COMP 551 Mini-Project 1
Introduction to Regression Modeling

Yang Kai Yam, Xu Michael, and Tian Yu Hao

School of Computer Science, McGill University

Abstract

In this project, we explored various facets of machine learning by applying linear regression and logistic classification algorithms on two different datasets. After having preprocessed the data and run some rudimentary visualizations of both datasets, we implemented from scratch linear and logistic regression models. We found that for linear regression, the model that performs the best is the analytical one, although that would certainly change for a bigger dataset. As for logistic regression, we found that mini-batch gradient descent yields us slightly better results than batch gradient descent.

Introduction

The heart of this mini-project lies in trying all kinds of different combinations of parameters, models, and methods of data preprocessing for us to familiarize ourselves to the process of machine learning. We found out that the theory learned in class corresponds closely to the results we have obtained experimentally: adding regularization to our cost function stabilizes parameters whose values are too large, adding momentum in gradient descent makes the algorithm converge faster, etc. Also, we realized that 1. analytical linear regression works the best out of all models and 2. that there were severe limitations to our logistic regression algorithms caused by the small sample size available.

Dataset

Upon examining the datasets, we observed no significant feature abnormalities or missing data. However, we made the ethical choice to drop the 'B' column from the Boston dataset. Additionally, in the Boston dataset pairplots, we noticed skewness in features like 'CRIM' and non-linearity in certain trends. Categorical features like 'CHAS' also exhibited skewness. The correlation matrices (Appendix A1) provided insights into feature influence on the target variable and helped us address collinearity concerns during feature selection.

Results

A. Linear Regression

In the Dataset 1 experiment, we implemented three models (Analytical, Gradient Descent, Mini-Batch Gradient Descent Linear Regression) to explore the effects of parameter changes, and identifying optimal settings by comparing their performance.

A1. 80/20 Train/Test Split As the primary metric to assess predictive accuracy, we employed the mean squared error (MSE). Table 1 provides a reference point for further evaluation. Appendix A2 contains a test of different normalizations, we chose to leave the original data as-is for training the analytical model and min-max normalization for the Gradient Descent Model due to their lower MSE.

Analytical	GD	SGD
23.18	30.65	30.46

Table 1: *Baseline MSE Scores for Linear Regression models. Learning rate=0.6, iterations=500, batch size=30, train size = 0.8. Lower MSE value is better*

A2. 5-Fold Cross Validation MSE As seen in Table 2, the analytical model performs the best in terms of mean squared error (MSE) and mean absolute error (MAE) on both training and testing sets. The models also demonstrate similar R^2 values, indicating comparable explanatory power in the data. For detailed metrics, Ap-

pendix A3 shows detailed evaluations.

Model	Train MSE	Test MSE
Analytical	21.482	23.282
Gradient Descent	22.252	23.998
Mini-Batch SGD	21.550	23.388

Table 2: *5-Fold CV Metrics for Different Models*

To Investigate the Mini-Batch SGD Model's optimal parameters, we also tested the 5-Fold CV MSE while varying the learning rates (Appendix A4.1), iterations (Appendix A4.2), and batch sizes (Appendix A4.3). Learning rates ranging [0.05, 0.1, 0.2], iteration counts [500, 700, 800, 900, 1000], and batch sizes [4, 8, 16, 32, 64] consistently led to reduced MSE. The parameters will be the focus of further tuning experiments in the next three sections.

A3. Influence of Training Data Size We test the range of training set size from 0.2 to 0.8, the training size [0.55-0.65] gets the lowest MSE in the three models (1 & 2). We will use a training set of size 0.65 going forward.

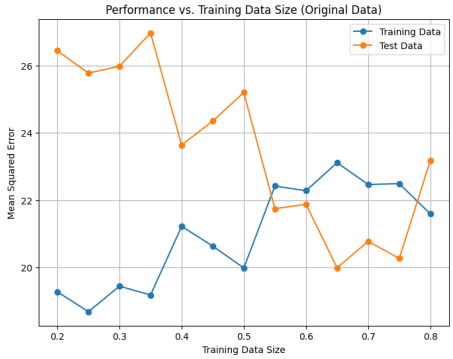


Figure 1: *Influence of Training Sizes on Analytical Model (No L2 Regularization)*

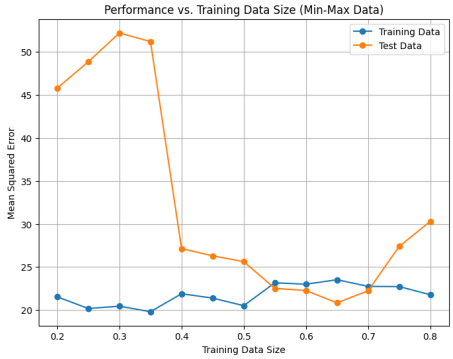


Figure 2: *Influence of Training Sizes on Mini-Batch SGD Model*

A4. Influence of Mini-Batch Sizes In Appendix A5.1, we experimented with increasing batch sizes from 1 to 128 and including a fully batched option. We maintained a learning rate of 0.1 and 1000 iterations for all runs, as determined in Section A2. Further analysis focused on batch sizes of 4, 8, 16, 32, and 64, which exhibited low MSE (22-23) and rapid convergence, aligning with the 5-fold cross-validation results.

A5. Influence of Learning Rates (LR) Figure 3 analyzes loss curves for different LR in the Mini-Batch model, focusing on the LR [0.05, 0.1, 0.2] that resulted in low MSE, based on 5-fold CV performance. These learning rates exhibited steeper loss curves, indicating their effectiveness. Also, Appendix A6.1 revealed that a LR of 0.5 performed optimally in the gradient descent model.

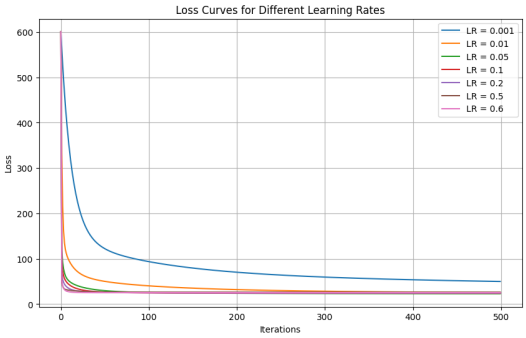


Figure 3: *Loss Curve in Mini-Batch SGD Linear Model with Various Learning Rates*

A6. Influence of the Number of Iterations For the mini-batch model, assuming a LR 0.1 and batch size of 30, we run each iteration choice 10 times and sum up the MSE values each time. Iterations [500, 1200, 1600], for their decent convergence speeds, achieved the lowest MSE scores ($\approx$ 22) shown in Appendix A5.2 and A5.3 respectively.

For the gradient descent model, iteration 100 is the fastest to converge (0.12s) with an MSE score of (22.69), as shown in Appendix A6.2.

A7. The Optimal Parameter Configuration For the analytical model, no hyperparameter tuning is needed, and we found 0.65 training size to obtain the best performance. The optimal MSE we could obtain is 19.991.

For the batch gradient descent model, hyperparameter tuning function is implemented with grid search. We found the optimal learning rate at 0.6 and number of iterations at 300. The calculation is shown in Appendix A6.3. With the training set size of 0.65, the optimal MSE obtained is 21.47.

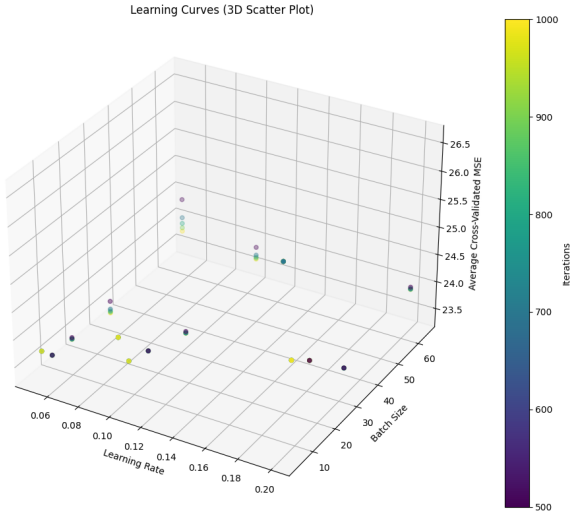


Figure 4: *The Graph of Hyperparameter Tuning for Mini-Batch SGD Linear Model*

To optimize the mini-batch SGD model, we performed hyperparameter tuning for learning rate, iterations, and batch sizes in a 3D grid shown in Figure 4. To ensure accuracy, we ran each parameter combination 10 times and averaged the results. To expedite the tuning process, we narrowed down the parameter ranges based on previous experiments. Specifically, we tested learning rates of [0.05, 0.1, and 0.2], iterations of [500, 700, 800, 900, and 1000], and batch sizes of [4, 8, 16, 32, and 64]. Through this process, we identified the optimal configuration with learning rate 0.05, iterations 1000, batch size 32, and training set size 0.65, resulting in an MSE of 20.611.

A8. Model Comparisons With the optimal parameters obtained through hyperparameter tuning, Appendix A7 shows the model comparisons in MSE, MAE, R-squares, and training speed. The analytical model is the best in all comparisons, while the performance of the mini-batch model is slightly better than batch gradient descent although they show similar patterns. But for the training time, gradient descent is faster than the mini-batch SGD.

A9. Influence of Momentum We tried running different situations from no momentum growing to max. momentum, to see how it helps the mini-batch performance. Figure 5 shows that with the exception of the run with momentum = 1.0, with growing momentum, the training speed and cross-validation MSE scores decrease.

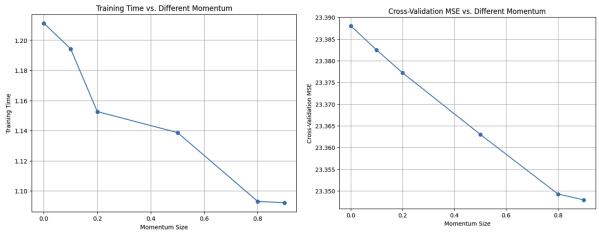


Figure 5: *Comparison of Different Momentum in Training Time and CV MSE*

A10. Feature Extraction using Gaussian basis functions In an analysis using the analytical linear regression model with identical training and test data, feature extraction was conducted employing five Gaussian basis functions. This approach resulted in a doubled mean squared error (MSE). The predicted values post-feature extraction were notably smaller than those derived without extraction, especially when the true values were significantly large, shown in Appendix A9.1. Nonetheless, upon conducting a grid search, shown in Appendix A9.2, it was observed that utilizing 65 Gaussian basis functions led to a substantial improvement in the MSE, decreasing it from 23.18 to 14.52, as detailed in Appendix A9.3.

B. Logistic Regression

B1. 80/20 Train/Test Split For the logistic regression part of the mini-project, we ran batch and mini-batch stochastic gradient descent as the sole two learning models. In order to obtain baseline performance results from our best initial guesses towards appropriate parameters, we first ran the two algorithms on a training set of size 0.8 and obtained their test set correlation matrices in Appendix B1.1 and B1.2. Going forward, we will mainly talk about accuracy and F1 scores to determine performance results for reasons described in the discussion.

Metrics	GD Train	GD Test	MB Train	MB Test
Accuracy	0.96	0.97	0.99	0.98
Precision	0.95	0.95	0.98	0.98
Recall	0.96	0.96	0.98	0.98
F1	1.91	1.91	1.97	1.96

Table 3: *Table of 5-Fold CV Performance Metrics for gradient descent and mini-batch models. Note: Higher is better*

B2. 5-Fold Cross Validation Scores As was done in section A2., we also ran performance metrics using the 5-fold cross validation method. We obtain higher performance metrics using the mini-batch gradient descent model as seen in Table 3.

B3. Varying Training Sizes For both models, we tried training set sizes varying from [0.2,0.8] and plotted them against their performance metrics in Figure 6. As for the performance for both models, the general trend follows that a larger training set yields higher metrics as seen in Appendix B3.2 and B3.3. Going forward we use a training set size of 0.8.

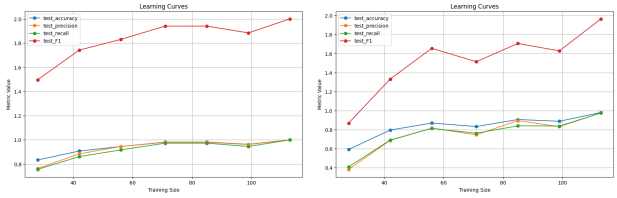


Figure 6: *Metrics curves plotted against learning rates for both GD and mini-batch GD*

B4. Varying Mini-Batch Sizes Looking at the table B4.1 in the appendix, we observe a general downward trend when increasing batch sizes. A mini-batch size of 8 seems to be the best compromise between adequate performance metrics and computing time, which we'll use going forward.

B5. Varying Learning Rates For the gradient descent model, we had to use a test size of 0.95 to get any change on the graphs since it converges to the final parameters too quickly as in the graph of Appendix B5.1. A larger learning rate seems to lead to better metrics however (Appendix B5.2). For the mini-batch model, we find a range of [0.06, 0.17] to be most adequate for the mini-batch size of 8 (Appendix B5.3).

B6. Optimal parameter selection Due to the challenges presented by class imbalance, we opted for the F1 score as our performance metric. We evaluated batch sizes of [1, 4, 8, 16, 32] and learning rates spanning [0.00625, 0.0125, 0.025, 0.05, 0.075, 0.1, 0.15]. To optimize computational efficiency, a single epoch was used for each grid iteration. We found the combination of a batch size of 1 and a learning rate of 0.05 consistently ranked within the top five in our F1 benchmark table, as detailed in Appendix B6.1.

B9. Additional Tweaks In hopes of getting better results, we applied L1-regularization to our gradient descent model. Also, to see what effects it would have, we tried to get rid of the 'ASH' feature that had the lowest influence on our target parameter. As shown in Appendix B9.1, we ended up getting overall worse results than our previous setup.

Discussion and Conclusion

The analytical linear model outperforms other models by directly calculating optimal parameters without iterative optimization [1]. However, for very large datasets, its high computational costs are impractical. Mini-batch SGD strikes a balance between computation speed and accuracy, although the randomness in batch selection introduces instability. To mitigate this, we perform hyperparameter tuning to find relatively optimal parameters, using average MSE and 5-fold cross-validation MSE [2]. Momentum allows the model to move in the same direction as the previous iterations, and normalization expedites the convergence speed, contributing to stability. The figure 7 shows a relative improvement with the usage of momentum. While our efficient tuning approach shows promise, exploring a broader range of algorithms can further enhance accuracy. At the end of the day, linear regression stays limited in what it can do as we are technically trying to fit a straight line through data that can be highly noisy and/or non-linear. For this reason, when we try applying different non-linear bases to our input features (Gaussian bases here), we see that a

much better performance is achievable.

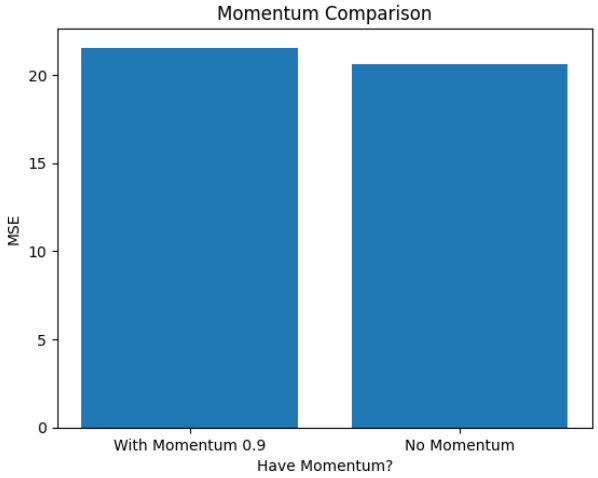


Figure 7: *Improvement shown by using Momentum in Mini-Batch*

The gradient descent linear model demonstrated shorter training time compared to the mini-batch SGD model, despite similar MAE and MSE results. This unexpected finding prompts further investigation into potential contributing factors. We believe hardware configuration, including parallel processing and GPU resources, may impact training time. Also, although our choice of learning rate (0.05) and batch size (32) achieved the best MSE, the small step size could influence training efficiency. It is believed the randomness of batch selection is also the factor leading the longer training time to the optimum. Future research should explore different hyperparameter patterns and optimize hardware resources and parallelization methods for more efficient training processes.

When it comes to logistic regression , it becomes a lot harder trying to determine the experimental setup that yields the best performance (B6) for a multitude of reasons. First of all, in this exact experiment, we had to work with a relatively small sample of n=189, which hinders greatly our ability to determine the validity of our performance metrics if we want a decently big enough training set (which the latter we would naturally want to prioritize based on our results found in section B3). Secondly, we found that the simplicity of the dataset coupled with the low amount of samples can be a problem in estimating the best parameters w^* as when we tried to estimate them in section B6, we found that given **enough epochs**, multitude of combinations of learning rates and batch-sizes could yield a F1 result of 2.0. On the other hand, the more we limited the amount of epochs given to the learning process, the more our results were relied on pure RNG. Where one specific combination could yield $F1 = 2.0$ on one run and $F1 = 0.9$ on the next. With that said, to conserve computational resources, we chose to use a single epoch for each iteration in the quest to identify the optimal parameters. Then we used ten epochs to refit the model with these optimal parameters and assessed its performance on the test dataset as a way to try to circumvent the issues listed above.

On the topic of performance metrics for logistic regression, we chose to use mainly the F1-score to determine goodness of a model configuration for two reasons: the first being that the wine dataset's class 2 has 47.9% more data points than class 3 (the latter being the most unpopular). And because of that, a model with high accuracy does not necessarily mean it performs as well on the third class as the others. On the other hand, we don't have any preferences for the lower Type I error or Type II error, so we should choose F1 over precision and recall.

Statement of Contribution

Yang Kai Yam - Implemented the analytical, GD, Mini-Batch SGD Linear Regression Model (Task 1, 2). Designed and run the full experiment (Task 3) on the linear models. Wrote the report for the linear regression models' results and discussion.

Xu Michael - Implemented the GD Logistic Regression Model and its experiments in task 3. Conducted Linear model's feature extraction using Gaussian Basis Functions. Assisted other teammates to finalize the report.

Tian Yu Hao - Wrote helper functions for linear and logistic regression. Implemented mini-batch GD in logistic. Wrote and finalized the most parts in report.

Reference

[1] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. Chapter 8 covers optimization algorithms, including stochastic gradient descent and mini-batch SGD.
Accessed: October 1, 2023 [Online]
Available: https://www.deeplearningbook.org/contents/optimization.html

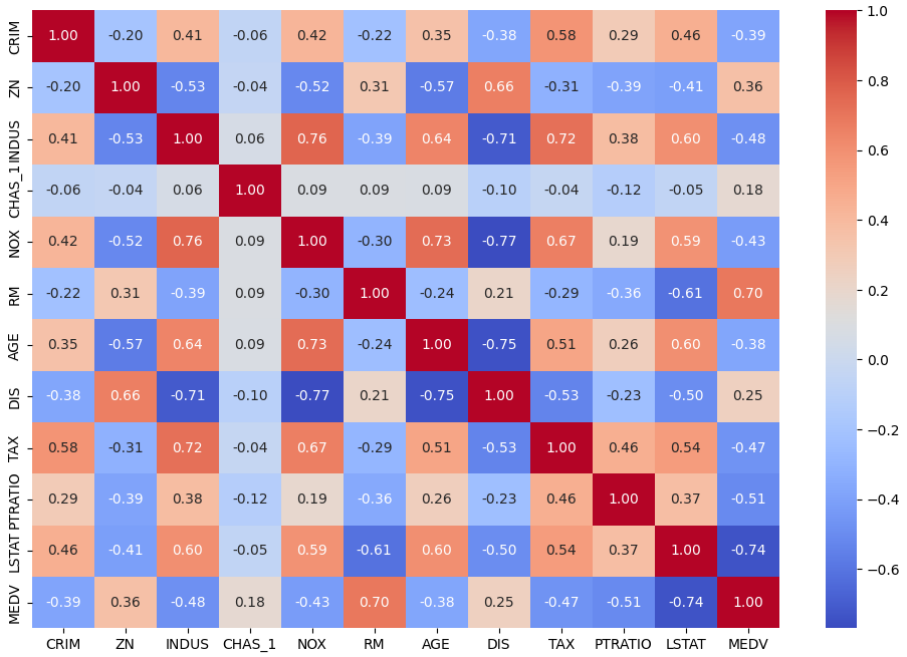
[2] Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13, 281-305. This paper discusses the use of random search for hyperparameter optimization, which can be useful when tuning the parameters of mini-batch SGD.
Accessed: October 1, 2023 [Online]
Available: https://jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Appendix

A. Linear Regression Model

1. Feature Correlation Matrix

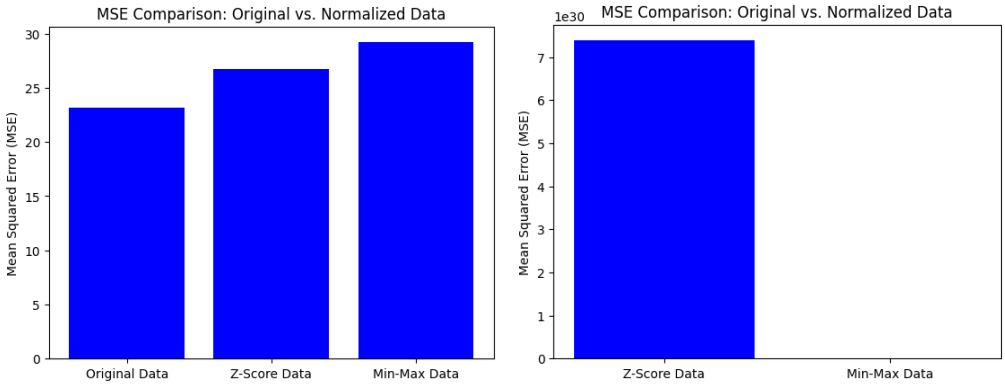
The matrix shows the correlation coefficients between all pairs of features. We can know the importance of each feature on the target output (MEDV)



(Figure A1. The correlation matrix of data set 1 feature and output)

2. The Influence of Different Normalization

The graphs show the MSE of different Normalization in the Analytical and Mini-Batch SGD Linear Models, with the original data and the min-max normalized data performing well respectively.



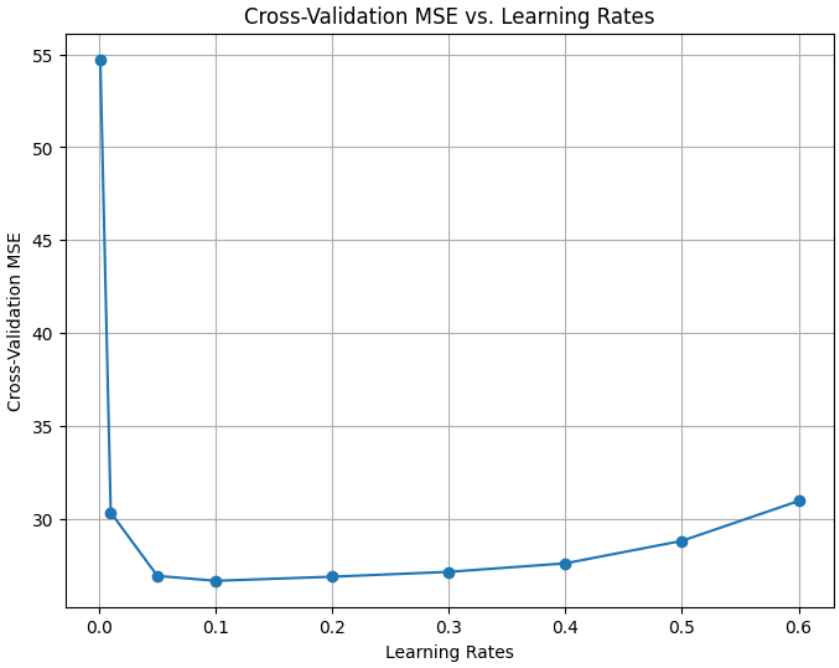
(Figure A2. The Influence of Normalization among analytical and GD Model)

3. **5-Fold Cross Validation of Test and Train Set Complete Table**
The tables are showing the performance metrics on both the training set and test set for each model.

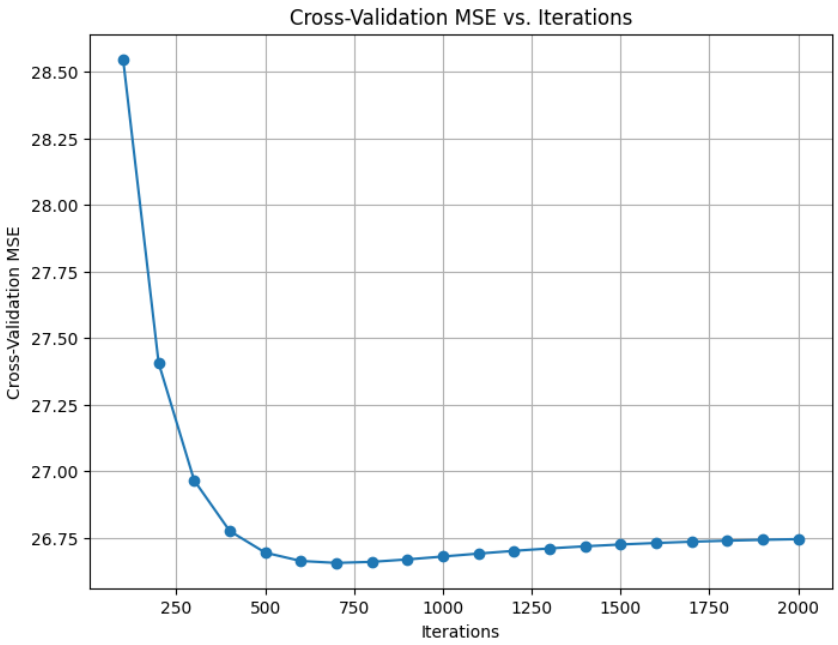
Model	Train MSE	Test MSE	Train MAE	Test MAE	Train R^2	Test R^2
Analytical	21.5	23.3	3.26	3.40	0.75	0.72
GD	22.3	23.9	3.25	3.38	0.74	0.71
Mini-Batch	21.6	23.4	3.24	3.39	0.74	0.72

(Figure A3. The 5-Fold Cross Validation Performance Metrics on Train and Test)

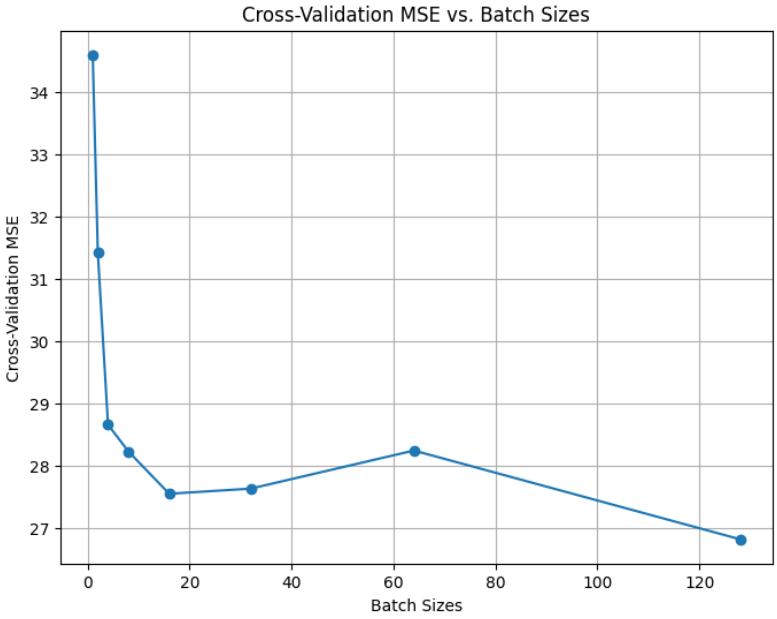
4. **5-Fold Cross Validation on Mini-Batch SGD**
The graphs show the 5-fold validation MSE performance of the Mini-Batch SGD Linear Model, changing by different learning rates, iterations, and batch sizes.



(Figure A4.1 The Mini-Batch CV MSE though increasing learning rate)



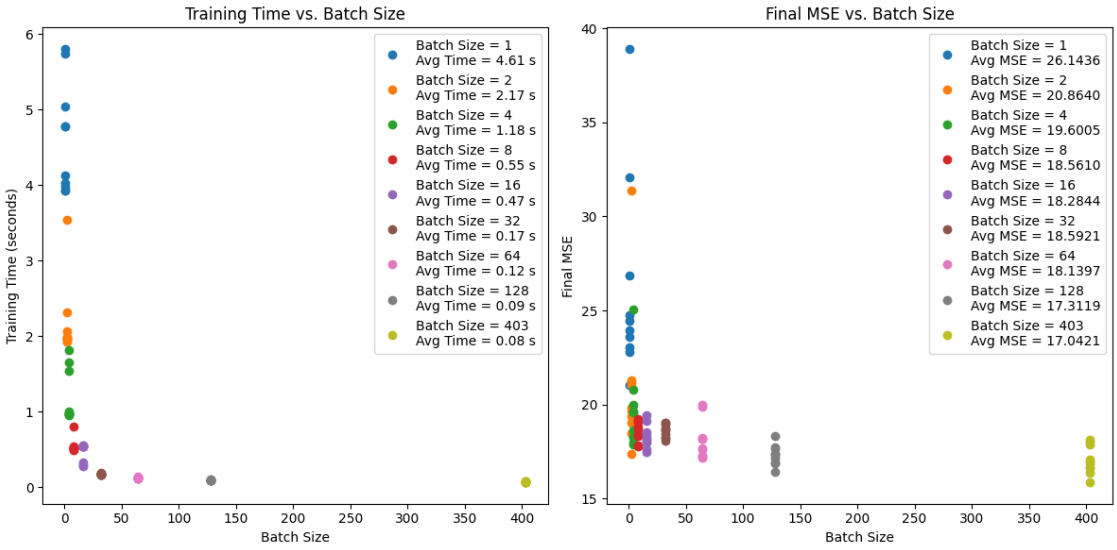
(Figure A4.2 The Mini-Batch CV MSE though increasing Iterations)



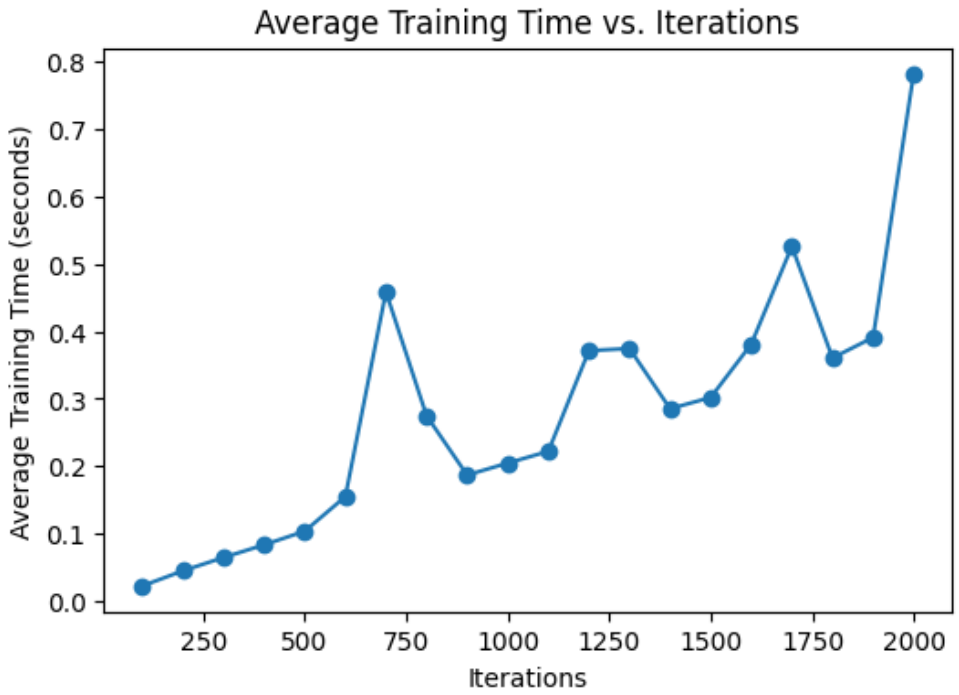
(Figure A4.3 The Mini-Batch CV MSE though increasing batch size)

5. Additional Analysis of Parameter Change in Mini-Batch

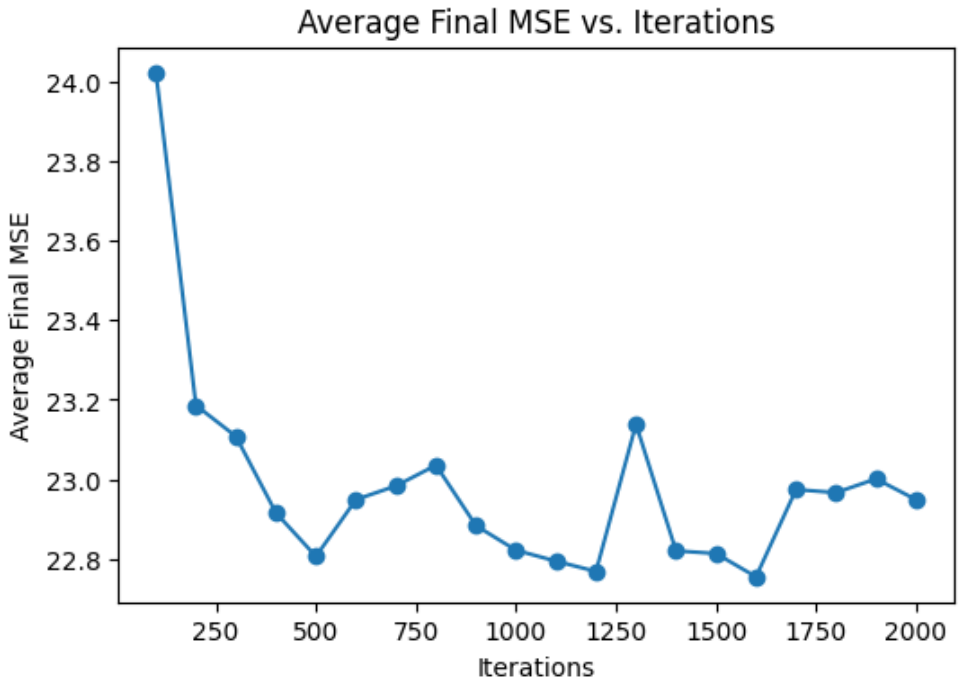
The additional analysis of how different batch sizes and iteration sizes influence the training time and average MSE



(Figure A5.1 The graph analyzes the training time and average MSE performance with different batch sizes)

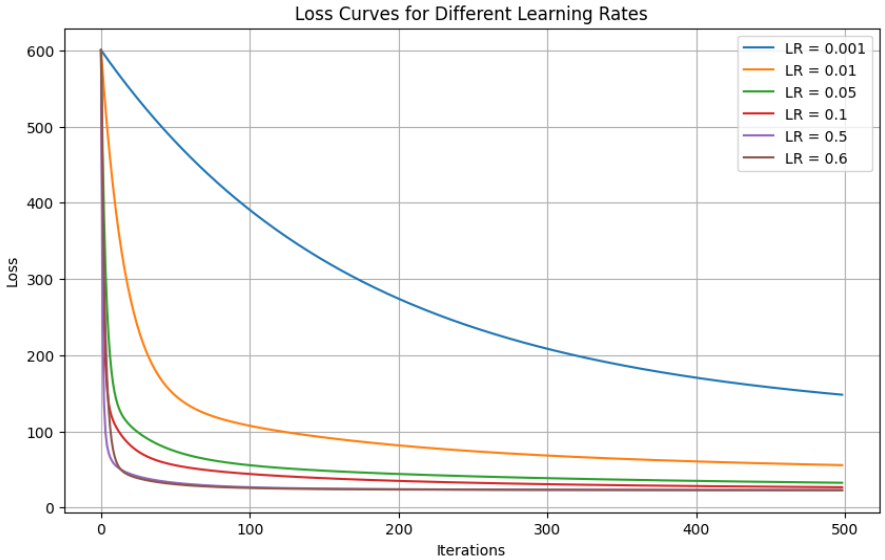


(Figure A5.2 The graph analyzes the training time with different iterations.)

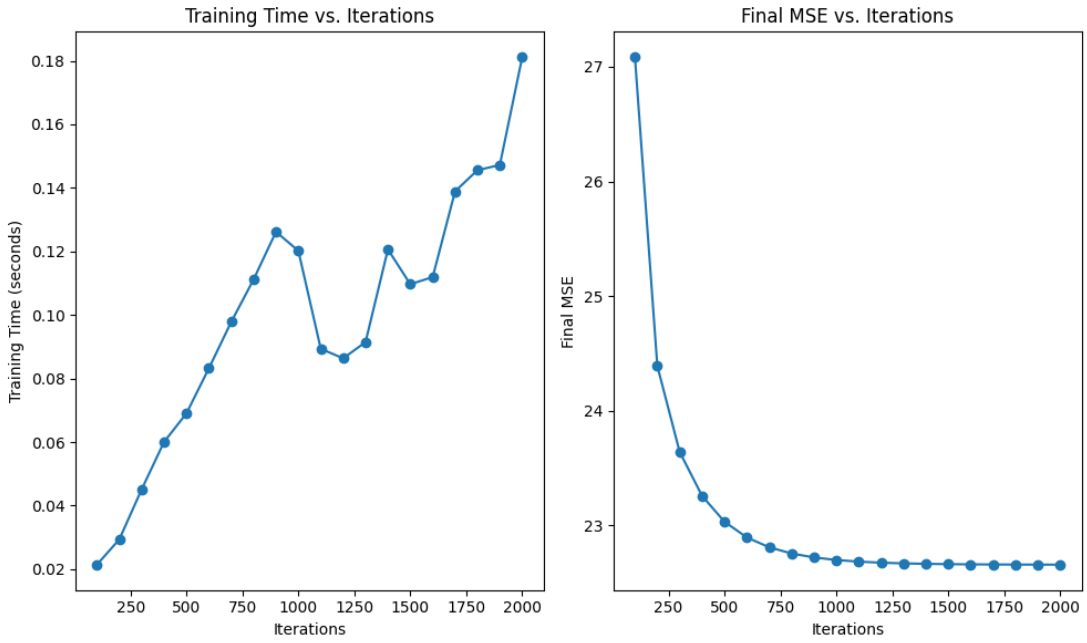


(Figure A5.3 The graph analyzes the average MSE performance with different iterations.)

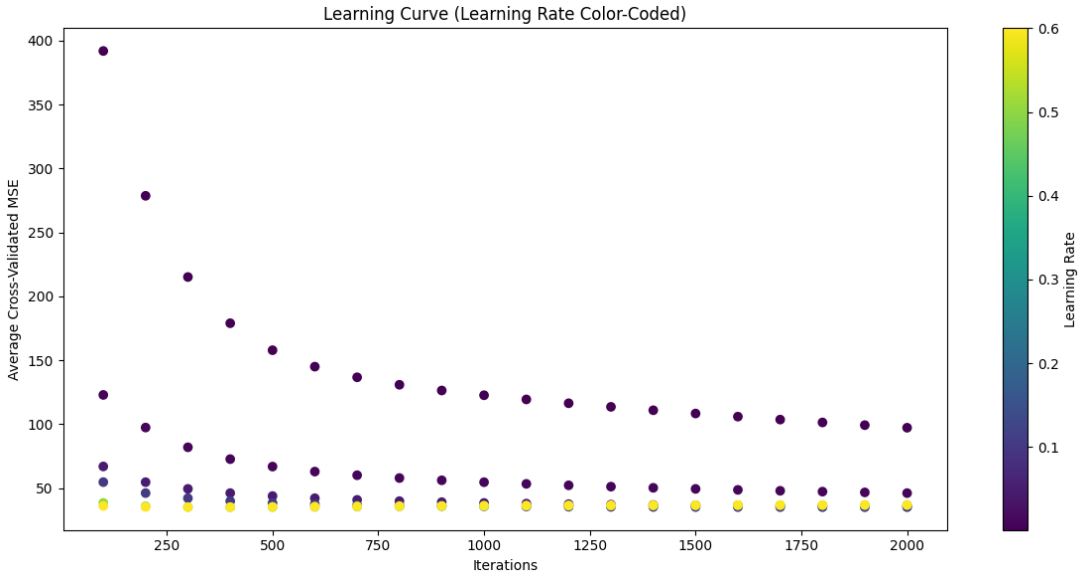
6. **Additional Analysis of Parameter Change in Gradient Descent**
Showing how the different learning rates and iterations affect the performance of the gradient descent model. Finally, hyperparameter tuning finds out the best parameter configurations for the GD linear model.



(Figure A6.1 The graph shows the Loss Curve in GD Linear Model with different Learning Rates)



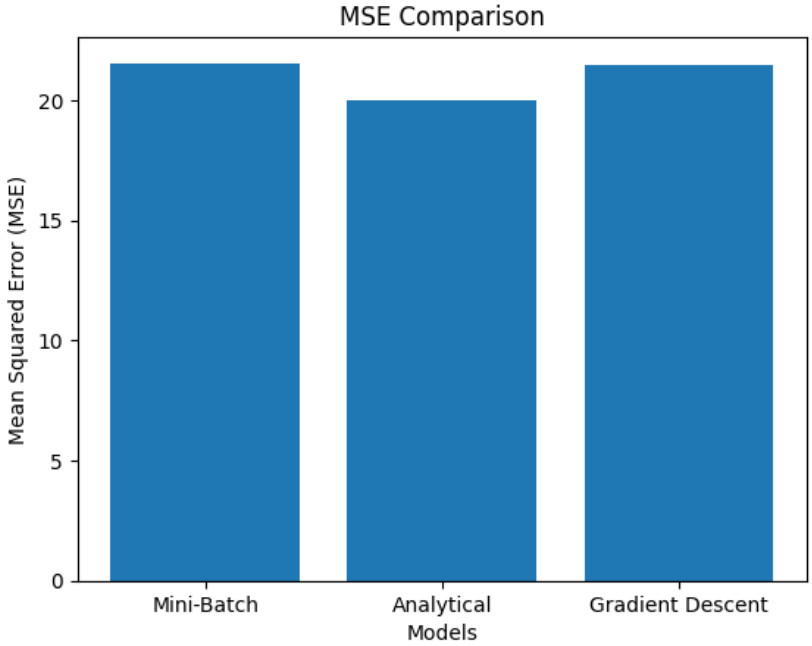
(Figure A6.2 The graph shows the training time and Average MSE in GD Linear Model with different iteration sizes.)



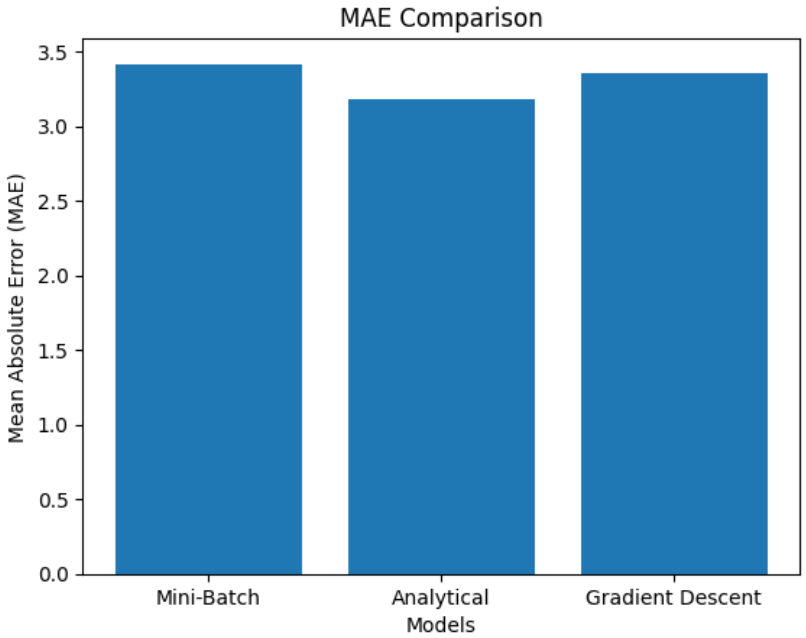
(Figure A6.3 The graph of hyperparameters tuning on the Gradient Descent Linear Model to find out the optimal parameters.)

7. The Graphs of Model Comparison

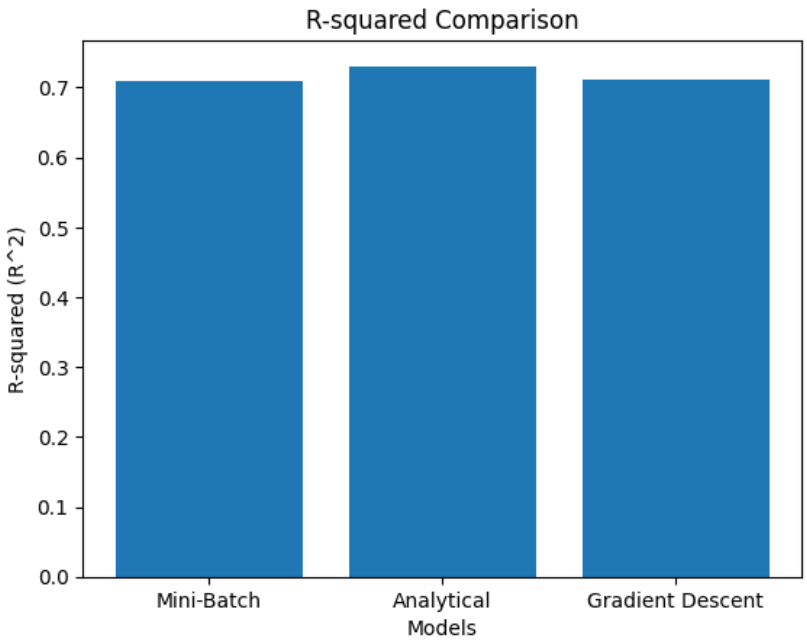
The comparisons of model performance (Analytical, GD, Mini-Batch SGD) in different aspects (MSE, MAE, R squares, Training Time)



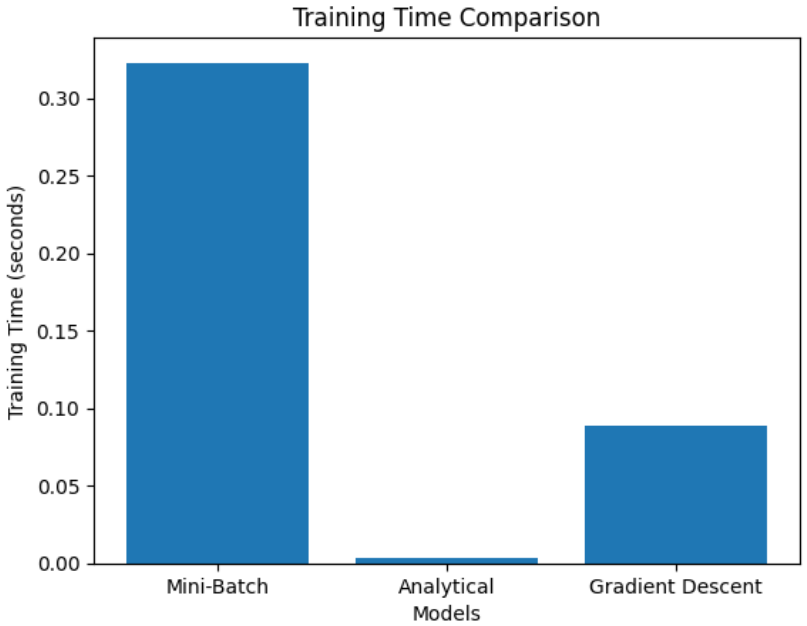
(Figure A7.1 The comparison of Models Performance (Analytical, GD, Mini-Batch SGD) in MSE)



(Figure A7.2 The comparison of Models Performance (Analytical, GD, Mini-Batch SGD) in MAE)



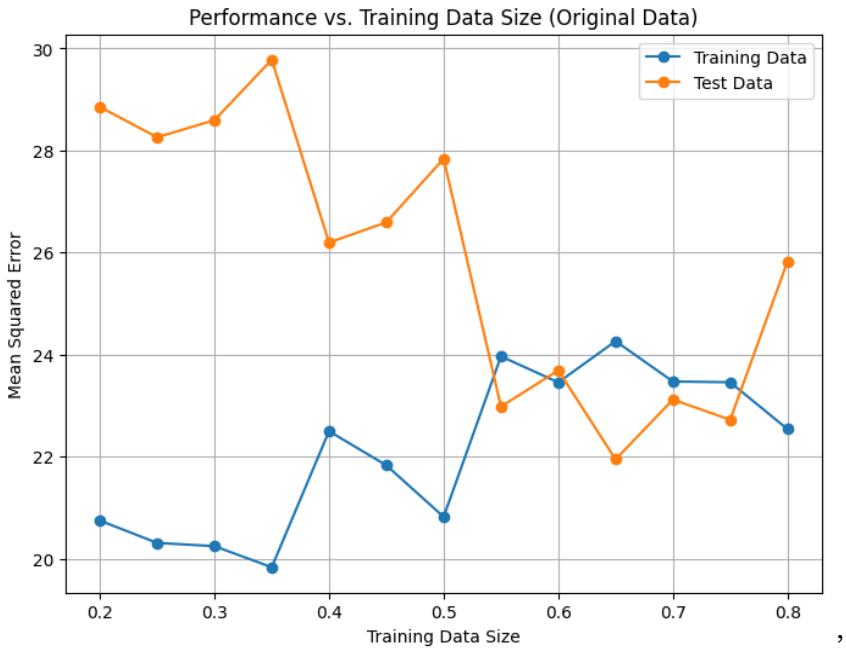
(Figure A7.3 The comparison of Models Performance (Analytical, GD, Mini-Batch SGD) in R squares)



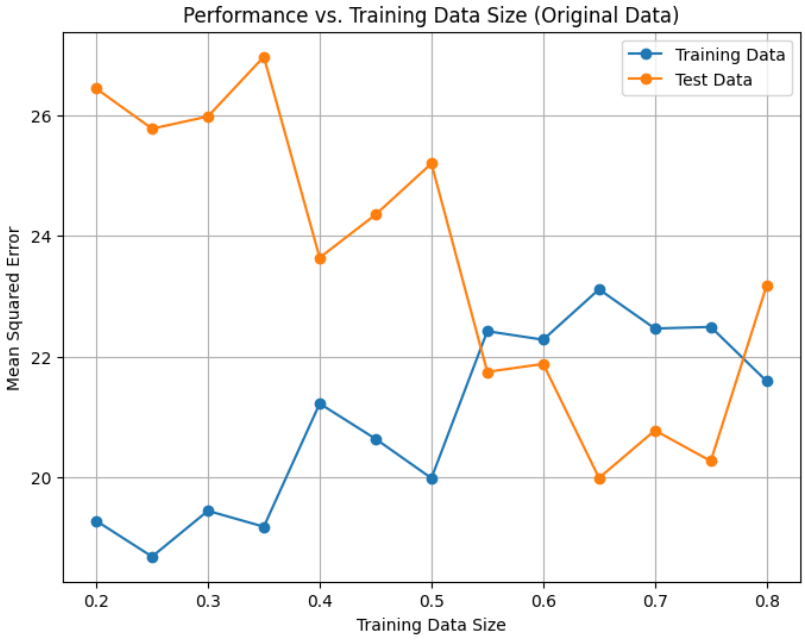
(Figure A7.4 The comparison of Models Performance (Analytical, GD, Mini-Batch SGD) in Training Time)

8. The L2 Regularization Influence

With various training sizes, we investigate the difference of the Average MSE of the analytical model with and without L2 Regularization). The graphs with and without L2 are similar shape and both obtain great MSE performance when the train size is 0.65. However, it shows L2 Regularization does not carry significant improvement to lower than the Average MSE, comparing to the situation without any regularization.



(Figure A8.1 The MSE Performance with L2 Regularization in Analytical Model)

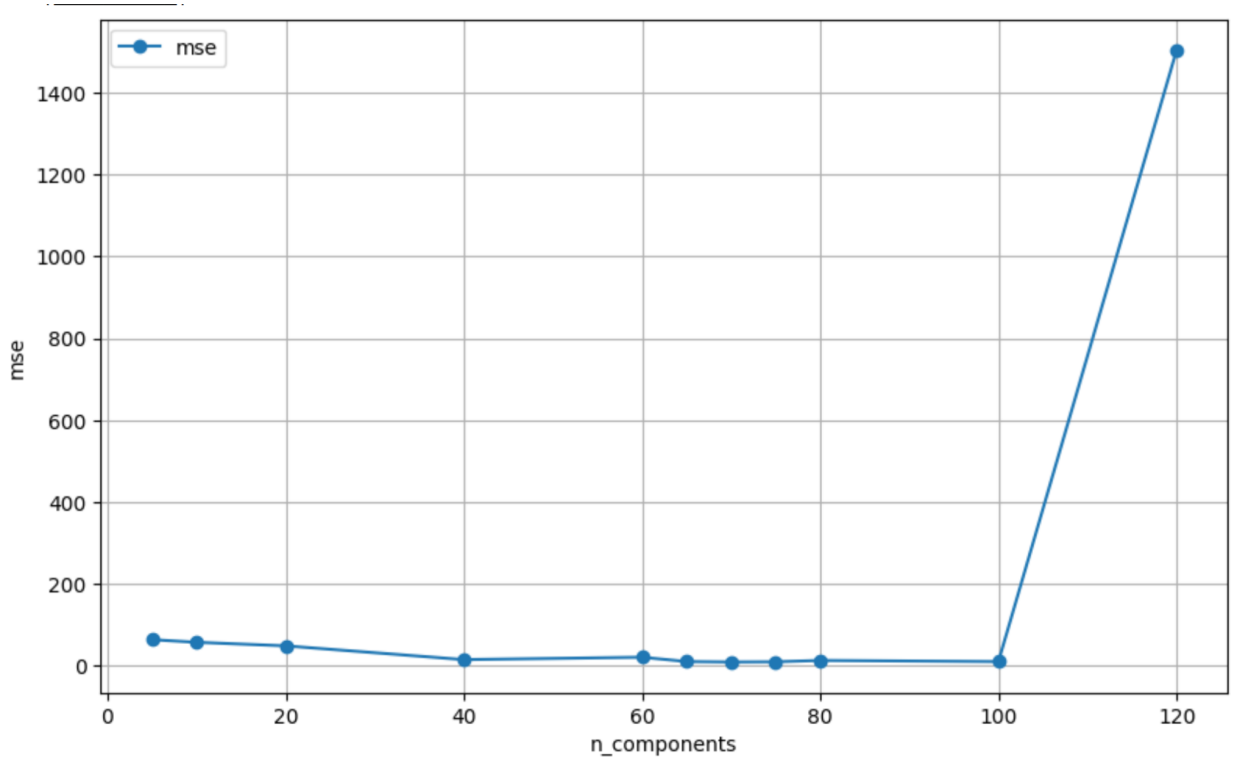


(Figure A8.2 The MSE Performance with no L2 Regularization in Analytical Model)

9. Predicted value comparison in Feature Extraction using Gaussian basis function

y_test	y_pred (before gs)	y_pred (after gs)
50.0	39.892550448581346	28.614770100668082
50.0	42.16300095435213	30.11147544767065
50.0	24.175011536543707	17.10119172213166
48.5	40.914458908719325	30.074946471671158
43.5	39.06585167546598	26.01041948325344
35.4	33.786123264123944	28.528567698337604
35.4	30.255921873950683	30.15809845922322
34.7	29.974269434456573	26.822844304385175
32.4	35.650576189856906	27.336656772045963
31.5	32.351101598083574	26.157753559171397
30.8	33.58963508002046	28.868788400452758
29.8	25.089869732415536	24.0916692523152

(Figure A9.1 The comparison of the predicted value before and after the feature extraction with target value)



(Figure A9.2 Grid search in training data to find the optimal numbers of Gaussian basis Function)

```
MSE from analytical linear regression (without Gaussian basis functions):  23.178598349478865
MSE from anlytical linear regression with 65 Gaussian basis functions):  14.515491430424108
```

(Figure A9.3 The comparison of the MSE in test data before and after applying 65 Gaussian basis functions)

B. Logistic Regression Model

B1. Baseline performance for logistic regression

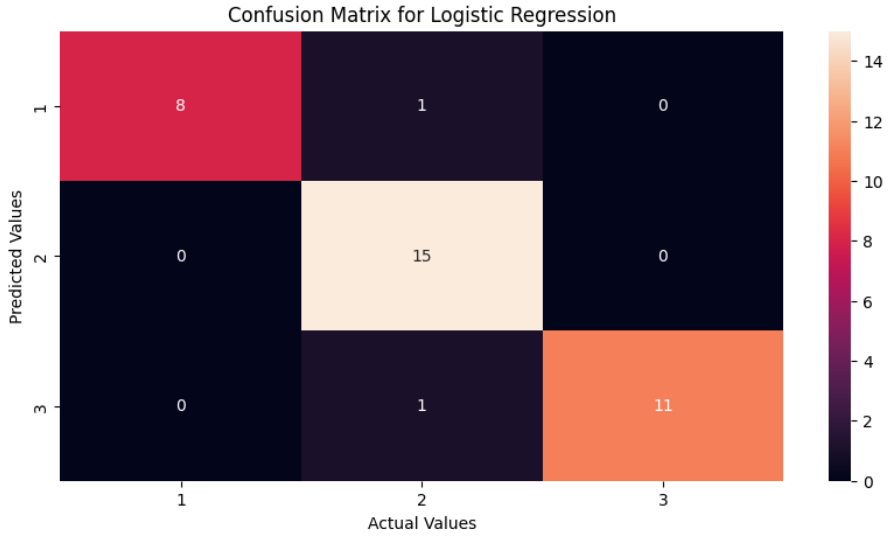


Figure B1.1: Cross validation matrix and its associated computed metrics for batch gradient descent logistic regression. Metrics are as presented for avg_accuracy: 0.962963, avg_precision: 0.935185, avg_recall: 0.960784, avg_F1: 1.890076

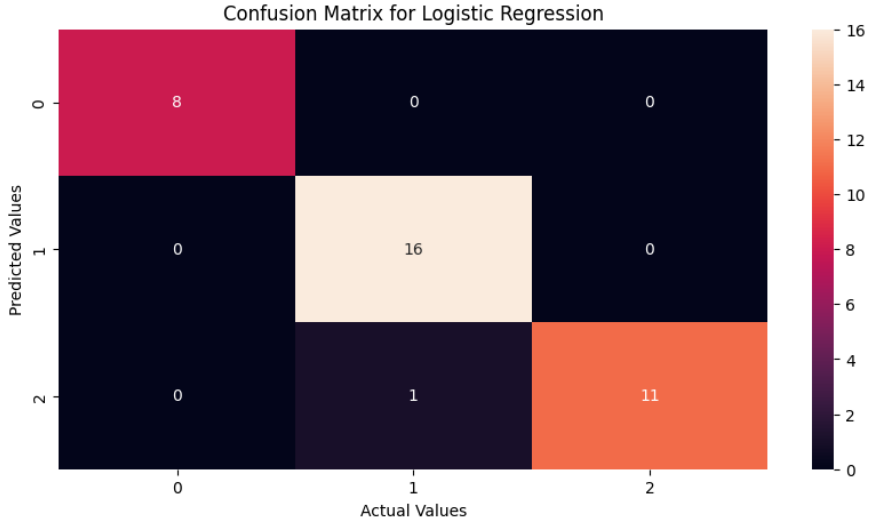


Figure B1.2: Cross validation matrix and its associated computed metrics for mini-batch gradient descent logistic regression. Metrics are as presented for avg_accuracy: 0.981481, avg_precision: 0.972222, avg_recall: 0.980392, avg_F1: 1.966340

B3. Varying Training Size

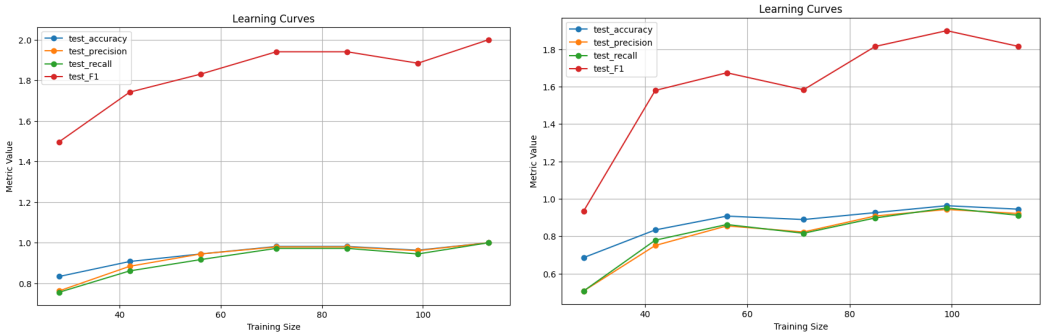


Figure B3.1: Graph of test metrics against differing training sizes for GD (left) and mini-batch GD (right).

	training_size	train_accuracy	train_precision	train_recall	train_F1	test_accuracy	test_precision	test_recall	test_F1
0	28	0.904762	0.857143	0.878788	1.755448	0.962963	0.940741	0.952381	1.899855
1	42	0.920635	0.889394	0.895833	1.787248	0.962963	0.940741	0.952381	1.899855
2	56	0.916667	0.894035	0.877193	1.766393	0.962963	0.940741	0.952381	1.899855
3	71	0.943662	0.922051	0.923077	1.855853	0.962963	0.940741	0.952381	1.899855
4	85	0.960784	0.941297	0.949495	1.896426	0.962963	0.940741	0.952381	1.899855
5	99	0.959596	0.936105	0.952381	1.890141	0.981481	0.962963	0.976190	1.957410
6	113	0.964602	0.943386	0.958333	1.910974	0.981481	0.962963	0.976190	1.957410

Figure B3.2: Table of metrics for the GD plot above.

	training_size	train_accuracy	train_precision	train_recall	train_F1	test_accuracy	test_precision	test_recall	test_F1
0	28	0.809524	0.700855	0.700000	1.445299	0.685185	0.506716	0.506716	0.934066
1	42	0.857143	0.774074	0.770370	1.605040	0.833333	0.750000	0.778388	1.579765
2	56	0.833333	0.752228	0.771086	1.533391	0.907407	0.854978	0.862230	1.674312
3	71	0.887324	0.827377	0.846232	1.685458	0.888889	0.822222	0.815629	1.583567
4	85	0.968627	0.942857	0.955429	1.930708	0.925926	0.907843	0.897436	1.815116
5	99	0.973064	0.952381	0.967480	1.943204	0.962963	0.942857	0.950549	1.899633
6	113	0.976401	0.958120	0.972222	1.942485	0.944444	0.921296	0.911681	1.815660

Figure B3.3: Table of metrics for the Mini-batch GD plot above.

B4. Varying Mini-Batch Sizes

Batch Size	Accuracy	Precision	Recall	F1
1	0.977528	0.961538	0.974359	1.954834
4	0.977528	0.961429	0.974359	1.942944
8	0.977528	0.961538	0.974359	1.954834
16	0.902622	0.853640	0.877493	1.720965
32	0.707865	0.564727	0.555184	1.159503
64	0.662921	0.488022	0.482266	1.064949
128	0.775281	0.669672	0.675379	1.328190

Table B4.1: Performance metrics against mini-batch size for MB gradient descent. We observe a general downward trend.

B5. Varying Learning Rates

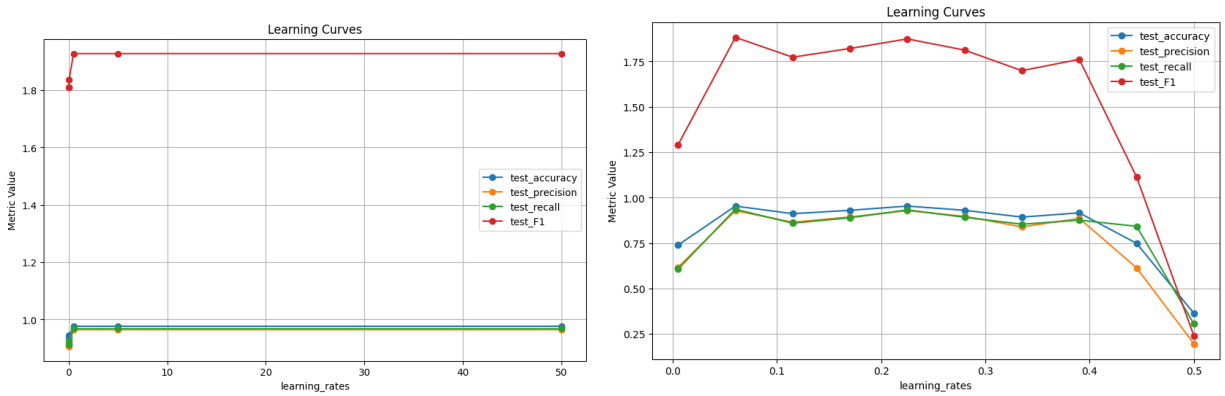


Figure B5.1: Plots of learning rates vs. performance metrics. (Mini-batch size = 8) GD on the left and Mini-batch on the right.

	learning_rates	train_accuracy	train_precision	train_recall	train_F1	test_accuracy	test_precision	test_recall	test_F1
0	0.0005	1.0	1.0	1.0	2.0	0.937255	0.906938	0.913610	1.808695
1	0.0050	1.0	1.0	1.0	2.0	0.937255	0.906938	0.913610	1.808695
2	0.0500	1.0	1.0	1.0	2.0	0.945098	0.917201	0.925306	1.837185
3	0.5000	1.0	1.0	1.0	2.0	0.976471	0.964282	0.968102	1.927450
4	5.0000	1.0	1.0	1.0	2.0	0.976471	0.964282	0.968102	1.927450
5	50.0000	1.0	1.0	1.0	2.0	0.976471	0.964282	0.968102	1.927450

Figure B5.2: Table of metrics from the above GD plot.

	learning_rates	train_accuracy	train_precision	train_recall	train_F1	test_accuracy	test_precision	test_recall	test_F1
0	0.005	0.771429	0.651351	0.629902	1.273346	0.762238	0.643412	0.657493	1.277416
1	0.060	0.980952	0.969697	0.980392	1.965065	0.995338	0.993939	0.993464	1.984103
2	0.115	0.980952	0.969697	0.980392	1.965065	0.948718	0.921292	0.926174	1.856669
3	0.170	1.000000	1.000000	1.000000	2.000000	0.953380	0.927293	0.930474	1.878484
4	0.225	0.809524	0.691704	0.693627	1.240770	0.841492	0.795443	0.781046	1.469214
5	0.280	0.942857	0.923611	0.919118	1.846815	0.939394	0.911337	0.917211	1.813204
6	0.335	0.942857	0.923611	0.919118	1.846815	0.930070	0.895705	0.905229	1.780266
7	0.390	0.523810	0.357143	0.223529	0.447007	0.519814	0.276905	0.271032	0.468931
8	0.445	0.314286	0.150000	0.176471	0.162162	0.384615	0.251858	0.343682	0.446429
9	0.500	0.171429	0.035714	0.029412	0.032258	0.167832	0.035016	0.047386	0.056532

Figure B5.3: Table of metrics from the above Mini-batch plot. A range of [0.06, 0.17] seems most adequate for a batch-size of 8.

B6. Finding optimal parameters

batch_sizes	learning_rates	cv_F1
1	0.1	2.0
1	0.05	1.962421173547132
1	0.075	1.9580378250591013
1	0.075	1.9532171662453353
4	0.1	1.896434911546341
1	0.075	1.8953621749002956
1	0.05	1.8686478566153888
1	0.05	1.8555555555555554
1	0.1	1.8090014460941504
1	0.075	1.7974202280778069
1	0.05	1.76478891051827
8	0.15	1.7100023380016627
8	0.00625	1.7077173909438461
1	0.05	1.7023942461760253
4	0.15	1.6591942677925964
1	0.025	1.658337113599795
8	0.15	1.6559100745497315
8	0.05	1.6528194979508573
1	0.1	1.6459947037404834
4	0.15	1.6366280527556292
4	0.075	1.635828536816567
1	0.1	1.610217009763627
1	0.025	1.5960288977781178
1	0.025	1.594762670194843
4	0.1	1.5848724148514937

Figure B6.1: F1 benchmark table with different batch_sizes and learning_rates

B.9 Additional Tweaks

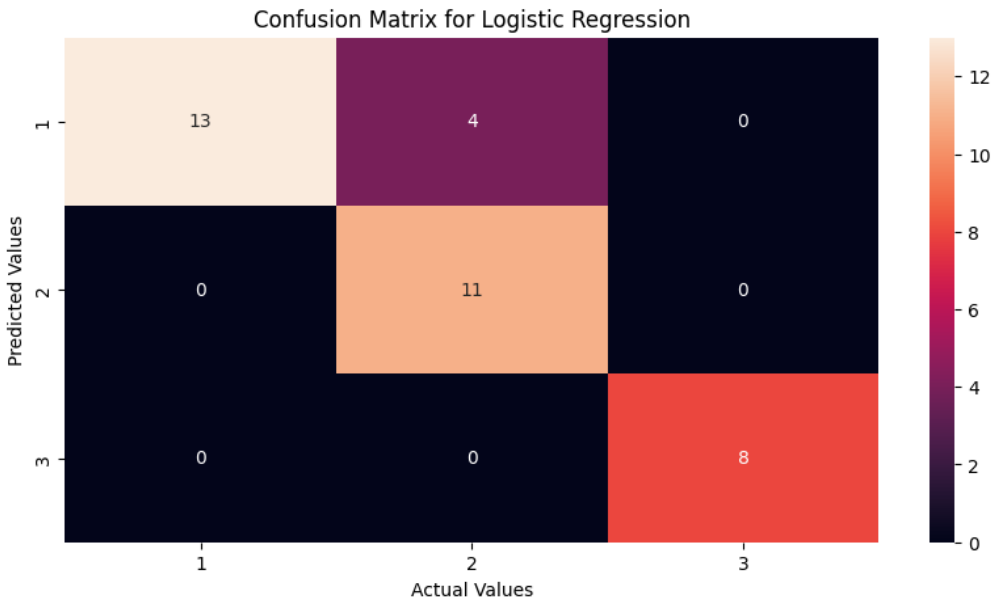


Figure B9.1 Resulting correlation matrix from L1-regularized and 'ASH'-less dataset. With metrics avg_accuracy: 0.925926, avg_precision: 0.921569, avg_recall: 0.911111, avg_F1: 1.788159.