Regression analysis and resampling methods

Ali Aslan Demir[1], Silja Borring Låstad[1], and Franziska Schöb[1]
[1] *The Njord Centre, Department of Physics, University of Oslo, P.O. Box 1048, 0316 Oslo, Norway*

This project explores a range of regression methods, including Ordinary Least Squares (OLS), Ridge regression, Lasso regression, and Logistic regression. The evaluation and exploration of these methods involve the utilization of optimization techniques such as Gradient Descent, as well as resampling techniques like bootstrap and cross-validation. While some of our findings are preliminary due to limitations in our code implementation, we have been able to conduct partial analyses. However, further work is necessary to comprehensively evaluate all the models under consideration.

I. INTRODUCTION

Regression is an analysis method to extract the relationship between dependent and independent variables in data. The model uses the independent variable as an input to predict the dependent variable and quantify the validity of the model with estimated parameters. Depending on the data type, researchers use various models such as linear, polynomial, or logistic regression. To investigate the impact of age and weight on high blood pressure in patients, the researchers used a regression model [1]. In the hypothesis, a positive correlation between age and weight is expected for high blood pressure, and the model is validated by analyzing this data.

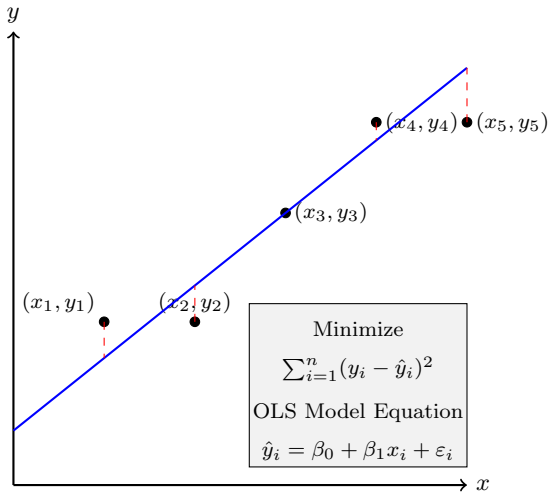


FIG. 1: Ordinary Least Squares Regression

II. METHODS

Data

For demonstration of the various regression and resampling methods, we make use of three distinct data sets. For the linear regression analysis (ordinary least squares, Ridge regression and Lasso regression) we use the Franke function, a function that is often used to test regression models. It contains both peaks and valleys and saddle points in the rage of [0, 1]. Due to its inherent non-linearity, it is an ideal function for exploration of regression approaches and their evaluation [2]:

For the Stochastic Gradient Descent we use a simpler, second order polynomial on the form $f(x) = a_0 + a_1 x + a_2 + x^2$, with x as before ranging from 0 to 1. Lastly, to study classification with logistic regression, we are using the publicly available Wisconsin Breast Cancer dataset.

Regression

Ordinary Least Squares (OLS) To determine minimizing residual errors between two or multiple data points, OLS uses maximum likelihood to find the best-fitting line and optimal coefficients. OLS builds up bases for linear analysis models [3]. In the OLS model, y_i refers to an actual value, and $\hat{y}_i$ refers to a predicted value. β_0 refers to an intercept of the model. β_1 is the coefficient of independent variable x_i. ε_i is the random error term. Corresponding functions are shown inside the Figure 1 with the gray box.

Lasso Regression Lasso regression, also often referred to as L1-regularization, is a common type of regularization used in modeling and machine learning used to tackle the problem of overfitting. It is based on simple linear regression but introduces a penalty for coefficients to make them shrink [4]. This can lead to coefficients being set to zero, thereby eliminating them from the model and simplifying it.

Ridge Regression Ridge regression is similar to Lasso regression but uses L2-regularization in order to shrink model coefficients to improve regression results and reduce overfitting. This means model parameters cannot be set to zero and therefore will not be completely eliminated [4].

Logistic Regression Logistic regression is commonly used for classification problems where unbounded linear regression is unsuitable. Most often, logistic regression is used to reach a binary prediction result such as true/false, malignant/benign. Logistic regression can be understood as a linear regression but with a sigmoidal function used for classification. Logistic regression can also be used

for multi-class classifications and is then referred to as multinomial logistic regression.

Resampling

Bias-Variance Tradeoff Bias-Variance is a way of understanding the model generalization and accuracy capacity based on how well the error is minimized. In other words, it acts like an uncertainty principle while trying to have low bias and variance [5]. Specifically, bias is trying to minimize the average prediction close to the correct value. On the other hand, variance determines the spreading of the predicted results from the actual expectation. Possible scenarios for bias-variance are illustrated in Figure 2.

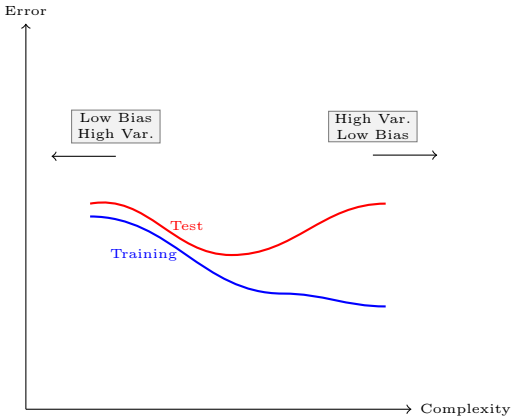
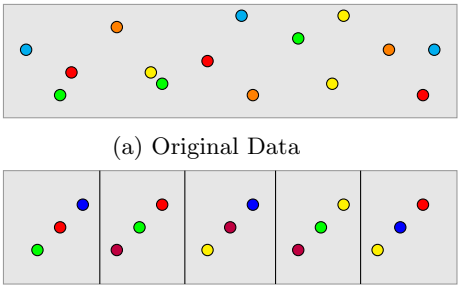


FIG. 2: Bias-Variance Tradeoff

Bootstrapping Bootstrapping is a random data replacement method for a variety of estimation problems [6]. It gives a summary of the statistical model to understand accuracy and variability. Bootstrap sampling takes a small dataset and learns the features by statistical analysis. In machine learning, bootstrapping is used with unseen data to understand the generalization capacity of the trained model and avoids overfitting problems depending on the dataset size. In the case of a randomly distributed dataset, small bootstrap samples can be taken with the replacement of the data to understand the model performance as illustrated in Figure 3.

Optimizers - Stochastic Gradient Descent

Stochastic Gradient Descent, in the following referred to as SGD, is an iterative process fro optimization used in machine learning. The method uses a stochastic approximation of the gradient instead of the actual gradient, thereby allowing us to use it for high-dimensional problems which would require immense computational power in order to calculate the actual gradient. In SGD,



(a) Original Data

(b) Bootstrap Samples

FIG. 3: Bootstrapping Data

the gradient estimation is updated for each training data point often for several iterations through the whole training set. The algorithm is usually stopped when it converges and the update falls under a chosen threshold [7].

$$w \leftarrow w - \eta \left[\alpha \frac{\partial R(w)}{\partial w} + \frac{\partial L(w^T x_i + b, y_i)}{\partial w} \right] \tag{1}$$

Many variants of SGD exist, some of the most used are SGD with Momentum, AdaGrad, and RMSProp.

Momentum The main difference between SGD with and without Momentum is that Momentum allows the SGD algorithm to keep track of the previous update of the gradient. This memory of the previous update is then used in the new update by building a linear combination of the gradient estimation and the previous update [8]. Momentum prevents the gradient updates from oscillating, helping with reaching convergence.

AdaGrad AdaGrad stands for Adaptive Gradient Descent and describes a form of SGD where the learning rate is adapted for each parameter [8]. This is especially useful for problems with sparse data and parameters, as it tends to keep the learning rate higher for these cases.

RMSProp RMSProp is short for Root Mean Square Propagation and resembles AdaGrad closely, as also here the learning rate is adapted for each parameter. However, instead of just using memory of previous gradients for the update, RMSProp uses a running average of the size of the most recent gradients and divides the learning rate by it. This gives stronger influence to more recent gradients and helps solve AdaGrads problem of vanishing gradients [8].

Adam Adam or Adaptive Moment Estimation uses RMSProp and adds Momentum to it. This combination leads to strong influence of recent gradients and momentums, while less recent ones are forgotten by the algo-

rithm [9]. There exist several improved versions of Adam (AdaMax, AMSGrad, etc), which will not be described in detail in this report.

III. RESULTS & DISCUSSION

Linear regression

For all of the results in this section we have split our data in test and training sets. Here, we have used 1/4 of the total data for testing, which is an arbitrary choice. When increasing the size of the test data we expect the error of our predictions to go down, which is also what we see. However, when we increase the absolute size of our data set (by decreasing the step size in the mesh) we observe that the increasing the test size affects our predictions less. It is reasonable that the error on the prediction goes down when we keep the fraction of test data constant and increase the size of our data set, because we have more data to train on.

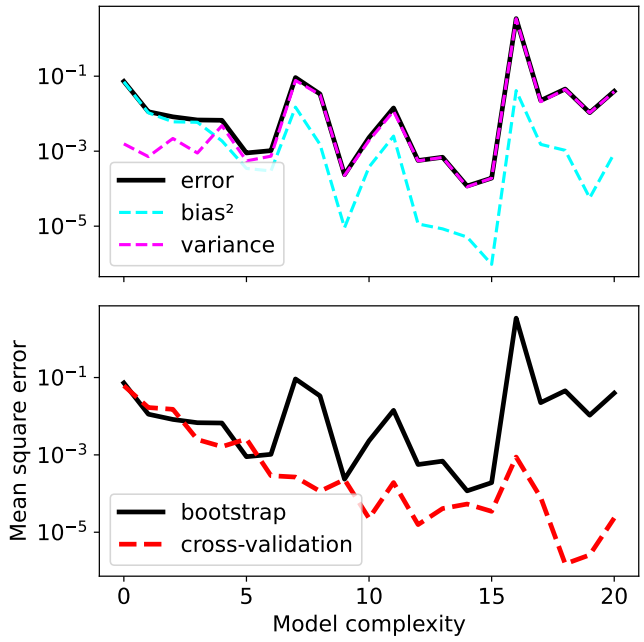


FIG. 4: **Ordinary Least Squares** on the Franke function. Model complexity corresponds to the polynomial degree of the design matrix. Here, 1/4 of the total data is used as test data. **Upper:** Bias-variance trade off computed with the bootstrap resamplig technique. Here, we have used 100 bootstraps. **Lower:** Mean square error computed using either bootstrap or k-fold cross-validation for resampling.

First, we study the ordinary least squares regression. In FIG. 4 we have plotted the errors from using OLS, and resampling either with bootstrap or cross-validation. The plots are noisy and missing a clear trend. This does not appear to be a statistical feature, as increasing the number of bootstraps does not improve the plot, indicating instead that there is an issue with the code. In the upper plot we have plotted the bias-variance tradeoff, computed with bootstrapping. The figure is very noisy and it is hard to confirm whether the error behaves as expected (First decreasing with complexity as the model becomes more accurate, then increasing with complexity as we are over-fitting). However, we see that the squared bias and the variance sum to the error (note the logarithmic axes). Furthermore, we see that at low model complexity the error is dominated by the bias. This is what we would expect since a model of low complexity has poor accuracy but higher precision. However, as the model complexity increases the variance dominates the error, which is again what we would expect since a model of high complexity is more flexible and thus more accurate at the cost of precision.

In the lower plot, we compare the mean square error for bootstrap (the same as in the upper plot) and k-fold cross-validation, respectively. The cross-validation error is less noisy, and somewhat consistently decreasing with model complexity. It seems unlikely that the optimal model is a polynomial of power 20, hence this is probably also due to a bug in the algorithm. In general, we expect the error to follow the same curve for the two resampling techniques, the error from the bootstrap algorithm being slightly higher than for the cross-validation. This is because the bootstrap algorithm keeps the test data out of the resampling, and therefore does not "see" the test data before making the predicten. In constrast, the k-fold cross-validation algorithm iterates through the data used as test data, as the test data of one iteration is part of the training data of another iteration. Based on this, we also argue that bootstrap is a little more sensitive to fluctuations, as the algorithm does not have access to the entire data set. However, this sensitivity should disappear when we increase the statistics, hence it is not related to the fluctuations we see in FIG. 4.

Then we compare OLS with the corresponding results from Ridge- and Lasso regression. In Figs. 5-6 we have plotted the mean square error, bias-term and variance-term using bootstrap, as well as the mean square error using cross-validation, for Ridge- and Lasso regression, respectively. We have chosen three values for the additional parameter $\lambda \in \{10^{-10}, 10^{-5}, 1\}$. for both Ridge and Lasso $\lambda = 0$ corresponds to the normal OLS, and increasing λ monotonically increases their difference from the OLS. Just like in FIG. 4, the errors are fluctuating heavily and it is not obvious what the trend is. The upper figures ($\lambda = 10^{-10}$) appear very similar to the case of OLS, which is as expected when λ is very small. As λ increases, the cross-validation error increases, which is again what we would expect, since increasing λ increases the significance of the distortion of our original system.

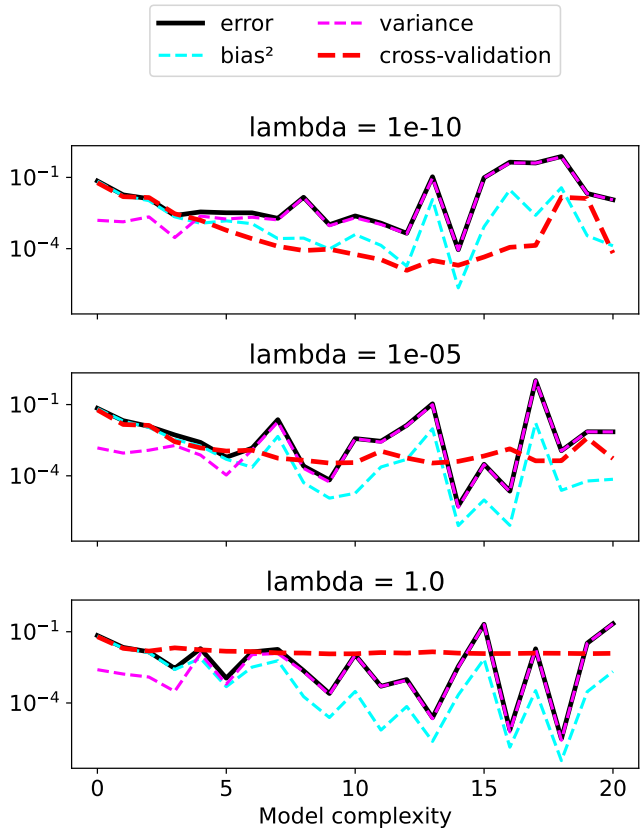


FIG. 5: **Rigde bias-variance tradeoff** on the Franke function. Model complexity corresponds to the polynomial degree of the design matrix. **Upper:** Using the bootstrap resampling technique. **Lower:** Using cross-validation as resampling technique.

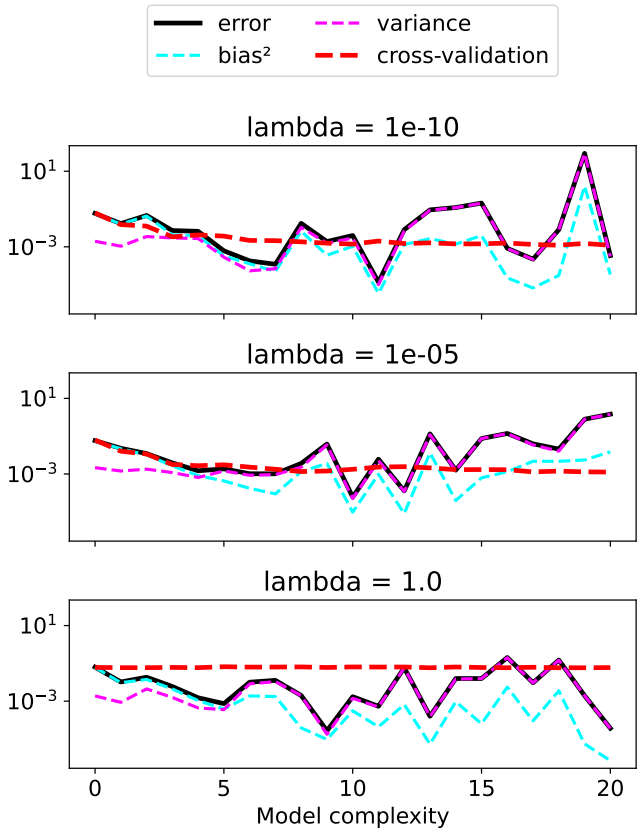


FIG. 6: **Lasso bias-variance tradeoff** on the Franke function. Model complexity corresponds to the polynomial degree of the design matrix. **Upper:** Using the bootstrap resampling technique. **Lower:** Using cross-validation as resampling technique.

Contrary to what we would expect, the bootstrap values do not appear to be affected by λ. Again, this is likely related to a bug in our code.

Convex optimization

Then we turn the task of optimization, using gradient descent to estimate the coefficients of our model, instead of by matrix inversion. First, we study just the plain gradient descent, where our estimate for β is updated with the its gradient at every iteration. In FIG. 7 we have plotted the number of iterations it takes for β to converge, as well as the cost function of our final estimate for β. Both in the case of OLS and Ridge, we see that the number of iterations decreases with learning rate, until the point where they don't converge, because the learning rate is too large, and we therefore miss the minimum. For almost all learning rates we have that Ridge uses fewer iterations to converge than standard OLS, whereas the cost function is higher for Ridge than OLS. The cost appears to be almost constant, but this is just because the fluctuations are on a much smaller scale than the difference between the two.

We then extend our gradient descent function to also include a momentum/memory term, and plot this against learning rate for several values of momentum. This can be seen in FIG. 8, where we have plotted number of iterations it takes to converge along with the cost function. Not surprisingly, we see that the number of iterations decreases with learning rate, as it should since including momentum has no effect on the learning rate. Furthermore, we see that the number of iterations decrease non-linearly with the momentum value, meaning that including a small momentum term has a good effect, but that increasing the momentum further has limited effect. This seems meaningful, as the point of the momentum term is to make the gradient descent less sensitive to fluctuations in the gradient, and thus converge easier. However, if the momentum term becomes too large, it will out-compete

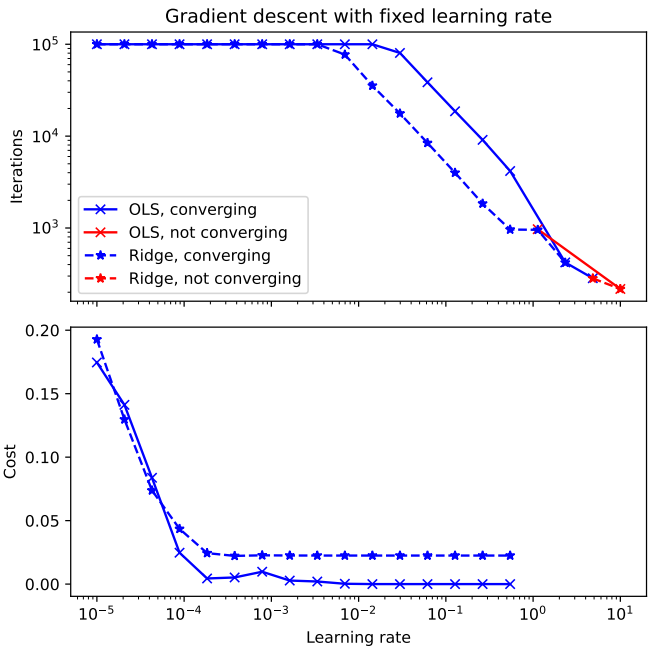


FIG. 7: **Gradient descent with fixed learning rate** for OLS and Ridge regression. For Ridge we have used $\lambda = 0.01$. **Upper:** Number of iterations before the algorithm converges as function of learning rate. Blue marks the results that actually converge to the expected β, and red marks the one that get stuck in a local minimum/sadlepoint. **Lower:** Cost function as function of learning rate.

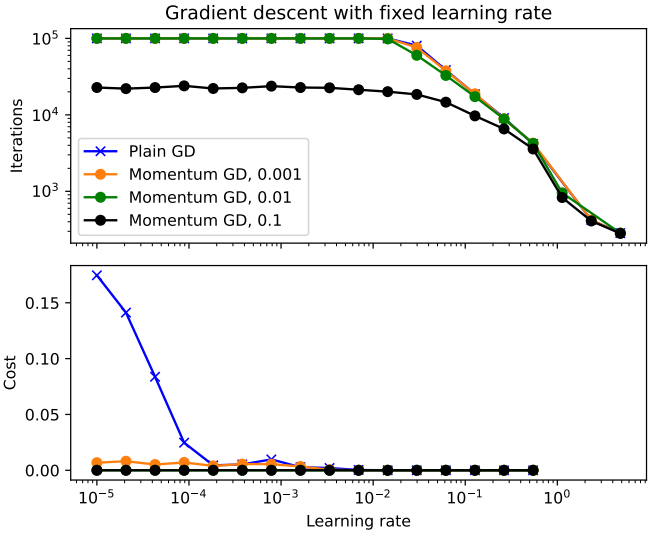


FIG. 8: **Gradient descent with momentum and fixed learning rate** for OLS.

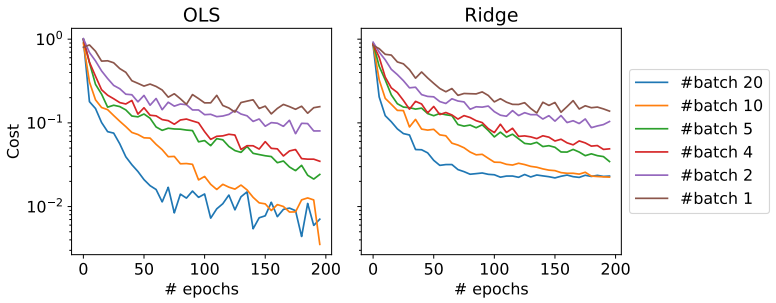


FIG. 9: **Stochastic gradient descent** without momentum as function of number of epochs. '#batch' is the total number of mini-batches of the SGD. **Left:** For Ordinary Least Squares method. **Right:** For Ridge regression, here $\lambda = 10^{-2}$.

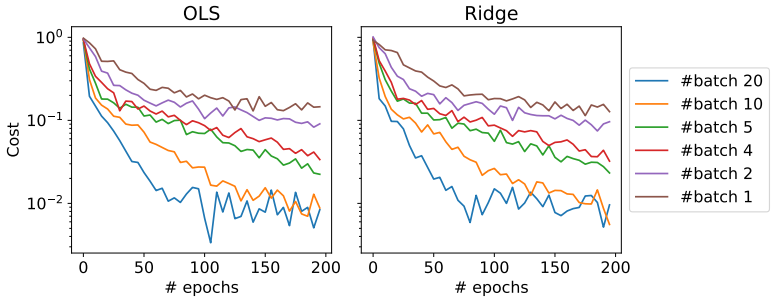


FIG. 10: **Stochastic gradient descent** without momentum as function of number of epochs. '#batch' is the total number of mini-batches of the SGD. **Left:** For Ordinary Least Squares method. **Right:** For Ridge regression, here $\lambda = 10^{-4}$.

the learning rate and thus prevent convergence.

Then we explore SGD, both for OLS and Ridge regression In FIG. 9-10 we have plotted the cost function of SGD without momentum as a function of the number of epochs, for number of mini-batches ranging from 1 to 20. In FIG. 9 $\lambda = 10^{-2}$, whereas in FIG. 10 $\lambda = 10^{-4}$. We see that the cost function decreases with the number of epoches, in an exponentially decreasing manner. This is what we would expect as our model should converge as we get more statistics, We also observe that the cost function decreases with the number of mini-batches, with the case of one single batch corresponding to the case of the plain GD, which is the one that preforms the poorest. Lastly, we also notice that the cost is higher for the higher λ, which is in correspondence with what we have observed earlier.

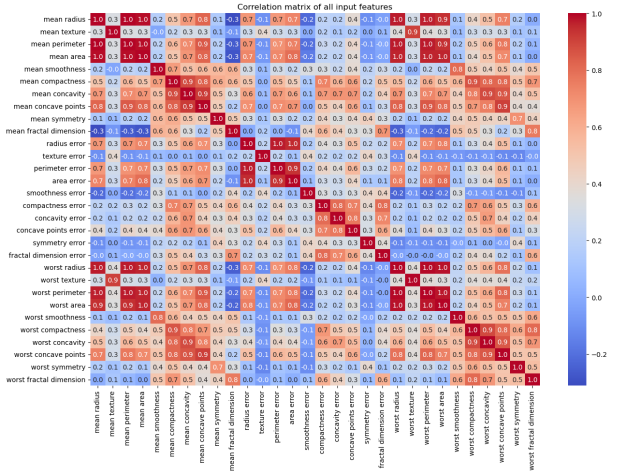


FIG. 11: **Correlation matrix of input features** in the Wisconsin breast cancer dataset. There is one clear cluster of correlated features consisting of mean/worst radius, perimeter and area.

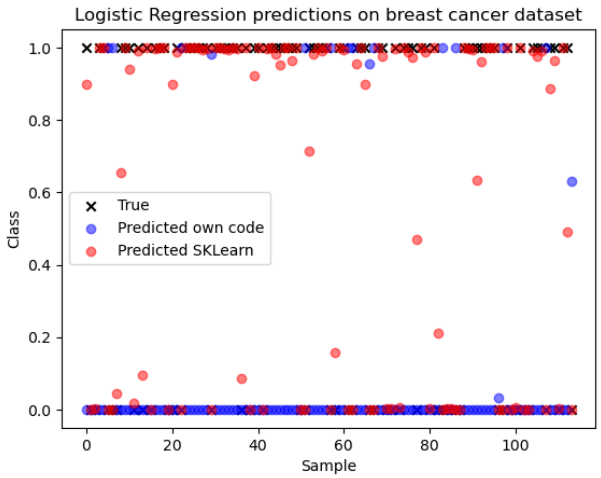


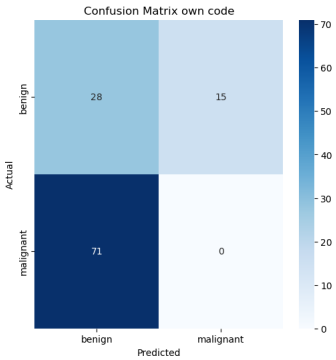
FIG. 12: **Predictions results** of both implementations. Black crosses display true labels, while blue and red dots show predictions results. Blue dots correspond to our own code and red dots represent results from sklearn.

Classification with logistic regression

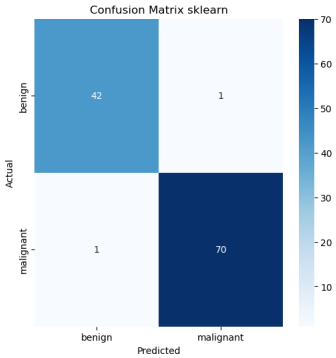
The Wisconsin Breast Cancer dataset consists of two classes with 212 malignant and 357 benign tumor samples. It has 30 input features which can be used for classification. In fig 11 we have plotted the correlation matrix of input features to explore the dataset. Most input features seem to have no strong correlation to each other.

For our logistic regression, we have built a function which uses our own implementation of stochastic gradient descent and a simple sigmoid function as cost function. We compare this function with the implementation provided by the scikit learn package. We have used the following learning rates: 0.1, 0.01, 0.001 and 1000 iterations. However, we did not find any differences in accuracy for the classification results based on the different learning rates. We reached a maximal accuracy of 25.4 for our own implementation, while sklearn's implementation reached 98.2 accuracy. In figure 12 it is already visible that our code predicts far more 0.0 class labels, which represent benign tumors compared to the true label number. As is clear from these results, there is lots of room for improvement in our own implementation. In future work, we suggest to try to include different gradient descent methods and selection of different cost functions.

Figure 13 shows confusion matrices for the classification results of both our own code and the provided sklearn implementation. Our own implementation classifies many malignant tumors as benign, while the sklearn implementation classifies both malignant and benign tumors mostly correctly.



(a) Confusion matrix for classification done by our own implementation



(b) Confusion matrix fro classification by sklearn implementation

FIG. 13: Confusion matrices for both our own code and the sklearn implementation. It is clearly visible that the provided implementation can classify almost all samples correctly, while our code classifies a lot of malignant tumors as benign.

IV. CONCLUSION

We have conducted three different methods of linear regression on the Franke function and observed that the Ordinary Least Squares (OLS) method outperforms Ridge regression in terms of error/cost. However, OLS exhibits slower convergence due to the additional parameter introduced by Ridge and Lasso regression, which modifies the cost function. Because of an unidentified error in our resampling function, we where prevented from accurately comparing the bootstrap method and the cross-validation methods.

For the optimization methods, we found that including stochasticity or momentum indeed decreased the cost function, as expected. However, also here our results were limited by non-functioning code, as we did not get to exlpore the adaptive learning rates corresponding to AdaGrad, RMSProp and Adam. In further work, these should be imlpemented properly.

Lastly, we used the SGD to implement logistic regression in order to classify breast cancer tumors. As our classifiers only reached a accuracy of 25.4 %, also this parts need further work.

[1] A. Schneider, G. Hommel, and M. Blettner, Dtsch Arztebl International **107**, 776 (2010).

[2] R. Franke, Technology Report NPS-53-79-003 (1979), 10.21236/ADA081688.

[3] J. T. Pohlman and D. W. Leitner (2003).

[4] L. E. Melkumova and S. Y. Shatskikh, Procedia Engineering **201**, 746 (2017).

[5] S. Geman, E. Bienenstock, and R. Doursat, Neural Computation **4**, 1 (1992).

[6] B. Efron, The Annals of Statistics **7**, 1 (1979).

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Nature 1986 323:6088 **323**, 533 (1986).

[8] E. Yazan and M. F. Talu, IDAP 2017 - International Artificial Intelligence and Data Processing Symposium (2017), 10.1109/IDAP.2017.8090299.

[9] D. P. Kingma and J. Lei Ba, *ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION*, Tech. Rep., arXiv:1412.6980v9.