COMP 551 Assignment 1: Linear and Logistic Regressions, Classification, and Gradient Descent on Infrared Thermography Temperature and CDC Diabetes Health Indicators Datasets

Group 44: Ayaz Ciplak - 261013785, Christopher Smith - 261013926, Jacqueline Silver - 261012002

Abstract

In this assignment, we explored methods for making both categorical and continuous predictions on raw datasets using basic machine learning models. The process involved full-scale implementation, ranging from data pre-processing to post-hoc model analysis.

At its core, this assignment involved the manual implementation of two widely-used models — linear regression and logistic regression. For the linear regression task, we used a dataset on patients subjected to infrared thermography to predict temperature measured in monitor mode (column aveOralM). For the logistic regression task, we worked with a dataset of patient health metrics to predict whether or not a patient would have diabetes. We also implemented different optimization techniques for both models, including the analytical (closed-form) solution and mini-batch stochastic gradient descent (SGD) for linear regression, as well as fully-batched and mini-batch SGD for logistic regression.

After executing all variations of our models on both datasets and performing various experiments on them, we gained lots of insight onto the behaviours of different models. These were mainly centered around the differences in training times and accuracies beween the same categories of models, but with varying implementations or hyperparameters.

Introduction

Despite the ongoing advancement of increasingly complex machine learning models, recent data suggests that linear and logistic regression remain the most commonly used algorithms (*Kaggle*). In this project, we implemented and evaluated these models using Google Colab, leveraging Python and widely used statistical libraries like Pandas, NumPy and Matplotlib.

The datasets utilized in this project were sourced from the UC Irvidne Machine Learning Repository, a well-established resource for machine learning datasets (*ICS*). The first dataset, related to infrared thermography temperature, primarily consisted of float64 numerical values with three features encoded categorically. The second dataset, which pertained to health metrics and diabetes, was entirely encoded as int64 data though most features were either categorical or binary in nature.

For ease of use and extensibility, we implemented analytical linear regression and fully-batched logistic regression models as base classes, and later extended them with subclasses that incorporated stochastic mini-batch gradient descent for parameter optimization. We rigorously evaluated the performance of these models through experiments that included varying train-test split ratios, comparing optimization methods within models of the same type, and testing different hyperparameter configurations. Additional experiments also examined the impact of feature data representation on model performance. Notably, we found that in both SGD-based models, specific mini-batch sizes outperformed their fully-batched counterparts in both accuracy and training time.

Datasets

The first dataset pertains to infrared thermography temperatures, while the second includes health metrics related to diabetes.

Preprocessing:

- **Dataset 1:** This dataset initially contained 2 null values, which were removed due to their insignificance relative to the total number of rows. Object-encoded data was eliminated, and two non-ordinal categorical variables, gender and ethnicity, were converted into binary true-false features. The ordinal-categorical variable, age, was encoded as integers from 1 to 6 after combining overlapping categories, such as "21-25", "26-30", and "21-30".
- **Dataset 2:** This dataset had no null values. Most features were binary, with some categorical and one continuous feature (BMI). All features were scaled to improve the optimization path for gradient descent and minimize bias toward higher-scale features.

Exploratory Analysis:

- **Dataset 1:** The target variable exhibits a normal, slightly right-skewed distribution centered around 37. Continuous feature variables largely followed a similar normal distribution, reflecting typical human body temperatures in Celsius. Their distributions can be found in Figure 1. Analysis of categorical features revealed a slightly higher number of female patients compared to males, along with a disproportionate representation of patients aged 18-30 and those identified as white.
- **Dataset 2:** The categorical target variable is unbalanced, with a significantly larger population of non-diabetic patients compared to those diagnosed with diabetes. The continuous feature, BMI, displayed a right-skewed normal distribution around 28. Disproportionate counts were also observed in binary categorical features, particularly in the columns for anyHealthCare, diffWalk, Stroke, NoDocbcCost, and HvyAlcoholConsumption.

Ethical Concerns: Although no names or addresses were included in either dataset, several ethical concerns arise with access to large datasets containing personal data. These include:

- **Re-identification risks:** Sufficient data points about a patient, even without direct identifiers, can potentially lead to tracing back to individuals.
- **Informed consent violations:** If a patient consented to share data for specific purposes, aggregating that data for different uses without their knowledge can breach ethical standards regarding informed consent.

Results

Dataset 1:

- **Performance (Analytical):** After training our analytical linear regression model on an 80-20 train-test data split, we got the following values: **Training set:** Mean Absolute Error (MAE): 0.19, Mean Squared Error (MSE): 0.06, R-Squared: 0.78. **Test set:** Mean Absolute Error (MAE): 0.21, Mean Squared Error (MSE): 0.07, R-Squared: 0.63

  According to our computed weights, the analytical linear regression model performs moderately well on both the training set and the test set.

  - **Training set:** We achieve reasonably good values for MAE, MSE and RMSE, explained by the fact that our model immediately finds the optimal weight set for our dataset. Our R-squared value is also good – It indicates that our model explains 78 percent of the variance in the training set.
  - **Test set:** All measures perform fine on the test set, but are still uniformly worse than the measures for the training set. Since we have not implemented any sort of regularization or other prevention against overfitting, this makes sense - The model is designed to fit most optimally towards the information it has seen (training set).

- **Weights Analysis (Analytical):** Figure 3 illustrates the weights that were calculated using the analytical linear regression model on dataset 1. Note that the relative magnitude of weights is relatively low - Barely any weights exceed an absolute value of 1. This can be explained in part by the fact that most features had very similar distributions, and none had extremely large or small average values.

  - **High-Influence Weights:** Factors like TRCMax1, TLC1, TLCMax1 and TRC1 have relatively high absolute weight values. This indicates that they have more effect/influence on the value of our target column AveOralM. Columns with higher (positive) non-absolute values indicate that an increase in the feature values lead to an increase in the target value. Columns with lower (negative) non-absolute values indicate the opposite.
  - **Low-Influence Wights:** Features like Humidity and Distance have almost negligible weight values. This indicates that they contribute very little to the prediction. The feature values may be irrelevant or redundant to the behaviour of our target value AveOralM.

- **Train-Test Split Analysis:** Figure 3 illustrates the performance of the linear analyical regression model (MSE) on the train and test sets as a function of train-test split proportions. We identify a few trends that have manifested in our data:

  - **Initial high test set error and low training set error:** This can be explained by the fact that the model is overfitting to a small random subset of our data. Since we have such limited data in this step, the model does very well at predicting this set but struggles to make accurate predictions on the unseen data.
  - **Gradual increase in training error:** This suggests that the model is having trouble finding a perfect fit for larger and larger samples of training data. This isn't necessarily a bad thing, since it does not translate to any increase in test set error.
  - **Test error remains relatively stable:** Aside from the 20-80 test-train split, the error on the test set remains relatively stable between splits. This indicates that beyond a certain point, the model's ability to generalize to unseen data does not improve significantly. From our graph, we can observe that this point occurs around a 30-70 train-test split and onwards.

- **Mini-Batch Variation Analysis (SGD):** Figures 4 and 6 illustrate the model performance as a function of mini-batch size, while figures 5 and 6 demonstrate the effect of mini-batch size on training speed. We tested batch size values in powers of 2, from the 3rd power to the 9th. Below, we make some observations on the trends that occur within the data.
  - **Convergence Speed Trends:** The lower the batch size, the higher the execution time, even keeping max. iterations and learning rates constant. This could be due to a multitude of factors, notably:
    * Higher per-iteration overhead: Each weight update involves memory access and data transfer, which is an overhead incurred more often with more frequent weight updates for smaller batch sizes.
    * Higher batch-size models cna converge faster since they capture more of the dataset's trend per iteration

    Note: In the big-picture trend, there sometimes seems to be a point where this trend reverses - Mid-size batch SGD seems to run faster than full-batch. This can be explained by the fact that full-batch SGD requires computing the gradient over the entire dataset at each iteration, which can be computationally expensive.
  - **Model Performance Trends:** We can observe that very small batch sizes have a performance very similar to that of the full-batched SGD model initially. Batch sizes between 16-128 increasingly surpass the accuracy of the full-batched model for both the train and test sets, and this performance slowly trends towards that of the baseline as we increase batch sizes. This can be explained by a variety of factors.
    * Models with smaller batch sizes may be overfitting the data to some extent
    * We seem to reach an equilibrium between under and over-fitting the test data around a batch size of 16, where we achieve the lowest training error.
  - **Best Configuration:** For this model in specific, we believe the model with batch size 32 provides maximal optimality. It completes training in much less time than the model with batch size of 8, while also having the minimal test set error and a relatively small training set error.

- **Learning Rate Variation Analysis (SGD):** We analysed the effects of learning rate variation by keeping all other hyper-parameters constant and testing our SGD model with LR values ranging from 0.0001 t0 5e-07. Figure 7 illustrates the relation between learning rate and performance (MSE), as well as its relation with training time.
  - **Performance vs. Learning Rate:** There is a clear trend between an increase in learning rate and an improvement in the model's performance. This can be explained by the fact that hyperparameters like batch size and max iterations are being kept constant, while an increase in learning rate lets the model converge more quickly at each iteration. This means that within the set amount of iterations, models with a higher learning rate are able to get closer and closer to the optimal weight value. *(Note: This only holds for learning rates up to a certain value, since learning rates that are too large run the risk of diverging.)*
  - **Training time vs. Learning Rate:** There is no clear trend between an increase in learning rate and a change in the amount of time taken to train the model. This can be explained by the fact that a change in learning rate doesn't significantly change the computational complexity at each iteration of the model. Since all other hyperparameters are kept constant, and a change in learning rate still requires the same operations at each iteration, it is logical that there is a constant relation between the learning rate and the training time.

- **Optimization Method Comparison - Analytical vs. SGD:** To test both of these models fairly, we used the standard analytical linear regression model and the optimal SGD model found in the mini-batch variation analysis (32 batches). The training time results were 0.0109s for the analytical model, vs. 6.1s for the SGD model. Performance-wise, the analytical model achieved MSE values of 0.060 and 0.073 on the train and test sets respectively, while the SGD model achieved results of 0.119 and 0.146 respectively. Comparing these two models highlights the advantage of being able to solve directly for the optimal solution during linear regression:
  - **Significantly lower training time for Analytical Linear Regression:** The analytical model uses matrix computation to directly compute the direct weight set, giving it an optimal solution in very little time. It does not have to go through thousands of iterations to optimize the weight set, like the SGD model does.
  - **Lower error for both train and test data for Analytical Linear Regression:** The Analytical Linear Regression model already solved for the optimal weight set, whereas we "interrupt" the SGD model from its optimizations after 10 000 iterations. Therefore, the Analytical model has already found its most optimal solution on the training set, reflcted as lower MSE values in both the train and test sets.

  We also observe an interesting difference in the weights computed by each model. Both models behave in relatively similar ways when considering the sign of the weights — Most positively-weighted parameters in the analytical model are also positively-weighted in the SGD model, and vice-versa. However, we notice a significant difference in the magnitude of the weights - The SGD model's weighs are of a much higher magnitude than those of the Analytical model. This could be the result of the SGD model not having enough iterations to fully converge to an optimal weight set.

- **Extra - Data Representation Variation Analysis:** In this subsection, we attempted to analyse whether the representation of the feature data had any effects on the model performance in both Analytical Linear Regression and SGD Linear Regression. We compared training times and performance for both models on our initial, preprocessed dataset 1, as well as another variation of dataset 1 that entirely uses inefficient, float64 types for data representation.

  On the regular dataset 1, the analytical and SGD models trained on an 80-20 train-test split in 0.002 seconds and 6.25 seconds respectively. They had (train, test) MSEs of (0.06, 0.07) and (0.12, 0.15) respectively. On the augmented dataset 1, the analytical and SGD models trained on an 80-20 train-test split in 0.01 seconds and 6.85 seconds respectively. They had (train, test) MSEs of (0.06, 0.07) and (0.13, 0.15) respectively.

  - **Training time:** We do observe a slightly higher training time for both models when using the more precise dataset with larger representations. This can be attributed to the fact that larger types can occupy more space in RAM during data processing, leading to higher training times. However, we do believe this value would be amplified for larger, more RAM-consuming datasets.
  - **Accuracy:** As predicted, there is negligible effect on the accuracy of each model between the precise and less-precise data representations, since the underlying data stored in each is identical.

Dataset 2:

- **Performance (Full-Batch):** To gain a comprehensive understanding of the performance of the Logistic Regression model on dataset 2, we used the following five popular performance metrics for classification tasks: accuracy score, precision, recall, F1 score, and a confusion matrix, as seen in the table in Figure 9.

  The accuracy score serves as a basic measure of the model's correctness, with a score of 0.86 indicating that our model is relatively successful in predicting output overall. However, when we take a more nuanced approach by examining precision and recall, we observe that the model struggles with classifying minority cases. A precision score of 0.53 suggests that when predicting the positive presence of diabetes, the model is correct just over half of the time. The recall score of 0.15 indicates that the model identifies only a small portion of the true positive cases, highlighting a real risk of false negatives. The confusion matrix in Figure 10 provides a breakdown in absolute terms of these scores. This is further reflected in the low F1 score of 0.23, which serves as a more comprehensive measure of performance — especially for imbalanced datasets like this one. This more nuanced approach reveals, that overall, this model's performance is limited.

- **Weights Analysis (Full-Batch):** Figure 11 in the Appendix display the feature weights from the Full-Batch Logistic Regression model. They indicate that higher BMI, age, high blood pressure, and especially general health are strongly associated with an increased likelihood of diabetes, while heavy alcohol consumption and certain lifestyle factors, like smoking and physical activity, have a negative but minor influence. The negative bias term suggests that, in the absence of other features, the model predicts a lower probability of diabetes. Again, this is due to the imbalanced nature of the dataset.

  Overall, the presence of several significant positive weights associated with features that are generally linked to an increased likelihood of diabetes (e.g. general health, BMI) implies that the model has some validity to it.

- **Train-Test Split Analysis (Full-Batch):** Figures 12 and 13 depict the trends seen in testing out various train-test splits. In this experiment, we sampled growing subsets of the training data, ranging from 20% to 80% of the dataset.

  - **Initial high performance, followed by gradual decline:** At first, the model overfits, learning specific patterns or noise in the small training set, which leads to lower accuracy and F1 scores for the test set. As the training set grows, the model becomes exposed to more surprising patterns, slightly reducing training accuracy and F1 scores.
  - **Decreasing Performance from 70% to 80% Training Size:** The smaller test set provides fewer samples to evaluate, making performance measurement less reliable. As a result, overfitting becomes more likely as the model learns very specific training data patterns, reducing generalization.
  - **Peak Performance around 60-70% Training Size:** At this size, the model has enough data to learn general patterns without overfitting. The test set remains large enough to reliably evaluate performance, leading to the best test accuracy and f1 scores. As the training set increases beyond 70%, test accuracy declines due to overfitting and smaller test sets
  - **Fluctuating F1 Scores:** Smaller test sets make precision-recall trade-offs less stable, particularly with more imbalanced proportions.

  Thus, somewhere between a 60-70% train-test split achieves the best balance between model learning and generalization.

- **Mini-Batch Variation Analysis (SGD):** Another experiment we ran was to compare full-batch gradient descent (keeping the original 80-20 train-test split) to our mini-batch Stochastic Gradient Descent (SGD) implementation in order to test the effects of different mini-batch sizes (e.g. 32, 64, 128, 256, 512, and 1024) on performance (measured by compute time and the more comprehensive F1 score), and to ultimately determine the higher performing model and method of optimization. The results of this experiment are illustrated in Figure 14 and 15. Certain trends we noticed were:

  - **Initial spike in F1 score as the batch size increases:** as the batch size increases from 32 to size 128, we notice a significant 20% increase in both the train and test f1 scores. This comes as no surprise as smaller batch sizes result in noisier updates to the weights during training and therefore a poorer performance. We have a peak at batch size 128, one of the only batch sizes to exceed the F1 score baseline comparison. This suggests that this batch size offers a good balance between update stability and convergence speed, and that therefore a batch size around this number would be optimal for overall model performance.
  - **Decreasing Training Time:** There is a significant decrease in training time between batch size 32 and 256. This is to be expected as larger batch sizes allow the model to process more data per iteration, reducing the total number of iterations needed to complete training, while smaller batch sizes result in higher training times due to the higher number of gradient updates. There is a slight rebound in training time past batch size 256, however this is most likely due to inefficiencies due to memory overhead.
  - **Full-Batch vs. Mini-batch Stochastic Gradient Descent (Baseline Comparison):** Overall, increasing the batch size leads to a faster training time, and, on average, higher performance. Higher performance can be achieved depending on the specific batch size. It is therefore a game of trial and error to determine what is ultimately the ideal batch size to optimize performance and training time.

- **Learning Rate Variation Analysis (SGD):** Finally, we wanted to explore the effects of various learning rates on the performance (as measured by the more comprehensive F1 score) of the mini-batch gradient descent logistic regression model (keeping the original 80-20 train-test split). The key takeaways from this test are illustrated in Figures 16 and 17:

  - **Performance vs. Learning Rate:** In learning rates below 0.1, we note incremental, but relatively stable improvements in performance. This initial consistency in performance is interrupted by a sharp drop between learning rate 0.5 and 0.8, thereby highlighting instability around the 0.2 to 0.8 range. However, performance seems to rebound to its peak when the learning rate is equal to 0.9.
  - **Training Time vs. Learning Rate:** AS expected, training time is highest at very low learning rates (e.g. 0.001) where smaller updates require more iterations to converge. As the learning rate increases, training time drops significantly and stabilizes around learning rate values of 0.05 to 0.5. Interestingly, there is slight rebound at a learning rate of 0.8, which is most likely due to instability and oscillations in the gradient updates.

  Thus, lower learning rates provide more stable performance but result in longer training times due to smaller incremental updates. As the learning rate increases, training time decreases, but the model becomes less stable, resulting in performance losses.

Creative Additions

In order to thoroughly understand our models and the data they were being trained on, we took several measures to go above and beyond the standard questions listed in the assignment.

- **Extra Experiment - Dataset 1:** When initially processing our datasets, we noticed that many features were not represented in their optimal forms - For example, dataset 2 had multiple binary features represented as float64 entries. In the 7th experiment of dataset 1, we go into detail on the effects of data representation on model training speed and performance.
- **Scaling Experimentation:** We decided to play around with scalers for our data, specifically for dataset 2. We experimented with different combinations of scaling data points and downcasting columns, and found that no downcasting + scaling all features in dataset 2 provided optimal training times.
- **Additional values - Learning Rates and Batch Sizes:** We were very interested in the trends batch size and learning rate variations had on the more extreme end. Our team decided to add many more measurement points for these hyperparameters in their respective analysis sections in order to get a better view of their behaviour.

Discussion and Conclusion

Overall, this project has served as an excellent introduction to machine learning. We have implemented our first models and explored the many factors (e.g. batch-size, learning rate) and trade offs (e.g. train-test split) that are at play in the development of these models in machine learning. From pre-processing the data and handling computational challenges, to fine-tuning hyperparameters like batch sizes and learning rates, and choosing performance metrics, we have gained hands-on experience with every step of the model development process. All of this has provided valuable insights into the complexities of machine learning, but also raised questions on whether there are different forms of gradient descent or other advanced optimization techniques, but also how can we optimize our models further to increase performance.

Statement of Contribution

We completed this assignment together collaboratively; Task 1 was focused on by Ayaz; Ayaz, Christopher, and Jacqueline worked on Task 2; Task 3 was focused on by Christopher and Ayaz, and the report was focused on by all team members.

Sources

- ICS: https://archive.ics.uci.edu/
- Kaggle: https://www.kaggle.com/c/kaggle-survey-2019

Appendix

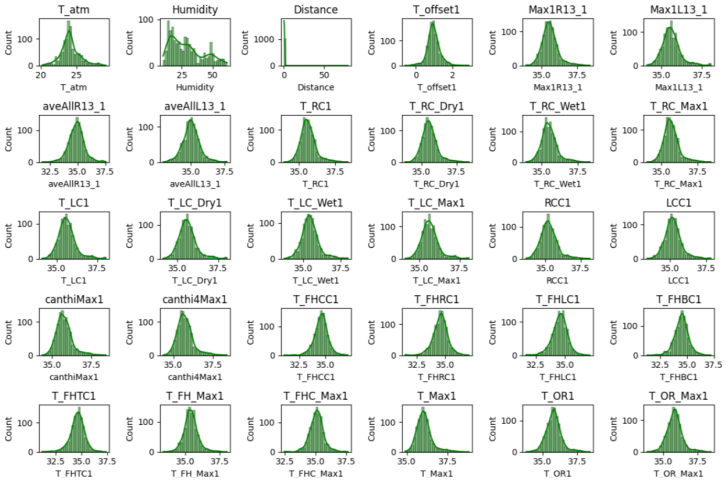

Figure 1: Continuous Feature Distributions - Dataset 1

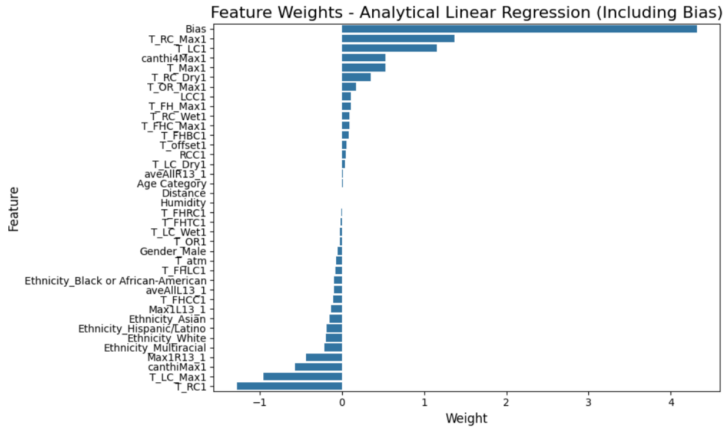

Figure 2: Analytical Linear Regression Weight Results

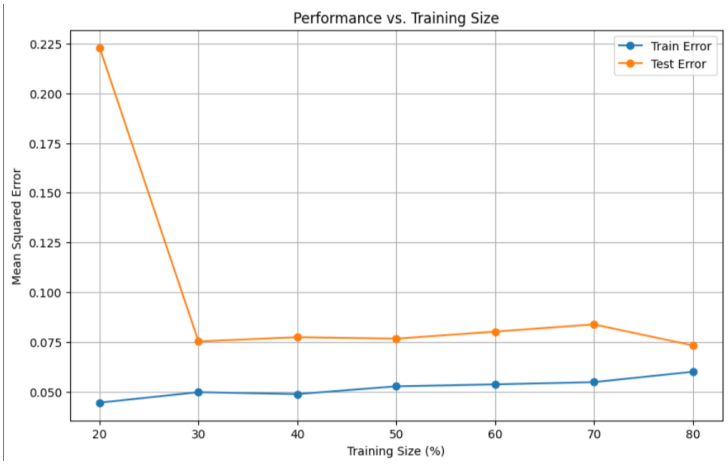

Figure 3: Train-Test Split Performance Comparisons - Analytical Linear Regression

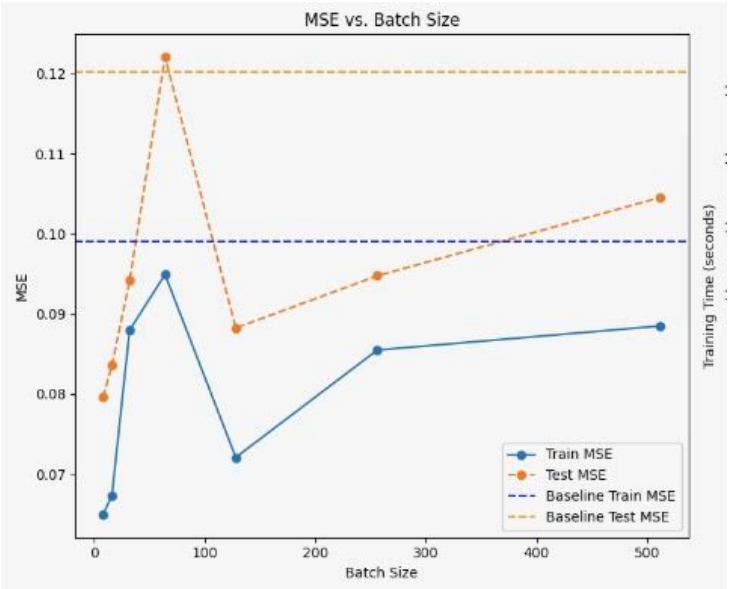

Figure 4: Batch Size Performance Comparisons - SGD Linear Regression

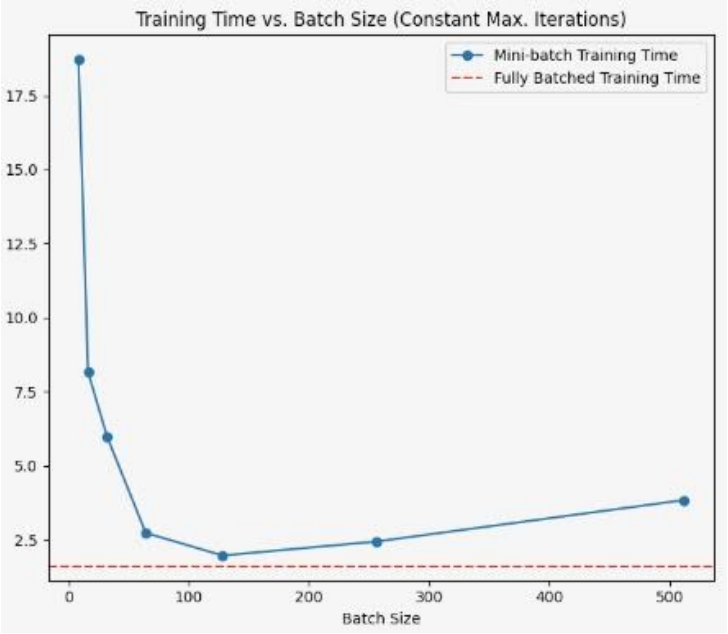

Figure 5: Batch Size Training Time Comparisons - SGD Linear Regression

```
Baseline MSE (train): 0.09907383599337377
Baseline MSE (test): 0.12026548349011965
Training time - Baseline (s):  1.5899968147277832
Batch Size: 8, Train MSE: 0.0649, Test MSE: 0.0796, Time: 18.7168 seconds
Batch Size: 16, Train MSE: 0.0673, Test MSE: 0.0836, Time: 8.1637 seconds
Batch Size: 32, Train MSE: 0.0879, Test MSE: 0.0941, Time: 5.9629 seconds
Batch Size: 64, Train MSE: 0.0949, Test MSE: 0.1221, Time: 2.7283 seconds
Batch Size: 128, Train MSE: 0.0721, Test MSE: 0.0882, Time: 1.9634 seconds
Batch Size: 256, Train MSE: 0.0855, Test MSE: 0.0947, Time: 2.4361 seconds
Batch Size: 512, Train MSE: 0.0885, Test MSE: 0.1045, Time: 3.8348 seconds
```

Figure 6: Batch Size Time and Performance - SGD Linear Regression

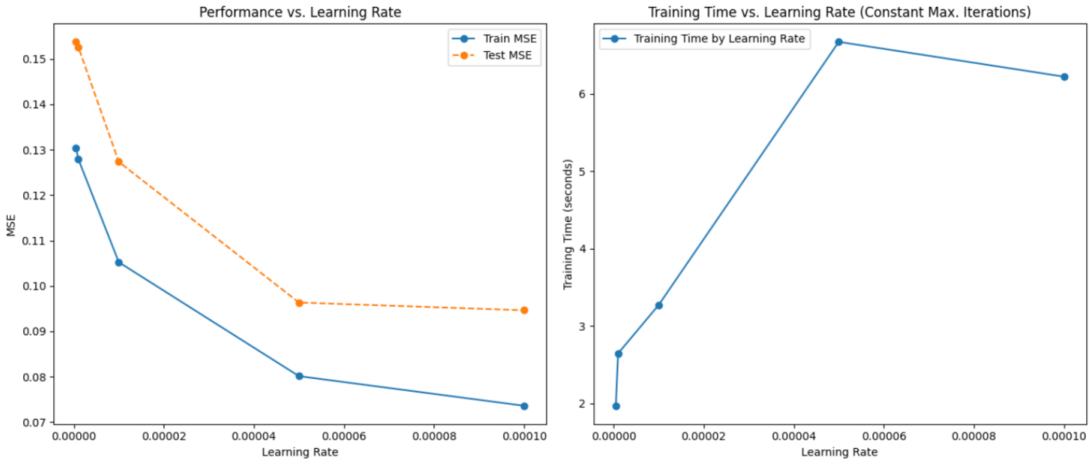

Figure 7: Learning Rate, Performance and Training Time Comparison - SGD Linear Regression

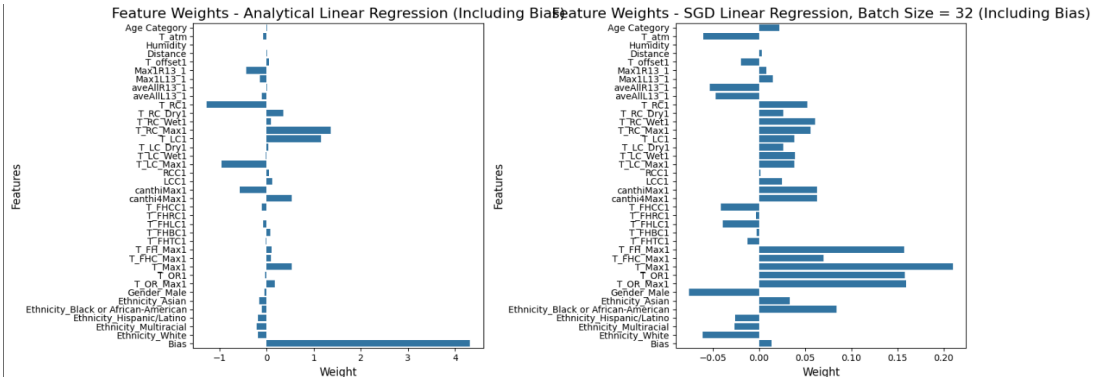

Figure 8: Analytical vs. Mini-Batch SGD Weights

Metric	Score	Description
Accuracy	0.86	The ratio of correctly predicted instances to the total instances (i.e. the model's overall performance in terms of correctness).
F1 Score	0.23	The harmonic mean of precision and recall (i.e. a balanced metric that considers both precision and recall scores, a better metric for overall performance when the dataset is imbalanced, like it is in this case).
Precision	0.53	The ratio of true positive predictions to total predicted positives (i.e. the model's ability to produce correct positive predictions and avoid false positives).
Recall	0.15	The ratio of true positive predictions to total actual positives (i.e. the model's effectiveness in identifying actual positive cases and avoiding false negatives).

Figure 9: Logistic Regression Model Performance Metrics

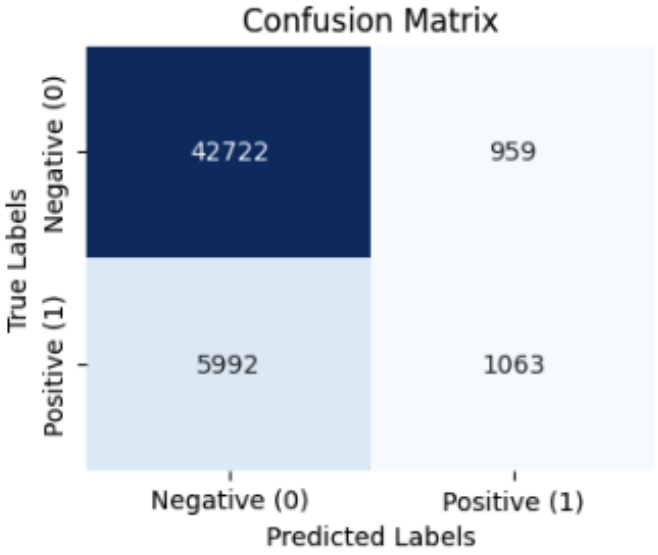

Figure 10: Confusion Matrix - Full-Batch Logistic Regression

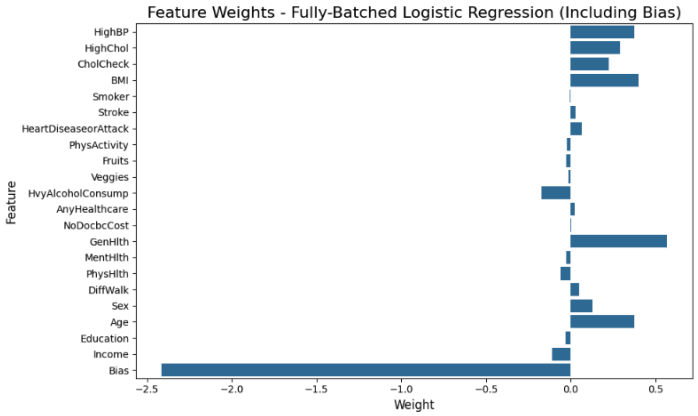

Figure 11: Dataset2: Weights Analysis - Full-Batch Logistic Regression

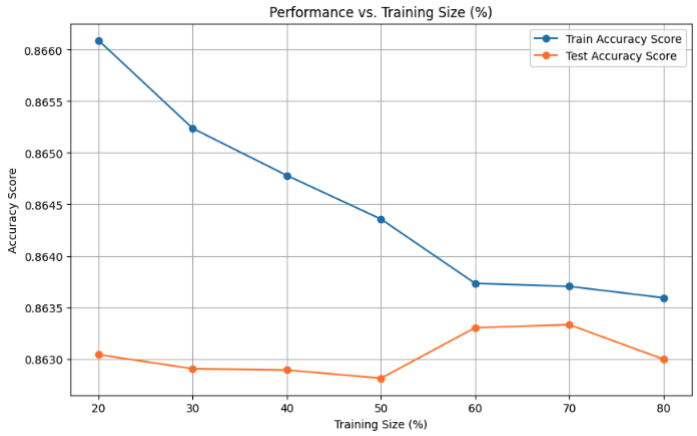

Figure 12: Train-Test Split Accuracy Score Comparison - Full Batch Logistic Regression

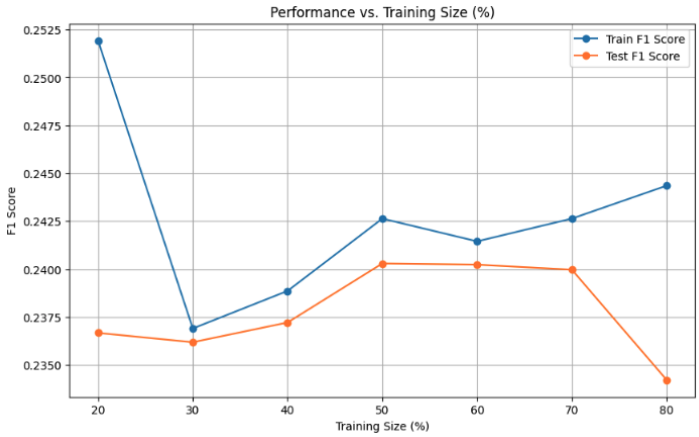

Figure 13: Train-Test Split F1 Score Comparison - Full Batch Logistic Regression

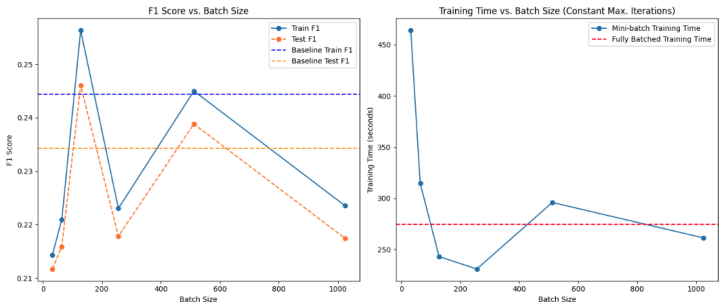

Figure 14: Batch Size Time and Performance Graph - Mini-batch SGD Logistic Regression

```
Baseline F1 (train): 0.2444
Baseline F1 (test): 0.2342
Training time - Baseline (s):  274.47154664993286
Batch Size: 32, Train F1: 0.2143, Test F1: 0.2117, Time: 463.9391 seconds
Batch Size: 64, Train F1: 0.2209, Test F1: 0.2159, Time: 314.4825 seconds
Batch Size: 128, Train F1: 0.2563, Test F1: 0.2460, Time: 243.0541 seconds
Batch Size: 256, Train F1: 0.2230, Test F1: 0.2178, Time: 230.9598 seconds
Batch Size: 512, Train F1: 0.2450, Test F1: 0.2388, Time: 295.9408 seconds
Batch Size: 1024, Train F1: 0.2236, Test F1: 0.2175, Time: 261.3483 seconds
```

Figure 15: Batch Size Time and Performance Values - Mini-batch SGD Logistic Regression

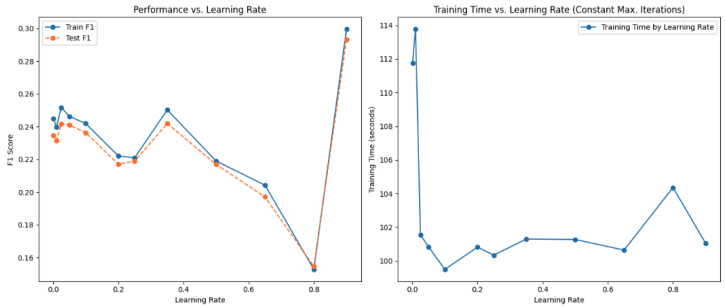

Figure 16: Learning Rate, Performance and Training Time Comparison Graph - Mini-batch SGD Logistic Regression

```
Learning rate: 0.001, Train F1 Score: 0.2449, Test F1 Score: 0.2348, Time: 111.7606 seconds
Learning rate: 0.01, Train F1 Score: 0.2398, Test F1 Score: 0.2315, Time: 113.7644 seconds
Learning rate: 0.025, Train F1 Score: 0.2517, Test F1 Score: 0.2417, Time: 101.5512 seconds
Learning rate: 0.05, Train F1 Score: 0.2463, Test F1 Score: 0.2410, Time: 100.8333 seconds
Learning rate: 0.1, Train F1 Score: 0.2419, Test F1 Score: 0.2364, Time: 99.4960 seconds
Learning rate: 0.2, Train F1 Score: 0.2221, Test F1 Score: 0.2172, Time: 100.8176 seconds
Learning rate: 0.25, Train F1 Score: 0.2209, Test F1 Score: 0.2189, Time: 100.3404 seconds
Learning rate: 0.35, Train F1 Score: 0.2503, Test F1 Score: 0.2419, Time: 101.3023 seconds
Learning rate: 0.5, Train F1 Score: 0.2190, Test F1 Score: 0.2169, Time: 101.2707 seconds
Learning rate: 0.65, Train F1 Score: 0.2043, Test F1 Score: 0.1971, Time: 100.6438 seconds
Learning rate: 0.8, Train F1 Score: 0.1528, Test F1 Score: 0.1545, Time: 104.3586 seconds
Learning rate: 0.9, Train F1 Score: 0.2995, Test F1 Score: 0.2932, Time: 101.0488 seconds
```

Figure 17: Learning Rate, Performance and Training Time Comparison Values - Mini-batch SGD Logistic Regression