COL774 Assignment 1.2

2022CS11621 - Jakharia Rishit

2022EE11152 - Pratham Malhotra

1 Part B

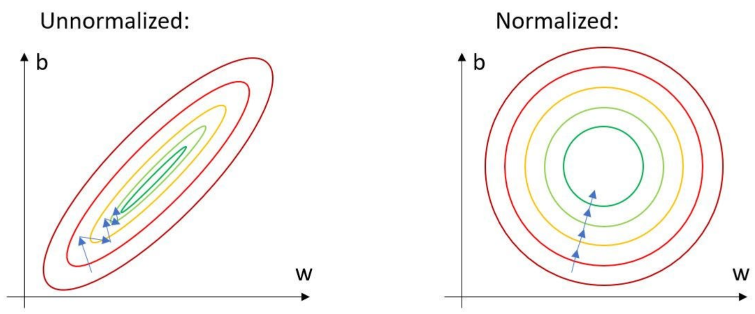
In Part B, we explore working hyper-parameters and learning rate strategies that minimize the loss as quickly as possible and use the model to train and make predictions on the evaluation data. We explored:

1. Pre-processing (Normalization)
2. Varying weight initialization
    - Random Initialization
    - Zeros Initialization
3. Learning Rate Strategies
    - Constant Rate
    - Decaying Rate
    - Ternary Search
    - Step Function
    - Momentum
    - RMSProp
    - Adam
4. Varying Hyperparameters

1.1 Pre-Processing

The necessity of data normalization arises from the inherent differences in the scales of various features within a dataset. For instance, consider a dataset containing features such as age (0-100) and income (thousands to millions). Larger-scale features can dominate the learning process without normalization, leading to sub-optimal model performance.

$$X_{norm} = \frac{X - \mu}{\sigma + \epsilon}$$



1.2 Varying Weight Initialization

Typically, in neural networks, weights are initialized with small random values to break symmetry and allow the model to learn unique features during training. Different initialization strategies, such as zero initialization, random initialization, or initialization based on specific distributions (like Gaussian or Xavier initialization), can influence how quickly the model learns and the quality of the final solution.

1.2.1 Random Initialization

In this initialization method, we randomly sample numbers, usually from a normal distribution, for the initial weights of the model.
However, keeping the other hyper-parameters the same, the random initialization did not yield much difference.

Table 1: Random vs Zero Initialization

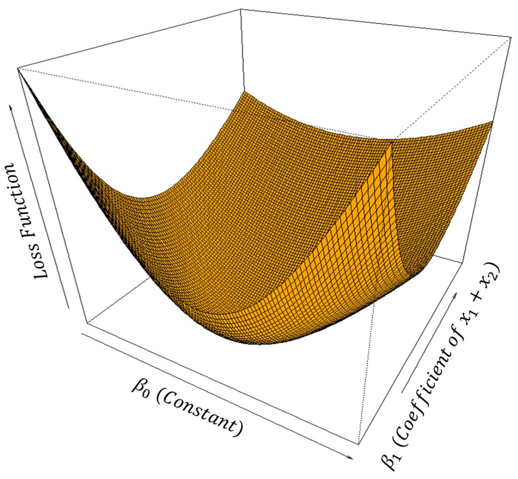
Hyper-Params	Random Init. Error	Zero Init. Error
1, 25, 100	4.10e-5	4.09e-5
10, 25, 100	3.487e-5	3.488e-5
70, 25, 100	3.57e-5	3.53e-5
100, 25, 100	3.57e-5	3.56e-5

Hyper-Params: learning_rate, epochs, batch_size
Errors have been noted up to the first digit of difference

1.2.2 Zero Initialization

In this initialization, we initialize all weights to 0, and since the initial values of weights are 0, this helps in not *"Blowing up"* the values of *probs*

1.2.3 Explanation



Convex Loss Function: The loss function used in logistic regression is convex with respect to the model parameters (*As shown above for a simpler model with 2 coefficients*).
Thus, there is a single global minimum and no local minimum to get trapped. As a result, the optimization process is straightforward, and the model will converge to the same solution regardless of the initial weights as long as the learning rate is appropriate.

1.3 Learning Rate Strategies

As discussed above, initialization of weights does not have much of an effect on the gradient descent; rather, the choice of learning rate and learning rate strategies play a crucial part in finding the minima of the loss function.

1.3.1 Constant Rate

This is the simplest learning rate strategy; here, we do not change or adapt the value of the learning rate throughout the gradient descent.

$$\lambda = \eta_0$$

Results

Table 2: Constant Learning Rate

Learning Rate	(32, 100)	(64, 25)	(100, 25)	(128, 25)
1	3.472e-5	3.934e-5	4.091e-5	3.877e-5
10	3.487e-5	3.465e-5	3.485e-5	3.472e-5
25	3.571e-5	3.489e-5	3.464e-5	3.464e-5
40	3.576e-5	3.520e-5	3.495e-5	3.477e-5
70	3.667e-5	3.556e-5	3.533e-5	3.513e-5
100	3.879e-5	3.675e-5	3.561e-5	3.544e-5
1e5	3.991e-4	4.041e-4	4.045e-4	3.842e-4

Hyper-Params: batch_size, epoch

Currently, best result is
algorithm = constant, batch_size = 128, epochs = 25, learning_rate = 25 $\rightarrow$ 3.46384121341326e-05

1.3.2 Decaying Rate

Here, we use the logic: Earlier in gradient descent, we can have a larger learning rate; however, in later stages of gradient descent, to not overshoot the minima, It is better to have smaller learning rates.

$$\lambda = \frac{\eta_0}{1 + kt}$$

eta_0 = initial learning rate
k = decaying strength
t = number of epoch
Results

Table 3: Decaying Learning Rate

Learning Rate (η_0)	k=1	k=0.9	k=0.09
25	3.711e-5	3.711e-5	3.472e-5
40	3.595e-5	3.594e-5	3.510e-5
50	3.544e-5	3.544e-5	3.522e-5
100	3.477e-5	3.477e-5	3.567e-5
150	3.466e-5	3.465e-5	3.581e-5
200	3.468e-5	3.468e-5	3.594e-5
300	3.490e-5	3.488e-5	3.701e-5

Hyper-Params: epochs = 25, batch_size = 100
Errors have been noted up to the first digit of difference

Currently, best result is
algorithm = constant, batch_size = 128, epochs = 25, learning_rate = 25 $\rightarrow$ 3.46384121341326e-05

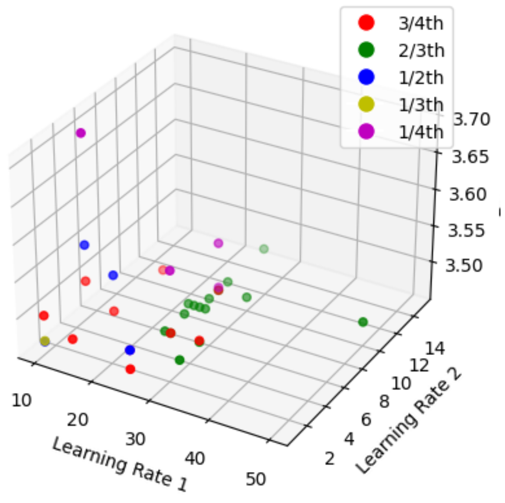
1.3.3 Ternary Search

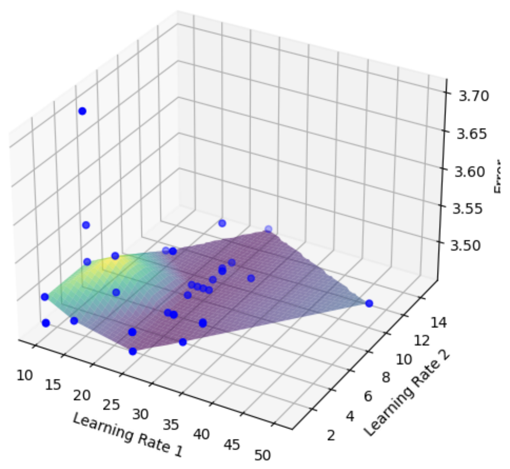
Since the loss function in the learning rate is a one-dimensional function in η, and since it can be proved that the multiclass logistic regression loss function is convex, the 1-D convex optimizations technique can be applied to it to find such η.
Due to time constraints on this part, Ternary Search could not be implemented with higher epoch numbers and better batch_sizes; thus, the

1.3.4 Step Function

Similar logic to the Decaying Learning Rate, however, the decrease in the learning rate is sharper and in much more control of us.

$$\lambda = \begin{cases} \eta_0, & 1 \le \text{epoch} < a_1 \\ \eta_1, & a_1 \le \text{epoch} < \text{n\_features} \end{cases}$$





The above are two graphs with axes:
learning rate 1 $= \eta_0$
learning rate 2 $= \eta_1$
Error (e-5) = grade_b.py error
also (3/4th , 2/3th, ..., 1/4th) represent: $\frac{epoch}{total\_epochs}$

Currently, best result is
algorithm = step, batch_size = 100, epochs = 25, $\eta_0 = 24$, $\eta_1 = 8 \to$ 3.4638161013840806e-05

1.3.5 Momentum

Momentum is useful in optimization because it helps accelerate gradient descent by using past gradients to maintain a consistent direction. This reduces oscillations and speeds up convergence, especially in areas with gentle slopes or complex loss surfaces.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla L(w)$$
$$m_t = \frac{m_t}{1 - \beta_1^t}$$
$$w_{t+1} = w_t - \lambda m_t$$

Results

Table 4: Momentum

Learning Rate (λ)	Error
5e-5	4.07e-5
1	3.83e-5
5	3.48e-5
9	3.464e-5
10	3.46e-5
11	3.466e-5
15	3.477e-5
30	3.53e-5
50	3.55e-5
100	3.68e-5

1.3.6 RMSProp

RMSProp adjusts the learning rate dynamically, reducing it for parameters with large gradients to prevent oscillations and increasing it for parameters with small gradients to ensure faster convergence. The update rule for RMSProp involves maintaining a moving average of the squared gradients.

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\nabla L(w)^2$$

$$v_t = \frac{v_t}{1 - \beta_2^t}$$

$$w_{t+1} = w_t - \lambda \frac{\nabla L(w)}{\sqrt{v_t} + \epsilon}$$

Since not very good results were obtained here, results are not being mentioned.

1.3.7 Adam

Adam (Adaptive Moment Estimation) is an optimization algorithm that combines the benefits of both momentum and RMSProp to provide an adaptive learning rate for each parameter. It maintains an exponentially decaying average of past gradients (momentum) and an exponentially decaying average of past squared gradients (RMSProp), allowing it to adaptively adjust the learning rates based on the first and second moments of the gradients.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Results

Table 5: Adam

Learning Rate (λ)	Error
10	4e-4
1e-3	3.77e-5
3e-4	3.60e-5
7e-4	3.72e-5
1e-4	3.53e-5
3e-5	3.73e-5
5e-5	3.60e-5

1.3.8 Conculsion

After the above analysis, currently, the best configuration we have is:
algorithm = step, batch_size = 100, epochs = 25, $\eta_0 = 24$, $\eta_1 = 8 \rightarrow$ 3.4638161013840806e-05

1.4 Varying Hyper-Parameters

Small Batch Sizes lead to noisy gradients, which can help escape local minima but might cause instability during training.
Large Batch Sizes result in more stable gradients and faster convergence but can lead to convergence at sharp minima, which might not generalize well.
But since we have one minimum, larger batch size is not a big problem.

Results

Table 6: Batch Size and Epochs

Epoch ↓ BatchSize →	50	75	100	200
20	3.557e-5	3.521e-5	3.467e-5	3.523e-5
24	3.533e-5	3.510e-5	3.464e-5	3.510e-5
25	3.484e-5	3.467e-5	3.463e-5	3.499e-5
26	3.488e-5	3.471e-5	3.464e-5	3.501e-5
30	3.490e-5	3.473e-5	3.465e-5	3.503e-5
50	3.506e-5	3.498e-5	3.480e-5	3.512e-5

Thus, the best configuration remains
batch_size = 100
epochs = 25
$\eta_0 = 24$
$\eta_1 = 8$
algorithm = Step Function

2 Part c

Feature Selection and feature creation approaches are very important in machine learning. In this part, we explore feature selection and feature creation techniques to produce the best possible accuracy. We explored:

1. Feature Creation
   (a) One Hot Encoding
   (b) Smooth Target Encoding
2. Feature Selection
   (a) ANOVA for feature selection
3. Data Preprocessing
   (a) Normalization of Input data
   (b) Skewness analysis of input data

2.1 Feature Creation

2.1.1 Ont-Hot Encoding

All categorical features were ordinally encoded, which instills a sense of order between the training examples; however, for nominal features in this data, this can be a problem; hence, one-hot encoding was employed for the following columns.

- APR Severity of Illness Description
- Length of Stay
- APR MDC Code
- APR Risk of Mortality
- APR Medical Surgical Description
- Age Group

- Patient Disposition
- Permanent Facility Id
- Race
- Ethnicity
- Payment Typology 1
- Payment Typology 2
- Payment Typology 3

2.1.2 Smooth Target Encoding

While the input data has ordinal encoding, we do not know if the order is correct. To ensure this, we implement smooth target encoding. However, there are some problems with Target encoding, mainly because our dataset has many outliers.

Table 7: Target vs Smooth Target Encoding

Feature	Target Encoding	Smooth Target Encoding
Overfitting	Prone to Overfitting	Less prone to overfitting
Rare Categories	Sensitive to rare categories	Handles rare categories better

Thus, here, we have used Smooth target encoding, implemented below,

$$E[Y|X = x_i] = \frac{\sum_{j=1}^{n_i} y_{ij}}{n_i}$$

$$n_i = \text{count of category } x_i$$

$$\hat{y_i} = \frac{n_i E[Y|X = x_i] + \alpha\mu}{n_i + \alpha}$$

Where α is the smoothing factor.

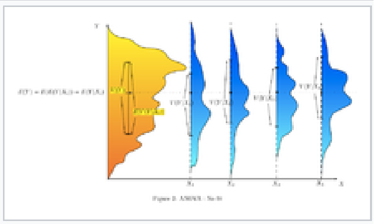
2.2 Feature Selection

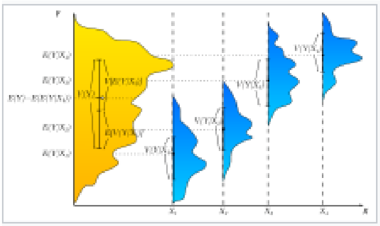
2.2.1 ANOVA

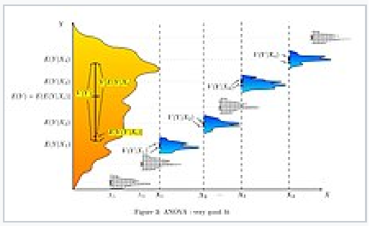
Analysis of Variance (ANOVA) is a statistical technique used to determine if there are any statistically significant differences between the means of three or more independent groups. In the context of feature selection, ANOVA can be used to assess the importance of different features by analyzing how much variance in the dependent variable can be explained by each feature.

When applying ANOVA for feature selection, each feature is considered individually, and an F-test is conducted to compare the variance between groups defined by the feature values to the variance within those groups. A high F-value indicates that the feature has a significant effect on the target variable, making it a good candidate for selection.

This method is particularly effective when the data is normally distributed and helps to identify features that contribute the most to the variance in the target variable, aiding in the development of more accurate and efficient predictive models.


No fit: Young vs old, and short-haired vs long-haired


Fair fit: Pet vs Working breed and less athletic vs more athletic

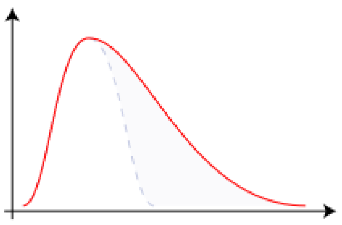

Very good fit: Weight by breed

2.3 Data Pre-processing

2.3.1 Skewness Analysis

(a) **Introduction**
Skewness analysis is a critical component in understanding data distribution within a dataset. Skewness refers to the degree of asymmetry observed in a frequency distribution. It is a measure that indicates whether the data points in a dataset are more concentrated on one side of the mean than the other.

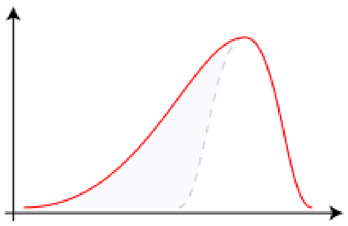
(b) **Positive Skewness**
Positive skewness (or right skew) occurs when the tail on the right side of the distribution is longer or fatter than the left side, indicating that the mean and median are greater than the mode. This suggests that more data points are clustered at lower values with fewer high-value outliers.



Positive skew

To Tackle this, a **logarithmic transformation** can help compress the range of higher values, bringing the distribution closer to normality.

(c) **Negative Skewness**
Negative skewness (or left skew) happens when the tail on the left side of the distribution is longer or fatter than the right side, indicating that the mean and median are less than the mode, with a larger number of data points clustered at higher values and fewer low-value outliers.



Negative skew

For negatively skewed data, techniques such as Yeojhonson can stretch the lower values, redistributing the data points more evenly around the mean.

Note: However, in the current assignment, the training accuracy seemed to decrease as it became harder for the model to converge to a solution after the removal of skewness; hence, the code for this has not been included in logistic_competitive.py

2.4 Conclusion

The models below were tried in Kaggle's environment, and the following error values were achieved using the 'grade c.py' file given in the assignment. The best accuracy achieved was **62.51**