Mini-Project 1: Getting Started with Machine Learning

Yijie Zhang
McGill University
yj.zhang@mail.mcgill.ca

Tian Bai
McGill University
tian.bai3@mail.mcgill.ca

Yiping Liu
McGill University
yiping.liu@mail.mcgill.ca

Abstract

In this project, we implement two machine learning (ML) models - linear regression and softmax regression (with gradient descent (GD)) on the Boston Housing dataset and Wine dataset respectively, and test the performance of different evaluation methods and hyperparameters. The primary discovery of this study all evaluation methods and hyperparameters should be chosen based on the characteristics of specific datasets and experiments on the performance of different hyperparameters. For linear regression, adding Gaussian basis to enrich the features of the dataset could improve the performance but it could also cause problems like overfitting. Comparing analytical linear regression and SGD-based linear regression, we find the performance of analytical linear regression is better. The exploration for best performance is a balance between various factors.

1 Introduction

The widespread adoption of ML has seen remarkable acceleration across various domains, including medicine, marketing, sciences, and social networks. Various techniques have been employed within machine ML to anticipate diverse forms of data. Linear regression and softmax regression are two of the most basic and classical applications in the classification. Linear regression is a mathematical approach used to perform predictive analysis that allows continuous/real or mathematical variable projections. It is commonly used in mathematical research methods, where it is possible to measure the predicted effects and model them against multiple input variables (1). softmax regression is a statistical model to describes the relationship between a binary or dichotomous outcome and a set of independent predictor and explanatory variables. It trains input text data and returns a function that maps input features to a probability distribution across output categories (2).

Creating a proficient machine learning model is an intricate and time-intensive endeavor. It encompasses the selection of a suitable algorithm and the refinement of its hyperparameters to achieve an optimal model architecture. The essence of most machine learning models is rooted in constructing an optimization model and determining its parameters from the given data. In today's data-rich environment, the role of numerical optimization in enhancing the efficacy of machine learning models is undeniable, playing a key role in their widespread use and real-world application (3; 4).

The primary object of this project is a) to implement two models on two datasets and explore the influence of different evaluation methods and hyperparameters on the performance of both models; b) to explore the influence of enriching dataset features and different optimization methods for linear regression.

In the experiments, we observed that larger hyperparameters and adding nonlinear features could improve the performance of models. However, too large hyperparameters can also cause several problems such as higher computational complexity and overfitting. Besides, the performance of different evaluation methods and optimization methods is influenced by many factors including the datasets' characteristics and targets for modeling. To find the optimal parameters and methods, we need to carry out more experiments and analyze the datasets carefully.

2 Datasets

2.1 Overview

The Boston Housing dataset comprises 506 data samples and originally includes 14 real attributes. For ethical reasons, we removed the feature "B" and utilized 13 attributes for analysis. The target variable is the median value of owner-occupied homes in $1000s, denoted by the 'MEDV' feature. The Wine dataset consists of 178 data samples and 13 attributes. The target variable is the class of wine, which ranges from 1 to 3.

2.2 Feature Analysis

In this project, a feature analysis was conducted to determine the appropriate features for both datasets. Firstly, we checked whether there is any missing value for the attributives and outputs to ensure we have the full dataset for the experiment, the results are shown in Figure 1 and Figure 2. Seeing from the results, we could obviously find that there are not any missing or malformed features or data oddities that need to be dealt with.
To visualize the attributes and output, we render the dataset statistics. The histogram of the datasets' attributives can be found in Figure 3 and 4. Figure 5 indicates the distribution of MEDV and Figure 6 shows the distribution of classes in the wine dataset.

3 Results

3.1 80/20 train/test split

In this part, we applied our machine learning models on respective datasets and tested the performance of our model under default hyperparameters and a 80/20 train/test split. As shown in figure 7 and figure 8, we used gradient's norm per step and training loss per step as metrics to keep track of our training process. We observe that the gradient's norm and training loss gradually converge to a stable value as the iteration increases. After the training phase, we also applied our model on the test set to see whether it generalizes well. We obtained a loss of 12.01265457 for analytical linear regression, a loss of 12.23696162 for linear regression with mini-batch stochastic gradient descent, and a loss 0.21947868197303977 for softmax regression. Note that the scale of loss for different models is different and should not be directly compared with each other(5). Generally, both of the two ML methods have a good performance on the respective dataset.

3.2 5-fold cross-validation

Figure 9 and Figure 10 show the performance of 5-fold cross-validation on the datasets. We can see that the figures here are very similar to those in the last section. It is calculated that the test loss of analytical linear regression and linear regression with minibatch SGD on the first dataset are 12.803416 and 13.2590642, and the test loss of softmax regression on the second dataset is 0.2702554424857716. Relative results for softmax regression are listed in Table 1.

Table 1: Accuracy, precision, recall, F1 score for softmax regression.

Method	accuracy	precision	recall	F1 score
80/20 train/test split	0.971428	0.964102	0.961538	0.961365
5-fold cross-validation	0.941836	0.932643	0.917285	0.922101

3.3 Performance under different training sizes

Figure 11 and Figure 12 show the performance of both models under different training sizes. For the Boston Housing dataset, as the training size increases, training and test loss fluctuate but gradually decrease to a stable value. However, for the wine dataset, the training loss increases with the training size while the test loss decreases as the size increases. This could

be due to the similarity of data within the small training set, and as we consider larger and larger datasets, the data is more heterogeneous. Figure 13 indicates the accuracy, precision, recall, and F1 score curves of the softmax regression. The curves have fluctuated but show an upward trend. Training loss (especially when the training set is very small) could not be viewed as a reliable metric for performance, and we conclude that in general the performance is positively correlated with the training size. This trend could have several reasons: a) Sample diversity: a larger training set contains a greater variety of samples, allowing the model to learn a wider range of features and patterns so that it generalizes well and performs well in predicting test sets. b) Reduced overfitting: a larger training set can make the model more robust to unseen data. c) Better gradient estimates: With more samples for parameter update, the gradient estimates would be more accurate as well, which makes the training process more stable and efficient. What we should notice is that if the training set is too large, it would cause high computational cost and diminishing returns as well(6). Moreover, the data quality also influences the results.

3.4 Performance under different minibatch sizes

Figure 14 and Figure 15 exhibit the gradient's norm per step in SGD under different batch sizes for both models. As the batch size increases, the gradient's norm decreases faster and the model becomes more stable. We could infer that this is due to the better stability of larger batch sizes: a larger batch size can reduce the noise between samples within each batch, which means the update direction becomes more stable, reducing randomness and making the training process more stable. However in practice, to find the best batch size, we should consider the changes in training performance as well as be mindful of memory constraints and learning rate adjustments.

3.5 Performance under different learning rates

Figure 16 and Figure 17 indicate the performance of both models under different learning rates. The situation is similar to the performance under different batch sizes. As the learning rate increases, the norm fluctuates less and decreases faster. We could also observe that when the learning rate is inappropriately small, such as 1e-5, the noisy SGD may fail to converge. In general, a large learning rate can bring the following benefits: a) Faster convergence speed: parameter updates are larger, causing the model to move more rapidly in parameter space. b) Stability: a smaller learning rate may cause the model to get stuck in the local minimum and a larger rate can skip over the local minimums. Because in both of our models the loss functions are smooth and convex, our data did not reveal the drawback of large learning rates.

3.6 Select appropriate metric and choose optimal parameter based on the metric

As listed in Table 2, considering the performance of different metrics, we take batch size=8, and learning rate=0.1 as the best parameters. From Table 3 and Table 4 we take batch size=32, and learning rate=0.1 as the best parameters. That smaller batch size results in better performance seems to contradict with our theory in 3.4. However, an important clarification has to be made: in our implementation of mini-batch SGD, we go through the whole dataset once per epoch no matter how small the batch size is. If the batch size is smaller, we go through the dataset step by step. This will give the mini-batch SGD with a smaller batch size to have more chances to update its gradient and learn. Since before every update, we record the metrics such as the gradient's norm and loss, small batch sizes give poor performance in 3.4. But here, since the number of epochs is fixed, mini-batch SGD's advantage is exposed.

3.7 Performance after adding Gaussian Basis for Boston Housing dataset

Generally, by introducing the Gaussian basis function, the model can learn the nonlinear relationship, which makes it possible to perform a good fit even when linear modeling is not possible in the original feature space. However, if too many Gaussian basis functions are used, the model may overfit the training data, resulting in poor performance on previously unseen data. In our case, adding the Gaussian basis feature to SGD linear regression gives a similar performance as linear regression without Gaussian features, but with low stability. Over repeated experiments, Gaussian basis linear regression gives a loss of about 11. Nevertheless, we observed disastrous performance when adding Gaussian features to analytical linear regression: the output is very unstable and the loss can be very large. We speculate that this could be due to the randomness when selecting the position of Gaussian functions. Moreover, the property of Gaussian functions also worsens this effect: Gaussian functions give nearly zero output when the input is far from its center, especially when the

Table 2: Test loss for different metrics-Boston Housing dataset.

batch size \ learning rate	1e-5	1e-4	1e-3	0.01	0.1
8	113.967	12.0390	12.0201	11.8770	11.0013
16	180.737	14.5119	12.0480	11.9679	11.0885
32	233.714	35.8424	12.0215	12.0024	11.5237
64	270.521	91.7083	11.9803	12.0141	11.8152
128	293.371	160.367	12.9066	12.0398	11.9827
256	306.330	218.441	25.2512	12.0408	12.0069
512	313.266	260.354	70.6978	11.9761	12.0142

Table 3: Test loss for different metrics-Wine dataset.

batch size \ learning rate	1e-5	1e-4	1e-3	0.01	0.1
8	0.672770	0.147368	0.0372037	0.015342	0.017562
16	0.931542	0.237220	0.054658	0.017872	0.015841
32	1.136913	0.380187	0.081803	0.023326	0.014850
64	1.265524	0.591355	0.126918	0.032981	0.014968
128	1.337070	0.850800	0.203308	0.048216	0.016784
256	1.374728	1.079436	0.327256	0.071605	0.021169

Table 4: Accuracy for different metrics-Wine dataset.

batch size \ learning rate	1e-5	1e-4	1e-3	0.01	0.1
8	0.854945	0.971428	1.0	1.0	1.0
16	0.742857	0.956043	1.0	1.0	1.0
32	0.628571	0.912087	1.0	1.0	1.0
64	0.657142	0.898901	0.971429	1.0	1.0
128	0.668132	0.813186	0.971429	1.0	1.0
256	0.668132	0.672527	0.927473	1.0	1.0

scale is large. Since we are asked to use a scale of 1, many of the entries in the augmented design matrix are zero, causing instability.

3.8 Comparison between analytical linear regression and linear regression with SGD

In our dataset, analytical linear regression and linear regression with SGD give similar performances. However, the runtime of analytical linear regression is significantly smaller than that of SGD linear regression, because of the fact that analytical linear regression calculates the best weights at once and there is no need to perform updates as in SGD. We could say that in our dataset, analytical linear regression works better. However, there are some drawbacks of analytical linear regression that are not revealed by our experiment. a) analytical linear regression cannot deal with the case where the the design matrix's product with its transpose, $X^T X$ is singular (we could use pseudo-inverse instead). b) taking the matrix's inverse is an unstable operation and is slow when the design matrix is huge, especially when its size is bigger than the memory.

3.9 Extra experiment: adding momentum to the GD implementation

We also tested the performance of our models when momentum is used in our GD and SGD. Due to the simplicity of our model and the convexity of the loss functions, adding momentum has basically no impact on the performance of our model, as shown in Fig. 18 (for dataset 1) and Fig. 19 (for dataset 2)

4 Discussion and Conclusion

4.1 Reflections, discussions, and conclusions

In this project, we implement two ML models on different datasets and explore the influence of different hyperparameters on the performance to select the optimal parameters for each dataset. Afterward, we further explore the influence of enriching the feature like adding Gaussian basis functions and different optimization methods for linear regression.
In the process, we find that larger hyperparameters could bring advantages such as higher stability, and higher accuracy as well as some drawbacks like more computational complexity and more memory. Through repeated experiments and evaluation of combinations of different hyperparameters, the optimal hyperparameter configuration can be found so that the model gets the best performance on a given task. In real applications, the best choice of hyperparameters usually depends on the specific data set and problem. Hence the researchers need to carry out more experiments and analyze the dataset carefully.
For linear regression, Gaussian basis function regression is a powerful nonlinear modeling technique, but care is needed to avoid overfitting and perform good hyperparameter tuning. Both analytical linear regression and SGD-based linear regression are good optimization methods. Which method to choose depends on the size of the data set, the number of features, the computational resources, and other factors.

4.2 Future investigation

There are potential areas for exploration in this project. Firstly, we could consider more parameters and methods for linear regression and softmax regression. Secondly, the interaction between different hyperparameters can also be studied further. For instance, when the batch size increases, the learning rate should increase or decrease to get a better performance. Thirdly, we can further explore how the datasets' characteristics and targets influence the choice of evaluation and optimization methods.

5 Statement of Contributions

This project is completed as a team of 3, and the work of this project is rather evenly distributed among the group members. Yijie Zhang implemented the models for linear regression and softmax regression, and Tian Bai carried out the tests on the two datasets. Yiping Liu is responsible for the report writing part. Three team members reviewed and finalized the report together.

References

[1] Maulud, Dastan, and Adnan M. Abdulazeez. "A review on linear regression comprehensive in machine learning." Journal of Applied Science and Technology Trends 1.4 (2020): 140-147.

[2] Das, Abhik. "softmax regression." Encyclopedia of Quality of Life and Well-Being Research. Cham: Springer International Publishing, 2021. 1-2.

[3] Claesen, Marc, and Bart De Moor. "Hyperparameter search in machine learning." arXiv preprint arXiv:1502.02127 (2015).

[4] S. Sun, Z. Cao, H. Zhu and J. Zhao, "A Survey of Optimization Methods From a Machine Learning Perspective," in IEEE Transactions on Cybernetics, vol. 50, no. 8, pp. 3668-3681, Aug. 2020, doi: 10.1109/TCYB.2019.2950779.

[5] Wagstaff, Kiri. "Machine learning that matters." arXiv preprint arXiv:1206.4656 (2012).

[6] Probst, Philipp, Anne-Laure Boulesteix, and Bernd Bischl. "Tunability: Importance of hyperparameters of machine learning algorithms." The Journal of Machine Learning Research 20.1 (2019): 1934-1965.

Appendix

```
Empty DataFrame
Columns: [CRIM, ZN, INDUS, CHAS, NOX, RM, AGE, DIS, RAD, TAX, PTRATIO, LSTAT, MEDV]
Index: []
Empty DataFrame
Columns: [CRIM, ZN, INDUS, CHAS, NOX, RM, AGE, DIS, RAD, TAX, PTRATIO, LSTAT, MEDV]
Index: []
```

Figure 1: Empty data frame of Boston Housing dataset.

```
Empty DataFrame
Columns: [Class, Alcohol, Malic acid, Ash, Alcalinity of ash, Magnesium, Total phenols, Flavanoids, Nonflavanoid phenols, Proanthocy
Index: []
Empty DataFrame
Columns: [Class, Alcohol, Malic acid, Ash, Alcalinity of ash, Magnesium, Total phenols, Flavanoids, Nonflavanoid phenols, Proanthocy
Index: []
```

Figure 2: Empty data frame of wine dataset.

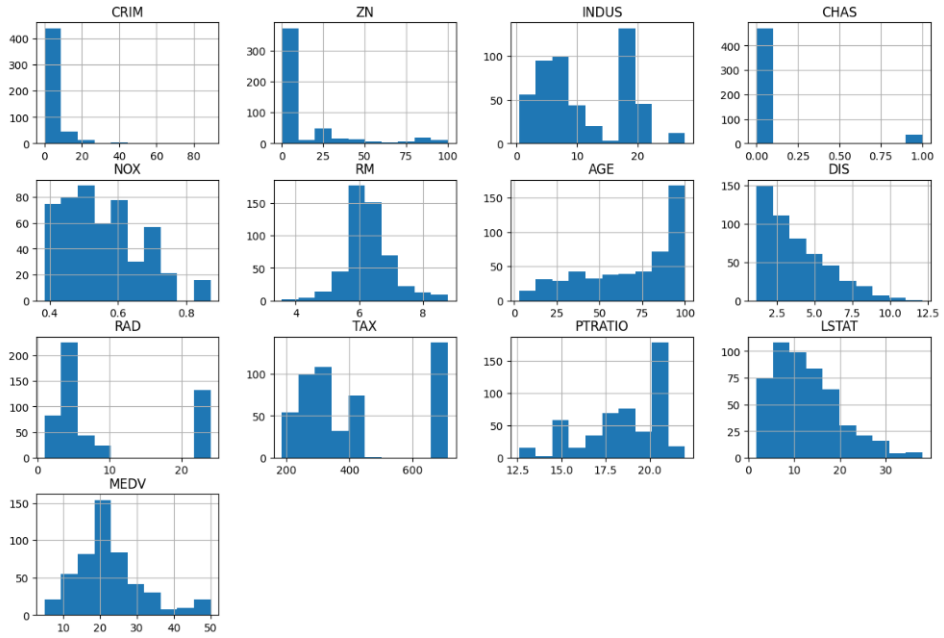

Figure 3: Histogram of housing features.

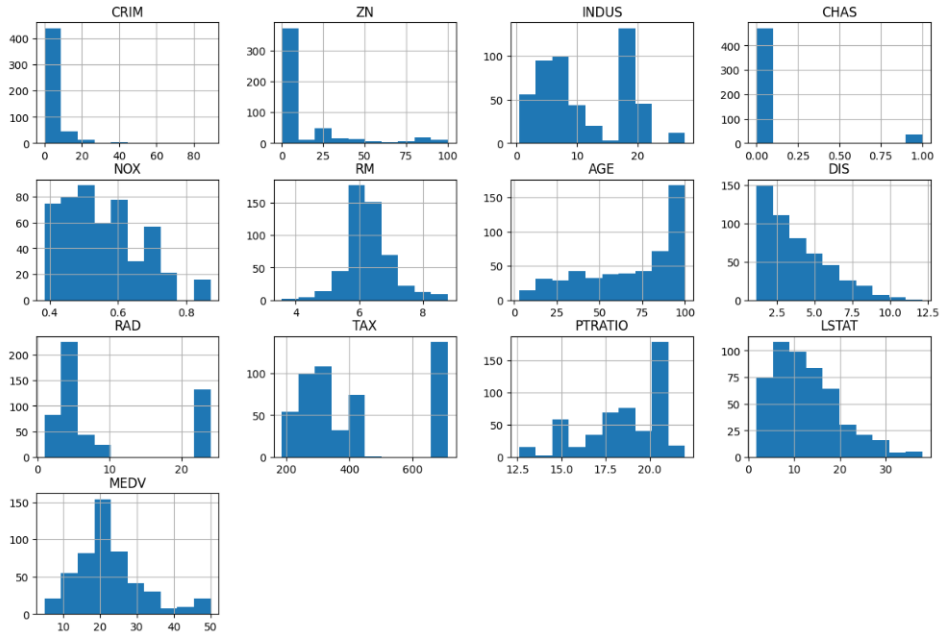

Figure 4: Histogram of wine features.

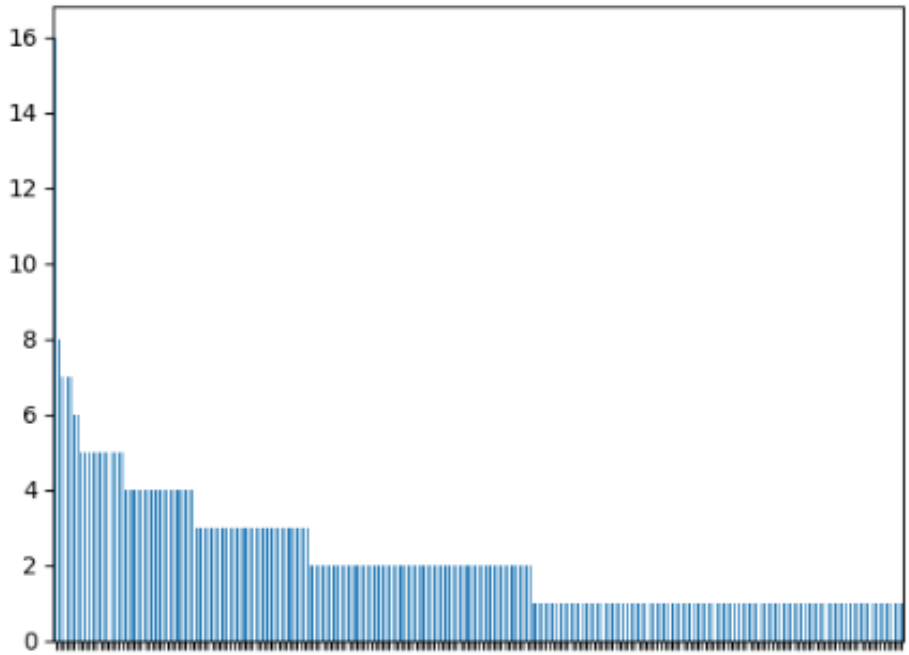

Figure 5: Distribution of MEDV.

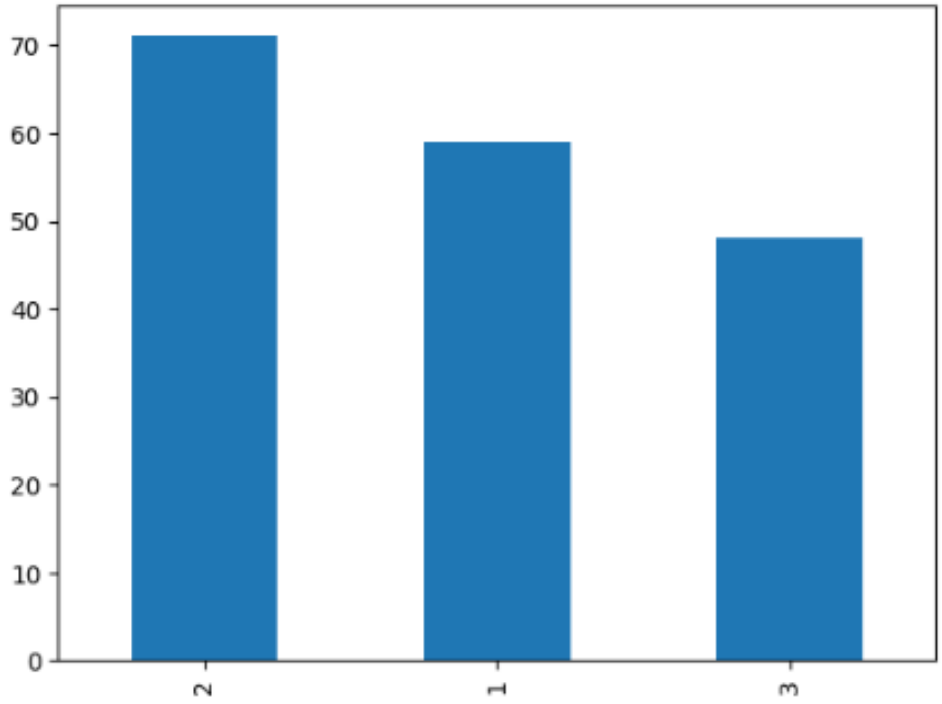

Figure 6: Distribution of classes.

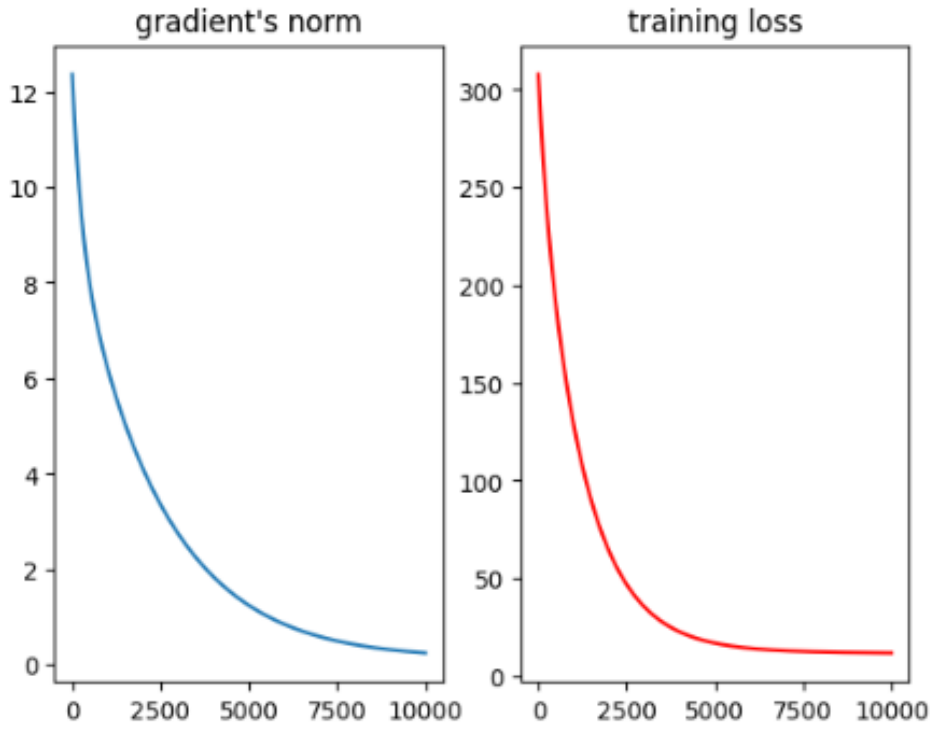

Figure 7: Performance of linear regression.

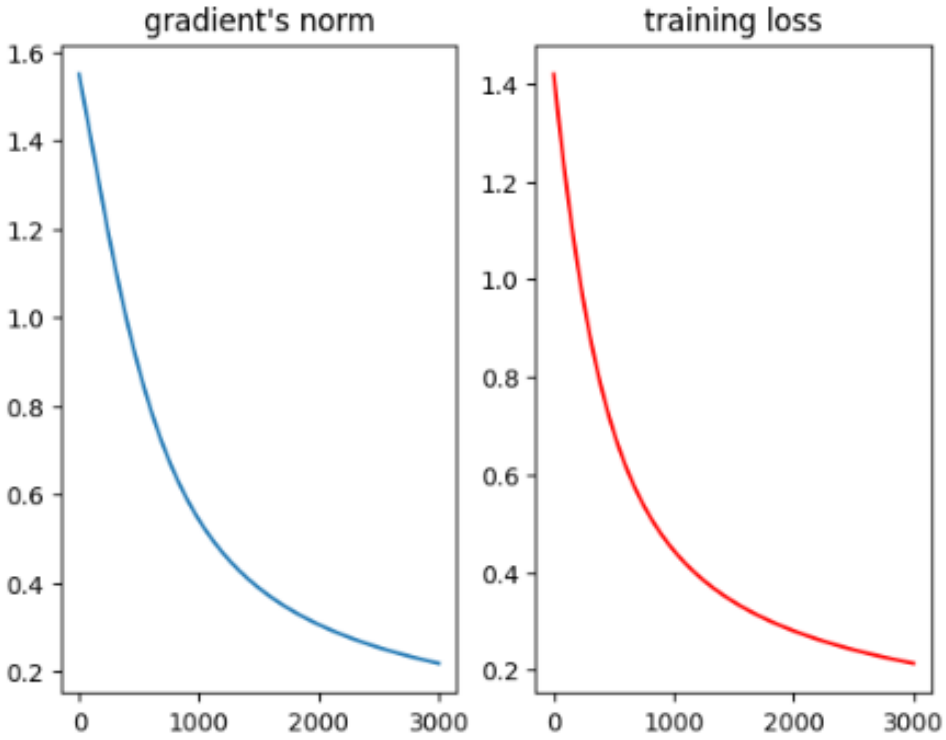

Figure 8: Performance of softmax regression.

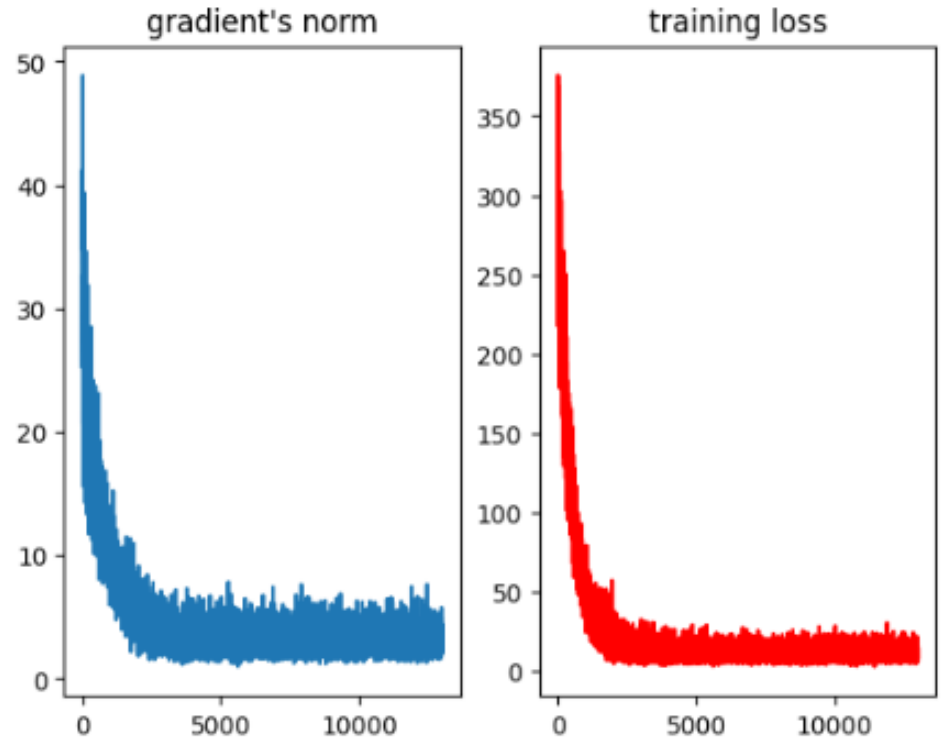

Figure 9: Performance of 5-fold cross-validation on Boston Housing dataset.

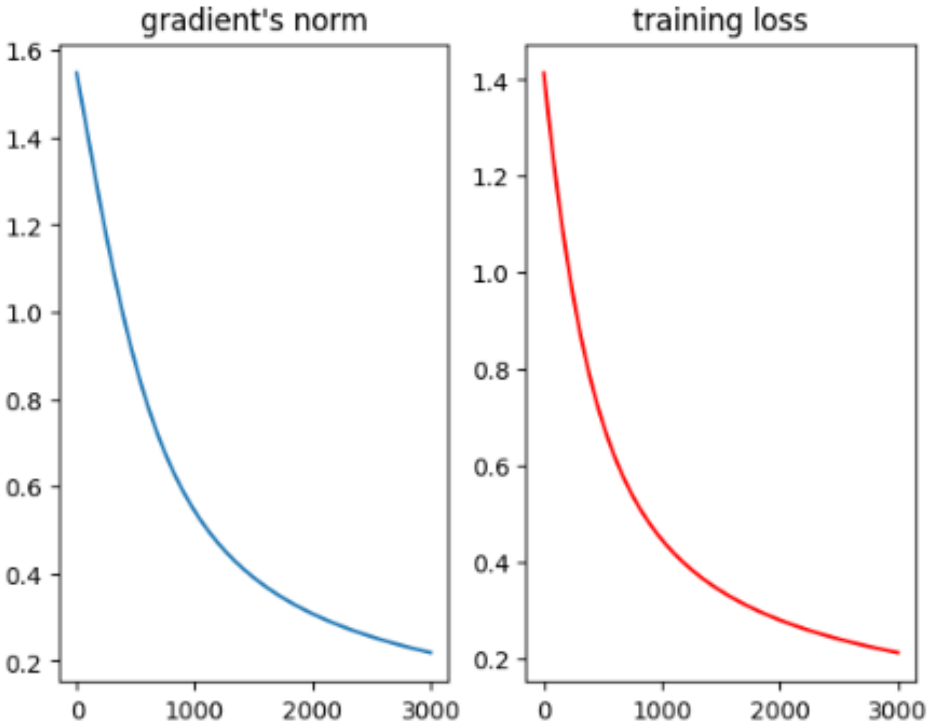

Figure 10: Performance of l5-fold cross-validation on wine dataset.

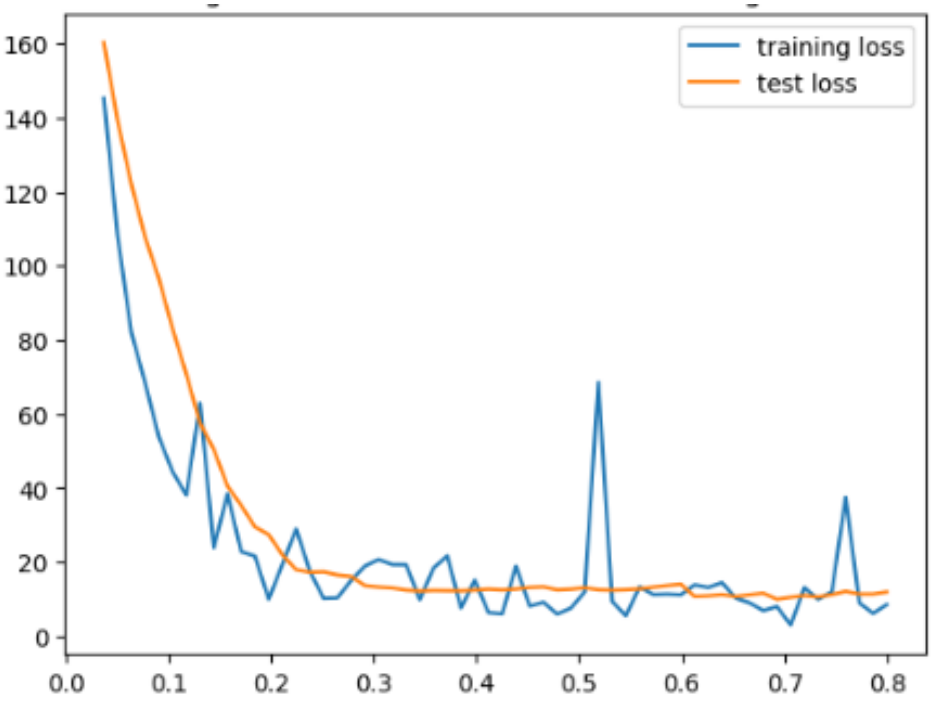

Figure 11: Training and test loss under different training set sizes for Boston Housing dataset.

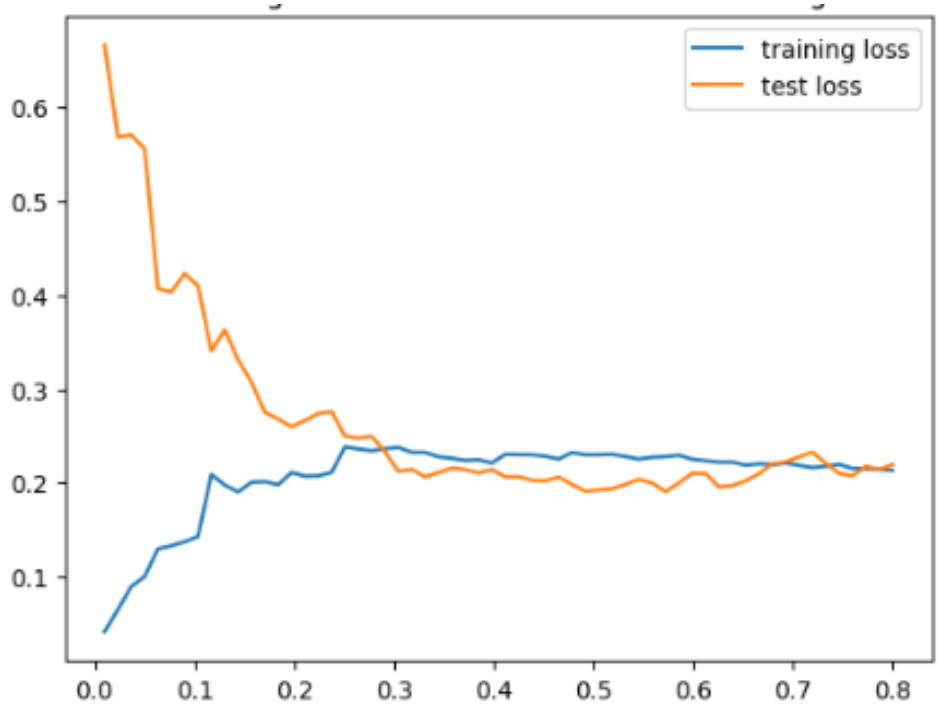

Figure 12: Training and test loss under different training set sizes for wine dataset.

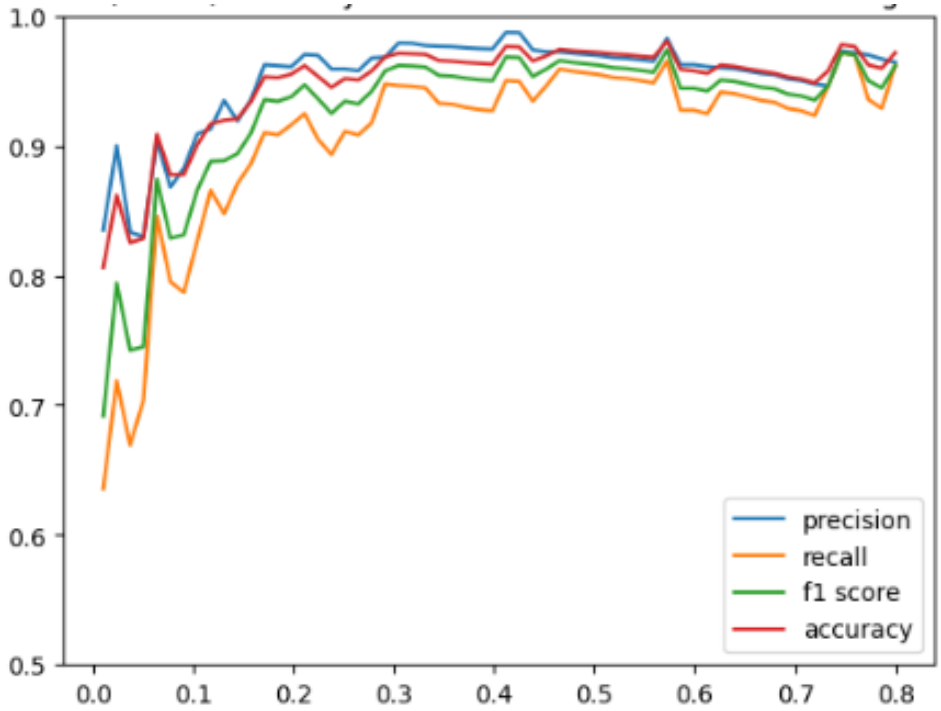

Figure 13: Precision, recall, accuracy and F1 score under different training set size (for wine dataset)

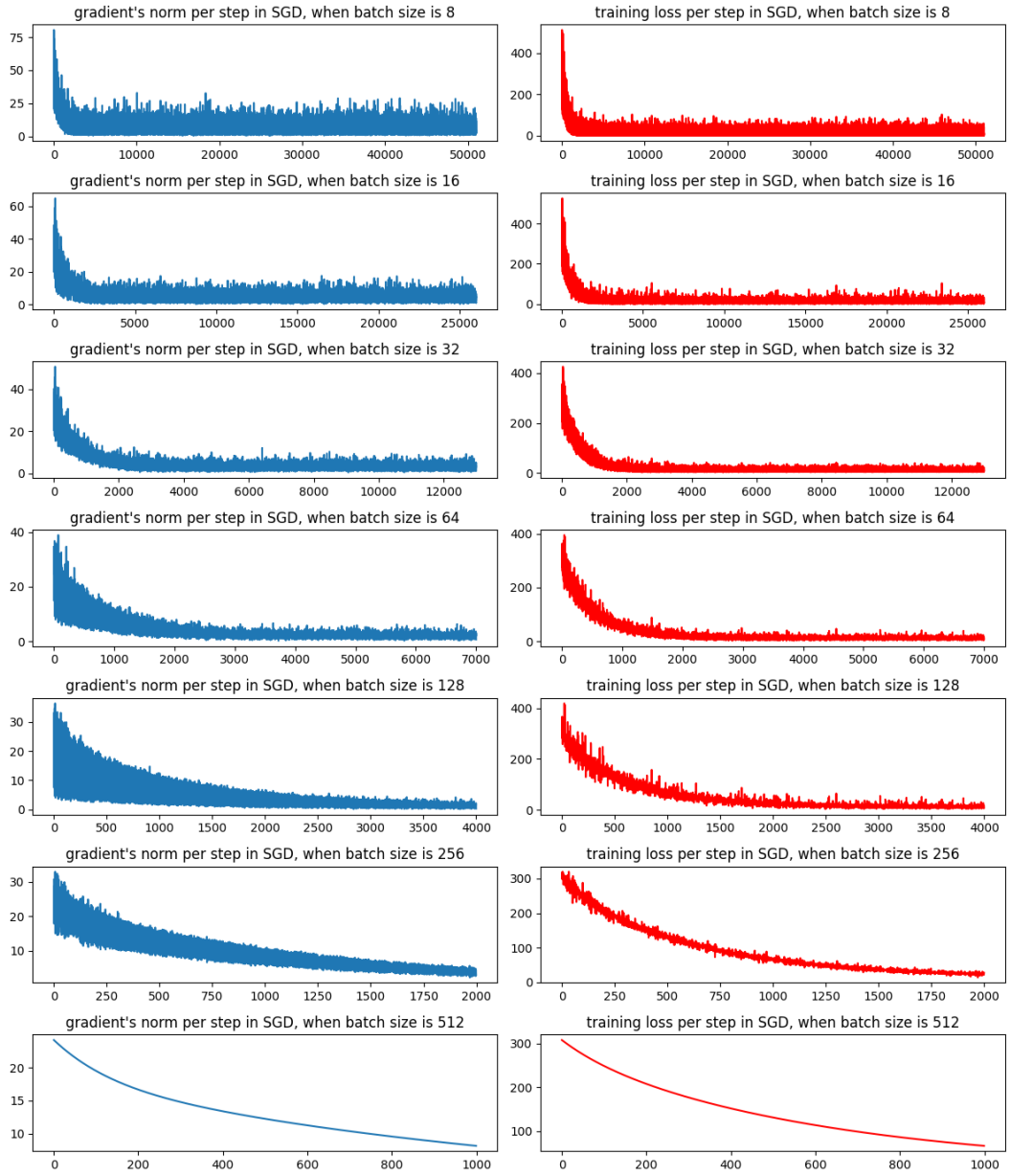

Figure 14: Gradient's norm per step in SGD under different batch sizes for Boston Housing dataset.

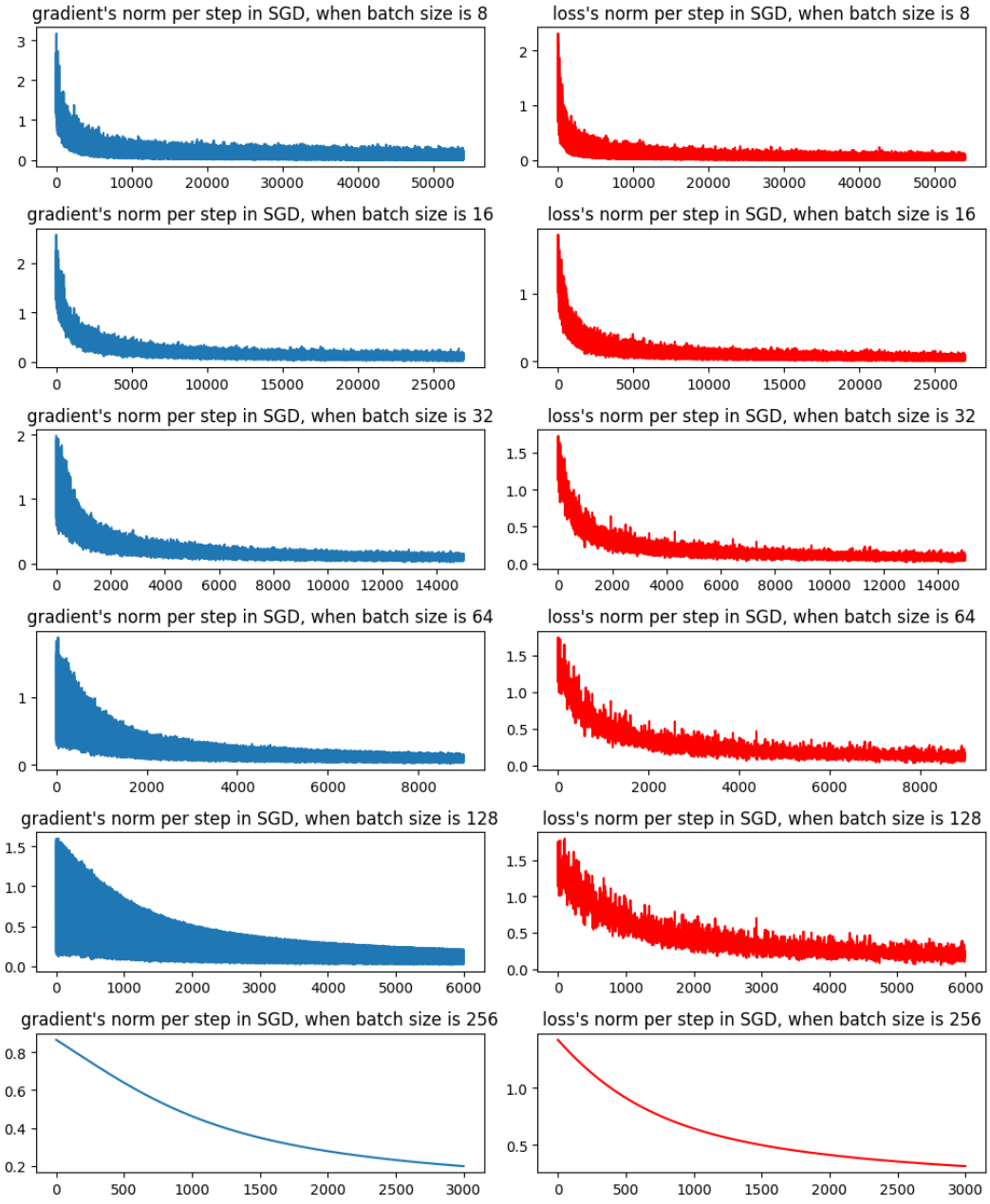

Figure 15: Gradient's norm per step in SGD under different batch sizes for wine dataset.

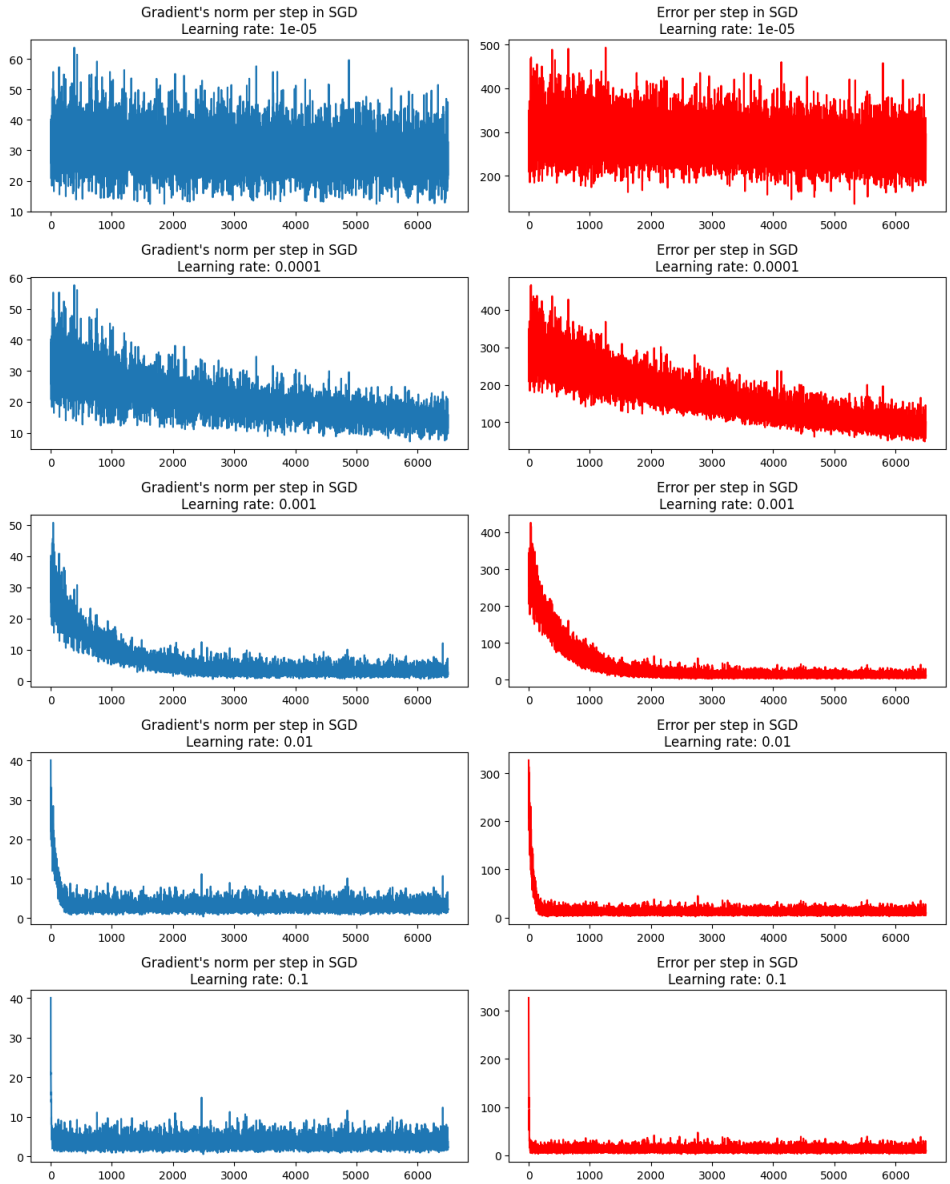

Figure 16: Test performance of Boston Housing dataset under different learning rates.

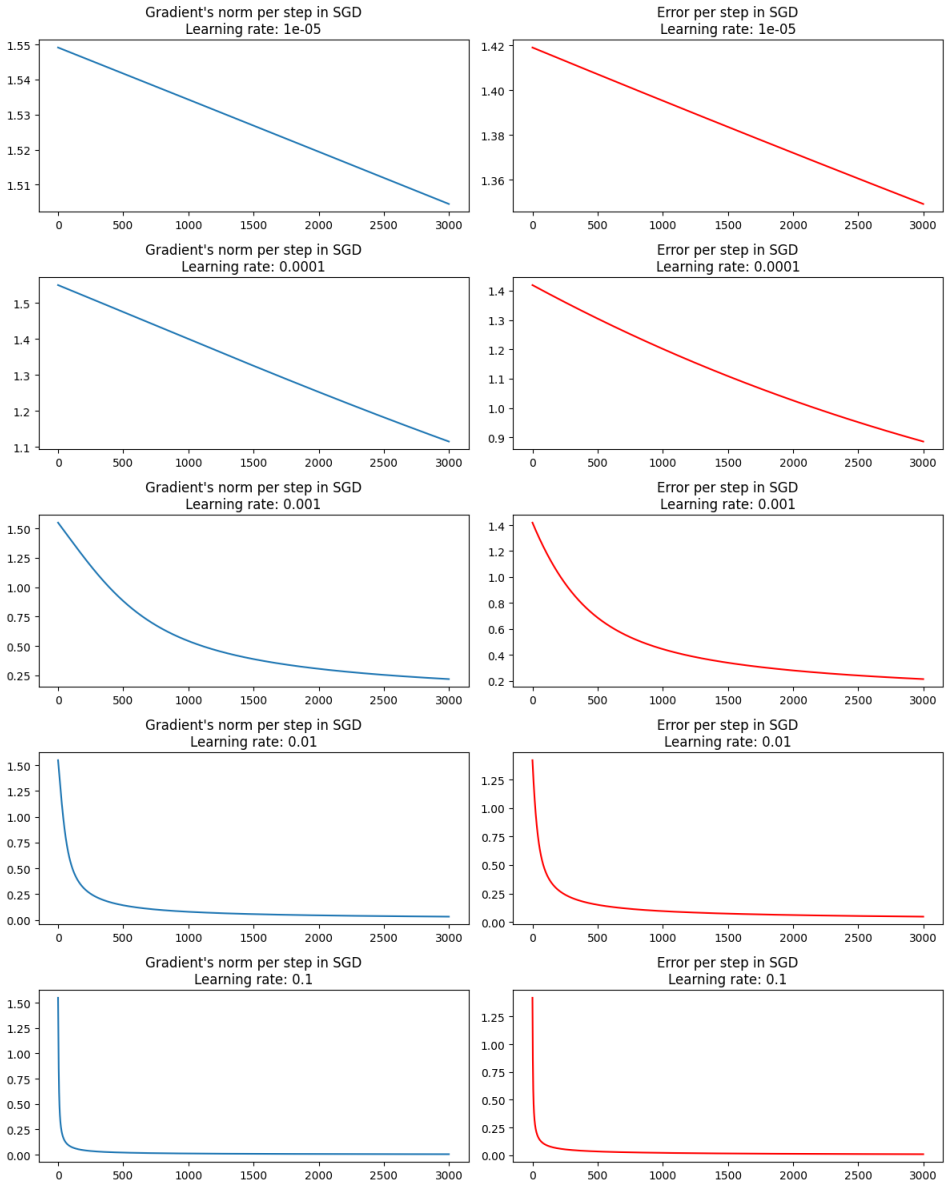

Figure 17: Test performance of wine dataset under different learning rates.

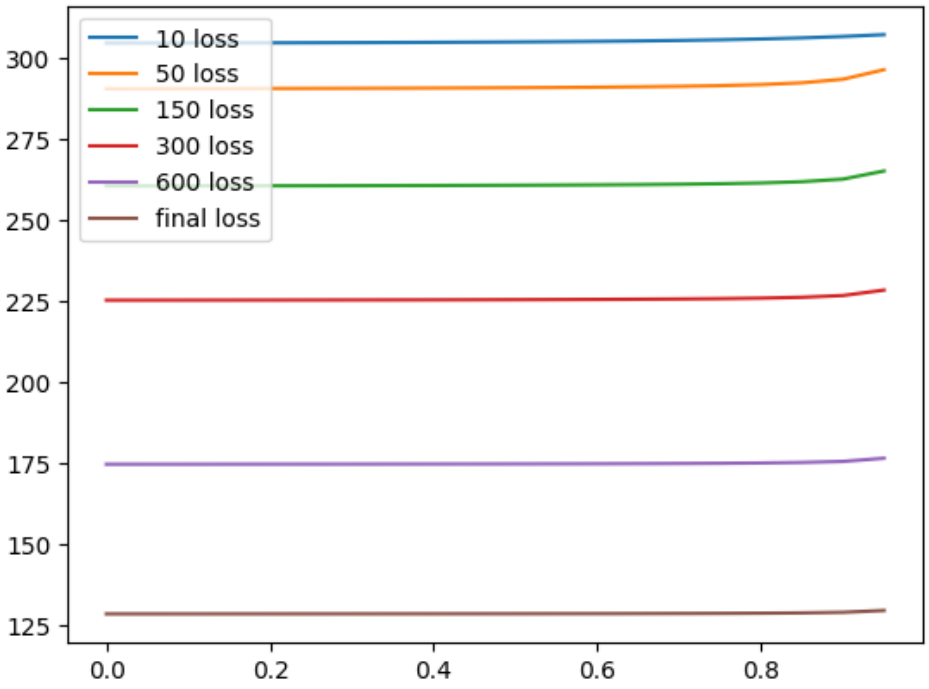

Figure 18: Boston Housing dataset: loss at different stages of GD/SGD when momemtum is added

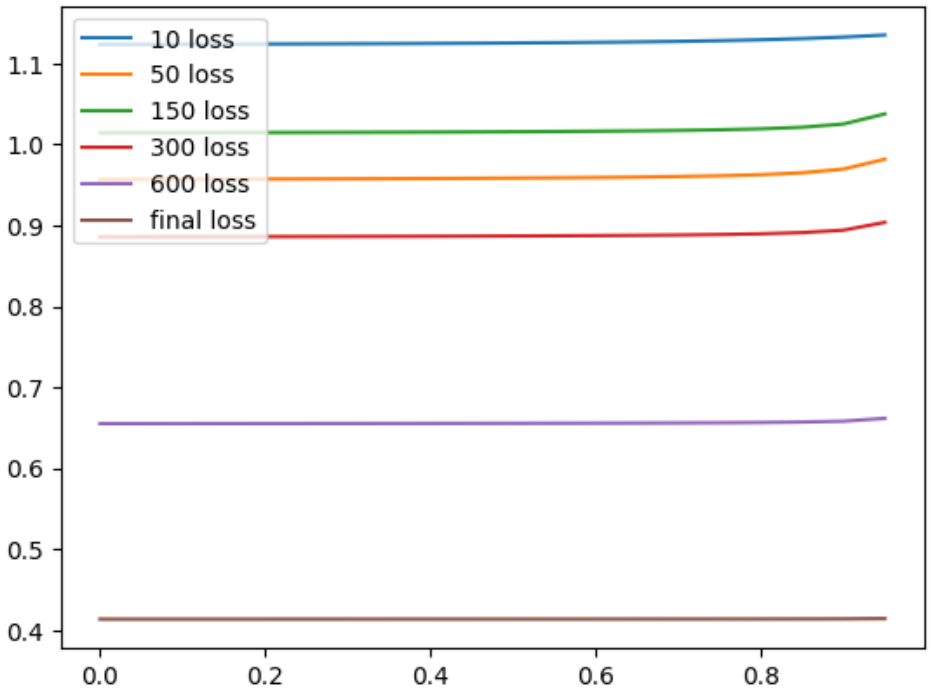

Figure 19: Wine dataset: loss at different stages of GD/SGD when momemtum is added