Mini-Project 1:

Getting Started With Machine Learning

Lily Gostovic
Mohammad Abdullah
Justin Novick

A report describing the findings of
Mini-Project 1 of COMP551

COMP551: Applied Machine Learning
McGill University
Montreal, Canada
October 7, 2023

1 Abstract

In this project, the two foundational models of Machine Learning: Linear Regression and Logistic Regression were implemented. Two types of linear regression (analytical and mini-batch stochastic gradient descent (SGD)) and two types of softmax logistic regression (gradient descent and mini-batch SGD) were implemented and compared in performance on the Boston Housing Dataset for linear regression models and the Wine Dataset for logistic regression models. The performance of linear regression models was measured using the mean squared error (MSE), while the performance of logistic regression models was measured using accuracy, precision, recall, and f1 scores. The performance of each model was analyzed across a variety of hyper-parameters such as the training dataset size, mini-batch size, and learning rate. The findings were then used to determine which models with which hyper-parameters perform best on each dataset. It was concluded that the Boston Housing Dataset is best modeled using mini-batch SGD linear regression with the training dataset being a random 60% of the original dataset, a batch size of 2, and a learning rate of 0.002. A lowest MSE of 18.5512 was achieved using these hyper-parameters. It was also concluded that the Wine Dataset is best modeled using mini-batch SGD logistic regression with a maximum average accuracy of 0.9861 for each class measured while the training dataset size was 60%, the batch size was 128, and the learning rate was 0.006. An implementation of the momentum optimizing technique was also added in the hyper-parameters Model to allow faster convergence.

2 Introduction

This project implements linear and logistic regression to model the Boston Housing Dataset and Wine dataset respectively. A variety of performance tests were run to identify trends in the models performances to best identify the optimal model for each dataset.

The goal while running linear regression models on the Boston Housing Dataset was to predict the median value of owner-occupied homes with the use of data regarding 13 different features. These features can be seen in more detail in the appendix under Boston Housing Dataset. The implementations below contain both an analytical solution and a mini-batch SGD solution. For continued reading on SGD, (**pmlr-v97-qian19b**) is a great resource. The analytical solution gives an exact solution and does not require the tuning of hyper-parameters during implementation. While analytical linear regression is an optimal solution for certain datasets, as discussed in section 4.8, we did not see it to be a good fit to model the Boston Housing Dataset. The mini-batch SGD linear regression model was deemed to be the optimal model of the Boston Housing dataset due to the dataset's large size and relatively low amounts of noise. Furthermore, we were able decrease the average time it takes for stochastic methods to converge, with the use of momentum.

The goal in modeling the Wine Dataset was to use the 13 parameters of the dataset to predict which of three cultivators in Italy samples were derived from. See specifities about the parameters in the appendix under Wine Dataset. Given the fact that these classifications have more than just a binary domain, we needed to implement softmax logistic regression to properly predict the wine classes. Similarly to the linear regression implementations, two logistic regression models were implemented: one using gradient descent, and one using mini-batch SGD. The implementation of logistic regression with gradient descent was for more consistent convergence at the expense of run time, while the implementation of mini-batch SGD was to provide better run time and the ability to fine-tune hyper-parameters to optimize the model for the wine dataset specifically.

The mini-batch SGD implementation of softmax regression proved to be better than the gradient descent implementation, even before tuning the hyper-parameters. The mini-batch SGD got near perfect scores for accuracy, precision, recall, and F1 Scores for all classes, while the gradient descent model consistently had near perfect accuracy, precision, recall, and F1 Scores when predicting class 1, however performed less favourably ranging from 88-97% for classes 2 and 3. Therefore, it was immediately clear that the mini-batch SGD model fit the data better than the gradient decent model. Additionally, once tuning the hyper-parameters of the mini-batch SGD model, it was evident that there were a variety of different combinations of hyper-parameters which resulted in near perfect accuracy, precision, recall, and F1 scores for all three classes.

To test the models and discover which model best predicted each dataset a variety of analyses were performed. The following analyses were done, where in each case the calculation of performance metrics were compared: for an 80/20 split of train/test data, 5-fold & 10-fold cross-validation, sampling growing training subsets, growing mini-batch sizes for SGD models, and adjusting the learning rate. Using the intuition gained from the aforementioned

analyses, 256 different combinations of training set size, batch size, and learning rate were tested to see which combination of the hyper-parameters best fit the models. The final analysis done was regarding the enrichment of the datasets using Gaussian features. Gaussian features were added to the datasets to investigate if the models could pick up underlying relations between the parameters to allow for non-linear relations and therefore more closely approximating the data.

3 Datasets

Two datasets were analyzed in this project. Both datasets were loaded into Google Collab then parsed to ensure standardized data. In both datasets, regular expressions were used to remove any non-numerical data to ensure computations can be done on all parameters without raising errors.
Once the data was loaded and cleaned, it was visualized to gain a better understanding of trends in the data using histograms of parameters for the boston dataset, box plots for the boston dataset (Fig. 2), correlation heatmap for boston (Fig. 3), histograms for the wine dataset (Fig. 4), and box plots of the wine dataset (Fig. 5) to better understand trends in the data.
The Boston Housing Dataset includes 13 parameters influencing the median value of owner-occupied house prices. For an in-depth explanation of the Boston Housing Dataset, continue reading (**article**). The goal of modeling the Boston Housing Dataset is to predict a house price given the above 13 parameters (including the target parameter). Price is a continuous variable, therefore linear regression will be used to model the Boston Housing Dataset.
An ethical concern arose when analyzing the Boston Housing Dataset. Specifically, with the column labeled `B` which is a measurement of African American population by town. We chose to remove this column from all modeling as this statistic is not a relevant factor when determining median house prices. This parameter should not be included inside the model, and if for whatever reason it needs to be included, then different types of demographics should be included as well. Otherwise a bias is being generated against that specific demographic. All other parameters were deemed to be ethical and were therefore used as input features for our model.
The Wine dataset includes 13 parameters which influence the wine class. The wine class is a separation of wines into classes 1, 2, and 3. The goal of modeling the Wine Dataset is to predict a wine's class, given the above set of parameters. The class can be either 1, 2, or 3, therefore the class is a discrete variable, therefore logistic regression will be used to model the Wine Dataset. No ethical concerns arose when analyzing the parameters of the Wine Dataset, therefore they were all used in modeling.

4 Results

A variety of different testing was done on the implemented linear and logistic regression models to examine how exactly they behave given different hyper-parameters and how the models compare in performance to one another. Prior to all analysis, the data was normalized using a normalization helper function and then split into training and testing sets using sci-kit learn's train_test_split function.

4.1 Baseline 80/20 test/train split

The first form of analysis was a simple 80/20 testing and training dataset split. Both datasets were split in two, one part taking 80% of the the data to form the training data, while the other took 20% and acted as the test set. All four models: AnalyticalLinearRegression, StochasticLinearRegression, LogisticRegression, and StochasticLogisticRegression were trained on the same training set and then were used to predict the y values of the test set, yh. The predicted values, yh, were then compared to the real y from the test set and the performances of the models were evaluated using the MSE for linear regression models and accuracy, precision, recall, and F1 scores for logistic regression models. The results found were as follows: MSE for Analytical and Mini-Batch SGD Linear Regression were 19.0956 and 18.9398 respectfully. The accuracy, precision, recall, and f1 scores for classes 1, 2, and 3 were the following: (1.0, 0.9722, 0.9722), (1.0, 1.0, 0.8888), (1.0, 0.9285, 1.0), and (1.0, 0.9629, 0.9411) for Logistic Regression and (1.0, 1.0, 1.0), (1.0, 1.0, 1.0), (1.0, 1.0, 1.0), (1.0, 1.0, 1.0) for Mini-Batch SGD Logistic Regression.
This analysis using a simple 80/20 split and all default hyper-parameter values gives a baseline of what kind of

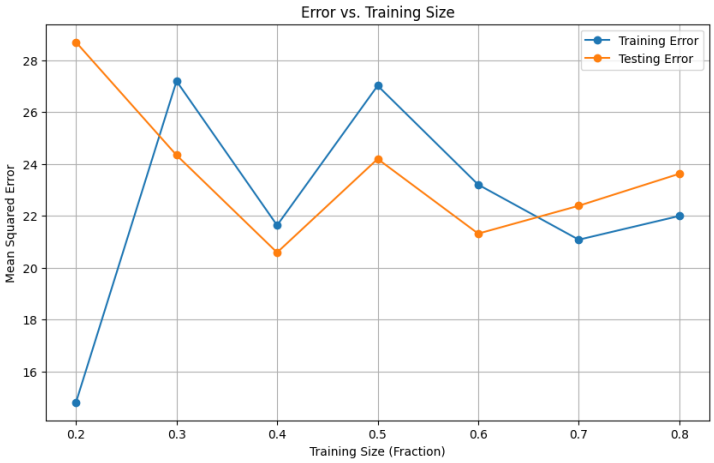
values to expect from each model. In the rest of the analyses, these values can be referred back to to ensure that the results are within a logical range.

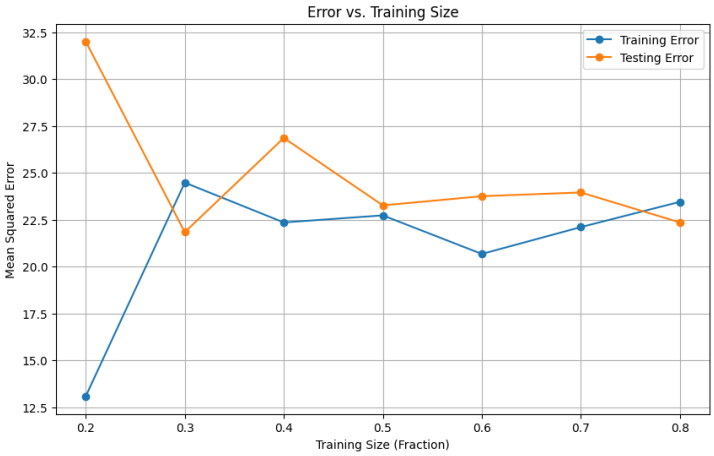
4.2 K-fold cross validation

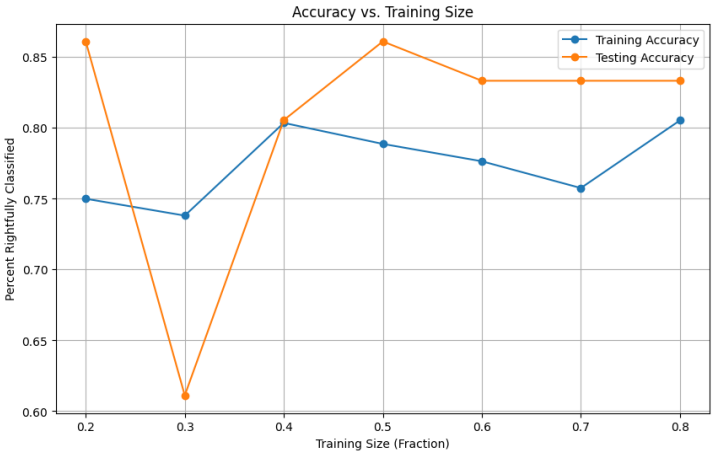
There is of course the possibility that on a regular 80/20 split, the models are trained and judged on an inconvenient or suboptimal slice of the original dataset. This problem is circumvented with the use of a K-fold cross-validation technique that iterates over that data K times, and at each iteration, a different training/validation split is used; the overall error, then becomes the average of all the errors found throughout the K tests. In this case, both a 5-fold and 10-fold technique were utilized; the 10 fold provided extra assurance, by mitigating the potential effects of noise. On the Boston Housing dataset, using analytical regression, 5-fold yielded an MSE = 36.327, while 10-fold yielded an MSE = 33.763. As for linear regression with hyper-parameters and a beta momentum parameter of 0.99, 5-fold MSE = 36.370, while 10-fold MSE = 34.120. In both cases, the 10-fold slightly outperformed the 5-fold, however, this result wasn't too significant. Both 5 and 10-fold techniques provided close results to the 80/20 split which implies the linear regression models handled noise well. On the wine dataset, for softmax regression (SGD), the 5-fold test revealed accuracy = 89.46%, while the 10-fold revealed accuracy = 96.25%. When mini-batch softmax regression (SGD) was used, accuracy was similar at 89.31% and 97.26% for the 5-fold and 10-fold tests respectively.

4.3 Sampling growing subsets of training data

The next test was to provide insight into how the models performed with different percentages of the dataset as training. Data was selected randomly that represented 20%, 30%, 40%, ..., and 80% of the entire dataset. Such subsets were then used to train the model and error statistics were calculated, as shown below (upper-left: analytical linear regression using MSE; upper-right: Linear Regression SGD using MSE; bottom-left: Softmax Regression using accuracy; bottom-right: Softmax Regression with mini-batch using accuracy).







For the Boston Housing dataset, MSE appeared to stabilize to lower values with trainings sets greater than 50% of the data, for both the analytical and SGD solutions; this stabilization was realized for both the testing and training sets, which implies over fitting is not a large concern. For Softmax Regression on the wine dataset, accuracy appeared to improve for both the mini-batch and regular SGD solutions, which implies the models got better, on average, with more training data. With the normal SGD Softmax Solution, there may be potential over fitting problems when using 50% to 60% as training data, as the training and testing accuracies differ, however, this levels out with training sizes greater than 60% of the dataset.

4.4 Growing mini-batch sizes for SGD models

This section of analysis explores how changing the mini-batch size effects a SGD model's performance.
When modeling the Boston Housing Dataset using the SGD linear regression model, the best batch size which optimizes both run time and MSE is 16. The batch size of 16 results in the lowest run time and MSE the most consistently in comparison to the other batch sizes tested, as seen in as seen in Figure 7 and as seen in Figure 8. The average run time for fitting a model with a batch size of 16 is 6ms, while the MSE 21. Additionally, it should be noted that the convergence graph for a batch size of 16 is the smallest batch size which has a mostly smooth curve. All batch sizes tested that are smaller than 16 had rough curves, while batch size 16 is the first convergence graph seen with a smooth curve. This can be seen in Figure 6 for the Linear Regression and Figure 8.5 for the Logistic Regression.
When modeling the Wine Dataset using the SGD logistic regression model, the best batch size found was 32. This batch size optimized both run time and the performance metrics as can be seen in Figure 9 and Figure 10.
For both datasets, it was observed that using batch sizes that are divisors of the total number of data points led to smoother convergences.

4.5 Learning rate adjustments

This section of analysis explores how adjusting the learning rate effects a SGD model's performance.
When modeling the Boston Housing Dataset, it was observed that a learning rate in the range 0.03-0.05 was optimal as can be seen in Figure 11 and Figure 12. This is the range where both the fitting run time and the MSE trend the lowest.
When modeling the Wine Dataset, it was observed that a learning rate in the range 0.04-0.45 was optimal with respect to fitting time as can be seen in Figure 13. The learning rate was observed to have no effect on the performance metrics as can be seen in Figure 14.
Therefore, the optimal learning rates for the Boston Housing Dataset and the Wine Dataset respectively were 0.03-0.05, and 0.04-0.45.

4.6 Finding the optimal hyper-parameters

When searching for the optimal hyper-parameters for both models it was important to take into account the findings from sections 4.3, 4.4, and 4.5. In these sections, we determined which training set sizes, batch sizes, and learning rates resulted in the best performing models. In this part of the analysis, we combined these three ranges of values for each model and tested each combination of each hyper-parameter to find the most optimal combinations for each model.
The MSE was used to evaluate the performance of the SGD Linear Regression model on the Boston Housing Dataset. MSE was used because it is always possible to derive the MSE for every different parameter configuration so the performance of every different configuration can be directly compared. The lowest MSE was achieved with a training set size of 60%, a batch size of 2, and a learning rate of 0.002, therefore this was deemed to be the most optimal configuration of hyper-parameters to model the Boston Housing Dataset.
Accuracy was used to evaluate the performance of the Mini-Batch SGD Logistic Regression model on the Wine Dataset. Accuracy was used because it can always be derived for every different parameter configuration so it is possible to directly compare the performance of every different configuration. Accuracy is a good overall impression of how well the model is fitting the data because it takes into account the true positives and true negatives, as opposed to other performance metrics such as precision, recall and f1 score which only consider one or the other. Additionally, it is possible to compute an overall accuracy for a specific set of hyper-parameters on

a model whereas precision, recall, and f1 scores cannot be generalized to an overall value and must be computed for each class. Having a singular number makes it possible to maximize and therefore find the most optimal set of hyper-parameters. The accuracy was maximized at a value of 0.9861 when the training set size was 60%, the batch size was 128, and the learning rate was 0.006.

4.7 Adding Gaussian basis functions

When performing linear regression, it must be true that each feature vector of the dataset displays normality. In the case that the assumption of normality is not met, it is common to apply Gaussian basis functions on irregular features to help improve training and the predictions that follow. Due to concerns the Boston Housing dataset violated the normality assumption, 5 random columns were selected to apply the Gaussian basis function (defined in the Google Colab). After such applications, the new 506x5 matrix alone, tested poorly with analytical linear regression; MSE was around 25% more than the magnitude of the original dataset. However, once these extra five features replaced the features they were derived from in the original dataset, the analytical regression MSE was typically driven down slightly, compared to the original dataset without the features.

4.8 Comparing Analytical and SGD Linear Regression

Two linear regression models, Analytical and SGD, were tested on the Boston Housing Dataset. Initially, Analytical Linear Regression had a lower MSE, but after optimizing SGD Linear Regression's hyper-parameters, it outperformed Analytical Linear Regression. It is important to note that the choice between the two models depends on factors such as dataset size, data collection method, and noise level. Analytical Linear Regression is faster for small datasets, while SGD Linear Regression is better for growing datasets and noisy data due to its adaptability. To read a more detailed persepctive on this topic, please refer to Comparing Linear Regression Models in the appendix.

4.9 Experimenting with momentum

The run time of mini-batch stochastic gradient descent for linear regression was drastically improved with the implementation of momentum. Momentum leverages a moving average of past gradients weighted by a factor β. Through a series of experiments, it became evident that $\beta = .99$ was most optimal amongst typical β values between 0.9 and 0.99. This value, giving significant weight to past gradients, allows for smoother updates in the face of noisy newer gradients from irregular regions of the data. This was done while also allowing for quicker convergence compared to SGD with the absence of momentum.

5 Discussion and Conclusion

In this study, Linear and Logistic Regression models were optimized using various hyper-parameters on the Boston Housing and Wine datasets, respectively. Mini-batch SGD consistently outperformed other implementations, showcasing its adaptability in both linear and logistic regression tasks. Specifically, the Boston Housing Dataset favored mini-batch SGD linear regression with hyper-parameters achieving the lowest MSE of 17.2029. Similarly, the Wine Dataset was best represented by mini-batch SGD logistic regression, reaching an impressive accuracy of 0.9861. The findings suggest the versatility of the SGD approach, and future research could further explore momentum optimization and integration of Gaussian basis functions to enhance performance.

6 Statement of Contributions

Abdullah focused on the implementations from part 2, while Novick and Gostovic split part 3 and the write up.

Appendix

Boston Housing Dataset Parameters

1. `CRIM`: Per capita crime rate by town.
2. `ZN`: Proportion of residential land zoned for lots over 25,000 sq.ft.
3. `INDUS`: Proportion of non-retail business acres per town.
4. `CHAS`: Charles River dummy variable (1 if tract bounds river; 0 otherwise).
5. `NOX`: Nitrogen oxides concentration (parts per 10 million).
6. `RM`: Average number of rooms per dwelling.
7. `AGE`: Proportion of owner-occupied units built prior to 1940.
8. `DIS`: Weighted distances to five Boston employment centers.
9. `RAD`: Index of accessibility to radial highways.
10. `TAX`: Full-value property tax rate per $10,000.
11. `PTRATIO`: Pupil-teacher ratio by town.
12. `B`: $(1000(Bk - 0.63)^2)$ Where Bk is the number of people of African descent by town.
13. `LSTAT`: Percentage of lower status of the population.

Wine Dataset Parameters

1. Alcohol: The alcohol content of the wine, usually represented as a percentage.
2. `Malicacid`: The amount of malic acid in the wine, which can influence its taste and acidity.
3. `Ash`: The amount of ash in the wine, which is a measure of the mineral content.
4. `Alkalinity of ash`: The alkalinity of the ash, which can affect the pH level of the wine.
5. `Magnesium`: The concentration of magnesium in the wine, which is one of the minerals found in wine.
6. `Total phenols`: The total phenolic content, which includes compounds like tannins and other polyphenols that contribute to the wine's flavor and color.
7. `Flavanoids`: The concentration of flavonoids, which are a type of phenolic compound found in wine and can contribute to its antioxidant properties.
8. `Nonflavenoid phenols`: The concentration of non-flavanoid phenolic compounds.
9. `Proanthocyanins`: The amount of proanthocyanins in the wine, which are a subclass of flavonoids.
10. `Color intensity`: A measure of the color intensity of the wine.
11. `Hue`: The hue or color of the wine.
12. `OD280/OD315 of diluted wines`: The ratio of optical density at 280nm and 315nm of the wine, which can provide information about the wine's color and concentration.
13. `Proline`: The concentration of proline, an amino acid found in wine.

Comparing Linear Regression Models

Two linear regression models were implemented to model the Boston Housing Dataset: Analytical Linear Regression and Linear Regression using Stochastic Gradient Descent.

Upon first trials of comparing the mean squared error of both models, it appeared that the analytical solution was more optimal and would return a lower test error. However, once the hyper-paramters of the model using Stochastic Gradient Descent were optimized, the Analytical solution quickly became less optimal than the perfectly tuned model using Stochastic Gradient Descent.

Therefore, to represent the Boston Housing Dataset, Linear Regression using Stochastic Gradient Descent gave better results than Analytical Linear Regression. However, this does not necessarily mean that Stochastic Gradient Descent is always the better choice than Analytical Linear Regression, there are many factors that can lead to one model outperforming the other.

First, the size of the dataset can greatly influence the results of both methods. With small datasets an analytical solution is very quick to compute and can save a lot of computational power compared to stochastic gradient descent methods. However, with a growing dataset the analytical solution becomes very computationally expensive since it involves taking the inverse of a matrix. This runs in $O(n^3)$ where n is the number of data points in the dataset.

The way in which the data is collected is also an important factor to consider. For example, if the dataset being worked with is continuously having data added to it then a linear regression model using stochastic gradient descent is more optimal. This is due to the fact that you can continue to train the existing model as more data is collected whereas with an analytical solution all computations would have to be redone with each data point added.

The amount of noise in the data is also an important factor. If a dataset has lots of noise then stochastic gradient descent may take a lot of time to converge because each step is likely to be quite off from the optimal direction. Therefore for a very noisy dataset, a linear regression model using stochastic gradient descent is better to use.

Figure 1: Histograms of Boston Housing Dataset parameters

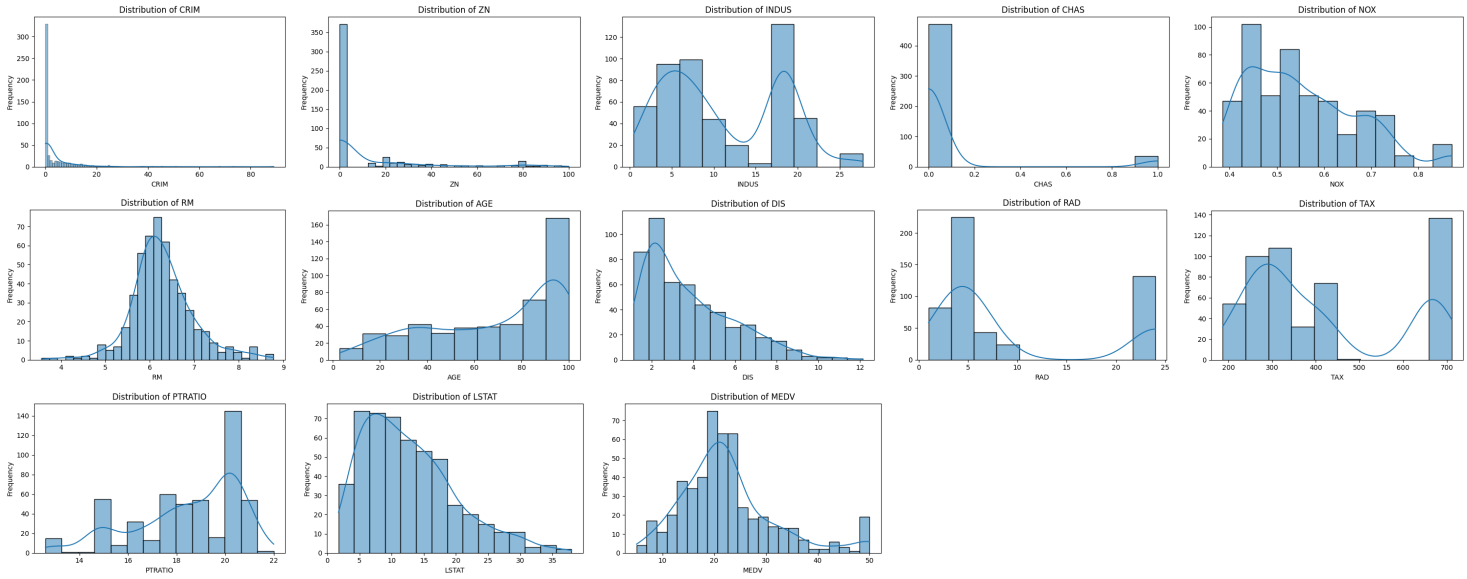


Figure 2: Boxplots of Boston Housing Dataset parameters

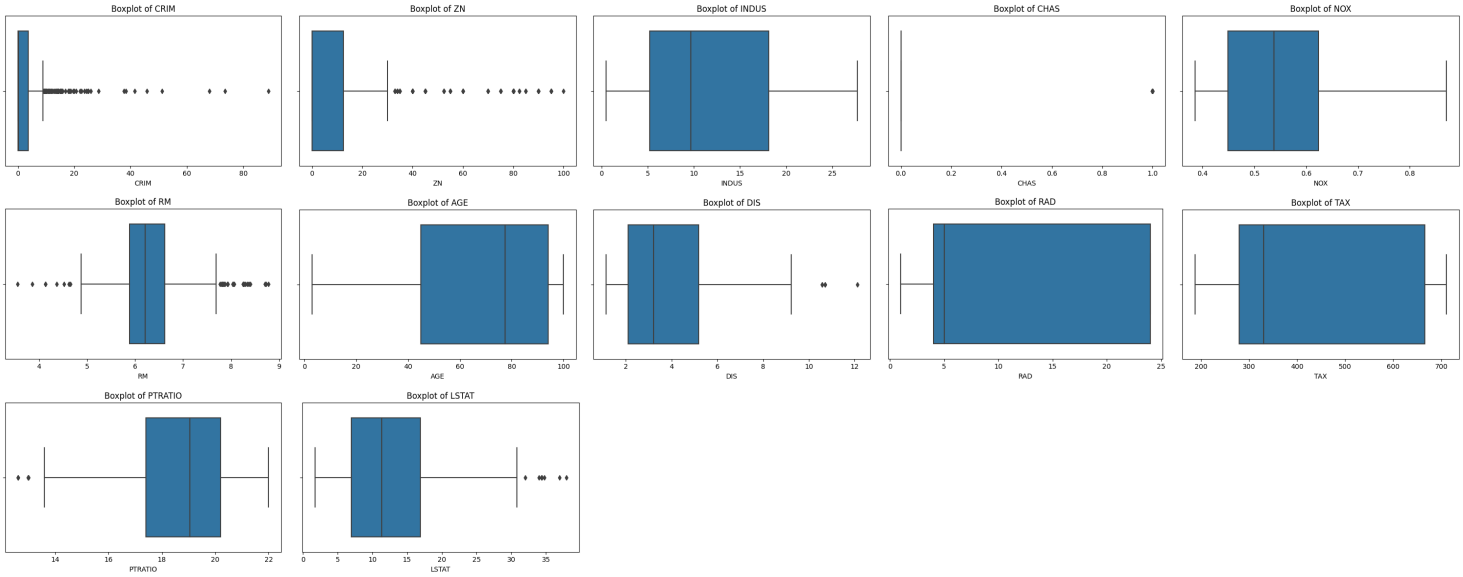


Figure 3: Correlation heatmap for Boston Housing Dataset

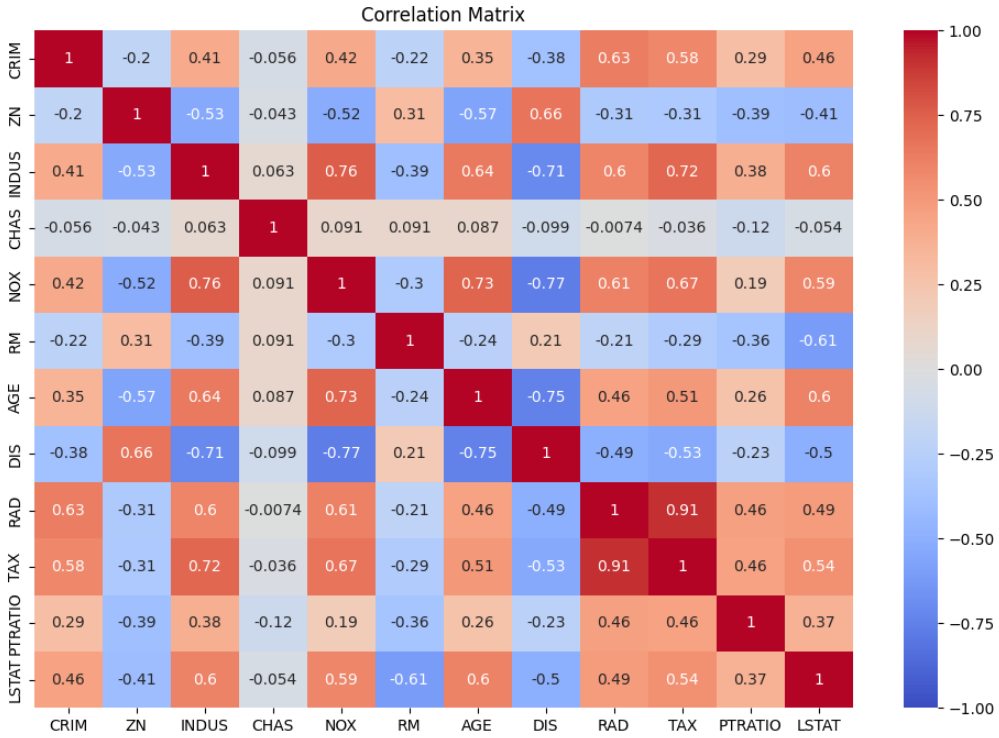


Figure 4: Histograms of Wine Dataset parameters

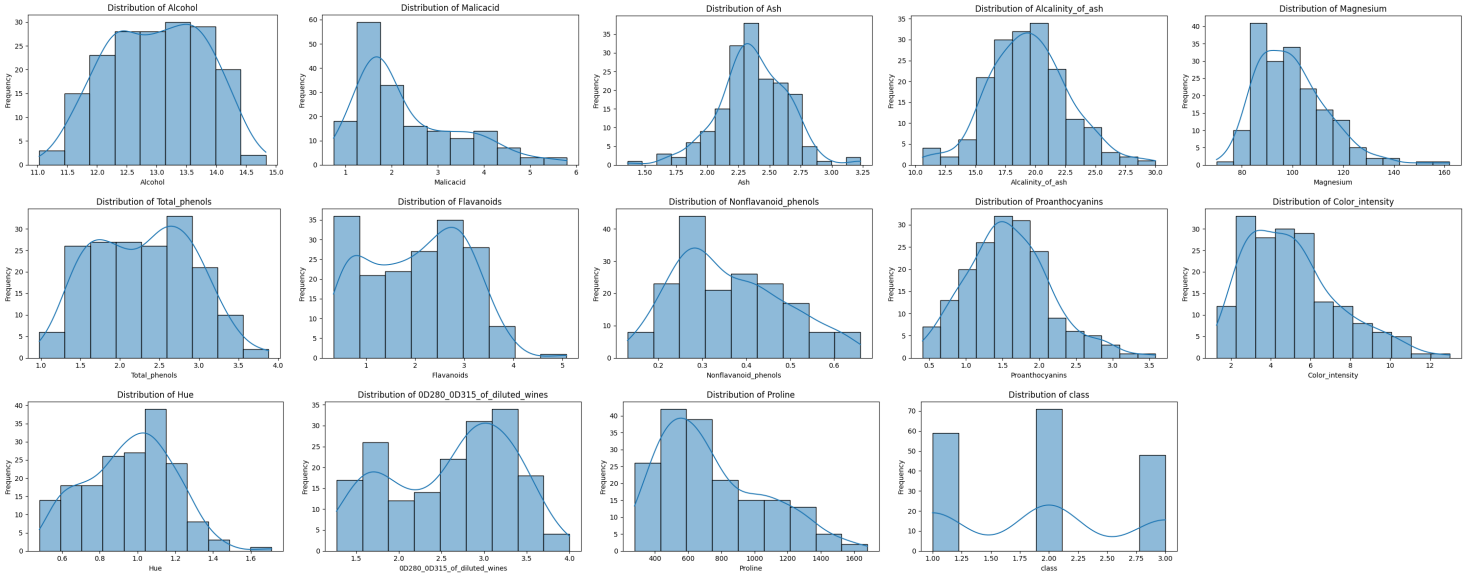


Figure 5: Box plots for parameters of Wine Dataset

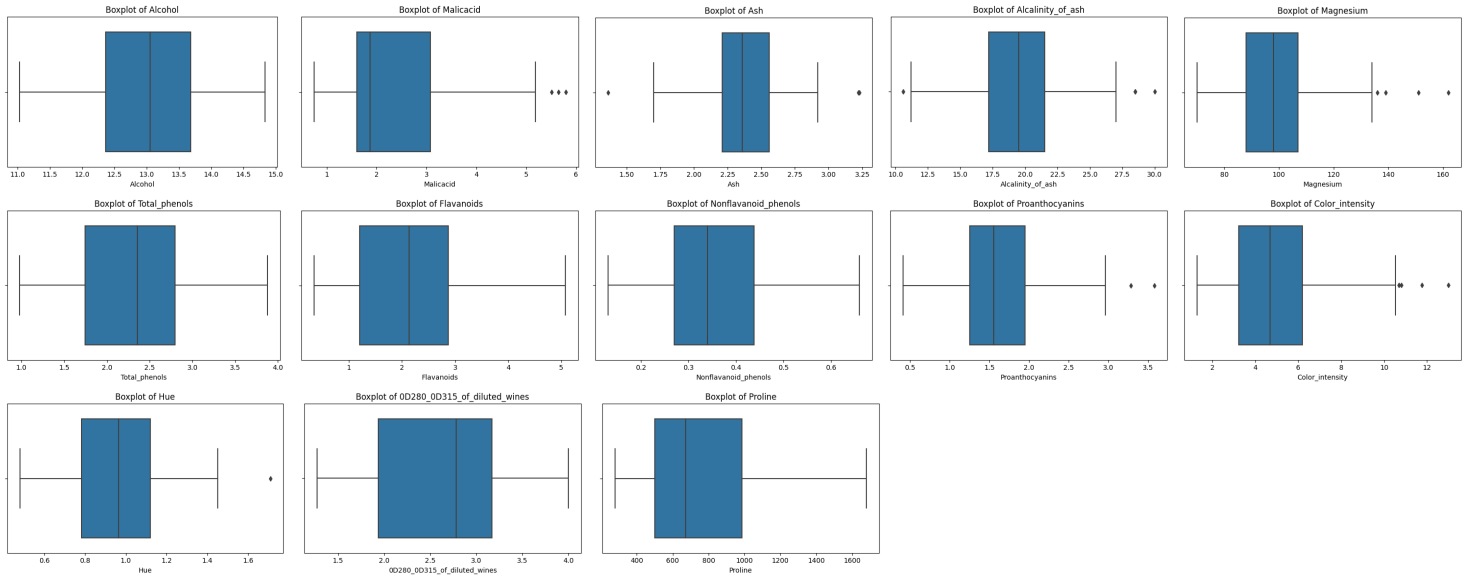


Figure 6: Boston Housing Dataset convergence rates with varying batch sizes

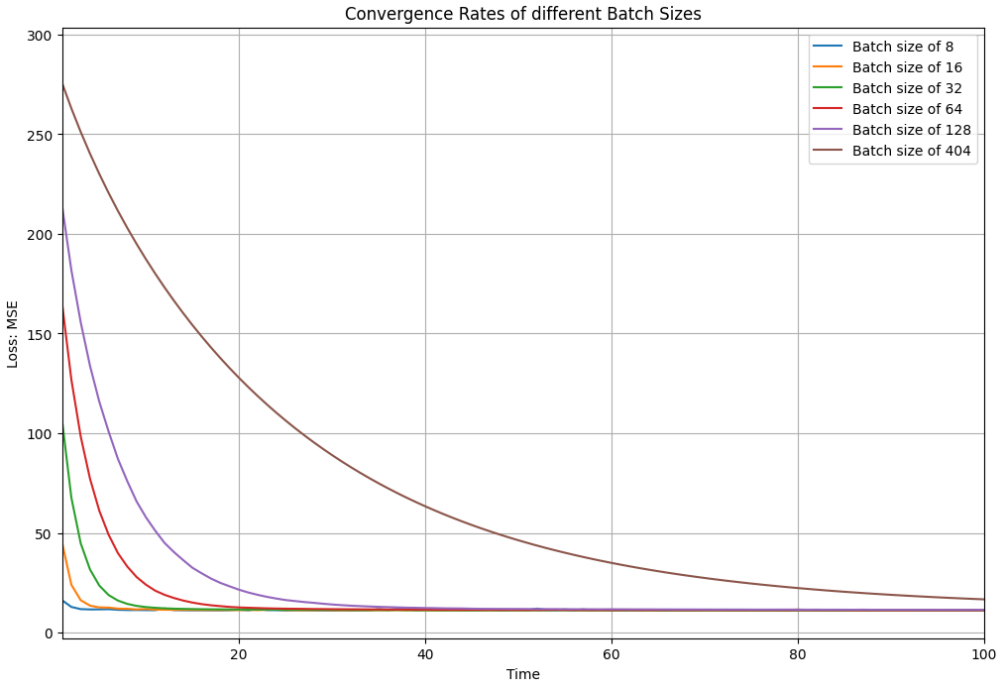


Figure 7: Boston Housing Dataset batch sizes vs fitting run time

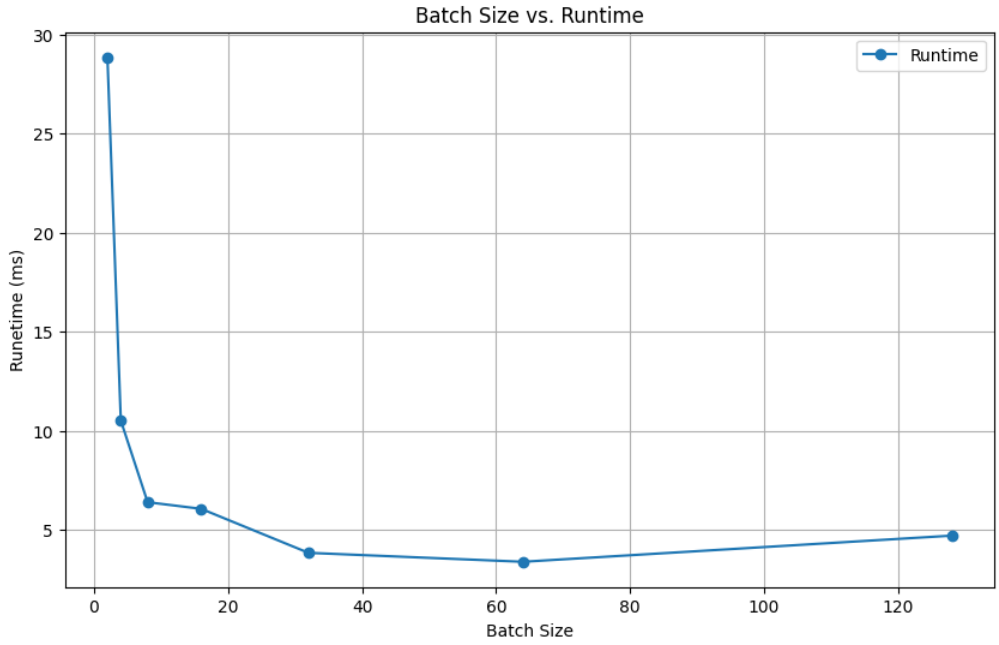


Figure 8: Boston Housing Dataset batch sizes vs MSEs

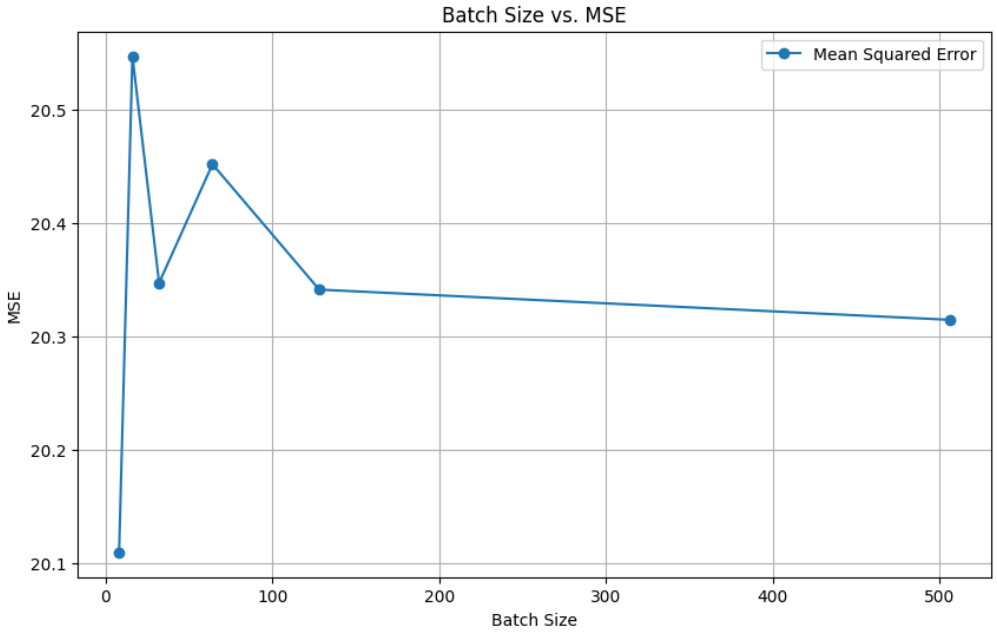


Figure 8.5: Wine Dataset convergence rates of different batch sizes

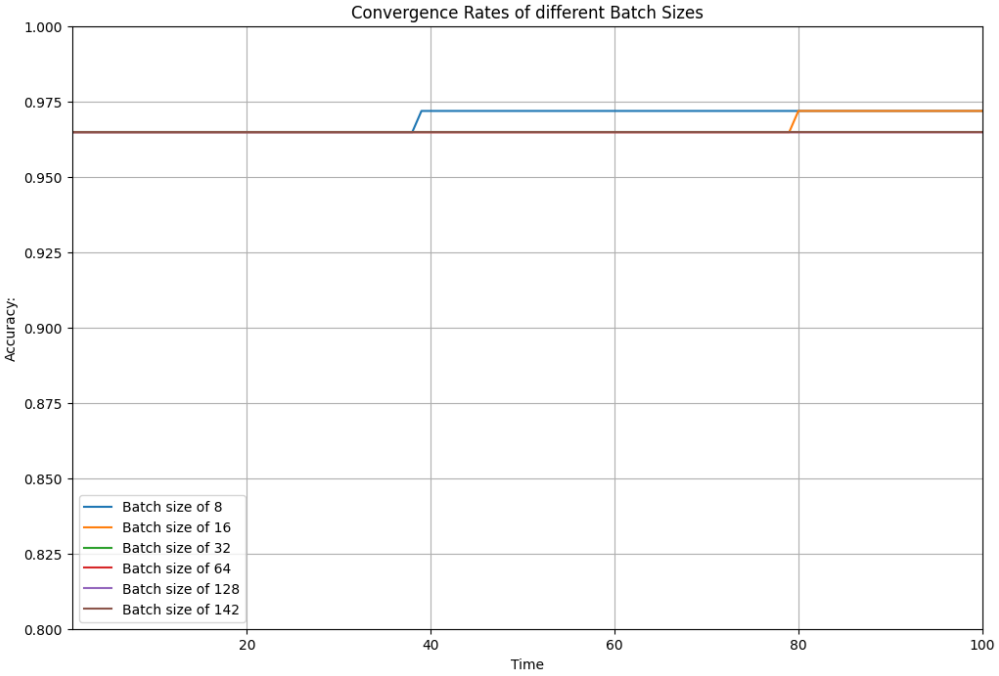


Figure 9: Wine Dataset batch sizes vs fitting run time

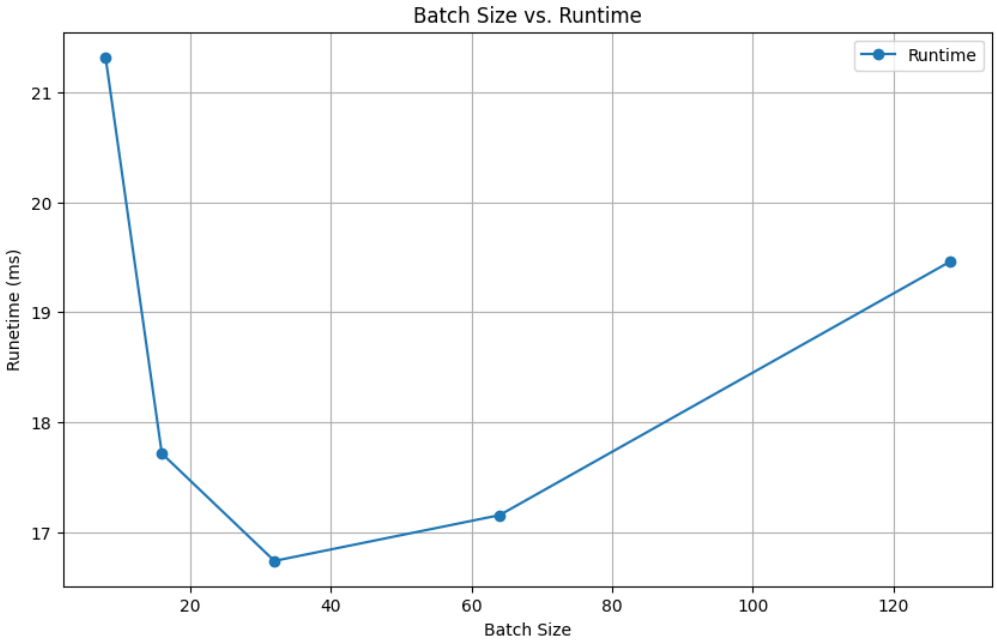


Figure 10: Wine Dataset batch sizes vs performance metrics

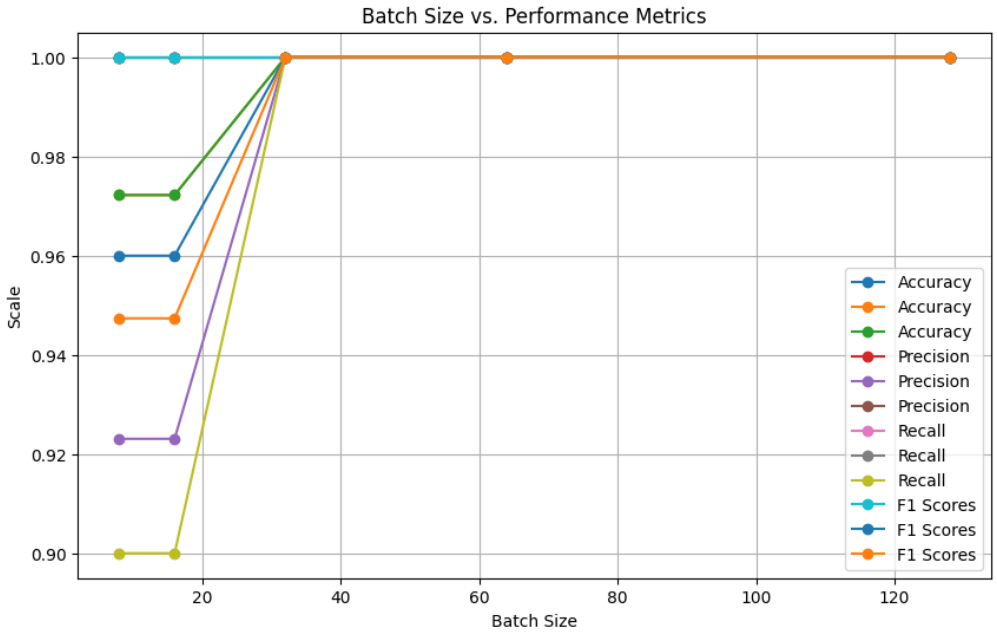


Figure 11: Boston Housing Dataset learning rate vs fitting time

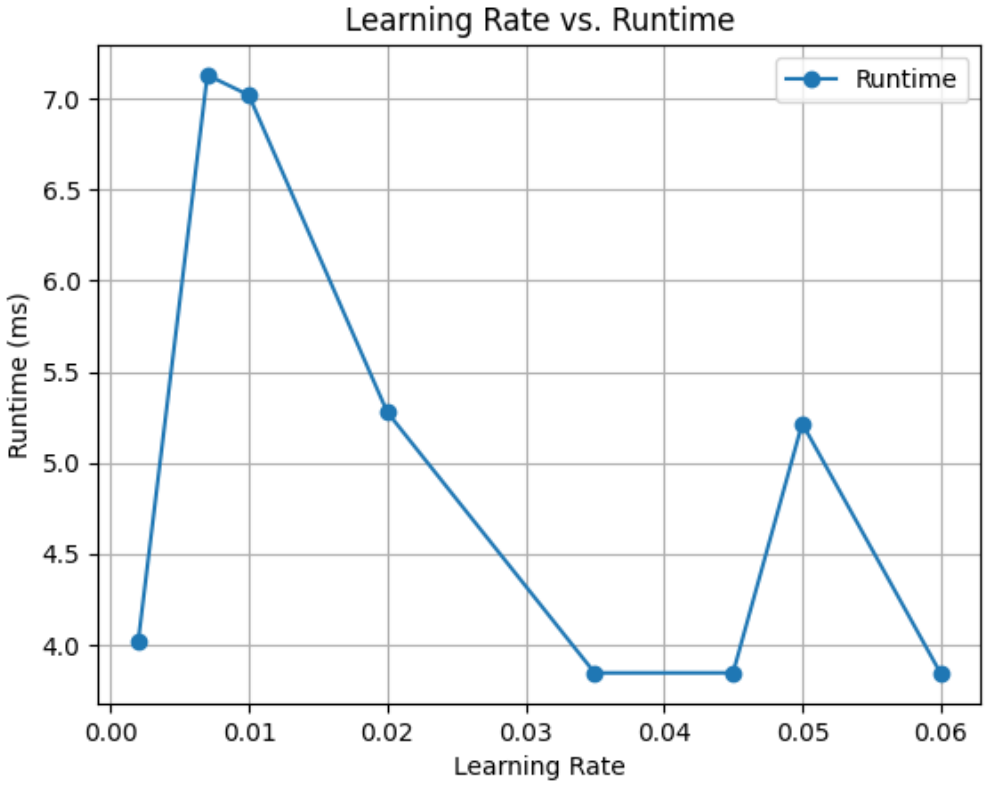


Figure 12: Boston Housing Dataset learning rate vs MSE

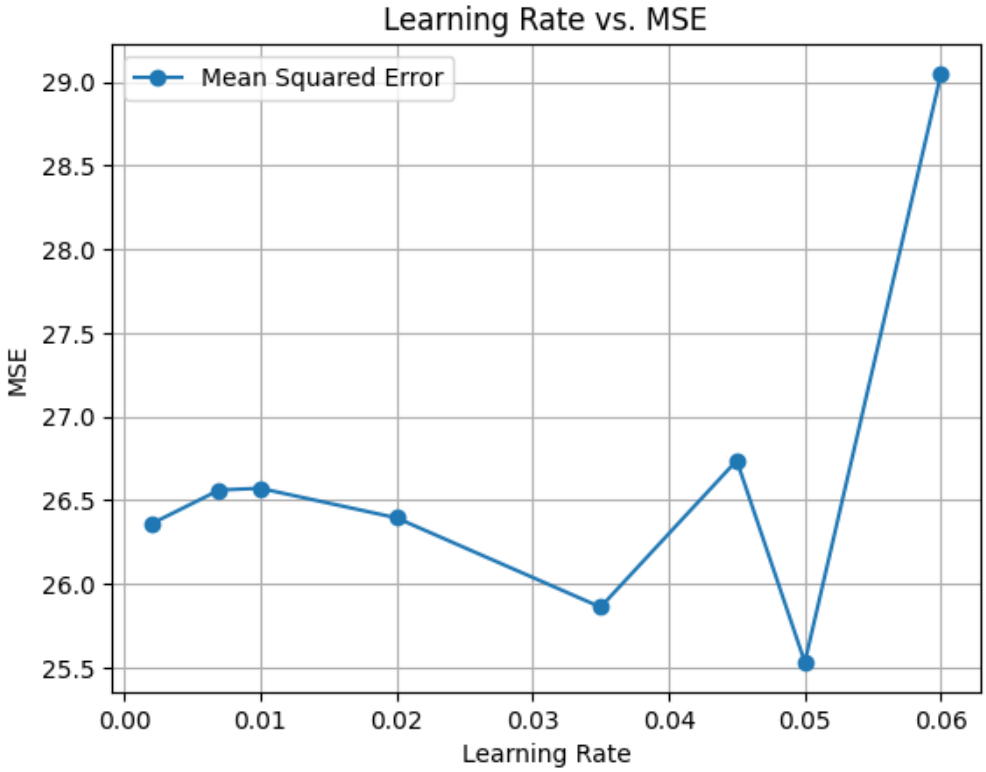


Figure 13: Wine Dataset learning rate vs fitting time

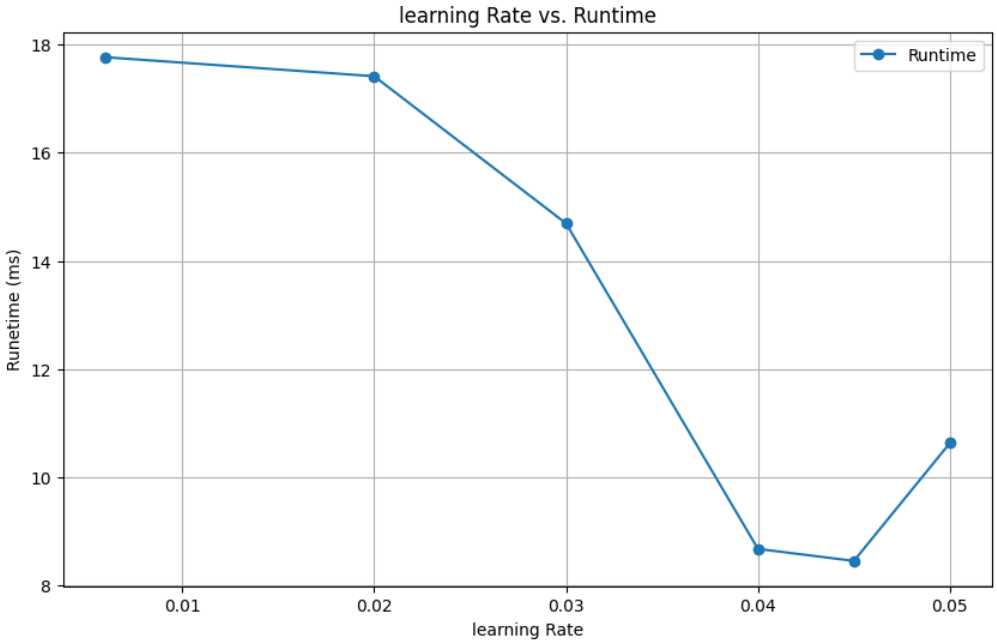
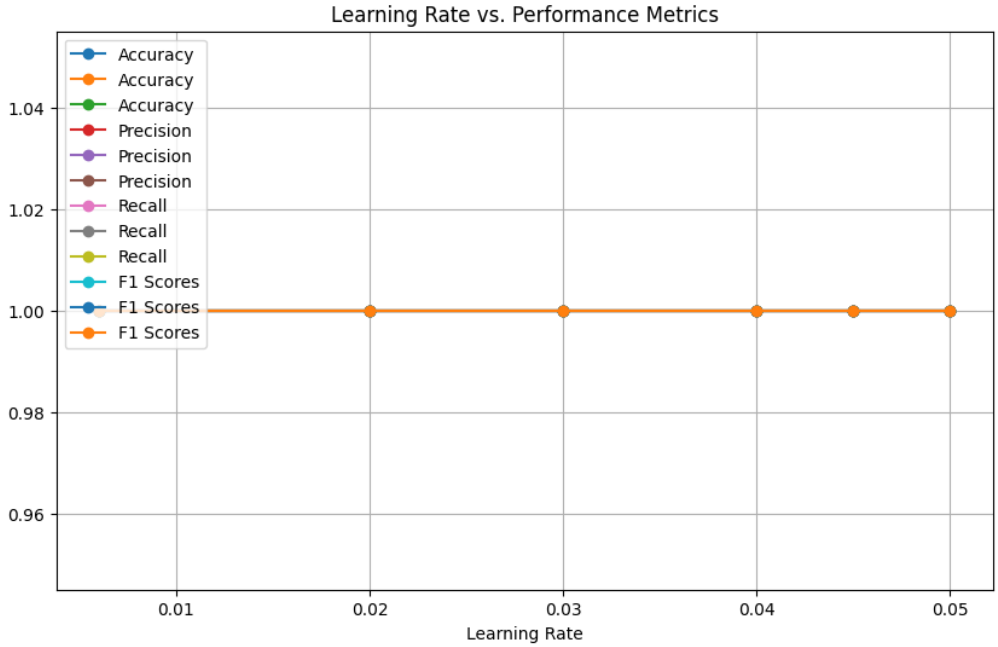


Figure 14: Wine Dataset learning rate vs performance metrics



Normalize helper function

```
def normalize(x):
  return (x - np.mean(x, axis=0)) / np.std(x, axis=0)
```