CMPE 462

Machine Learning

Assignment 1 Report

Harun Reşid Ergen 2019400141

Meriç Keskin 2019400024

Github Repository:

MericKeskin/CmpE462-Spring24 (github.com)

Perceptron Learning Algorithm

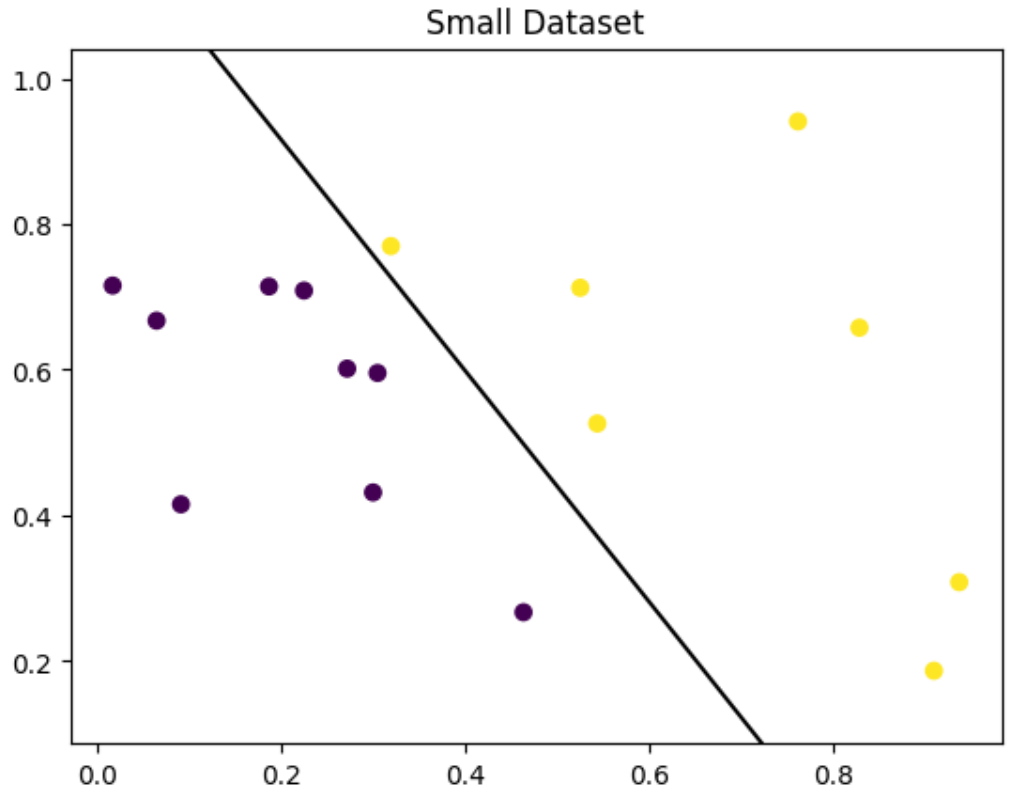
In this part of our project, we created a Perceptron Learning Algorithm to classify datasets of 2-D inputs and two categories. The perceptron is trained with given datasets.

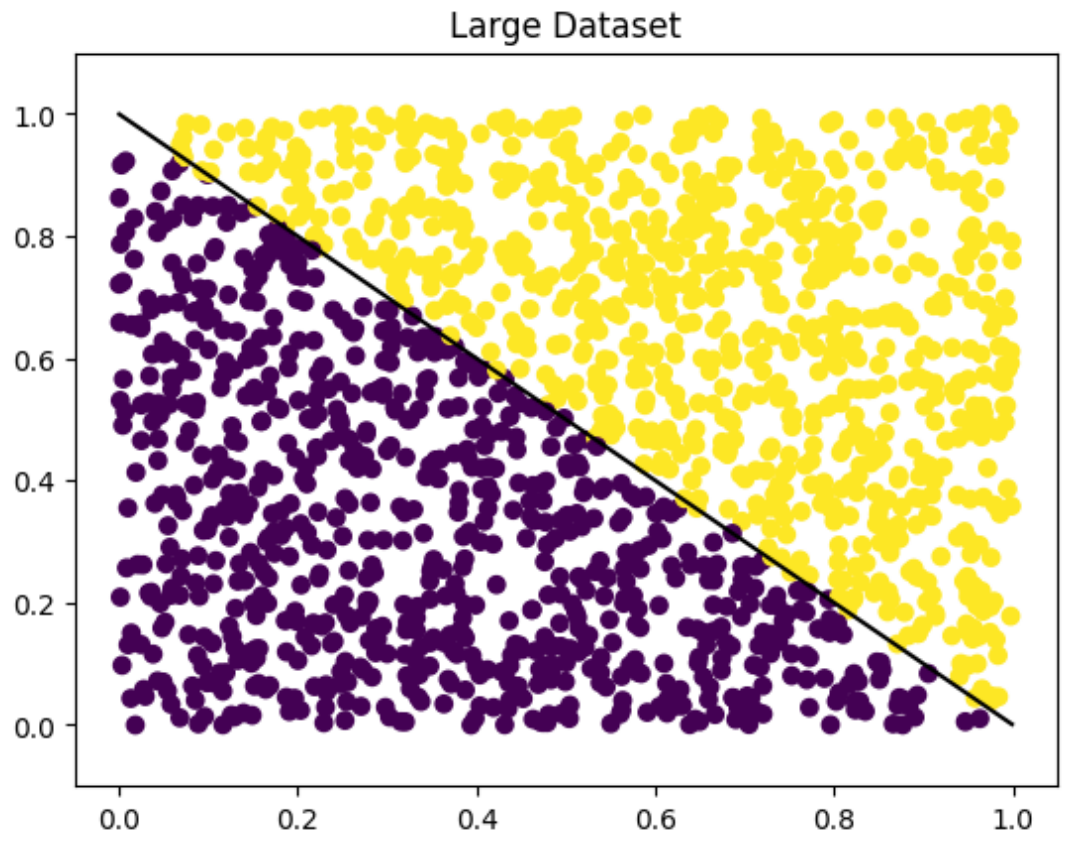
1) Comparison of the Number of Iterations

Between the small and large datasets, the differing number of inputs causes the number of iterations to differ either. The increase of the number of input data affects the number of iterations to grow generally.

The number of iterations also depends on the distribution of the training data, as it is easier or harder to distribute. Thus, the observed number of iterations may be higher in the large data in special cases. When tested, small data is converged in 20 iterations on average, whereas large data is converged in 240 iterations on average.

2) Plotting the Datasets and Decision Boundaries





3) Giving Different Initial Points

In our base algorithm, the initial weight vector is a zero vector. At the end of our main file for the Perceptron algorithm, the algorithm can be run with an initialize parameter for initializing the weight vector with random numbers in the given range repeatably, outputting varying final weight vectors each time. This occurrence of difference shows us that PLA is sensitive to initialization.

The effect of the initialization is caused because of the uncertainty of the decision boundary. Even when the classes are still, the separation of training dataset can create differences on decision boundary.

Also, the initialization of the weight vector can affect how the learning rate of the algorithm affects the direction of it. This effect is observable between the zero and nonzero initializing.

Logistic Regression

1) Dataset

This dataset is from a study where researchers took pictures of 3,810 rice grains. These grains come from two different rice species: Osmancik and Cammeo. The researchers processed the images to get 7 different measurements for each grain, and these are the features of the dataset: Area, Perimeter, Major Axis Length, Minor Axis Length, Eccentricity, Convex Area, and Extent.

Relationships Between Features

Some of these features are related. For example, a larger area usually means a longer perimeter. The major and minor axis lengths tell us about the grain's orientation and how wide it might be. If the rice grain is more stretched out, we'll see a longer major axis and a smaller minor axis. Eccentricity is connected to these lengths as well; the more elongated the grain, the higher the eccentricity. So it seems that the features are not completely independent of each other.

Possible Impact of Dependence on Logistic Regression

When we use logistic regression, we're trying to predict whether a grain is Osmancik or Cammeo based on these features. If some features give us overlapping information (like area and perimeter), it might influence the logistic regression model. This is because logistic regression weighs each feature's input when making a decision. If two features are saying the same thing, the model might think that's more important than it really is.

So, when preparing data for logistic regression, it's good to check if features are giving unique information. If they're not, we might need to pick the best ones or combine them in a way that makes sense for the model to learn the real differences between Osmancik and Cammeo rice grains. However, we observed that even if all features were used, we could get a good classification accuracy. But we wanted to make this point in our report anyways.

Normalization

	Mean	Std	Min	Max
Area	12667.728	1732.368	7551	18913
Perimeter	454.234	35.597	359.1	548.446
Major Axis L.	188.777	17.449	145.264	239.011
Minor Axis L.	86.314	5.730	59.532	107.542
Eccentricity	0.887	0.021	0.777	0.948
Convex Area	12952.497	1776.972	7723	19099
Extent	0.662	0.077	0.497	0.861

We have different kinds of measurements. Some are really big numbers that tell us about the size of the rice, and some are smaller numbers that describe the shape. If we put all these numbers into our logistic regression model as they are, the big numbers might throw everything off and cause overflow. The model might think that just because the number is bigger, it's more important, which isn't always true.

To fix this, we have two ways to adjust the numbers:

Standardization (Z-score): This method turns the numbers into something like a rating. If a rice grain's size is average, it gets a score around zero. If it's bigger than most, it gets a positive score. If it's smaller, it gets a negative score. This way, the model can better understand which rice grains are bigger or smaller compared to the average rice grain.

$$Z = \frac{(X - \mu)}{\sigma}$$

Min-Max Scaling: This is like turning the size of everything into a percentage. The smallest grain gets 0%, the biggest gets 100%, and all the others fall somewhere in between. This makes sure none of the measurements are too big or too small when we compare them.

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

We tried both. When Min-Max scaling scales the data between 0 and 1, adding regularization adds unnecessary loss to the model and does not increase accuracy. We shared our 5-fold cross-validation results for both methods. You can see that when we apply min max scaling, the best

result is obtained when lambda is 0. This shows that the model does not need regularization and underfit occurs. Hence, we continue with “standardization”.

2) Regularization parameter (5-fold cross-validation)

Regularization is crucial in preventing the model from overfitting, which occurs when the model learns the training data too well, leading to poor performance on unseen data.

To find the most effective lambda value, we employed a technique known as 5-fold cross-validation. This method involves dividing the dataset into five equal parts, then iteratively using four parts for training and one part for testing. This process is repeated five times, ensuring each part is used as a test set once. This approach provides a robust way to estimate the model's performance on unseen data.

We tested five different lambda values: 0, 0.0001, 0.001, 0.01, 0.1, and 1. These values range from allowing the model more freedom to fit the data to more restricting the model's complexity. The choice of lambda values was based on a balance between allowing the model enough flexibility to learn from the data and preventing it from overfitting.

Important note: The learning rate for our gradient descent process was set to 0.01 by choice, without doing any validation to determine its optimality. This decision was guided by common practices. It's important to acknowledge that this chosen learning rate may not be ideal for all phases of our project, especially as we transition to using stochastic gradient descent (SGD) for training. SGD introduces variability in the training process since it updates the model weights using only a single data point at a time. This can lead to fluctuations in the loss value trajectory towards the minimum. If the learning rate is too high in the context of SGD, these fluctuations could be exaggerated, preventing the model from converging on an optimal set of weights. In the cross-validation, only gradient descent is used, not stochastic! Same lambda value will be used for stochastic gradient descent for proper comparison between GD and SGD. Then, we could change the learning rate of SGD.

After applying 5-fold cross-validation to each lamda value, we observed the following accuracies on average:

Standardization	
λ	Accuracy
0	89.87%
0.0001	89.87%
0.001	89.95%
0.01	90.13%
0.1	90.16%
1	89.87%

Min-max Scaling	
λ	Accuracy
0	82.52%
0.0001	82.41%
0.001	81.31%
0.01	67.22%
0.1	11.21%
1	2.78%

For X normalized with standardization:

- The accuracy increases slightly as the lambda value goes from 0 to 0.1, peaking at 90.16% accuracy for $\lambda = 0.1$.
- The table indicates that the model benefits from a small amount of regularization. A lambda value of 0.1 provides the best balance between bias and variance.
- There is no significant overfitting detected since the highest accuracy is not at $\lambda = 0$, which implies that regularization slightly helps the model's generalization.

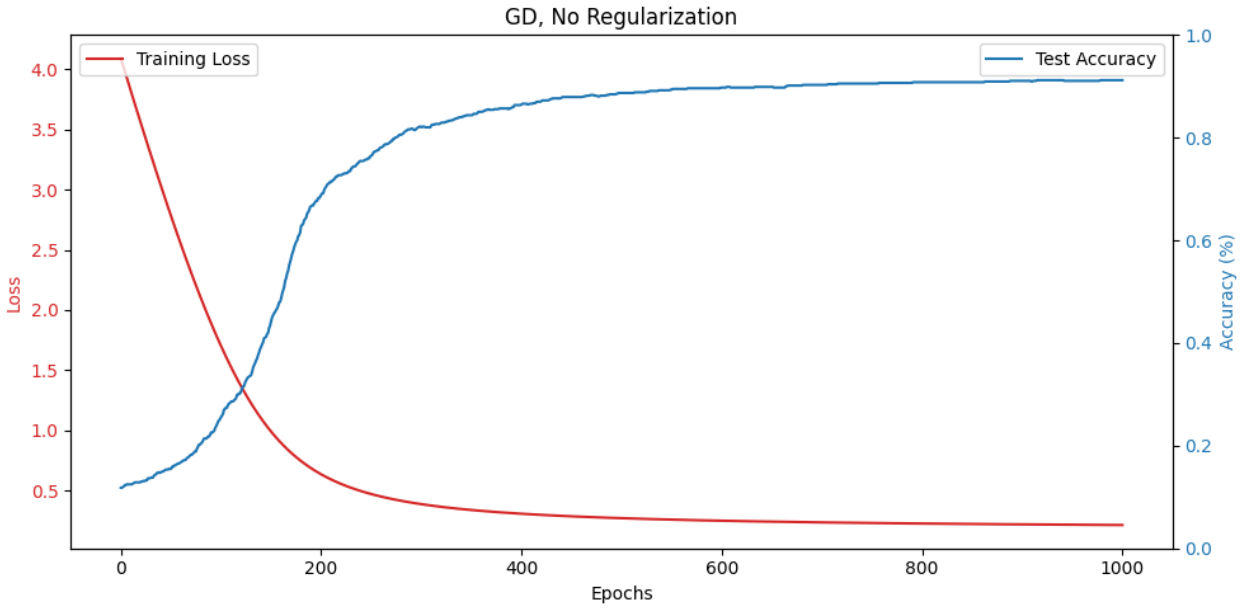
For X normalized with Min-Max Scaling:

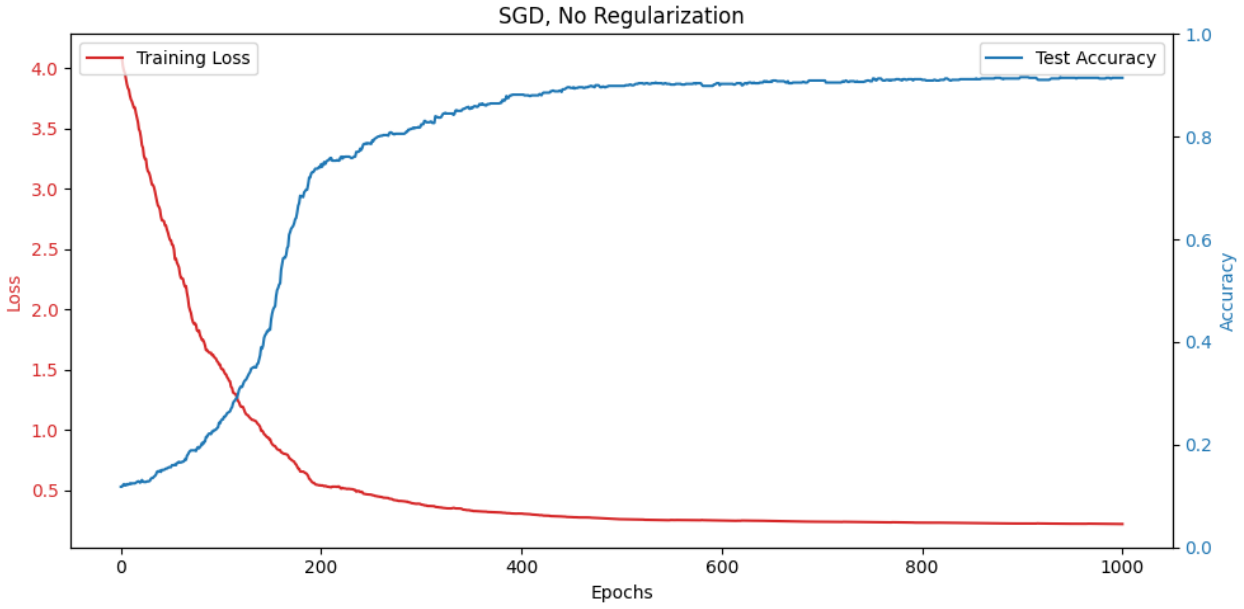
- A sharp decrease in accuracy is observed as lambda increases when using min-max scaling. The best result occurs at $\lambda = 0$, with an accuracy of 82.52%.
- It's clear that for min-max scaling, the regularization penalizes the model too much, as seen with the dramatic drop in accuracy, especially at higher lambda values ($\lambda = 0.01$, 0.1, and 1).
- The overall lower accuracies compared to standardization suggest that min-max scaling is not as effective for this dataset and model combination, and regularization does not improve the model performance.

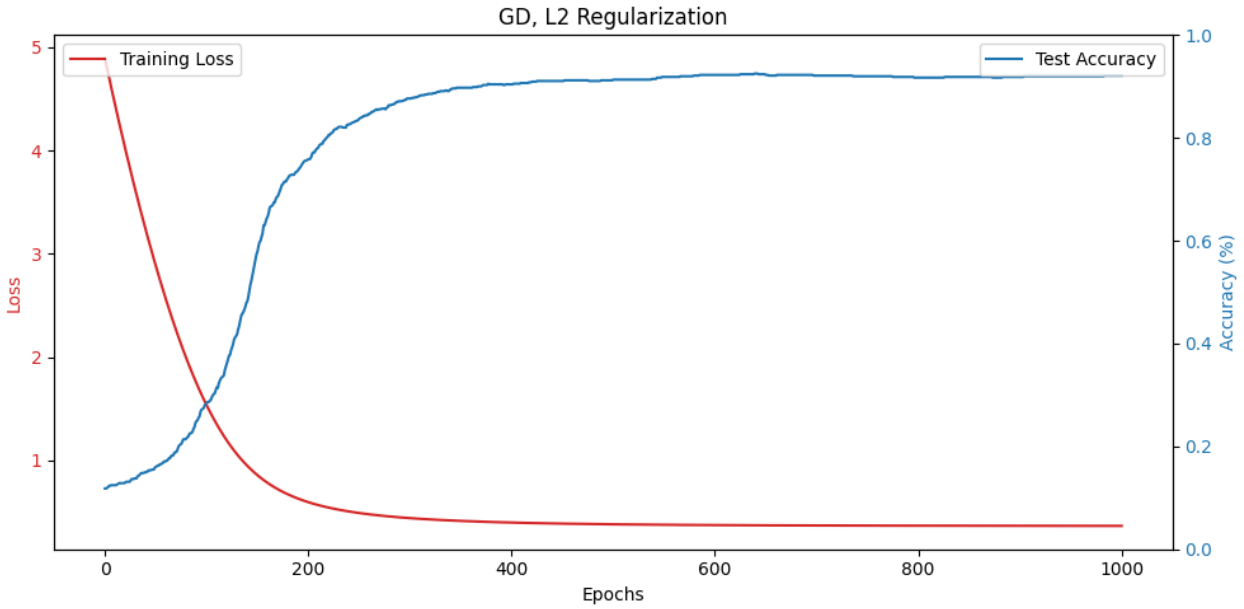
The results indicate that standardization is a more suitable scaling method for this dataset when used with logistic regression.

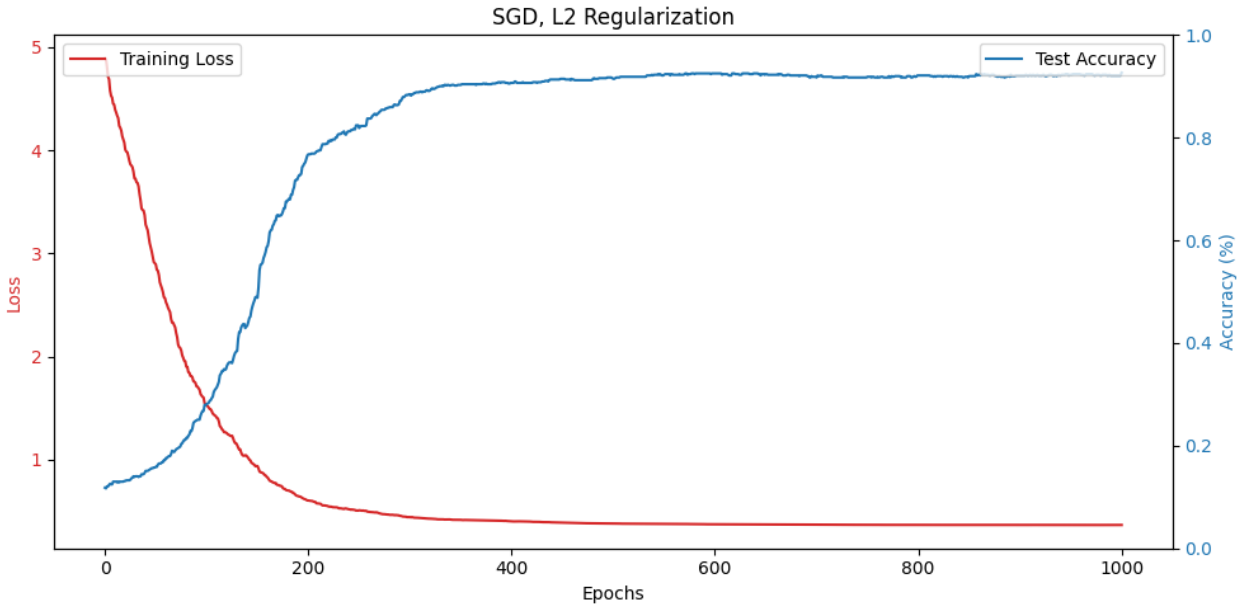
Given the learning rate was set without validation, the model's performance might be further optimized by tuning this hyperparameter, especially when SGD is applied

3) Training and Test Performance









	Training Acc.	Testing Acc.
GD	92.1916%	91.2073%
SGD	92.2244%	91.4698%
GD L2	92.6181%	92.1260%
SGD L2	92.8150%	92.6509%

GD and SGD without Regularization:

- GD: The training loss decreases sharply and converges smoothly. The test accuracy is relatively stable after the initial epochs. The final training accuracy is slightly higher than the testing accuracy, which simply reflects the variance within the testing set.
- SGD: The training loss decreases in a similar sharp manner but has some fluctuations, which is characteristic of SGD due to its updating weights using a single data point from training data. The final test accuracy is slightly higher compared to GD, suggesting that SGD may be converging to the optimum slightly more effectively in this case.

GD and SGD with L2 Regularization

- GD with L2: The introduction of L2 regularization to the GD model appears to very slightly enhance its generalization capability. Test accuracy increases from 91.20% to 92.12%. While the increase in accuracy is not significant, it indicates that regularization is beneficial, helping to improve the model's performance on the test set.
- SGD with L2: The introduction of L2 regularization to the SGD model provides a small uplift in test accuracy, with an improvement from 91.46% to 92.65%. This increment is not significant, however, the positive role of regularization can be observed.

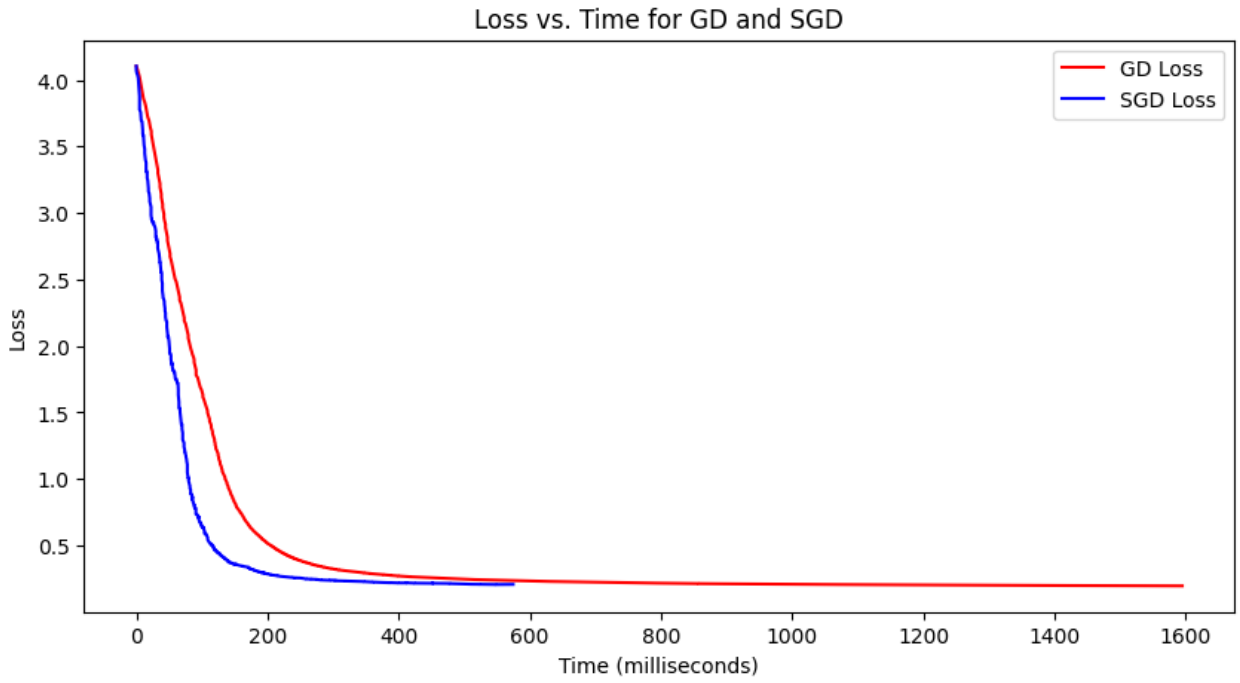
The increase in accuracy due to L2 regularization is relatively minor for both GD and SGD. However, the consistent improvement across both models suggests that the regularization is effectively refining the models' ability to generalize, even if the magnitude of the effect is small.

The fact that L2 regularization does not dramatically change the accuracies may imply that the base models without regularization were already performing quite well and not overfitting significantly to the training data.

4) Training Times of GD and SGD

We prepared a setup to compare training times of GD and SGD.

- We set a minimum loss improvement of 0.001.
- If the average loss improvement of last 200 epochs is smaller than 0.001, training stops (loss converges).
- At each epoch, time is measured.
- Approximately, SGD converges after 600 ms and GD converges after 1600 ms.



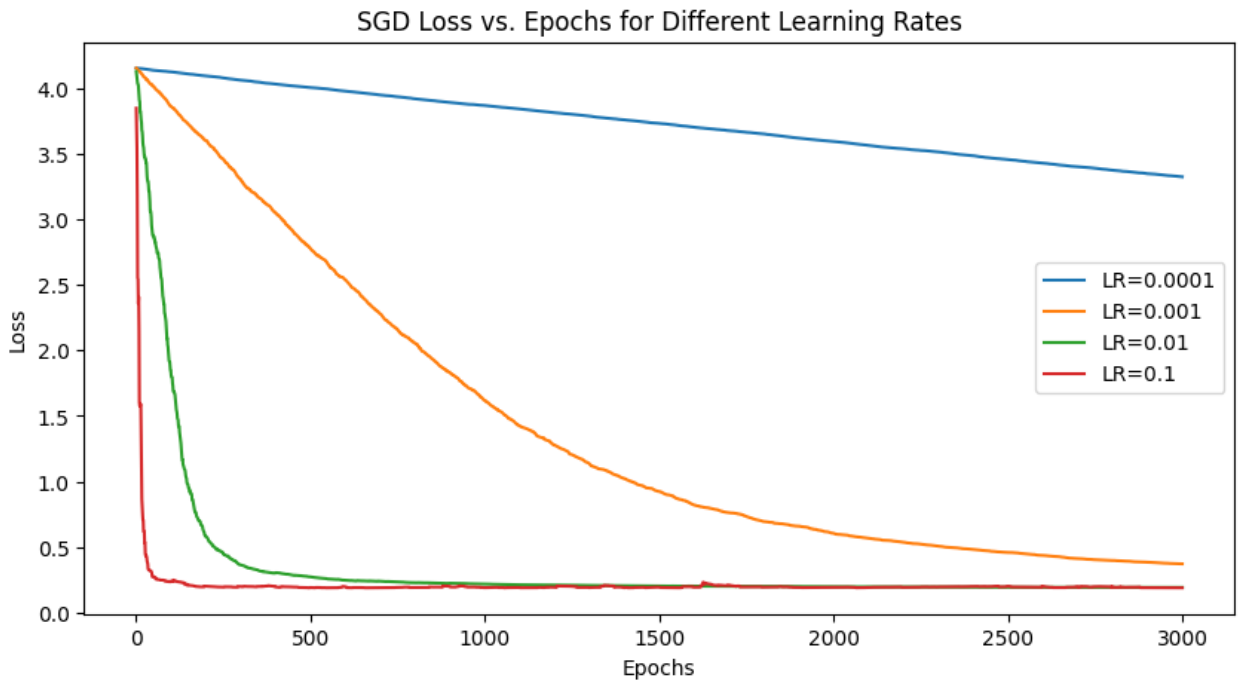
5) Step Size in SGD

The choice of initial learning rate for Stochastic Gradient Descent (SGD) is crucial since it can determine the efficiency and effectiveness of the entire training process. A rate that's too low leads to slow convergence, potentially requiring more time to reach the desired accuracy. Conversely, a rate that's too high may cause the model to overshoot the optimal solution, leading to instability or divergence in the training process.

We chose constant learning rates to keep things simple and straightforward. A constant learning rate means we take steps of the same size throughout the process. By doing this, we can clearly

see how different step sizes affect the model's learning process over time, as shown in our plot below.

Learning rates from 0.0001 to 0.1



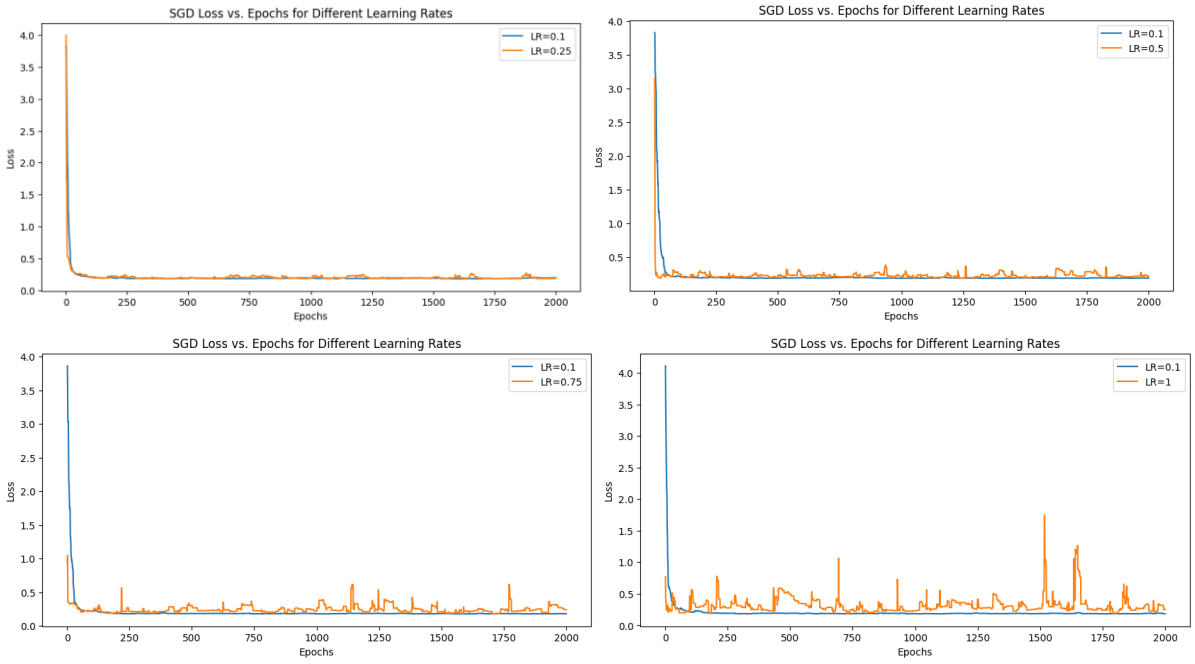
Blue Line (LR=0.0001): This model is taking tiny steps. It's gets better very slowly, but after many epochs of learning, it still hasn't reached as low a loss as the others. It definitely needs more time to converge since steps are too small.

Orange Line (LR=0.001): This one is taking slightly bigger steps. It's not as slow as the blue line and eventually gets to a low loss, but higher than the models with lower learning rates.

Green Line (LR=0.01): Here, the model is taking confident steps and learns quickly. It gets to a low loss faster than the blue and orange lines. It seems to find a good solution without wasting time, making it a strong choice for this task.

Red Line (LR=0.1): This one starts off strong, reducing loss very fast, but then it seems to hit a wall. It doesn't get much better after the initial drop. This might be because the steps are too big, and it's missing the best solution, or it's found the best solution it can, given its large step size.

Learning rates from 0.1 to 1



Further increasing learning rate from 0.1 makes fluctuations increase dramatically and convergence becomes difficult. Instead of seeing a faster convergence, we see models that constantly miss the optimal solution and stumble constantly.

Naïve Bayes

In this part of our project, we focused on creating and evaluating a Naive Bayes classifier. Our goal was to classify samples from a dataset to find out whether the presence or absence of breast cancer.

1) Training and Test Accuracies

Trained with 455 samples and tested with 114 samples. (Additionally, under the comparison with logistic regression part, we trained 10,000 times and stored the accuracies for each run. Then, we reported the minimum and maximum accuracies. This part only includes 1 training).

Training accuracy: 93.63%

Test accuracy: 95.61%

2) Compare the number of parameters

In Naive Bayes, the conditional independence assumption simplifies the model by assuming that all features are independent of each other given the class label.

In the dataset, there are 2 classes and 30 features having continuous values.

With Conditional Independence

$$P(X_1, X_2, \dots, X_d \mid Y = c) = P(X_1 \mid Y = c) \cdot P(X_2 \mid Y = c) \cdot \dots \cdot P(X_d \mid Y = c)$$

For each feature, values are continuous and assumed to follow a normal distribution. We need to estimate two parameters: the mean (μ) and the variance (σ^2) for each class.

30 features × 2 parameters per feature × 2 classes = 120 parameters

Without Conditional Independence

Without the conditional independence assumption, we have to consider the joint distribution of all the features given the class label.

$$P(X_1, X_2, \dots, X_d \mid Y = c) \neq P(X_1 \mid Y = c) \cdot P(X_2 \mid Y = c) \cdot \dots \cdot P(X_d \mid Y = c)$$

The means for each feature for each class:

30 × 2 = 60 parameters to estimate

The covariances and variances for each class:

	Feature 1	Feature 2	Feature 3	…	Feature d
Feature 1	$Var(F1)$	$Cov(F1, F2)$	$Cov(F1, F3)$	…	$Cov(F1, Fd)$
Feature 2	$Cov(F_2, F_1)$	$Var(F_2)$	$Cov(F_2, F_3)$	…	$Cov(F_2, F_d)$
Feature 3	$Cov(F_3, F_1)$	$Cov(F_3, F_2)$	$Var(F_3)$	…	$Cov(F_3, F_d)$
…	…	…	…	…	…
Feature d	$Cov(F_d, F_1)$	$Cov(F_d, F_2)$	$Cov(F_d, F_3)$	…	$Var(F_d)$

The covariance matrix shows how many variances and covariances we need to estimate. Since $Cov(F_a, F_b)$ and $Cov(F_b, F_a)$ are the same, we need to calculate unique ones. It is calculated as $\frac{d \times (d+1)}{2}$. So, $\frac{30 \times (30+1)}{2} = 465$ unique variances and covariances per class.

There are 2 classes. $465 \times 2 = 930$

60 parameters from means, 930 parameters from variances and covariances:

$60 + 930 = 960$ parameters to estimate.

3) Compare Logistic Regression with Naïve Bayes

We explored and compared the performance of two classification methods: Naive Bayes and Logistic Regression. Our focus is on the Wisconsin Diagnostic Breast Cancer dataset. This dataset includes 569 samples with 30 input features each. We aim to evaluate how well each model performs and understand their respective classification accuracies.

The Naive Bayes classifier was run 10,000 times (re-trained each time) to gather a range of accuracy outcomes. This iterative process allowed us to get a broader understanding of the performance of the classifier.

Logistic Regression was trained using two optimization techniques, Gradient Descent (GD) and Stochastic Gradient Descent (SGD), with and without L2 regularization.

Naïve Bayes Performance

The Naive Bayes classifier outcome the following range of accuracies after 10,000 runs:

Minimum Test Accuracy: 83.33%

Maximum Test Accuracy: 100.00%

Minimum Train Accuracy: 91.87%

Maximum Train Accuracy: 96.04%

Logistic Regression Performance

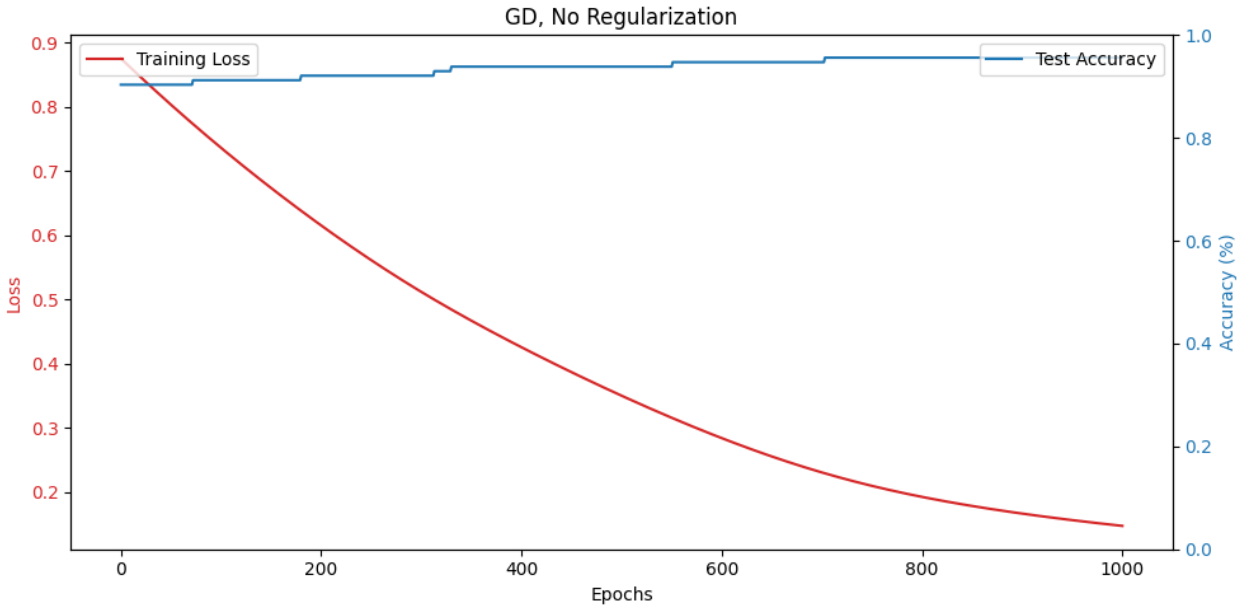
	Test Acc.	Train Acc.
GD	95.61%	94.51%
SGD	95.38%	95.61%
GD L2	98.25%	97.14%
SGD L2	96.49%	96.26%

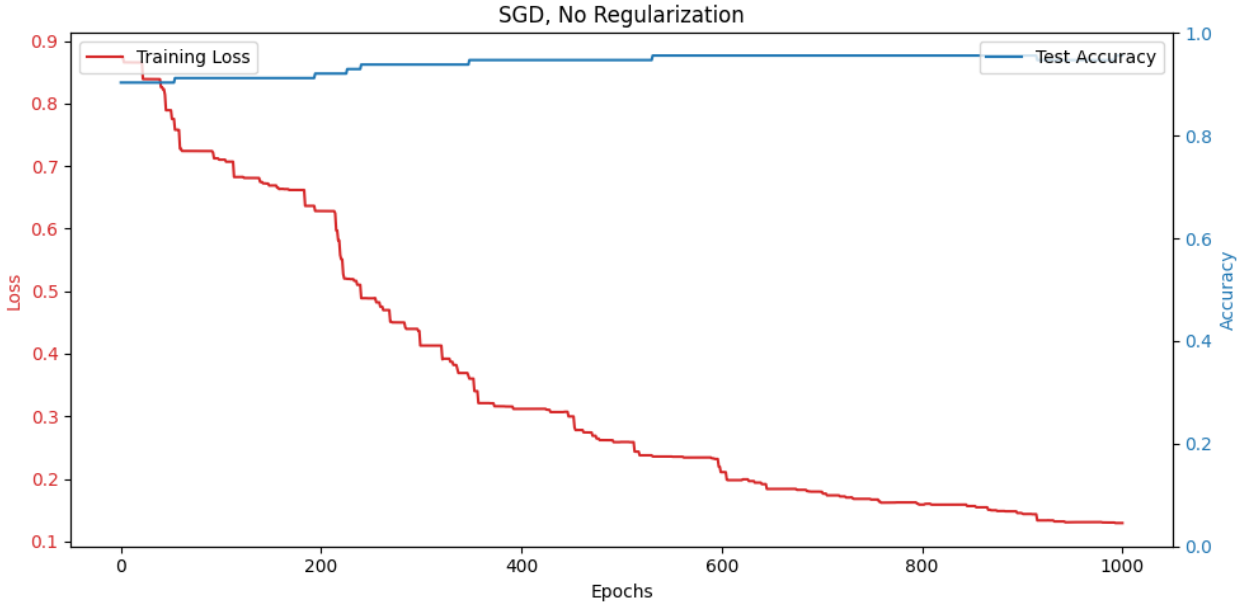
Both the Naive Bayes and Logistic Regression models have demonstrated high performance on the dataset. The variation in Naive Bayes' test accuracy can be attributed to its underlying assumption of feature independence. In real life, this is not realistic and not always true. Especially in health metrics, these values can be possibly dependent given the diagnosis result. It can still give accurate predictions when the features are conditionally independent given the class.

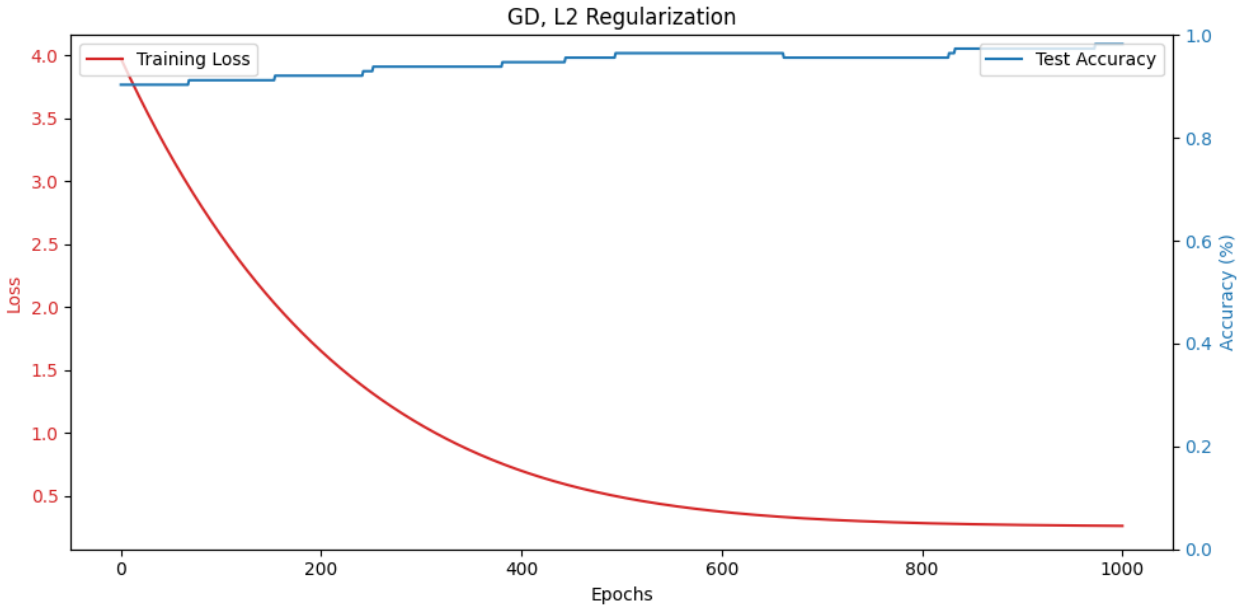
Logistic Regression, especially with L2 regularization, generally showed high consistency in performance. This can be due to its ability to model the relationships between features without assuming their independence. Regularization helps in preventing overfitting, which is crucial given the high dimensionality of the dataset.

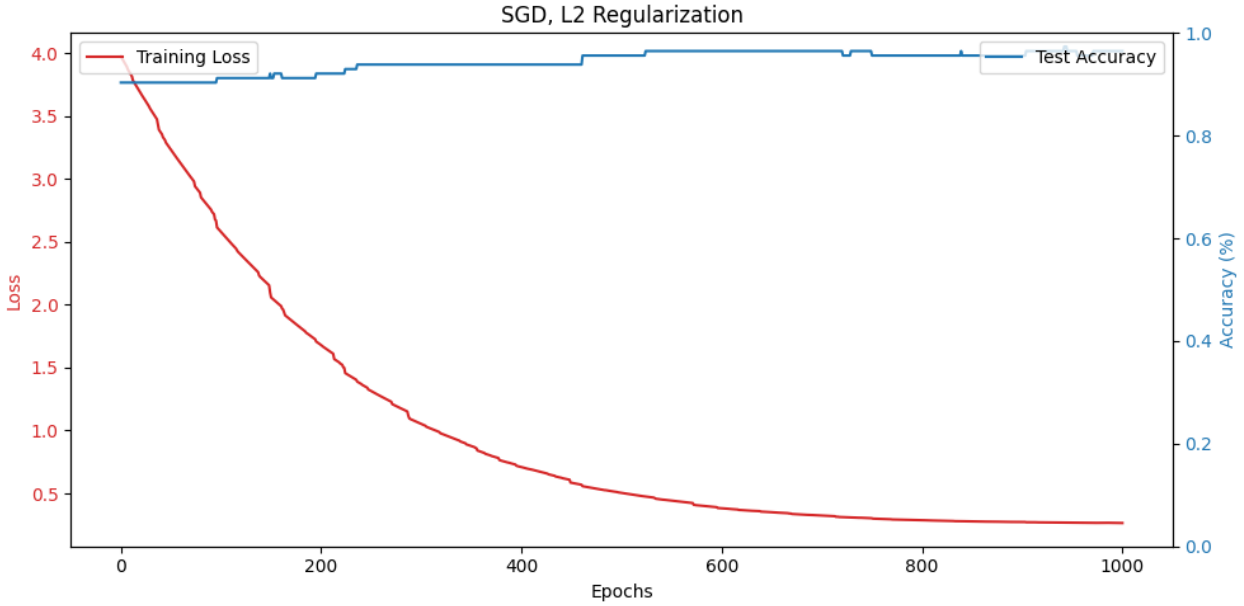
Considering the observed ranges and consistent outputs, neither classifier distinctly outperforms the other. Logistic Regression showed high accuracies. Naïve Bayes reached a perfect test accuracy (100%) in some runs. These outcomes underscore that the performance of a model can greatly depend on the dataset characteristics.

Both Naïve Bayes and Logistic Regression have their own good sides and can be well-suited for different scenarios within the same dataset. Naive Bayes can perform exceptionally well under certain conditions despite its simplicity. Logistic Regression offers robustness and consistency, particularly when enhanced with regularization if there is a overfit risk. We do not see one model outperforms the other.


GD, No Regularization
Training Loss
Test Accuracy
Loss
Accuracy (%)
Epochs
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000


SGD, No Regularization
Training Loss
Test Accuracy
Loss
Accuracy
Epochs
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000


GD, L2 Regularization
Training Loss
Test Accuracy
Loss
Accuracy (%)
Epochs
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000


SGD, L2 Regularization
Training Loss
Test Accuracy
Loss
Accuracy (%)
Epochs
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000