Assignment 1: Linear & Logistic Regression

Alexandre St-Aubin, Jonathan Campana, & Jake Gameroff

Comp 551: Applied Machine Learning

September 30, 2024

Abstract

In this first assignment, we study the performance of our implementations of two machine learning models, in which each were trained on their own distinct dataset. The first machine learning model to be implemented was a linear regression on the facial and oral temperature data from a large set of human subject volunteers [1] dataset. For this part, we compare the performance of the closed form solution of the linear regression to that of a model trained with a mini-batch stochastic gradient descent. Our findings showed that the model using the closed form solution had a better performance while being computationally less expensive when compared to the model that was trained using a mini-batch stochastic gradient descent.

The second machine learning model we implemented was a logistic regression on the Diabetes Health Indicators dataset [2]. In its analysis, we compare the performance of a model trained with batch gradient descent with a model that was trained with mini-batch stochastic gradient descent. Our findings showed that mini-batch stochastic gradient descent outperformed batch gradient descent, offering faster training times while maintaining competitive predictive accuracy.

Introduction

This paper details the implementation of two machine learning models from scratch, without using Python ML libraries. The first model we implement is a linear regression on a dataset of facial and oral temperature data from a large set of human subject volunteers [1]. The dataset contains 33 features and two target variables, and our goal is to predict the target called `aveOralM`, which represents temperature in monitor mode. The linear regression is developed using two approaches: the closed-form solution and mini-batch stochastic gradient descent. These methods are compared based on their predictive power on test data and the computation/training time required.

Prior to building the models, we clean the dataset to account for data points with missing values in some of its features. There are many ways to handle these missing data points, but as found in [3], using k-nearest neighbors (kNN) or taking the average of the data points and setting the NaN value to that average does not heavily affect performance. Rather, they suggest deleting the data points with NaN values to reduce noise and complexity, which is how we proceed, especially since so few data points have missing feature data.

The second machine learning model we study is logistic regression, which we implement using batch gradient descent and mini-batch stochastic gradient descent on the Diabetes Health Indicators Dataset [2]. This dataset does not have any missing data and contains mainly binary indicators. The first approach we explore is batch gradient descent, which processes the entire dataset at each iteration. The second is mini-batch gradient descent, which processes a random subset of size B at each iteration. We hypothesize that the batch approach should be more computationally expensive but more stable. We compare both methods through various evaluation metrics such as accuracy, F1 score, and ROC AUC score.

We note that throughout all tests, we fixed a seed to ensure consistency for both the linear regression and logistic regression. This made it easier to reproduce results and observe how modifications, such as changing the batch size, affect our ML models [4].

1 Logistic Regression

Logistic regression is a popular and simple model used for binary classification problems. This model is trained using gradient-based methods. We experiment with different gradient descent techniques and hyperparameters and report our findings.

1.1 Data Processing

We train our classification models using the Diabetes Health Indicators Dataset, which includes 21 features and 1 target variable. The features consist of binary indicators (e.g., whether the individual smokes, has had a stroke, their sex) as well as integer values like age and self-reported health ratings. The goal is to predict whether an individual has diabetes based on this information.

We see that the dataset is imbalanced with only 13.86% of instances indicating a positive diabetes diagnosis. This nearly 7-to-1 ratio is common in many classification datasets but can lead to challenges. In particular, it may result in the *accuracy paradox*, where high accuracy at test-time appears impressive but primarily reflects the skewed class distribution rather than true model performance [5]. For this reason, we use Synthetic Minority Oversampling TEchnique (SMOTE) [6] to balance our data. SMOTE identifies minority class instances and their nearest neighbors in the feature space, then generates new samples by selecting a minority instance and a neighbor, drawing a line between them, and placing a new sample along that line. This increases both the quantity and diversity of minority class instances improving accuracy and generalization. We end up with 174,809 instances of each class.

To optimize memory usage and accelerate the training process, we load the data into pandas DataFrames and convert all values to 8-bit integers instead of retaining them as 64-bit. This conversion reduces the memory consumption from over 60 MB to just 9 MB.

Then, we scaled and shuffled the data before training. Scaling ensures that all features are on a similar range, preventing the model from being biased toward features with larger magnitudes. Shuffling introduces randomness, ensuring that the training and testing sets are representative of the overall population. It also helps prevent the model from overfitting to specific patterns or sequences in the data, leading to a more robust model.

1.2 Feature Importance

In logistic regression, the feature weights directly reflect the importance of each feature. Features with higher absolute weights contribute more to the decision boundary. As shown in Figure 3b, the features have different associated weights. Notably, features like Sex, MentHlth, NoDocbcCost, Stroke, and Smoker exhibit small weight magnitudes, suggesting minimal influence on predictions. We hypothesized that excluding these features might improve results, as prior research indicates that dropping low-importance features can enhance performance and accuracy [7]. To test this, we applied Recursive Feature Elimination (RFE) to recursively remove the least influential features. However, keeping the model hyperparameters constant, we saw no performance improvement when training on a reduced feature set. Also, since training was already computationally efficient, the reduction in features did not offer any meaningful speedup. As a result, we opted to retain all features in the final model.

1.3 Batch Gradient Descent

The batch approach processes the entire dataset in each iteration of gradient descent. This has the advantage of providing a steady and accurate convergence to the optimal weights. It is also the easiest to implement. However, in most contexts, it is impractical due to its expensive computation costs. We tested this approach with different learning rates on a $80-20$ train/test split. We first evaluate the accuracy of the model, which corresponds to the success rate. Then, we look at the confusion matrix, which shows True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). The F1 score is the mean of precision (precision $= 1$ when FP$= 0$) and recall (recall $= 0$ when FN$= 0$). A good model would have a F1 score of 1. Finally, we compute the ROC AUC score, which evaluates the model's ability to distinguish between classes, with a score of 1 indicating perfect classification, and 0.5 random guessing. It is useful for evaluating the trade-off between true positive and false positive rates.

Table 2 shows that the ROC AUC and F1 scores are the lowest when $lr = 10^{-8}$ since it takes too much time to converge, as seen in Figure 2c. We also observe from Figure 2a that when the learning rate is too high, the loss is high and fluctuates. We see from Figure 3 that $lr = 10^{-5}$ emerges as the optimal learning rate, as it achieves nearly the best performance across all evaluation criteria, and with far fewer iterations than $lr = 10^{-8}$.

In our next experiment, we sample growing subsets of the training data and evaluate against the performance of the models. As shown in Figure 1, there was no significant trend in performance, except for the 20% subset, which performed marginally worse than the others. However, we observed a clear decrease in the number of iterations needed as the training set size increased, which is expected.

1.4 Mini-Batch Gradient Descent

This second approach is cheaper than batch gradient descent. Instead of considering the whole dataset, we take a small random sample to approximate it. While using more examples improves the gradient estimate, the improvements are less-than-linear with respect to the number of added instances [8]. Additionally, redundancy in the training set can make sampling more efficient, as similar examples contribute the same information to the gradient [8].

To optimize the logistic model's performance, we implemented several strategies. **Early stopping** halts training when the validation loss stops improving, preventing overfitting. Early stopping is regulated by the *patience* parameter, which specifies how many iterations can pass without a decrease in validation loss before training is stopped. **L^2 regularization** adds a penalty for large weights, reducing overfitting by encouraging smaller, more generalized model parameters. Lastly, the **ADAM optimizer** adjusts learning rates dynamically during training, allowing the model to converge faster while avoiding issues like vanishing or exploding gradients. Together, these techniques aim to improve accuracy and model robustness.

We first tested the effect of batch size on the training time and overall model performance. We trained on batch sizes of 8, 16, 32, 64 and 128 instances, each with learning rates $10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$. All models were trained with a patience parameter of 5, no regularization and no ADAM optimizing. We found the best learning rate to be 10^{-1}, as smaller learning rates took longer to converge, and larger ones didn't converge. Table 2 compares the performance of different batch sizes, showing the number of iterations, training time, accuracy, F1 score and ROC AUC.

We see that the number of iterations required to converge to a minimum is inversely proportional to the batch size. This is expected because smaller batch sizes require more iterations for the model to process the same number of instances compared to larger batch sizes, which process more data per iteration. As a result, larger batch sizes also tend to require less training time, as seen in Table 2. We also observe a direct relationship between model performance and batch size, as evidenced by the upward trend in evaluation metrics as batch size increases. Larger batch sizes consistently lead to improved accuracy, F1 scores, and ROC AUC, indicating that the model performs better with more data per iteration. We conclude that Mini-batch is the better approach, as it requires less computations thus less training time than full batch, and achieves overall better results.

To optimize our model, we performed a grid search on the specified parameters (see Table 3) and selected the configuration that resulted in the highest accuracy. The best model utilized the following parameters: $lr = 0.001$, batch size $= 64$, patience $= 5$, $\lambda = 0.01$, $\beta_1 = 0.99$, and $\beta_2 = 0.99$. For comparison, we tested this optimal model against another model with batch size $= 128$ and $lr = 10^{-1}$, which was the best-performing mini-batch model without any additional parameter tuning. Both models were evaluated using 5 different random seeds, and we report the mean accuracy of each.

The mean accuracy for the best grid search model was 0.7406 with a standard deviation of 0.0004, while the mini-batch model achieved a mean accuracy of 0.7388 with a standard deviation of 0.0019. This demonstrates that the grid search model consistently outperforms the mini-batch model, albeit with a slight margin. We also observed a smoother convergence in the tuned model.

2 Linear Regression

Just as logistic regression models the probability of a categorical outcome, linear regression focuses on predicting a continuous dependent variable by analyzing its relationship with independent variables. It fits a linear equation to the data, minimizing the differences between predicted and actual values.

2.1 Data Processing & Analysis

The Infrared Thermography Temperature (ITT) dataset consists of temperature measurements from various body regions using infrared thermography, often used in medical diagnostics. The target variable, `aveOralM`, represents the average oral temperature of a person, and the goal is to predict this value based on other recorded factors, such as gender, age, and thermal features from different body regions.

To prepare the data for analysis, we performed several preprocessing steps, including cleaning any missing or irrelevant data. Furthermore, we applied encoding to convert categorical variables into numerical values, making them suitable for regression algorithms. We also scaled the features to normalize their ranges, which helps improve model performance by ensuring that all features contribute equally to the prediction process.

With this, we performed an elementary data analysis. Figure 5 shows a correlation heatmap that visualizes the relationship between various features in the dataset, helping to identify which features are most strongly correlated with the average oral temperature. Indeed, taking the three most correlated features (`canthiMax1`, `canthi4Max1`, and `T_RC_Max1`, Figure 7 displays their distributions, which are relatively normally distributed. This is important because normal distributions allow for more reliable linear regression modeling, as many statistical methods assume normally distributed input data to produce valid results.

2.2 Performance & Feature Importance

For our linear regression model, we used the closed-form solution to determine the optimal weight vector w^*. Given the data matrix X and the target values y, we solved for $w^* = (X^T X)^{-1} X^T y$, which minimizes the mean squared error between the predicted and actual values. Additionally, we used 80% of the data for training and 20% for testing to evaluate the model's performance.

Figures 4a and 4b display the performance of our linear regression model on the training and test data, respectively. As seen in Figure 4a, the model fits the training data well, but Figure 4b shows the performance on the test data, where we observe a slight drop in accuracy, indicating the model generalizes reasonably well but may be slightly overfitted to the training data.

We also conducted further experiments to evaluate how the size of the training data affects model performance. We sampled growing subsets of the training data and analyzed the model's performance in predicting both the training and test data. As expected, larger training set sizes generally improved test performance, reducing the model's tendency to overfit. The gap between training and test performance narrows as the training set size increases, indicating better generalization with more data. To ensure the robustness of our results, we averaged the performance across 100 different seed values, where each seed determines how the data is split into training and test sets. The resulting performance curves are provided in Figure 8 in the appendix.

Beyond performance, Figure 3a plots the contribution of each feature to the prediction. The three features with the highest weights were `canthiMax1`, `canthi4Max1`, and `T_RC_Max1`. This is expected, since these features were the most-heavily correlated with `aveOralM`. In contrast, several other features such as `Distance` or age-related features show near-zero weights, suggesting a minimal impact on the regression model's predictions.

2.3 Gradient Descent & Batches

Gradient descent is an optimization algorithm used to find the minimum of a cost function. In the context of linear regression, gradient descent approximates the closed-form solution by iteratively refining the weights to minimize the mean squared error. Mini-batch gradient descent, which updates weights based on small subsets of the data, is particularly useful for large datasets because it is an efficient way to attain fast convergence.

The plot in Figure 9 illustrates the performance of our mini-batch gradient descent algorithm as a function of batch size, showing both the R^2 score and runtime. Just as above, we average our results over multiple seed values. As expected, increasing the batch size results in longer runtimes, with a clear linear relationship between batch size and runtime. This is due to the increased computational effort required to process larger batches of data in each iteration. In contrast, the R^2 score remains relatively stable across different batch sizes, plateauing after batch sizes of around 50. This suggests that beyond a certain threshold, increasing the batch size offers no significant improvement in the model's ability to fit the data, when compared to the maximum possible batch size (which is about 800).

2.4 The Learning Rate

After analyzing the effect of batch size on performance, we shifted our focus to another critical hyperparameter, *the learning rate*, which controls how large of a step gradient descent takes at each iteration. This affects the speed and quality of convergence.

We tested various learning rates ranging from 0.01 to 0.08, evaluating the R^2 score on the test set for each case. The plot in Figure 10 visualizes the results. We observe that lower learning rates, such as 0.01, result in slower convergence and poorer performance, as indicated by the relatively lower R^2 score of 0.34. As the learning rate increases to around 0.07, the model reaches its peak performance, with an R^2 score of 0.64. Increasing or decreasing the learning rate outside of this range may lead to instability or overshooting, where performance could degrade.

2.5 Overall Performance

Throughout this analysis, we explored two linear regression models: the standard closed-form model and a gradient descent-based linear regression. While standard linear regression provides an exact solution, the gradient descent version offers an iterative approach that can handle large datasets efficiently by using gradient updates.

The plot in Figure 11 compares the overall performance of these two models, where the gradient descent model is measured by the number of iterations it takes to converge. As expected, the standard linear regression achieves the optimum, while the gradient descent model gradually approaches similar performance as the number of iterations increases.

3 Conclusion

In this assignment, we implemented a linear regression and logistical regression on datasets for the respective model. The linear regression was implemented using the closed form solution of linear regressions and a mini-batch stochastic gradient descent. The two implementations were then compared to see which method had better predictive power. Given that the closed form solution gives an exact solution, this implementation reaches the optimum quicker. The mini-batch stochastic gradient descent updates the weights and bias at each iteration, converging to the optimum as the number of iterations increases. So given this dataset, the closed form solution is a clear winner due to its accuracy and reduced complexity compared to a mini-batch SGD approach. We also tested the performance of different batch sizes, and found that beyond a certain plateau, the computation time increase due to the bigger batch size would out weigh the convergence speed. The dataset in which the models were deployed had 33 features and just over 1000 data points, which is relatively small, so it is not guaranteed that the closed form solution would perform better computationally if we had more data points. With a very large dataset, matrix multiplication can be computationally expensive, where the mini-batch SGD takes batches and avoids the issue of processing all the data points. Thus it is possible that a mini-batch SGD approach can be more computationally efficient compared to the closed form solution approach in that case.

The logistic regression was implemented with a gradient descent and a mini-batch stochastic gradient descent. The batch gradient descent has a more steady and accurate converge to the optimal weights at the cost of increased computation time. Since all the data points are processed in the updating of the weights, a batch approach has large memory consumption, and is prone to running out of memory. We actually had to convert our values in the pandas DataFrame to use 8-bit integers instead of 64-bit integers to avoid memory issues. The mini-batch stochastic gradient descent avoids this issue by taking random batches and updating the weights at each iteration. As we increased the batch size, the number of iterations needed decreased and the accuracy increased, as we would expect given the convergence of a batch approach. However, considering the predictive power of the model, and computation time of the training the model, mini-batch is preferred due to its better complexity, while having almost the same predictive performance.

4 Contributions

Jonathan: statistics/analysis of the linear regression; writing of abstract, introduction, conclusion. Jake: cleaned, encoded, and scaled, data for ITT data set; implemented linear regression and GD linear regression models; report on these models. Alexandre: cleaned, encoded, and scaled, data for IDH data set; implemented Logistic regression. Added ADAM optimizer, early stopping and L^2 regularization to the model. Wrote the logistic regression section of the report.

References

[1] Quanzeng Wang, Yangling Zhou, Pejman Ghassemi, Dwith Chenna, Michelle Chen, Jon Casamento, Joshua Pfefer, and David Mcbride. Facial and oral temperature data from a large set of human subject volunteers, May 2023.

[2] Alex Teboul. Diabetes health indicators dataset, Nov 2021.

[3] Parsa Razmara, Tina Khezresmaeilzadeh, and B Keith Jenkins. Fever detection with infrared thermography: Enhancing accuracy through machine learning techniques. *arXiv preprint arXiv:2407.15302*, 2024.

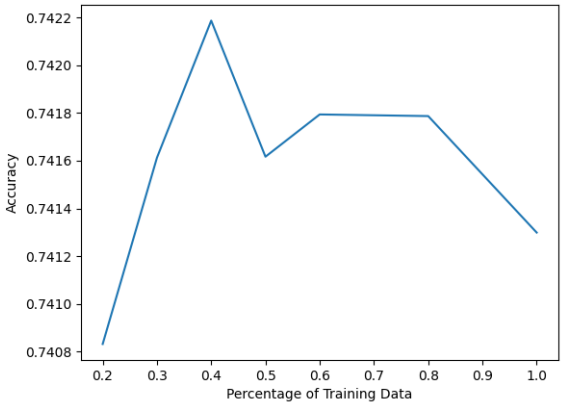
[4] ODSC Open Data Science. Properly setting the random seed in ml experiments. not as simple as you might imagine, May 2019.

[5] Jason Brownlee. 8 tactics to combat imbalanced classes in your machine learning dataset, August 2020. Accessed: 2020-08-15.

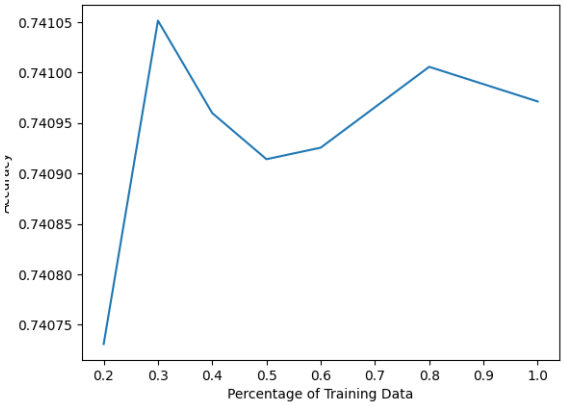
[6] Kevin W. Bowyer, Nitesh V. Chawla, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *CoRR*, abs/1106.1813, 2011.

[7] Isabelle M Guyon, Jason Weston, Stephen D. Barnhill, and Vladimir Naumovich Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46:389–422, 2002.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. In *Deep Learning*, chapter 8. MIT Press, 2016. `http://www.deeplearningbook.org`.



(a) Percentage of training data vs. Train Accuracy



(b) Percentage of training data vs. Test Accuracy

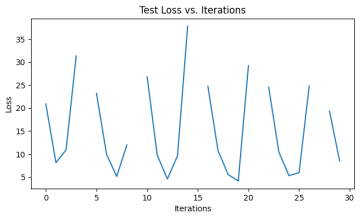
Figure 1: Percentage of training data vs. Accuracy of the resulting model.

Learning Rate	Accuracy	F1 Score	ROC AUC	Confusion Matrix	Iter.
1e-8	0.7139	0.7070	0.7139	[[32200 11471],[13514 30149]]	1000
1e-5	0.7405	0.7450	0.7405	[[31570 12101],[10560 33103]]	100
1e-2	0.7213	0.7341	0.7213	[[29400 14271],[10061 33602]]	30

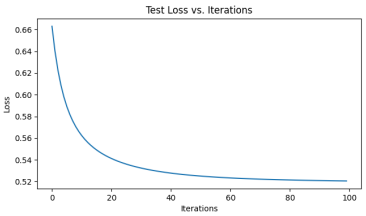
Table 1: Performance metrics for different learning rates using full-batch gradient descent.

Batch Size	Iterations	Training Time (s)	Accuracy	F1 Score	ROC AUC
8	77	0.4665	0.7190	0.7089	0.7191
16	52	0.2957	0.7244	0.7173	0.7244
32	64	0.3760	0.7359	0.7389	0.7359
64	25	0.1478	0.7328	0.7327	0.7328
128	35	0.2043	0.7402	0.7449	0.7402

Table 2: Performance Comparison for Different Batch Sizes with a learning rate of 10^{-1}.



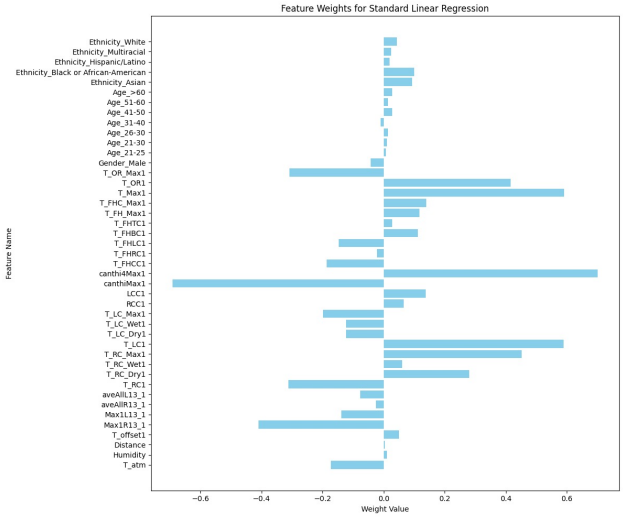
(a) Test loss vs. iterations, $lr = 10^{-2}$.



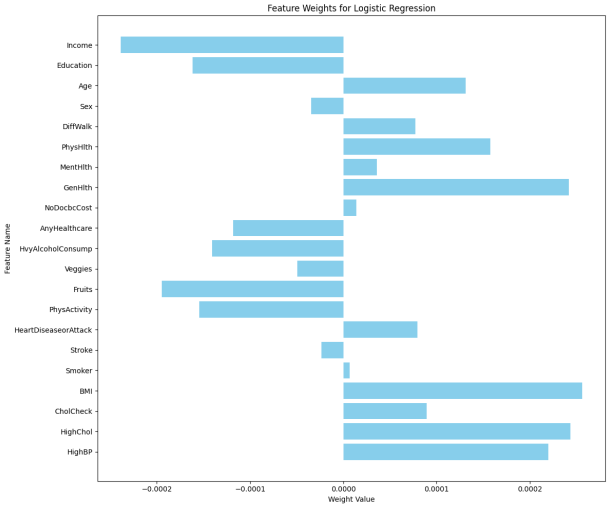
(b) Test loss vs. iterations, $lr = 10^{-5}$.

(c) Test loss vs. iterations, $lr = 10^{-8}$.

Figure 2: Test loss vs. iterations for different learning rates.



(a) Linear regression feature weights.

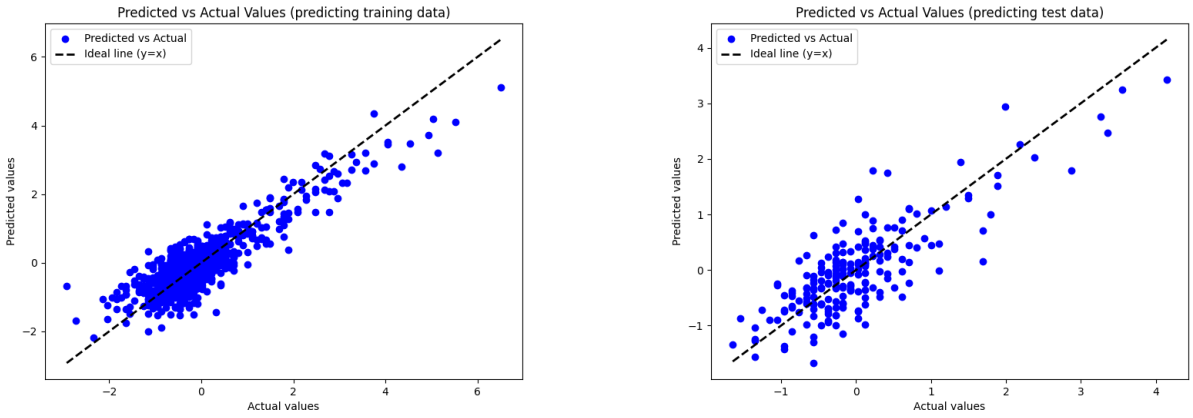


(b) Logistic regression feature weights.

Figure 3: Feature weights for both models

Parameter	Values
Learning Rate	0.01, 0.1, 0.001, 0.0001, 0.00001
Max Iterations	2000
Batch Size	8, 16, 32, 64, 128
Beta 1	0.9, 0.99, None
Beta 2	0.99, 0.999
Patience	5
Lambda (Regularization)	1, 0.1, 0.01, 0.001

Table 3: Grid search parameters for logistic regression optimization.



(a) Performance on training data.

(b) Performance on test data.

Figure 4: Performance of the linear regression model on training and test data.

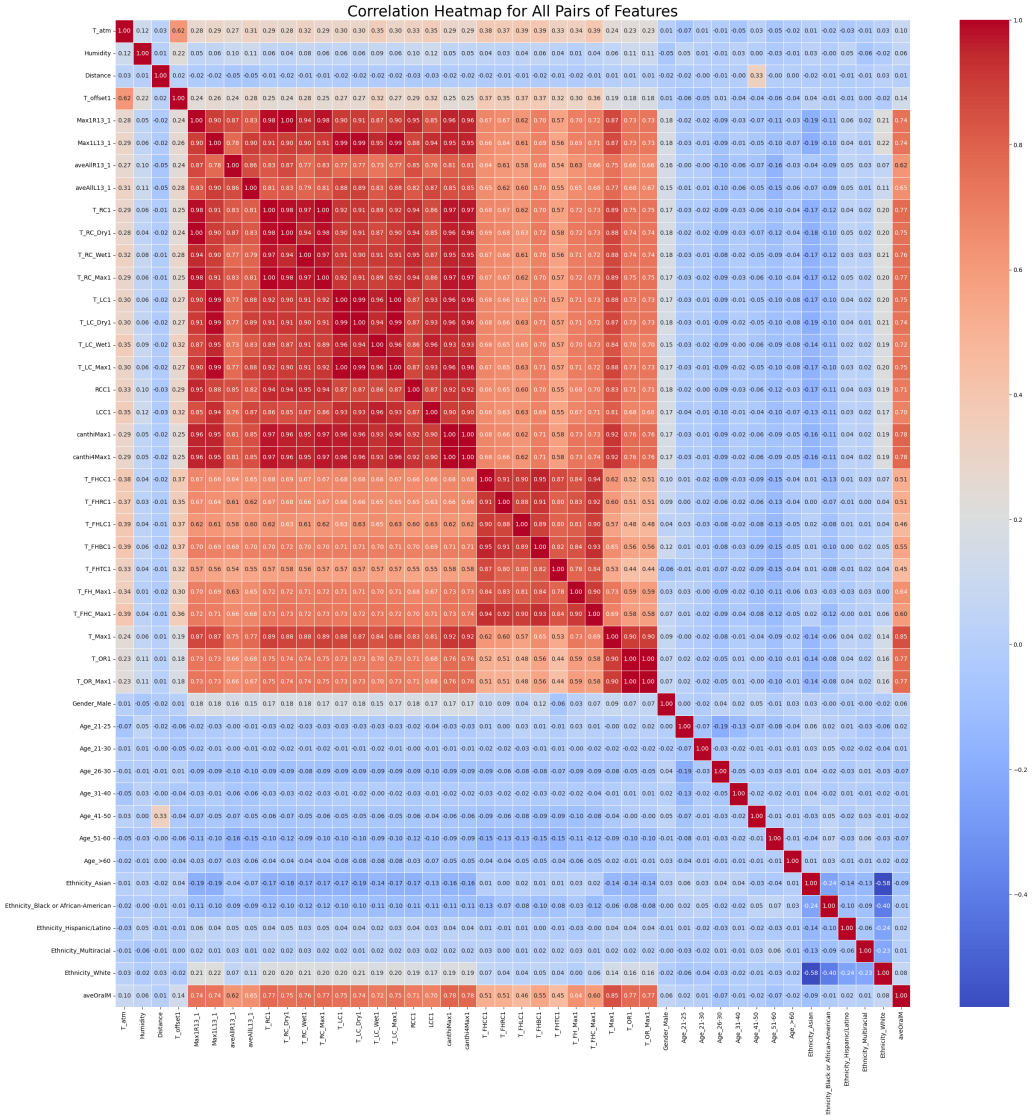


Figure 5: Correlation heatmap for all features with `aveOralM`.

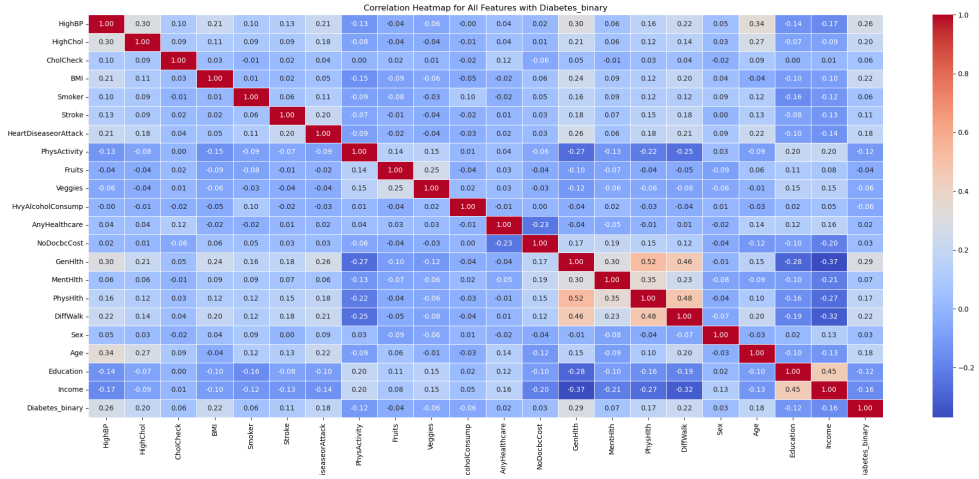
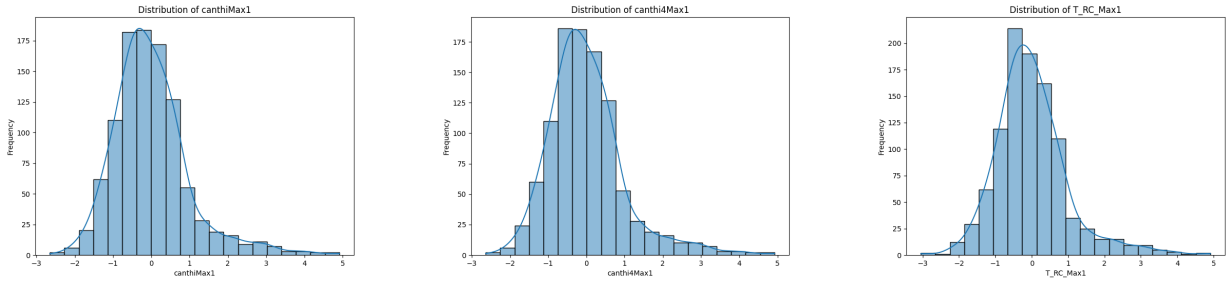


Figure 6: Correlation heatmap for all features with `Diabetes_binary`



(a) Distribution of `canthiMax1`. (b) Distribution of `canthi4Max1`. (c) Distribution of `T_RC_Max1`.

Figure 7: Distributions of the three features most highly correlated with `aveOralM`.

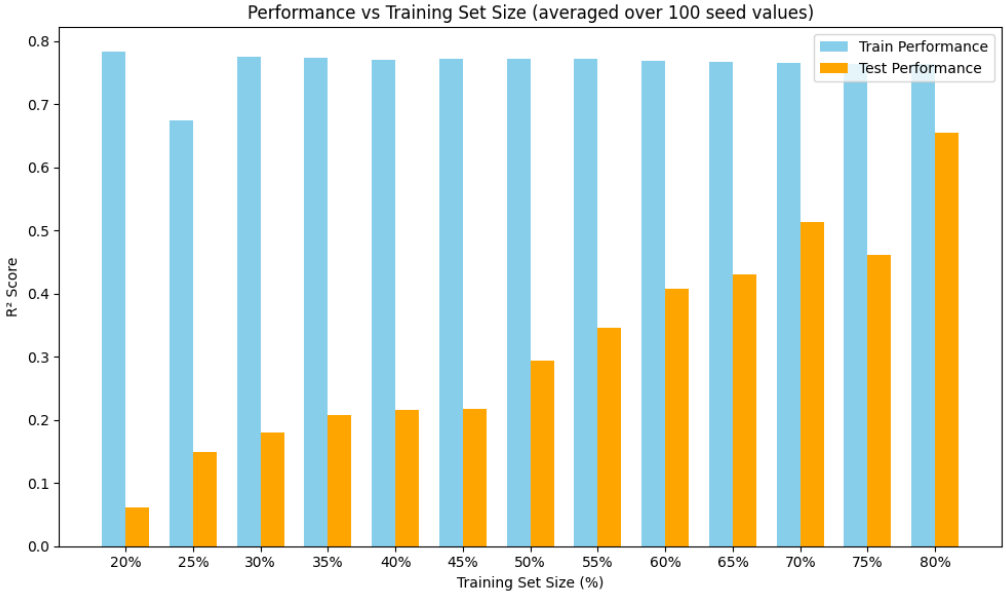


Figure 8: Effect of training size on performance for training and test data.

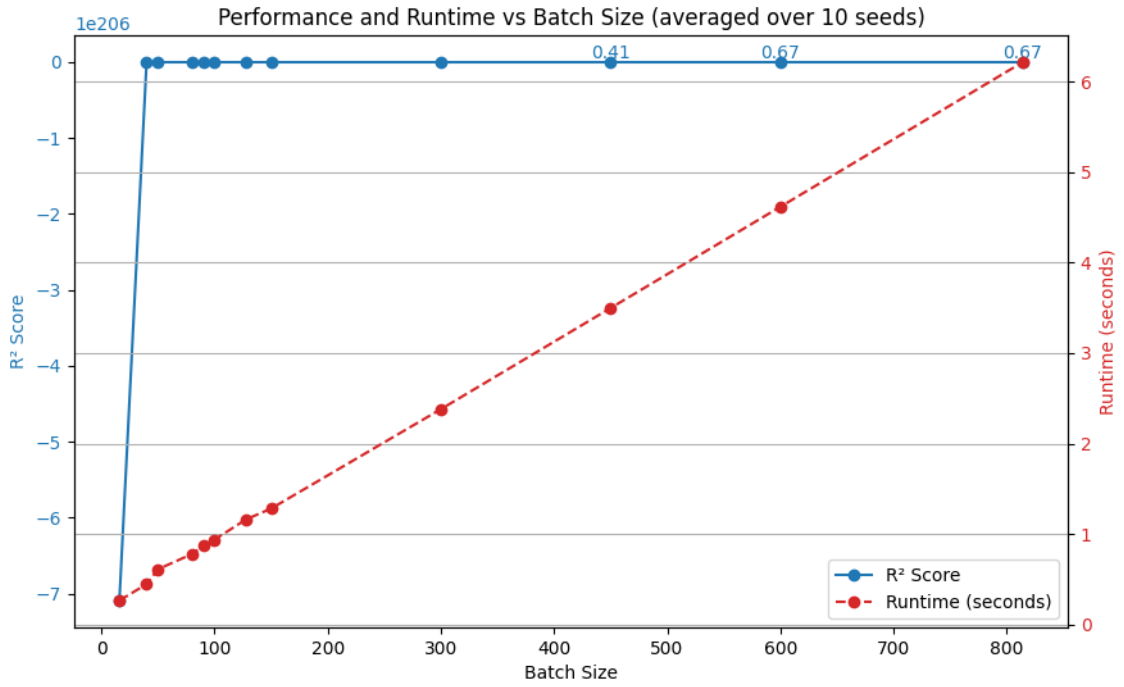


Figure 9: Performance and Runtime vs Batch Size (averaged over 10 seeds). The R^2 score and runtime are plotted against different batch sizes.

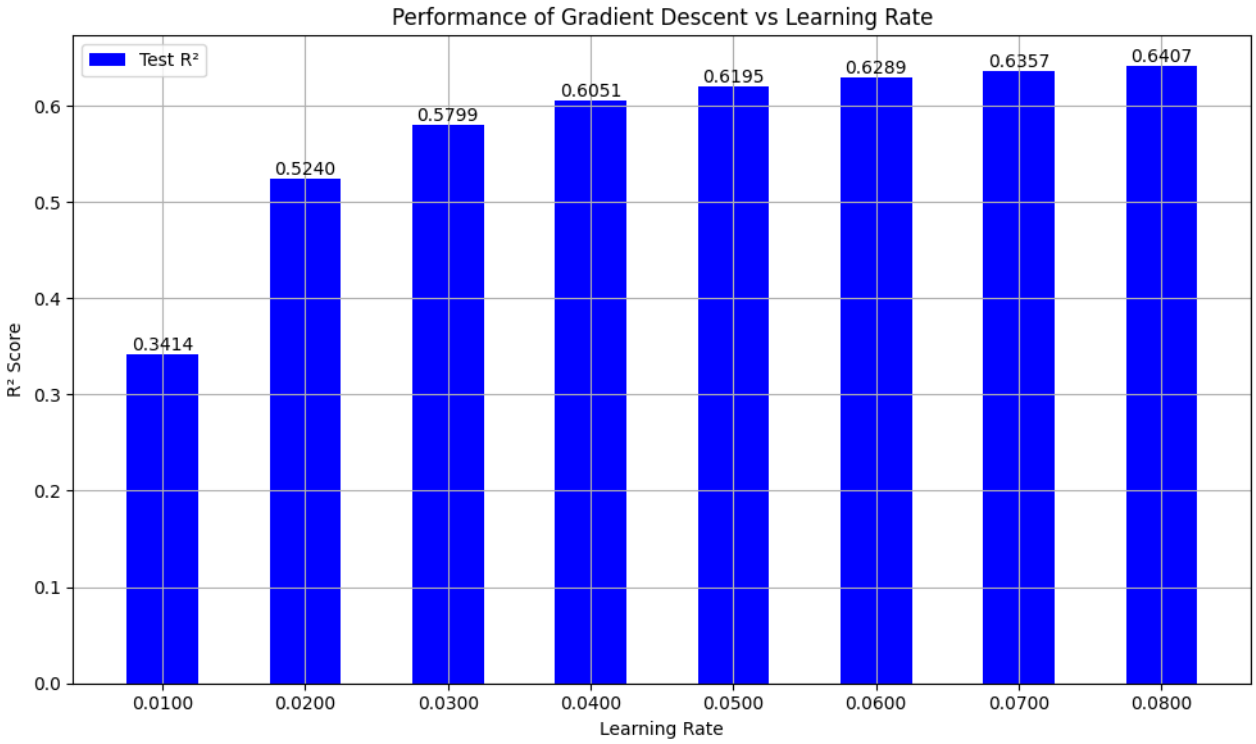


Figure 10: Performance of Gradient Descent vs Learning Rate.

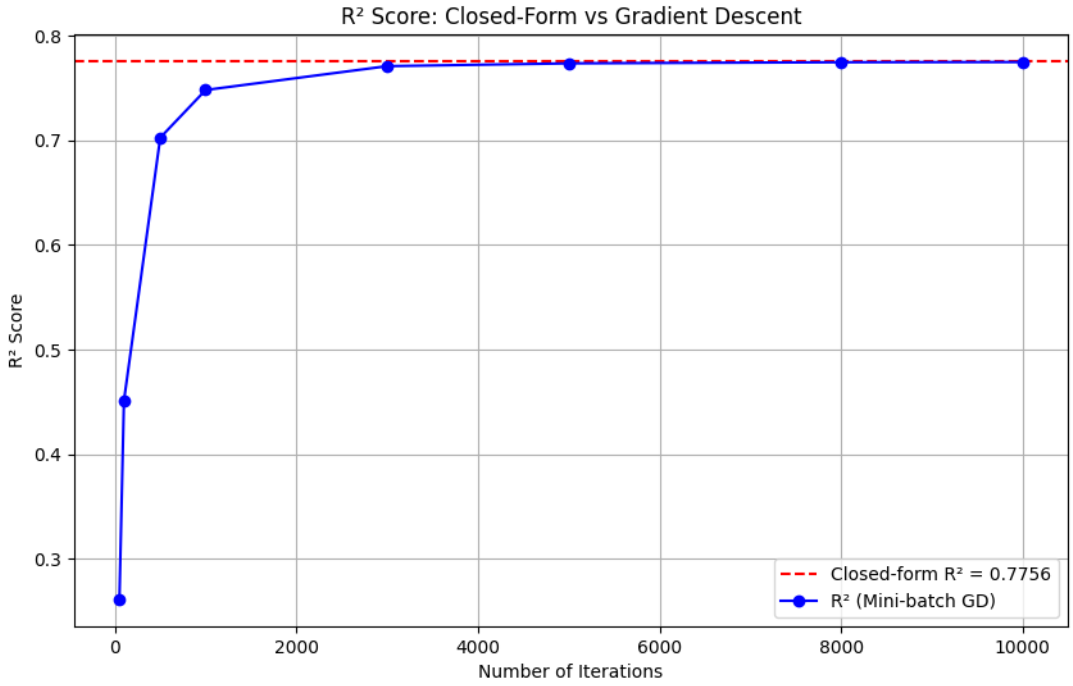


Figure 11: Comparison of Standard Linear Regression and Gradient Descent-based (mini-batch) Linear Regression. The performance of gradient descent performance is measured in terms of iterations, using an optimal learning rate $\alpha = 0.07$ and a batch size of 100.