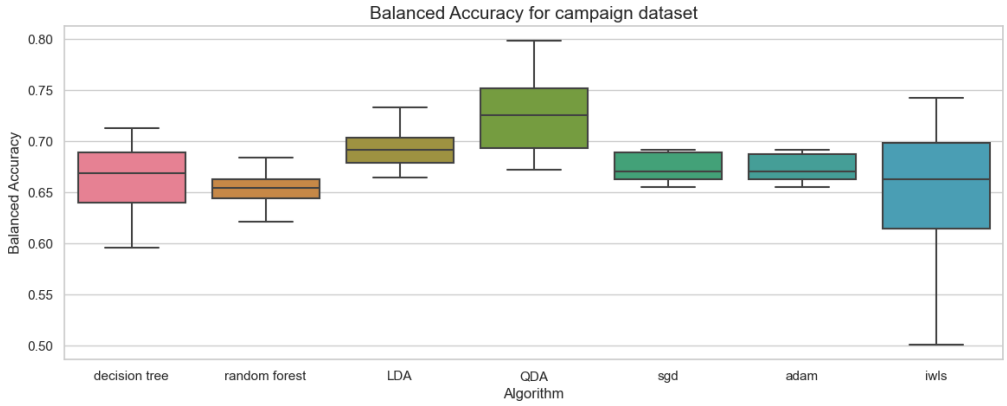

Figure 8: On challenging classification problem QDA perform the best, ADAM and SGD are in the middle of the pack.

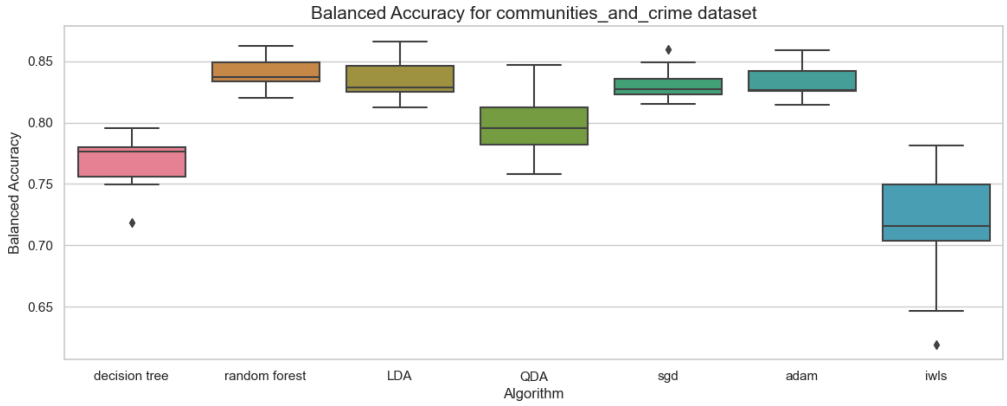

Figure 9: On a problem that is not as challenging, there are better candidates for classifier for example random forest or logistic regression with either ADAM or SGD

The plots were build on 10 different train-test splits, we picked the most interesting examples and visualizations for other datasets can be found here

5.3 Comparison of classification performance of models with and without interactions

Small datasets were not as challenging as large one. It was not that computationally expensive. While IWLS did manage to reach our time limit on diabetes data, it was able to make on average 45 iteration which is a huge improvement over one iteration, that was observed in certain datasets in previous section. 45 epochs is enough to reach satisfying results.

What is interesting, is the fact that it is harder to optimize problems with interactions. It was especially challenging for IWLS. To combat this phenomena we removed interactions that had correlation coefficient higher than 0.8 with original features, or with other interactions. While this procedure improved our performance metric, it did not solve issue completely. On the datasets we tested ADAM and SGD with interactions managed to beat their counterparts without interactions but only by a slim margin. IWLS with interactions perform about the same or slightly worse.

To summarize all algorithms performed exceptionally well. What is unexpected is the fact, that based on experiments we performed, our implementation of logistic regression is leading the charts, when it comes to balance accuracy.

The plots were build on 10 different train-test splits, we picked the most interesting examples and visualizations for other datasets can be found here

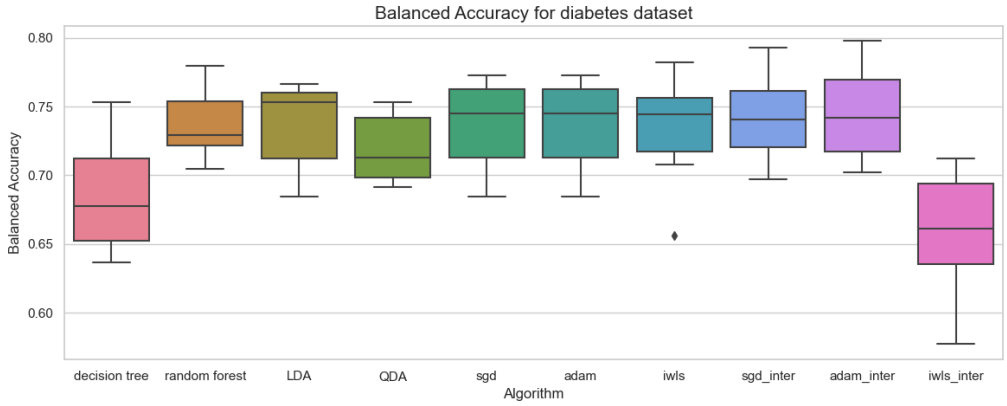

Figure 10: As we can see two of the best solutions were achieved by SGD with interactions and ADAM with interactions but at the same time the worst solution was produced by IWLS with interactions.

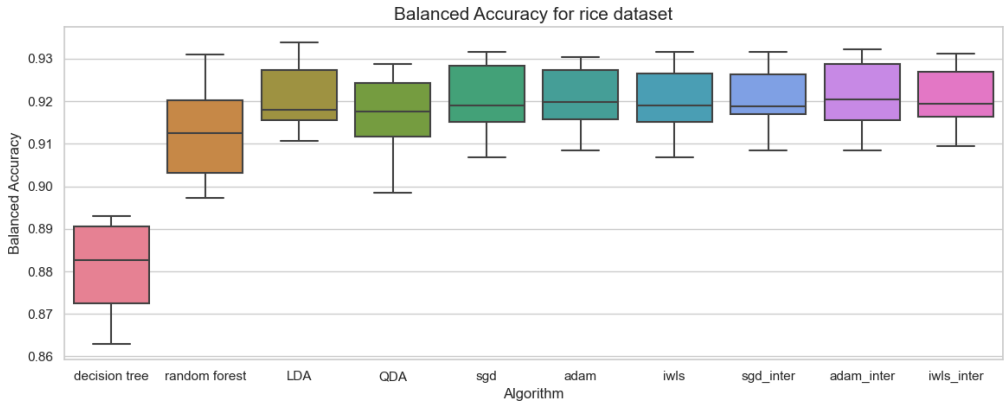

Figure 11: Here on rice dataset decision tree performed visible worse than the rest while all other methods are very close to each other

5.4 Time performance measurement

While we managed to reach 1 minute limit, especially in IWLS, where inverting a large matrices, is computationally expensive, most runs ended on 500 epoch or on our second stopping rule. We observe that we are not taking huge performance penalty by adding interactions (in general: additional columns). As we mentioned previously IWLS is the slowest algorithm, while ADAM is the fastest. To why ADAM is faster SGD, it is because we have implemented calculating gradients in a vectored manner, thus iterating over entire data set in chunks is faster than simply looping over elements. There is not a lot of variation in running time between different test-train splits. Plots for this section were build on 10 different test-train splits.

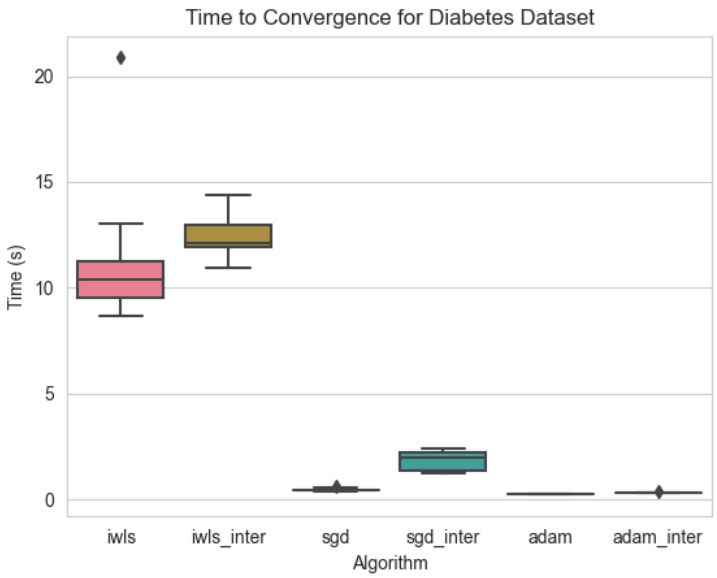

Figure 12: Example that visualizes the dependencies between the execution time of different algorithms

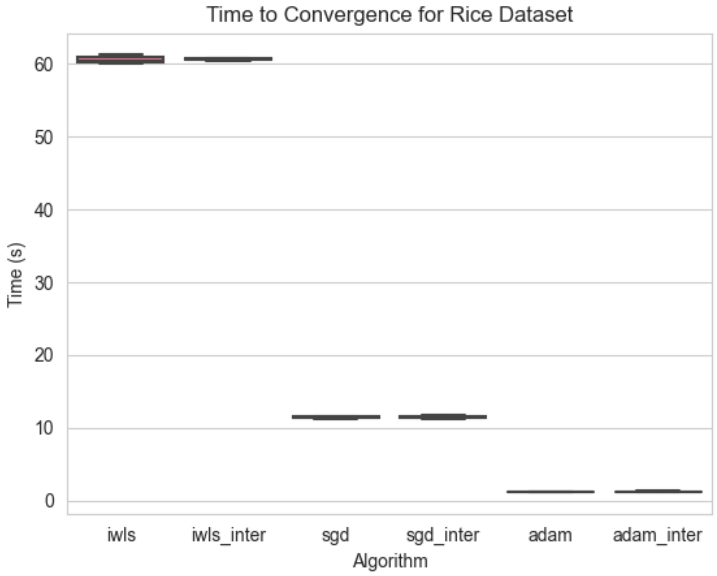

Figure 13: Example where each of IWLS runs ended with reaching time limit

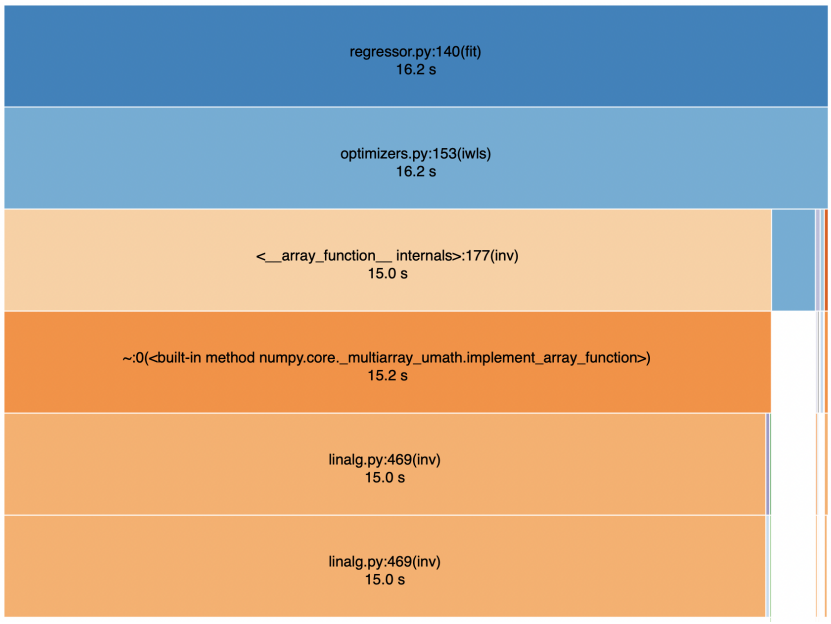

Figure 14: Screenshot from snakeviz (tool used to profiling), showing call stack of IWLS and time it spend running each function, here we see that over 90% of time it was busy with inverting matrices.

Other plots with time comparison for different datasets can be found here.

6 Summary

We learned a lot by doing this project. We have successfully implemented logistic regression and various optimization algorithms, used to find parameters that are minimizing cost function. Additionally we tested our implementation on artificial as well as on real data, in order to examine it thoroughly.

We had encounter many unexpected problems for example numerical instability or large volume of data. After discovery of those unwanted behaviors we had to understand and investigate issue and then come up with a solution that would solve the problem. This constant feedback loop has lead to improving our code to be more general and resilient to broken assumptions. During this process we had broaden our horizons when it comes to machine learning knowledge. We had and opportunity to face problems that we have previously only heard about in the lecture hall. And last but not least, one could argue it is the most important, we had a lot of fun while doing it.

References

[1] Léon Bottou. Online learning and stochastic approximations. 1998.

[2] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[3] prof. dr hab. inż. Radosław Pytlak. slides on optimization methods. accessed: 2023–10-10.

[4] prof. dr. hab. Jan Mielniczuk. slides on advanced machine learning. accessed: 2024–03-16.