
# Project Report

**Intern Name:** Ahmed Islam

**Department:** Artificial Intelligence

**Organization**: IR Solutions

**Designation:** AI Intern

**Week # 01**

**Submitted to:** Asad Ali

**Submission Date:** 11-July-2025

**Report Type:** Combined Tasks Documentation

# Theoretical Understanding

First, I learned about the machine learning fundamentals from the Google Developer program.

I explore the types of Machine learning

1. Supervised Learning: Where models learn from the labeled data, like we have multiple features and its crossponding label.
    I. Classification: In classification we have categorical data. Classification are also two types.
        1. *Binary Classification*
        2. *Multiclass classification*

    2. Regression:
2. Unsupervised Learning: In unsupervised learning we don't have a labels model learn through drawing a Clusters and extract the label data.
3. Reinforcement learning: Where model learn on the basis of trail and error. We give a reward if the model predicts correctly and give penalties for the bad prediction are action.
4. Generative model: Model generate text or images on the basis of user input.

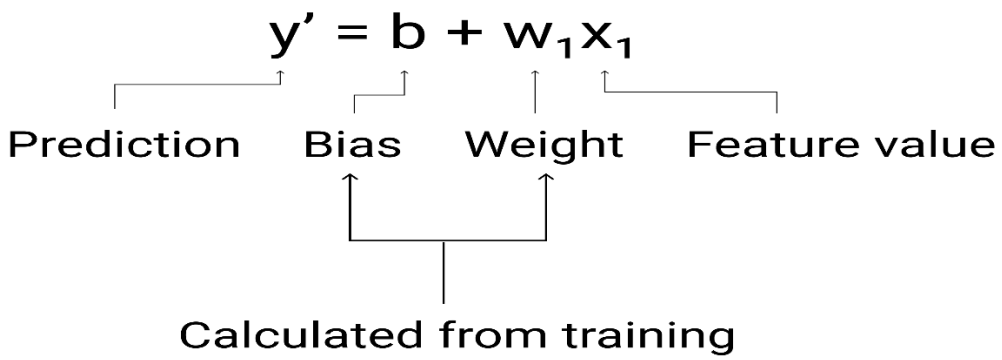
## Linear Regression:

Linear regression is a statistical technique used to find the relationship between variables. In an ML context, linear regression finds the relationship between features and a label.

Linear regression equation:

$$y= mx+b$$

- **y** is the predicted label, output.
- **b** is the bias of the model. Bias is the parameter of the model and calculates it during the training.
- **m or w1** is the weight of the Feature. Weight is the parameter of the model and calculates it during the training.
- **X** is the feature, input




**During training, the model updates the bias and weights.**

**Linear Regression Resource:** https://www.youtube.com/watch?v=jerPVDaHbEA

Loss: Difference between the actual and the predicted value.

## Types of loss

In linear regression, there are four main types of loss, which are outlined in the following table.

| Loss type | Definition | Equation |
|---|---|---|
| $L_1$ loss | The sum of the absolute values of the difference between the predicted values and the actual values. | $\sum \lvert actual\ value - predicted\ value \rvert$ |
| Mean absolute error (MAE) | The average of $L_1$ losses across a set of *N* examples. | $\frac{1}{N} \sum \lvert actual\ value - predicted\ value \rvert$ |
| $L_2$ loss | The sum of the squared difference between the predicted values and the actual values. | $\sum (actual\ value - predicted\ value)^2$ |
| Mean squared error (MSE) | The average of $L_2$ losses across a set of *N* examples. | $\frac{1}{N} \sum (actual\ value - predicted\ value)^2$ |

**Deciding whether to use MAE or MSE can depend on the dataset and the way you want to handle certain predictions.**

Gradient Decent: used to find the best weight and biases value that produces the model with the lowest loss.

Hyperparameters: are variables that control different aspects of training. Hyperparameters values are controlled by the user or developer.

Three common hyperparameters are:

1. *Learning Rate:* is a floating point number you set that influences how quickly the model converges.
2. *Epochs:* means that the model has processed every example in the training set once.
3. *Batch size:* is a hyperparameter that refers to the number of examples the model processes before updating its weights and bias.

# Logistic Regression

Used to predict the probability of given outcomes.

**Sigmoid Function:** Logistic regression use sigmoid function to predict the probability of the given outcome.

Sigmoid mean S shape.

$$f(x) = \frac{1}{1 + e^{-x}}$$

Figure 1 shows the corresponding graph of the sigmoid function.

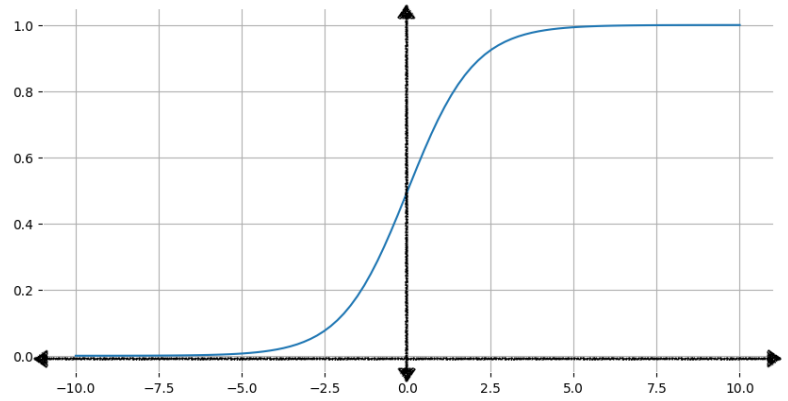



***Figure 1.*** *Graph of the sigmoid function. The curve approaches 0 as x values decrease to negative infinity, and 1 as x values increase toward infinity.*

no matter how large or how small the input value, the output will always be greater than 0 and less than 1.

Useful resources: https://www.youtube.com/watch?v=XNXzVfItWGY&t=42s

https://www.youtube.com/watch?v=_nvQKN8L1ZE

https://www.youtube.com/watch?v=n40hS9tQmcY

LogLoss: the loss function for **logistic regression** is Log Loss. The Log Loss equation returns the logarithm of the magnitude of the change, rather than just the distance from data to prediction.

$$\text{Log Loss} = \sum_{(x,y)\in D} -y \log(y') - (1-y) \log(1-y')$$

where:

- $(x, y) \in D$ is the dataset containing many labeled examples, which are $(x, y)$ pairs.
- $y$ is the label in a labeled example. Since this is logistic regression, every value of $y$ must either be 0 or 1.
- $y'$ is your model's prediction (somewhere between 0 and 1), given the set of features in $x$.

## Regularization in logistic regression:

Regularization is used to prevent model from outfilt. Regularization, a mechanism for penalizing model complexity during training.

There are two strategies to decrease the model complexity.

1. L2 regularization
2. Early stopping

## Confusion Matrix:

| | Actual positive | Actual negative |
|---|---|---|
| **Predicted positive** | **True positive (TP)**: A spam email correctly classified as a spam email. These are the spam messages automatically sent to the spam folder. | **False positive (FP)**: A not-spam email misclassified as spam. These are the legitimate emails that wind up in the spam folder. |
| **Predicted negative** | **False negative (FN)**: A spam email misclassified as not-spam. These are spam emails that aren't caught by the spam filter and make their way into the inbox. | **True negative (TN)**: A not-spam email correctly classified as not-spam. These are the legitimate emails that are sent directly to the inbox. |

## Accuracy:

**Accuracy** is the proportion of all classifications that were correct, whether positive or negative. It is mathematically defined as:

$$\text{Accuracy} = \frac{\text{correct classifications}}{\text{total classifications}} = \frac{TP + TN}{TP + TN + FP + FN}$$

## Recall:

### Recall, or true positive rate

The **true positive rate (TPR)**, or the proportion of all actual positives that were classified correctly as positives, is also known as **recall**.

Recall is mathematically defined as:

$$\text{Recall (or TPR)} = \frac{\text{correctly classified actual positives}}{\text{all actual positives}} = \frac{TP}{TP + FN}$$

## False positive rate:

The **false positive rate (FPR)** is the proportion of all actual negatives that were classified *incorrectly* as positives, also known as the **probability of false alarm.** It is mathematically defined as:

$$\text{FPR} = \frac{\text{incorrectly classified actual negatives}}{\text{all actual negatives}} = \frac{FP}{FP + TN}$$

## Precision:

**Precision** is the proportion of all the model's positive classifications that are actually positive. It is mathematically defined as:

$$\text{Precision} = \frac{\text{correctly classified actual positives}}{\text{everything classified as positive}} = \frac{TP}{TP + FP}$$

| Metric | Guidance |
|---|---|
| Accuracy | Use as a rough indicator of model training progress/convergence for balanced datasets.<br>For model performance, use only in combination with other metrics.<br>Avoid for imbalanced datasets. Consider using another metric. |
| Recall<br>(True positive rate) | Use when false negatives are more expensive than false positives. |
| False positive rate | Use when false positives are more expensive than false negatives. |
| Precision | Use when it's very important for positive predictions to be accurate. |

# Task 01: Insurance Charges Prediction using a Linear Regression model in PyTorch.

## 1. Understanding the Problem

The goal is to predict yearly **medical insurance charges** based on input features such as:

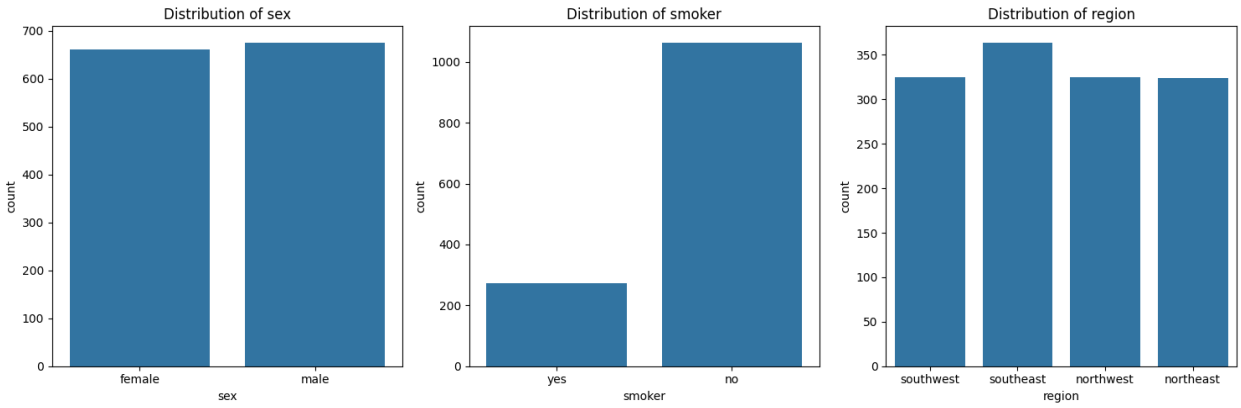
- Age
- Sex
- BMI
- Number of Children
- Smoking Status
- Region

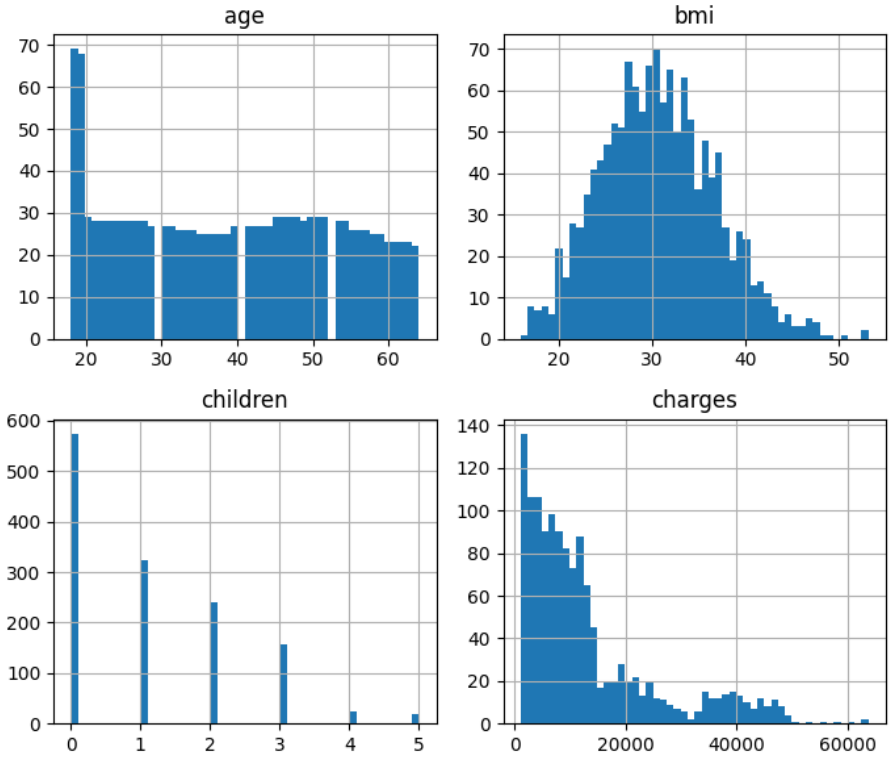
The dataset contains both numerical and categorical variables, with charges as the target variable.

## 2. Dataset Exploration

Initial exploration involved:

- Checking for null/missing values (none found)
- Examining data types and distributions
- Identifying categorical and numerical features
- Visualizing feature distributions




# 3. Preprocessing Steps

To prepare the data for training and ensure optimal model performance, the following preprocessing steps were performed:

## a. Handling Skewed Data

Upon inspecting the distribution of the target column charges, it was found to be **highly right-skewed** with several outliers.

To correct this:

- Applied **log1p transformation**:

**charges=log(1+charges)**

This helped in **normalizing the target distribution**, reducing variance, and improving model convergence.

## b. Categorical Encoding

The dataset includes three categorical features:

- sex
- smoker
- region

These were encoded using two techniques:

- Label Encoding: Converts categories into integers.

OR

- pd.factorize(): Automatically assigns integer codes to unique labels.

  I used Factorize method both work same.

## Impact:

Label encoding allows the model to interpret categorical inputs numerically. However, it may introduce ordinal relationships that don't exist. For this regression task, label encoding is acceptable because the model treats them as separate dimensions with learnable weights.

## c. Standard Scaling

Used **StandardScaler** from scikit-learn to standardize:

- All input features
- The target column charges (after log1p)

Standard scaling:

- Centers data around 0 (mean = 0)

- Scales it to unit variance (std = 1)

**Impact**:

- Improves numerical stability of training
- Ensures that features with different scales don't dominate the learning process

### d. Target Scaling and Inverse Transformation

Since the target variable (charges) was also scaled, we ensured:

- Predictions are made in the **normalized space**
- Final results are **inversely transformed** using StandardScaler.inverse_transform() and np.expm1() to bring them back to **original dollar amounts**

## ⚠️ Challenges & Solutions

### Challenge 1: Highly Skewed Target

- The charges column had extreme outliers and a right-skewed distribution.
- **Solution**: Used log1p() transformation to normalize it, resulting in faster convergence and reduced error variance.

### Challenge 2: Skewed Input Features

- Features like bmi and children were unevenly distributed.
- **Solution**: Standardized all inputs to ensure equal contribution to model training.

### Challenge 3: Categorical Variables

- Model can't work directly with text values like "male" or "southeast".
- **Solution**: Used Label Encoding and Factorization to convert them into numeric codes without creating high-dimensional one-hot vectors.

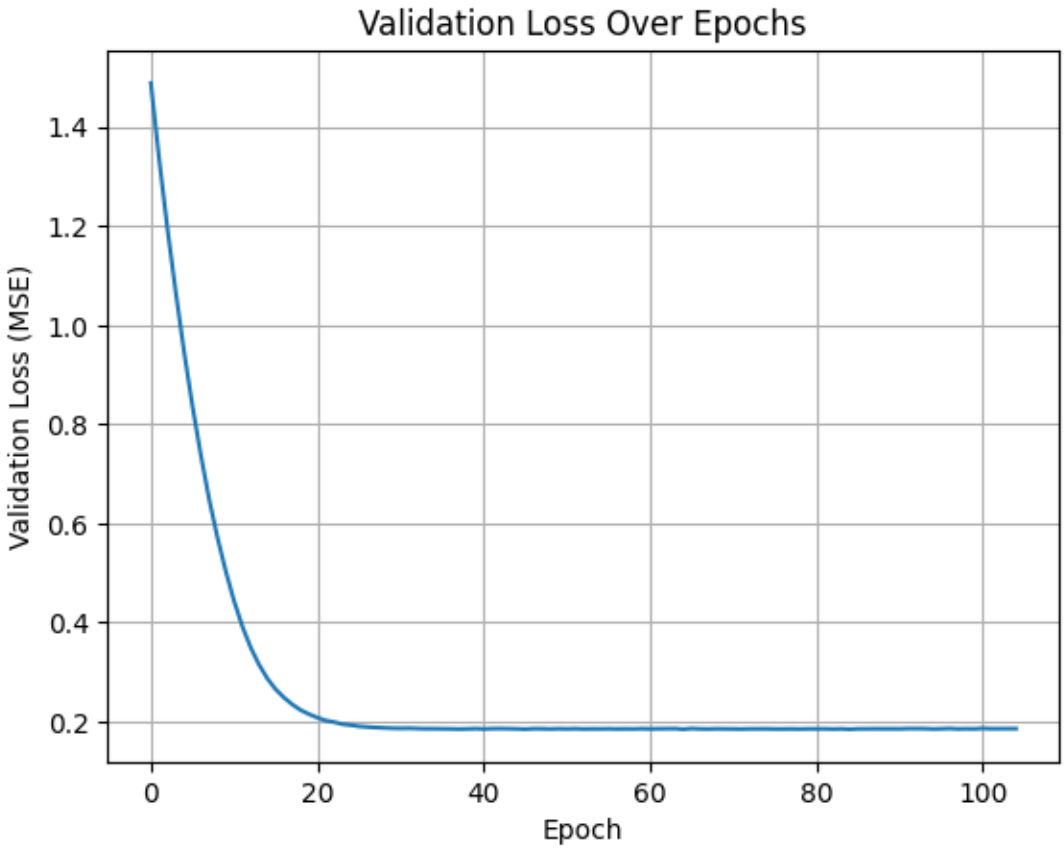
### Challenge 4: Scaling Target Variable

- Applying standard scaling to the target helped keep loss values small and training stable.
- **Solution**: During prediction, the target was **inversely scaled** and **exponentiated** to get the actual insurance charges.

## ✅ Outcome

After these preprocessing steps, the Linear Regression model was trained successfully using PyTorch.
Final model evaluation on the validation set showed:

- **Good convergence**
- **Low validation loss**
- **$R^2$ score > 0.80**, indicating strong predictive performance



# Task 02: SMS Spam Detection using Logistic Regression

## Objective:

The goal of this task was to build a binary text classification model to predict whether an SMS message is **spam** or **ham (non-spam)** using **Logistic Regression** implemented in **PyTorch**.

**Project Flow:**

### 1. Dataset Loading & Exploration

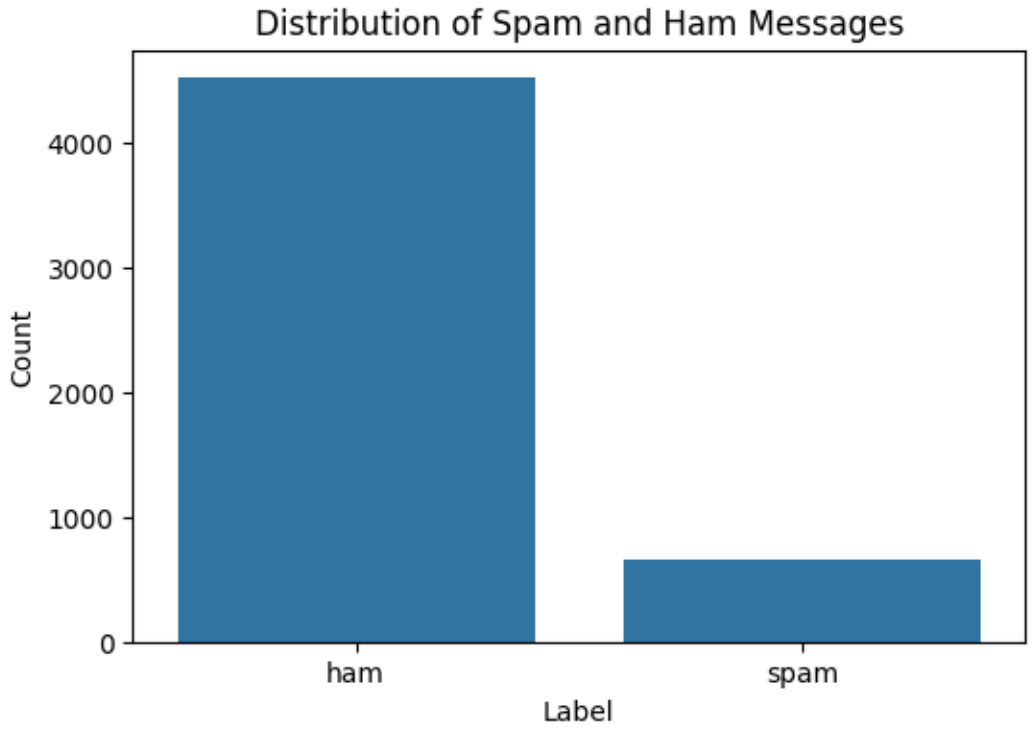
- Loaded the spam.csv dataset in Google Colab.
- Retained only two columns: label and text.
- Converted labels:
    - ham → 0
    - spam → 1
- Performed **EDA**:
    - Checked class distribution → found data is **imbalanced**.
    - Visualized text lengths and class counts.

## 2. Data Preprocessing

- Converted SMS text into numerical format using **TF-IDF Vectorizer**.
- Limited features to **3000 most important terms** to reduce sparsity and dimensionality.
- Split the data into training and validation sets (**85% train / 15% val**).
- Converted the data to PyTorch Tensors using a custom Dataset class and DataLoader.

## 3. Handling Imbalanced Target

The dataset had a **majority of "ham" messages**, and a **minority of "spam" messages**, which can cause bias toward predicting "ham".

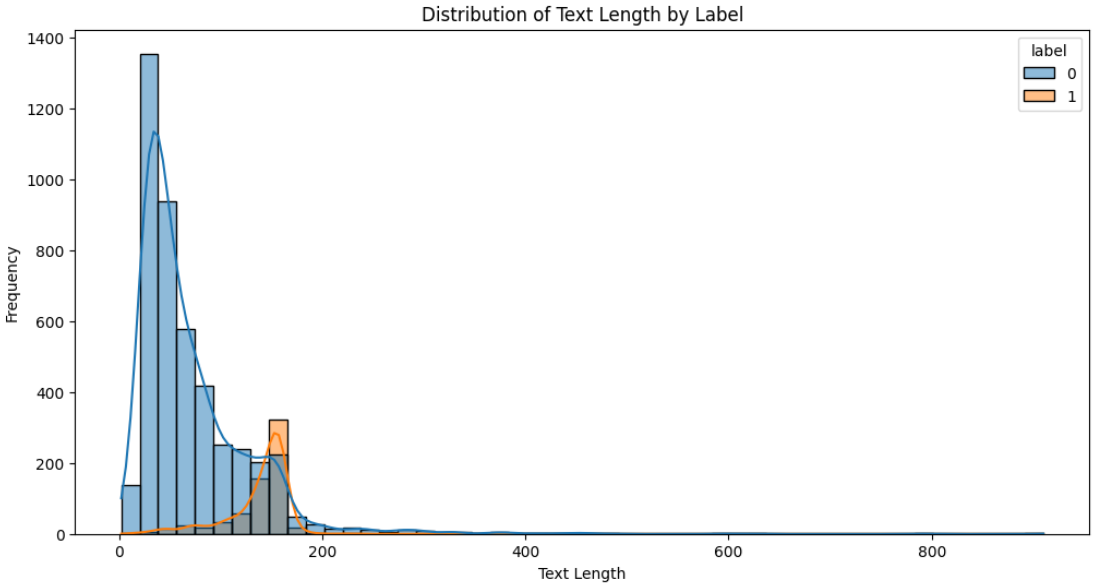



Solution Implemented:

- Used BCEWithLogitsLoss with **pos_weight** to give higher importance to the spam class:

$$\text{pos\_weight} = \text{ham\_count} / \text{spam\_count}$$

- This approach ensures balanced learning without duplicating or removing samples.




## 4. Model Architecture

- Implemented a **Logistic Regression model** using nn.Linear(input_dim, 1)
- Skipped sigmoid inside the model to use BCEWithLogitsLoss for better stability
- Model predicts logits which are later converted to probabilities using torch.sigmoid() during evaluation

## 5. Training

- Trained for **100 epochs**
- Used the **Adam optimizer** with learning rate 1e-3
- Tracked **training and validation loss** across epochs
- Plotted the loss curve to confirm convergence

## 6. Evaluation

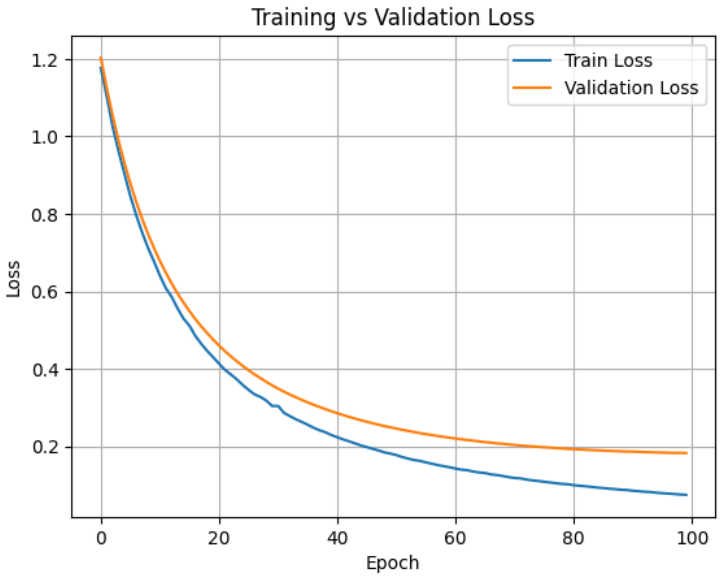
- Applied sigmoid to the outputs to convert logits to probabilities
- Thresholded predictions at 0.5
- Calculated:
    - **Accuracy**
    - **Precision**
    - **Recall**

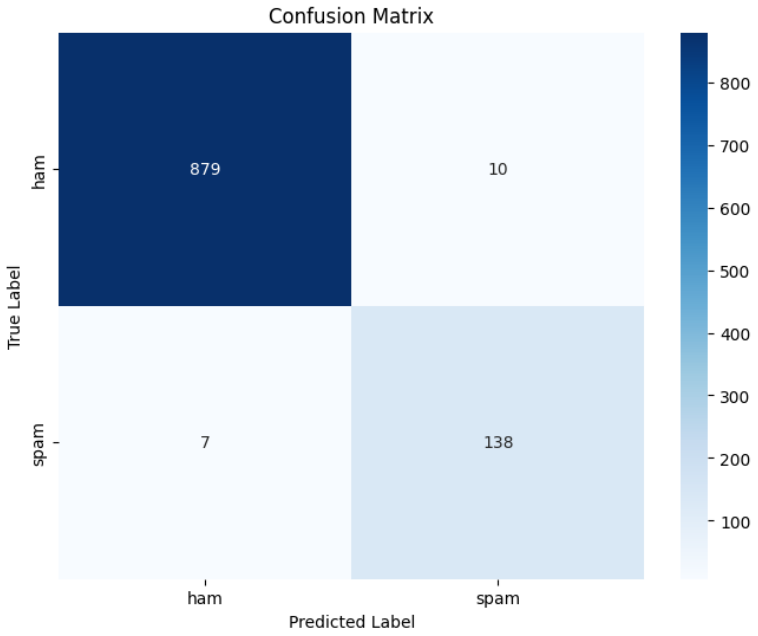
- **F1 Score**
- **Confusion Matrix**

## Outcome:

| Metric | Score |
|---|---|
| Accuracy | **98.0%** |
| Precision | High |
| Recall | High |
| F1 Score | High |

- The model was able to classify SMS messages with excellent performance.
- Achieved **98% accuracy after 100 epochs** despite class imbalance.






# Additional Learning

I am not familiar with the pytorch framework so I start learning pytorch framework.

I learned Basic of pytorch from this YouTube channel playlist:
https://youtu.be/QZsguRbcOBM?si=3ZIJFZf2oC3PoCdR

I also learned about the object-oriented programming in python:

https://www.rithmschool.com/courses/python-fundamentals-part-2/