# B1 Project: Optimisation

John Lee

## Introduction

This is the report for B1 Engineering Computation - Project B: Optimisation for regression and classification models. The project investigates how to apply different optimisation methods for learning optimal parameters of a model that can predict a value of interest for a given input data point. A total of 6 tasks were given and completed using the MATLAB programming language with the「Statistics and Machine Learning Toolbox version 23.2」and the「Optimization Toolbox version 23.2」. Results shown in this report are generated using `rng(12345)` unless otherwise stated.

## Content

# 1. Task 1: Linear Regression via analytical solution to MSE

## 1.1. Optimal Parameters

With 1000 training samples, the resulting $\boldsymbol{w}$ and $b$ are sensible given that they roughly equal to the coefficients of the equation within the data generating function: $\mathrm{y} = 1.5 + 0.6x^{(1)} + 0.35x^{(2)}$

Table 1: Training with 1000 samples

| | $b$ | $w^{(1)}$ | $w^{(2)}$ |
|---|---|---|---|
| Learnt | 1.4862 | 0.6083 | 0.3435 |
| Actual | 1.5 | 0.6 | 0.35 |

Table 2: MSE for 1000 training samples

| | Training | Test |
|---|---|---|
| MSE | 0.0478 | 0.0495 |

## 1.2. Changing to Training Sample Size

This set of parameters differs from the one obtained in 1.1. With too small a training sample size, there is insufficient data to accurately capture the coefficients for the linear regression.

Table 3: Training with 10 samples

| | $b$ | $w^{(1)}$ | $w^{(2)}$ |
|---|---|---|---|
| Learnt | 1.1395 | 0.6049 | 0.5678 |
| Actual | 1.5 | 0.6 | 0.35 |

Table 4: MSE for 10 training samples

| | Training | Test |
|---|---|---|
| MSE | 0.0548 | 0.0927 |

## 1.3. Experimentation with Training Sample Sizes and RNG Seeds

Prior to this, everything was done with `rng(12345)`. Now, we will vary seed values and training sample sizes to observe how the number of training samples affects `Training MSE` and `Test MSE`.

Table 5: Experimentation MSE Means and Standard Deviations

| Training Samples | Training MSE | Test MSE |
|---|---|---|
| 4 | $0.0083 \pm 0.0093$ | $0.2205 \pm 0.2989$ |
| 10 | $0.0355 \pm 0.0188$ | $0.0808 \pm 0.0571$ |
| 20 | $0.0414 \pm 0.0147$ | $0.0586 \pm 0.0068$ |
| 100 | $0.0493 \pm 0.0078$ | $0.0518 \pm 0.0012$ |
| 1000 | $0.0504 \pm 0.0028$ | $0.0503 \pm 0.0004$ |
| 10000 | $0.0499 \pm 0.0007$ | $0.0501 \pm 0.0003$ |

As the training sample size increases, both `Training MSE` and `Test MSE` converge to the variance for regression target as mentioned in the data generating function `r_noise_var = 0.05`. To understand this, we firstly introduce the $\varepsilon$ term as random error independent of $\mathbf{X}$ with mean of zero $\mathrm{E}[\varepsilon] = 0$ such that $\mathrm{y}$ can be expressed as

$$\mathbf{y} = f(\mathbf{X}) + \varepsilon$$

Now we can re-express the MSE in terms of an expectation and a variance

$$
\begin{aligned}
\mathrm{MSE} = \mathrm{E}\left[(\mathbf{y} - \hat{\mathbf{y}})^2\right] &= \mathrm{E}\left[\left(f(\mathbf{X}) + \varepsilon - \hat{f}(\mathbf{X})\right)^2\right] \\
&= \mathrm{E}\left[f(\mathbf{X})^2 + 2\varepsilon f(\mathbf{X}) - 2f(\mathbf{X})\hat{f}(\mathbf{X}) + \varepsilon^2 - 2\hat{f}(\mathbf{X})\varepsilon + \hat{f}(\mathbf{X})^2\right] \\
&= \mathrm{E}\left[\left(f(\mathbf{X}) - \hat{f}(\mathbf{X})\right)^2\right] + \cancel{2\mathrm{E}[\varepsilon]\mathrm{E}[f(\mathbf{X})]} + \mathrm{E}[\varepsilon^2] - \cancel{2\mathrm{E}[\varepsilon]\mathrm{E}\left[\hat{f}(\mathbf{X})\right]} \\
&= \mathrm{E}\left[\left(f(\mathbf{X}) - \hat{f}(\mathbf{X})\right)^2\right] + \mathrm{E}[\varepsilon^2] = \mathrm{E}\left[\left(f(\mathbf{X}) - \hat{f}(\mathbf{X})\right)^2\right] + \left(\mathrm{E}[\varepsilon^2] - \mathrm{E}[\varepsilon]^2\right) \\
&= \mathrm{E}\left[\left(f(\mathbf{X}) - \hat{f}(\mathbf{X})\right)^2\right] + \mathrm{Var}(\varepsilon)
\end{aligned}
$$

Given this re-expression, we observe that as the error between prediction and target goes to zero, there remains an irreducible error source in the form of the regression target's variance, explaining the convergence to 0.05 in agreement with `r_noise_var = 0.05`.

# 2. Task 2: Linear Regression via Gradient Descent

## 2.1. Optimal Learning Rate 1

To find the optimal learning rate $\lambda$, we keep `n_iters = 1000` and perform the experiment for `lambdas = [0.00001 0.0001 0.001 0.01 0.1 1]`, picking the one with lowest corresponding `Training MSE` and `Validation MSE` $\implies$ `lambda = 0.1`.

Table 6: Training and Validation MSEs for Different Learning Rates at 1000 iterations

| $\lambda$ | Training MSE | Validation MSE |
|---|---|---|
| 0.00001 | 6.5561 | 6.8993 |
| 0.0001 | 1.2430 | 1.2485 |
| 0.001 | 0.1428 | 0.1279 |
| 0.01 | 0.0472 | 0.0532 |
| 0.1 | 0.0465 | 0.0528 |
| 1 | NaN | NaN |

## 2.2. Test Set

With the optimal parameters derived via `lambda = 0.1`, we calculate the `Test MSE`. There is a slight difference between the two MSE values, and this is simply a result of `rng(12345)` "shuffling" the numbers. The MSE values obtained so far have been done with test set being created after the train-val set. The results will be different if the ordering was swapped.

Table 7: 200 validation and 20000 test samples

| | Validation | Test |
|---|---|---|
| MSE | 0.0528 | 0.0496 |

Table 8: Reversed order of data generation

| | Validation | Test |
|---|---|---|
| MSE | 0.0486 | 0.0497 |

## 2.3. Optimal Learning Rate 2

Changing from `iters_total = 1000` $\rightarrow$ `iters_total = 10000`, we get `lambda = 0.01`.

Table 9: Training and Validation MSEs for Different Learning Rates at 10000 iterations

| $\lambda$ | Training MSE | Validation MSE |
|---|---|---|
| 0.00001 | 1.2439 | 1.2495 |
| 0.0001 | 0.1428 | 0.1279 |
| 0.001 | 0.0473 | 0.0532 |
| 0.01 | 0.0465 | 0.0528 |
| 0.1 | 0.0465 | 0.0528 |
| 1 | NaN | NaN |

### 2.4. Relationship between $n_{\text{iters}}$ and $\lambda$

Based on the experiments in 2.1. and 2.3., we can see that as $n_{\text{iters}}$ decreases, the minimum optimal learning rate, $\lambda$, increases. This is reasonable since a suitable larger $\lambda$ would lead to faster convergence, thereby reducing the $n_{\text{iters}}$ necessary. Between the two, it is $n_{\text{iters}}$ that affects runtime, because it dictates the number of loops for the learning function while $\lambda$ is only involved in one multiplication step within each loop. Therefore, in practice, it is preferable to use a short $n_{\text{iters}}$ coupled with the optimal $\lambda$ so that runtime is shortened.

### 2.5. Comparison with Task 1

Running the analytical solution from Task 1 on the Task 2 training data produces the same optimal parameters, which is reasonable since Gradient Descent should converge to the exact solution.

Table 10: Comparison of optimal parameters between Task 1 and Task 2

| | $b$ | $w^{(1)}$ | $w^{(2)}$ |
|---|---|---|---|
| Task 1 | 1.4899 | 0.6089 | 0.3382 |
| Task 2 | 1.4899 | 0.6089 | 0.3382 |

## 3. Task 3: Logistic Regression using Gradient Descent

### 3.1. Derivation of Log-Loss Gradient

$$
\begin{aligned}
\nabla_\theta \mathcal{L}_C &= \nabla_\theta \{-y \log(\sigma(\hat{\mathbf{x}}^T \boldsymbol{\theta})) - (1-y)\log(1-\sigma(\hat{\mathbf{x}}^T \boldsymbol{\theta}))\} \\
&= \nabla_\theta \left\{ -y \log\left(\frac{1}{1+\exp(-\hat{\mathbf{x}}^{\boldsymbol{T}} \boldsymbol{\theta})}\right) - (1-y)\log\left(1 - \frac{1}{1+\exp(-\hat{\mathbf{x}}^{\boldsymbol{T}} \boldsymbol{\theta})}\right)\right\} \\
&= \nabla_\theta \left\{ y \log(1+\exp(-\hat{\mathbf{x}}^{\boldsymbol{T}} \boldsymbol{\theta})) - (1-y)\log\left(\frac{\exp(-\hat{\mathbf{x}}^{\boldsymbol{T}} \boldsymbol{\theta})}{1+\exp(-\hat{\mathbf{x}}^{\boldsymbol{T}} \boldsymbol{\theta})}\right)\right\} \\
&= \nabla_\theta \{ y \log(1+\exp(-\hat{\mathbf{x}}^{\boldsymbol{T}} \boldsymbol{\theta})) + (1-y)\log(1+\exp(-\hat{\mathbf{x}}^{\boldsymbol{T}} \boldsymbol{\theta})) - (1-y)\log(\exp(-\hat{\mathbf{x}}^{\boldsymbol{T}} \boldsymbol{\theta}))\}
\end{aligned}
$$

$$= \nabla_\theta\{\log(1+\exp(-\hat{\mathbf{x}}^{\boldsymbol{T}}\boldsymbol{\theta})) - (1-y)\log(\exp(-\hat{\mathbf{x}}^{\boldsymbol{T}}\boldsymbol{\theta}))\}$$

$$= \nabla_\theta\{\log(1+\exp(-\hat{\mathbf{x}}^{\boldsymbol{T}}\boldsymbol{\theta})) + (1-y)\hat{\mathbf{x}}^{\boldsymbol{T}}\boldsymbol{\theta}\}$$

$$= -\frac{\exp(-\hat{\mathbf{x}}^{\boldsymbol{T}}\boldsymbol{\theta})}{1+\exp(-\hat{\mathbf{x}}^{\boldsymbol{T}}\boldsymbol{\theta})}\hat{\mathbf{x}} + (1-y)\hat{\mathbf{x}}$$

$$= \left(\frac{-1-\exp(-\hat{\mathbf{x}}^{\boldsymbol{T}}\boldsymbol{\theta})}{1+\exp(-\hat{\mathbf{x}}^{\boldsymbol{T}}\boldsymbol{\theta})} + \frac{1}{1+\exp(-\hat{\mathbf{x}}^{\boldsymbol{T}}\boldsymbol{\theta})}\right)\hat{\mathbf{x}} + (1-y)\hat{\mathbf{x}}$$

$$= (-1+\bar{y})\hat{\mathbf{x}} + (1-y)\hat{\mathbf{x}} = (\bar{y}-y)\hat{\mathbf{x}} \qquad \blacksquare$$

## 3.2. Optimal Learning Rate 1

To find the optimal learning rate $\lambda$, we keep `n_iters = 1000` and perform the experiment for `lambdas = [0.00001 0.0001 0.001 0.01 0.1 1]`, picking the one with lowest corresponding `Training Error` and `Validation Error` $\Longrightarrow$ `lambda = 1`.

Table 11: Training and Validation $e$ for Different Learning Rates at 1000 iterations

| $\lambda$ | Training $e$ | Validation $e$ |
|---|---|---|
| 0.00001 | 0.5000 | 0.4450 |
| 0.0001 | 0.4988 | 0.4450 |
| 0.001 | 0.4537 | 0.4400 |
| 0.01 | 0.0963 | 0.1050 |
| 0.1 | 0.0275 | 0.0350 |
| 1 | 0.0200 | 0.0300 |

## 3.3. Optimal Learning Rate 2

With `n_iters = 1000` and `lambda = 1`, the loss plateaus at the end of training, suggesting convergence of mean log-loss.

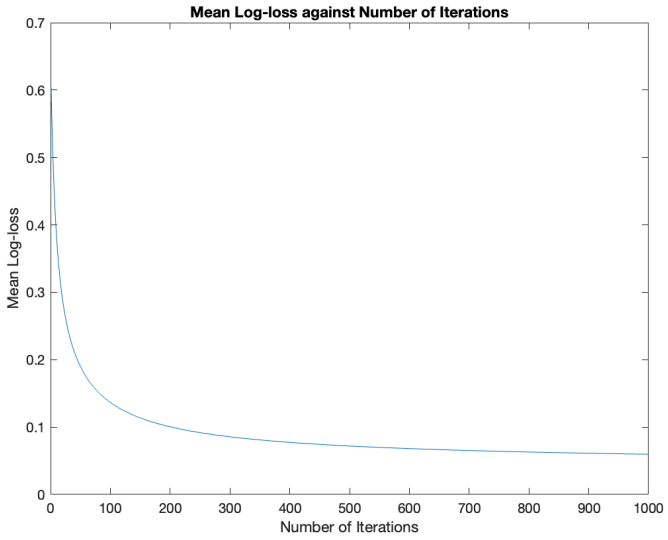


Figure 1: Mean Log-loss for `lambda = 1` against number of iterations

### 3.4. Test Set

Table 12: Validation and Test $e$ with Optimal Parameters

| | Validation | Test |
|---|---|---|
| $e$ | 0.0300 | 0.0242 |

### 3.5. Performance

Using seeds from `rng(1)` to `rng(20)`, we collect 20 classification error ratios for each dataset, over which we obtain their means and standard deviations.

Table 13: Experimentation $e$ Means and Standard Deviations

| Training Samples | Training Error | Validation Error | Test Error |
|---|---|---|---|
| 10 | $0.0000 \pm 0.0000$ | $0.0500 \pm 0.1500$ | $0.0600 \pm 0.0248$ |
| 20 | $0.0000 \pm 0.0000$ | $0.0250 \pm 0.0750$ | $0.0399 \pm 0.0119$ |
| 100 | $0.0150 \pm 0.0135$ | $0.0250 \pm 0.0296$ | $0.0285 \pm 0.0052$ |
| 1000 | $0.0247 \pm 0.0039$ | $0.0255 \pm 0.0086$ | $0.0253 \pm 0.0013$ |
| 10000 | $0.0249 \pm 0.0016$ | $0.0254 \pm 0.0032$ | $0.0251 \pm 0.0014$ |

Over-fitting is when the error ratio for one dataset is much lower than that for the other 2. This is observed in the first 2 rows, where `Training Error` deviates greatly from both `Validation Error` and `Test Error`. In fact, the `Training Error` reaches 0, implying perfect inference by the trained model. This over-fitting may be due to the training sample size being too small (10 and 20 samples). Another observation from the table is that `Validation Error` and `Test Error` are very similar. This suggests that the model's performance on the validation samples gives us a good indication of that for the test samples.

## 4. Task 4: Logistic Regression with Stochastic Gradient Descent

### 4.1. Configuring Hyperparameters

Using seeds from `rng(1)` to `rng(20)`, we collect 20 classification error ratios for each dataset, over which we obtain their means and standard deviations. From there, we pick the batch size that yields the lowest corresponding `Training Error` and `Validation Error` $\Longrightarrow$ `batch_size = 100`

Table 14: Training and Validation $e$ Means and Standard Deviations

| Batch Size | Training Error | Validation Error |
|---|---|---|
| 1 | $0.0276 \pm 0.0058$ | $0.0410 \pm 0.0041$ |
| 10 | $0.0218 \pm 0.0025$ | $0.0370 \pm 0.0029$ |
| 20 | $0.0210 \pm 0.0017$ | $0.0355 \pm 0.0035$ |
| 50 | $0.0209 \pm 0.0015$ | $0.0345 \pm 0.0035$ |
| 100 | $0.0207 \pm 0.0009$ | $0.0328 \pm 0.0029$ |

## 4.2. Convergence

With `n_iters = 1000`, `lambda = 1`, and `batch_size = 100`, the loss plateaus at the end of training, suggesting convergence of mean log-loss.

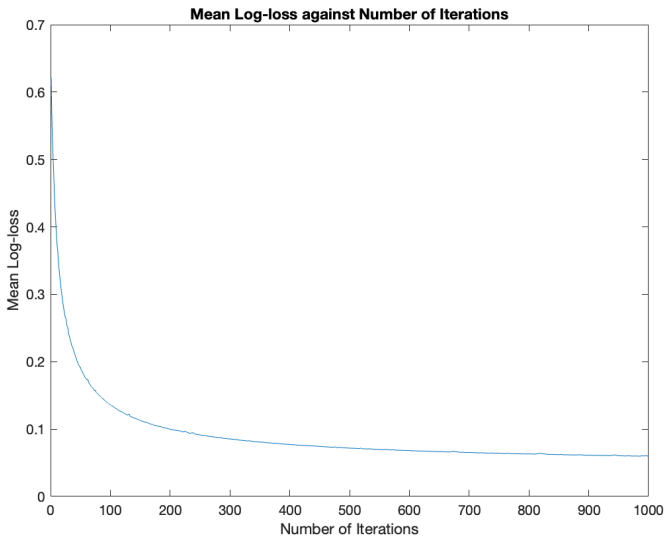


Figure 2: Mean Log-loss for `batch_size = 100` and `lambda = 1` against number of iterations

## 4.3. Comparison with Task 3.

By zooming into Figure 2, we can see that the plot is not smooth. This “noise” exists because the actual training sample size is smaller than the full training dataset, which leads to greater variance in mean log-loss when checked against the full training dataset.

## 4.4. Test Set

The resulting classification error ratios are very similar to those in 3.4., with test error ratio outperforming validation error ratio by a similar margin given the same order of data generation.

Table 15: Validation and Test Accuracy with Optimal Learning Rate and Batch Size

| | Validation | Test |
|---|---|---|
| $e$ | $0.0328 \pm 0.0029$ | $0.0251 \pm 0.0011$ |

## 4.5. Runtime and Memory Usage Differences

The number of steps within each of the $n_{\text{iters}}$ loops directly affect model training runtime. Unlike in GD, SGD only considers a subset of the full training dataset. As such, the number of calculations in each loop is smaller, resulting in shorter runtime. However, to continually get batches of random elements within the full training dataset, the algorithm requires extra memory as large as the `batch_size`. The larger the `batch_size`, the slower the training and the larger the extra memory needed. For large databases in real-world applications, applying an SGD makes sense because it has the potential of greatly shortening model training runtime. To realise this potential, however, there must be sufficient RAM to store the batches.

# 5. Task 5: Optimizing SVM via Linear Programming

## 5.1. Linear Programming implementation

To be compatible with MATLAB's `linprog` solver, $\theta$ and $\xi$ can be combined into $\psi$ where

$$\boldsymbol{\psi} = \begin{bmatrix} \xi_1 & \xi_2 & \xi_3 & \dots & \xi_{n-1} & \xi_n & b & w^{(1)} & w^{(2)} \end{bmatrix}^T$$

and since $\mathbf{f}^T\boldsymbol{\psi} = \sum_{i=1}^{n} \xi_i$

$$\mathbf{f} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 & 1 & 0 & 0 & 0 \end{bmatrix}^T$$

Another necessary part is to form $\mathbf{A}, \beta, \mathbf{A}_{\text{eq}}\ \beta_{\text{eq}}, \mathbf{lb}, \mathbf{ub}$ such that they map to the original conditions of the optimisation equation. These then can be used in `linprog(f, A, b, Aeq, beq, lb, ub)`. Four of these are trivial to setup due to the lack of an equality condition as well as there only being a lower bound for $\boldsymbol{\xi}$. Parameters $\mathbf{A}$ and $\beta$ can be derived by first manipulating the original condition into

$$-(\hat{\mathbf{x}}_i^T\theta)y_i - \xi_i \leq -1$$

```
function theta_opt = train_SVM_linear_progr(X, y)
    f = [ones(length(y), 1); 0; 0; 0];    % [1 1 1 ... 1 1 0 0 ]^ T with n + 3 elements
    b = -ones(length(y), 1);              % [-1 -1 -1 ... -1 -1] ^ T with n elements
    A = -eye(length(y));                  % Only ith slack param for the ith inequality
    % Based on an observation from the altered original condition
    for i = 1:length(y)
        if y(i, 1) == 1
            X(i, :) = -X(i, :);
        end
    end
    A = [A X];
    Aeq = [];                             % No equality condition
    beq = [];                             % No equality condition
    lb = [zeros(length(y), 1); -inf; -inf; -inf]; % Lower bound applies to slack params
    ub = [inf(length(lb), 1)];            % No upper bound
    theta_opt = linprog(f, A, b, Aeq, beq, lb, ub);
    theta_opt = theta_opt(end-2:end, 1);  % The last 3 elements are the optimal params
end
```

## 5.2. Performance

Compared to the `Test Error` in both the GD (0.0242) and SGD ($0.0251 \pm 0.0011$) cases for logistic regression, the SVM method yields very similar results.

Table 16: Training and Test Accuracy with SVM Linear Programming

| | Train | Test |
|---|---|---|
| $e$ | 0.0200 | 0.0241 |

### 5.3. Decision Boundary Derivation

Given the form $\bar{y} = b + w^{(1)}x^{(1)} + w^{(2)}x^{(2)}$, making $x^{(2)}$ the subject of formula generates the following form: $x^{(2)} = \alpha x^{(1)} + \beta$ where $\alpha = \left(-\frac{w^{(1)}}{w^{(2)}}\right)$ and $\beta = \frac{\bar{y}-b}{w^{(2)}}$.

Table 17: Optimal Parameters and Resultant $\alpha$ and $\beta$ for SVM, SGD and GD Logistic Regression

| | SVM ($\bar{y} = 0$) | SGD LR ($\bar{y} = 0.5$) | GD LR ($\bar{y} = 0.5$) |
|---|---|---|---|
| $b$ | −9.6567 | −10.9964 | −10.5063 |
| $w^{(1)}$ | 1.9256 | 1.9981 | 1.9365 |
| $w^{(2)}$ | 5.4760 | 6.5872 | 6.2485 |
| $\alpha$ | −0.3516 | −0.3033 | −0.3099 |
| $\beta$ | 1.7635 | 1.7453 | 1.7614 |

The $\alpha$ and $\beta$ values are very similar in all 3 cases, and their graphs reflect the similarity too.

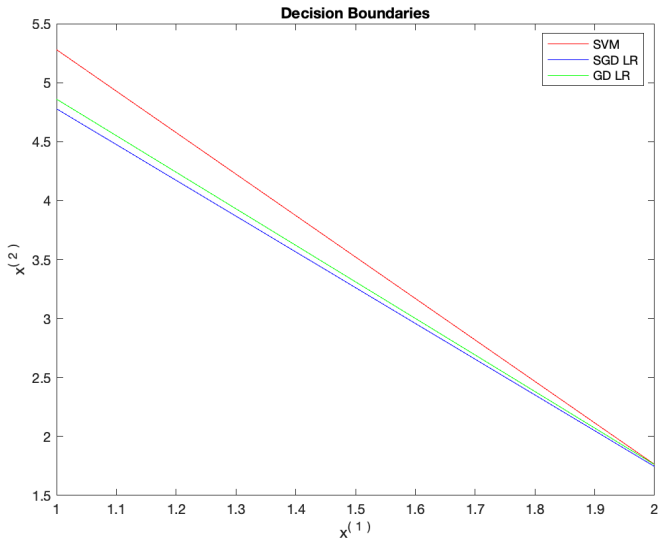


Figure 3: 2D Plots of Decision Boundaries for SVM, SGD, and GD Logistic Regression

## 6. Task 6: Optimizing SVM via Gradient Descent and Hinge Loss

### 6.1. Hinge Loss Gradient Computation

```
for i = 1:iters_total
    grad_loss = zeros(n_features, 1);                % Initialise Hinge Loss Gradients
    hinge_losses = hinge_loss_per_sample(X_train, y_train, theta_curr);
    for j = 1:length(hinge_losses)
        if hinge_losses(j) > 0                       % Satisfy the Gradient condition
            grad_loss = grad_loss - y_train(j) * X_train(j, :)';
        end
    end
    theta_curr = theta_curr - learning_rate / length(y_train) * grad_loss;
end
```

## 6.2. Configuring Hyperparameters

To find the optimal learning rate $\lambda$, we keep `n_iters = 10000` and perform the experiment for `lambdas = [0.00001 0.0001 0.001 0.01 0.1 1]`, picking the one with lowest corresponding `Training Error` and `Validation Error` $\Longrightarrow$ `lambda = 1`.

Table 18: Training and Validation $e$ for Different Learning Rates at 10000 iterations

| $\lambda$ | Training $e$ | Validation $e$ |
|---|---|---|
| 0.00001 | 0.5000 | 0.4450 |
| 0.0001 | 0.4250 | 0.4050 |
| 0.001 | 0.0438 | 0.0400 |
| 0.01 | 0.0200 | 0.0350 |
| 0.1 | 0.0175 | 0.0300 |
| 1 | 0.0163 | 0.0300 |
| 10 | 0.0163 | 0.0300 |

## 6.3. Average Loss

The optimal combination of `n_iters = 10000` and `lambda = 1` generates the following average loss behaviour against number of iterations. We can see that the plot plateaus, suggesting convergence of the average loss.

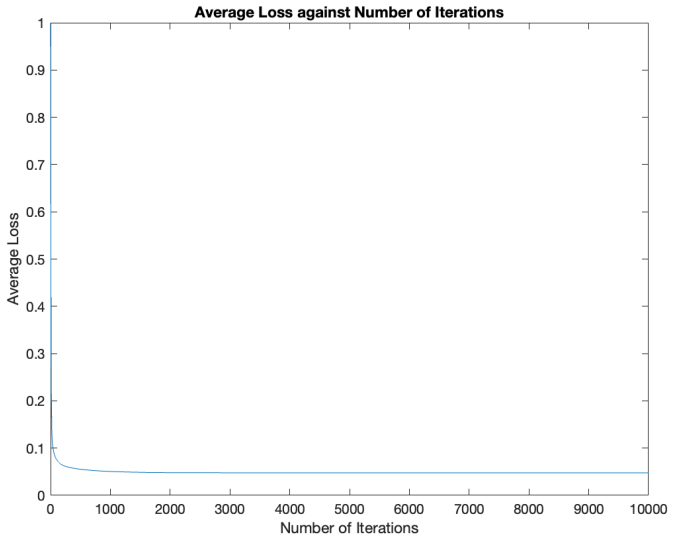


Figure 4: Average Loss for `n_iters = 10000` and `lambda = 1` against number of iterations

## 6.4. Performance

Compared with 5.2., the `Test Error` are very similar, implying similar performance of the learnt optimal parameters in both techniques.

Table 19: Errors with SVM Hinge Loss GD

| | Train | Validation | Test |
|---|---|---|---|
| $e$ | 0.0163 | 0.0300 | 0.0238 |

Table 20: Errors for Different SVM methods

| | Hinge Loss GD | Linear Programming |
|---|---|---|
| $e$ | 0.0238 | 0.0241 |

## 6.5. Comparison of Optimal Parameters

When compared with the Linear Programming method, we do not get the same parameters, explaining the slight real-world performance difference of the 2 sets of optimal parameters.

Table 21: Optimal Parameters learnt from Different methods

| | Hinge Loss GD | Linear Programming |
|---|---|---|
| $b$ | −10.0675 | −9.6567 |
| $w^{(1)}$ | 2.0265 | 1.9256 |
| $w^{(2)}$ | 5.7141 | 5.4760 |

## 6.6. Decision Boundary Derivation

Referring to the altered form: $x^{(2)} = \alpha x^{(1)} + \beta$ where $\alpha = \left(-\frac{w^{(1)}}{w^{(2)}}\right)$ and $\beta = \frac{\bar{y}-b}{w^{(2)}}$, since comparison is only made between different SVM methods, $\bar{y} = 0$, allowing for $\beta = \frac{-b}{w^{(2)}}$ and yielding the following $\alpha$ and $\beta$ values.

Table 22: Resultant $\alpha$ and $\beta$ for SVM, SGD and GD Logistic Regression

| | Hinge Loss GD | Linear Programming |
|---|---|---|
| $\alpha$ | −0.3546 | −0.3516 |
| $\beta$ | 1.7619 | 1.7635 |

The $\alpha$ and $\beta$ values are very similar for both cases, and their graphs reflect this too.

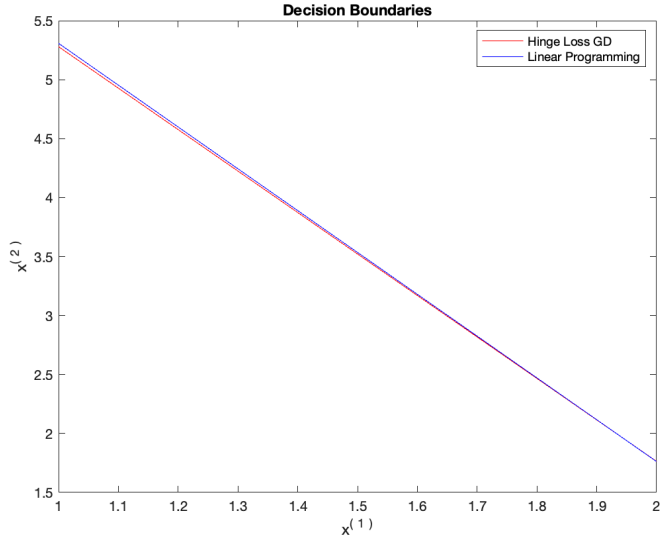


Figure 5: 2D Plots of Decision Boundaries for Hinge Loss GD and Linear Programming