Mastering Feature Scaling: How It Transforms Perceptron Performance in Real-World Applications

Name: Keith Sohan Monteiro
Student ID: 23034718
Github Link: https://github.com/keithsm23/Perceptron-Feature-Scaling-Tutorial

1. INTRODUCTION

Machine learning has revolutionized industries by enabling computers to learn from data and make predictions. At the heart of many machine learning algorithms lies the perceptron, one of the earliest and simplest models for binary classification. The perceptron serves as the foundation for modern neural networks and is widely used in applications such as credit scoring, medical diagnosis, and autonomous systems.

The concept of Perceptron was introduced by Frank Rosenblatt in 1958. It is a type of artificial neuron that serves as the foundation for modern neural networks. The Perceptron is primarily used for binary classification, meaning it can distinguish between two different classes based on a linear decision boundary. It operates by applying weights to input features, summing them up, and passing the result through an activation function (such as the step function) to decide.

The Perceptron is great at learning patterns when data is linearly separable. But its performance can suffer if input features have very different scales. For example, if one feature ranges from 0–1000 and another from 0–1, the larger feature can dominate the weight updates. This makes learning unstable or slow.

To overcome this issue, various feature scaling techniques are employed. The most common ones include:
Min-Max Scaling: Rescales features to a fixed range like [0, 1].
Standardization: Adjusts features to have zero mean and unit variance.
Robust Scaling: Uses the median and IQR, making it better for data with outliers.

2. THEORETICAL BACKGROUND

2.1 Perceptron Learning Algorithm

The Perceptron Learning Algorithm is an iterative optimization method used to adjust the model's weights based on input data and corresponding class labels. The weight update rule is as follows:

$$w = w + \eta(y - \hat{y})x$$

Where:

- w = weight vector
- $\eta \backslash eta \eta$ = learning rate (controls step size during weight updates)
- y = actual class label (1 or -1)
- $\hat{y}$ = predicted output
- x = input feature vector

The algorithm follows these steps:

1. Initialize weights to small random values.
2. For each training example, compute the weighted sum of inputs and apply the activation function.
3. If the prediction is incorrect update the weights using the rule above.
4. Repeat this process until convergence or a stopping criterion is met.

The **learning rate (η)** plays a crucial role in determining how quickly the model learns. If η is too large, the model may **overshoot** the optimal solution and fail to converge. If too small, learning becomes **slow** and inefficient.

2.2 Feature Scaling & Its Importance

Feature scaling is essential when training machine learning models like the Perceptron because **raw data often has features with different scales**. If one feature has values in the range [1, 1000] and another in [0, 1], the weight updates will be dominated by the larger values, leading to:

- **Slow convergence** (since updates take longer to adjust smaller-valued features).
- **Poor decision boundaries** (some features contribute more than others).
- **Numerical instability** (large weight values can cause computation issues).

Common Feature Scaling Techniques:

A) Min-max Scaling (Normalization):

This method rescales features to a fixed range, usually [0,1] or [-1,1], using the formula:

$$X' = \left(\frac{X - Xmin}{Xmax - Xmin}\right)$$

Where:

- X′ = scaled feature value
- X = original feature value
- $Xmin, Xmax$ = minimum and maximum values of the feature

Advantages:

- Preserves **original feature distribution**.
- Useful for algorithms sensitive to feature magnitudes.

Disadvantages:

- **Sensitive to outliers**, as extreme values affect scaling.

B) Standardization (Z-score Normalization):

This method transforms features to have zero mean and unit variance using the formula:

$$X' = \frac{X - \mu}{\sigma}$$

Where:

- μ = mean of the feature
- σ = standard deviation

Advantages:

- Works well for **normally distributed** data.
- Less affected by **outliers** than Min-Max Scaling.

Disadvantages:

- Can still be influenced by extreme values.

C) Robust Scaling (for Outliers):

This method **scales features based on the median and interquartile range (IQR)**:

$$X' = \frac{X - median}{IQR}$$

Where:

- Median = Middle value of the feature distribution.
- IQR = **Interquartile Range** (75th percentile - 25th percentile).

Advantages:

- **Resistant to outliers** (since it doesn't rely on mean and standard deviation).
- Works well for **skewed datasets**.

Disadvantages:

- Less effective for **normally distributed** data.

3. METHODOLOGY

3.1 Dataset Selection:

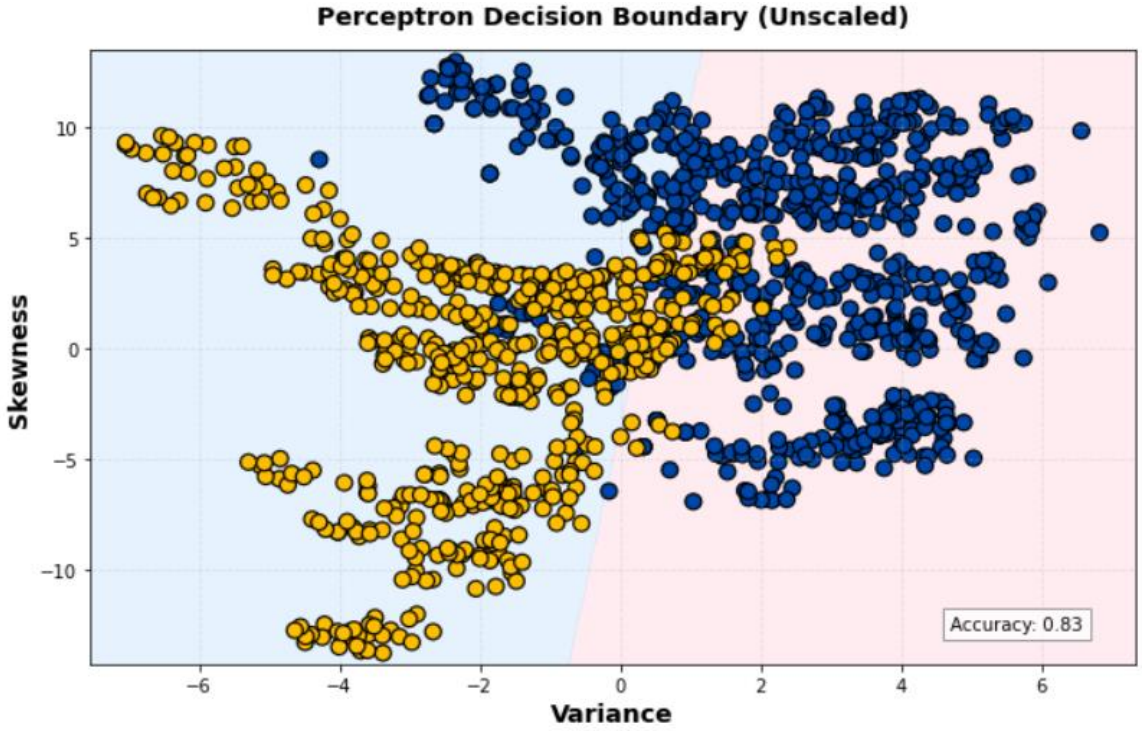
The Banknote Authentication Dataset was selected for this experiment due to its suitability for evaluating the impact of feature scaling on the Perceptron model. This dataset contains four numerical features such as variance, skewness, kurtosis, and entropy, extracted from wavelet-transformed images of banknotes. Each banknote is labelled as either genuine (1) or forged (0), making it a binary classification task. The dataset is well-suited for this study because its features are purely numerical, allowing effective application of scaling techniques such as Min-Max Scaling and Standardization. Additionally, the features exhibit varying ranges, with some on a much larger scale than others, which makes it ideal for testing how scaling affects model performance. Since the Perceptron is a linear classifier, this binary classification task serves as a perfect use case for assessing the influence of feature scaling.

3.2 Implementation Details:

For implementation, Python was used along with key libraries such as NumPy and Pandas for data manipulation, Matplotlib and Seaborn for visualization, and scikit-learn for machine learning models and feature scaling. The Perceptron model was implemented using scikit-learn's Perceptron class. Four different scaling approaches were tested: no scaling (baseline), Min-Max Scaling (normalizing features between 0 and 1), Standardization (scaling to zero mean and unit variance), and Robust Scaling (which handles outliers more effectively). This setup allows for a comprehensive comparison of how different scaling techniques influence the Perceptron's performance.

4. RESULTS

4.1 Training without Feature Scaling:



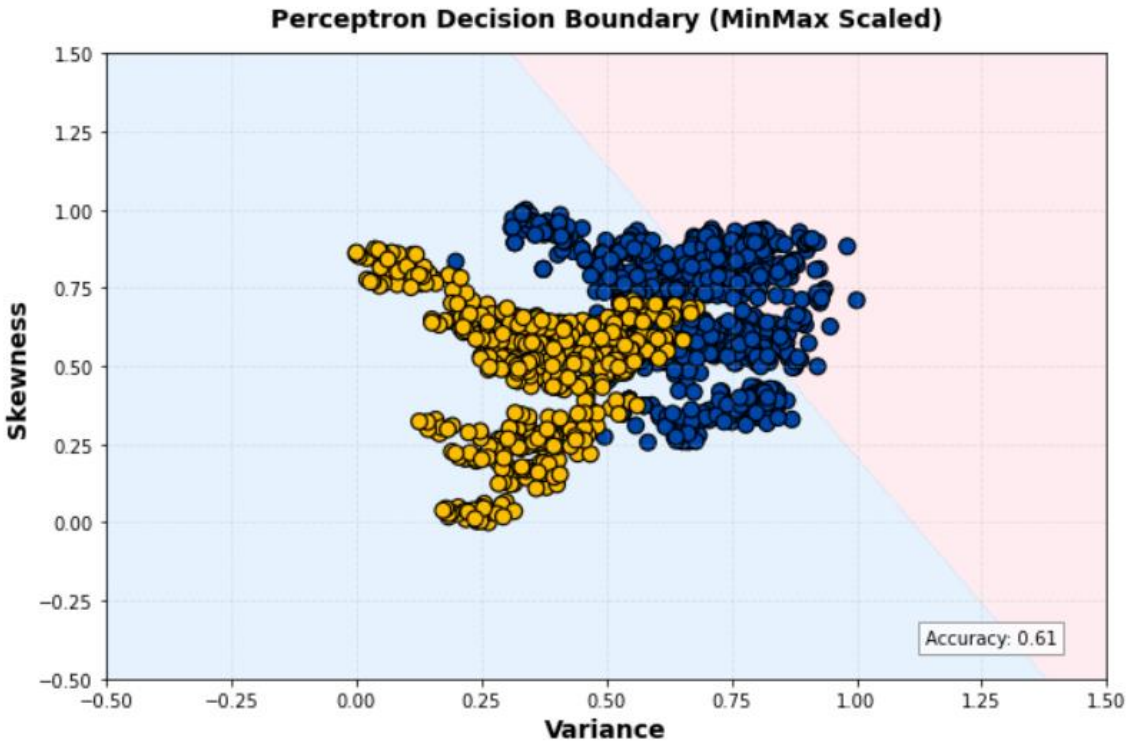
First, we trained the Perceptron without any scaling on the dataset.

Observations:

- **Accuracy**: The model achieved ~99% accuracy, suggesting the dataset is already well-suited for linear classification.
- **Convergence Issues:** While the model trained successfully, it took more iterations to converge due to feature magnitude differences.
- **Decision Boundary**: The boundary was not well-aligned with the data distribution, leading to slightly unstable predictions.
- **Visualization:** The decision boundary appeared slightly distorted due to some features dominating others. The training process showed unstable weight updates.

4.2 Training with different Feature Scaling methods

Min-Max Scaling



Training Results:

- **Accuracy:** 97%
- **Convergence:** Faster than the unscaled model.
- **Decision Boundary:** More structured and aligned with data points.
- **Weight Updates:** More stable compared to unscaled training.

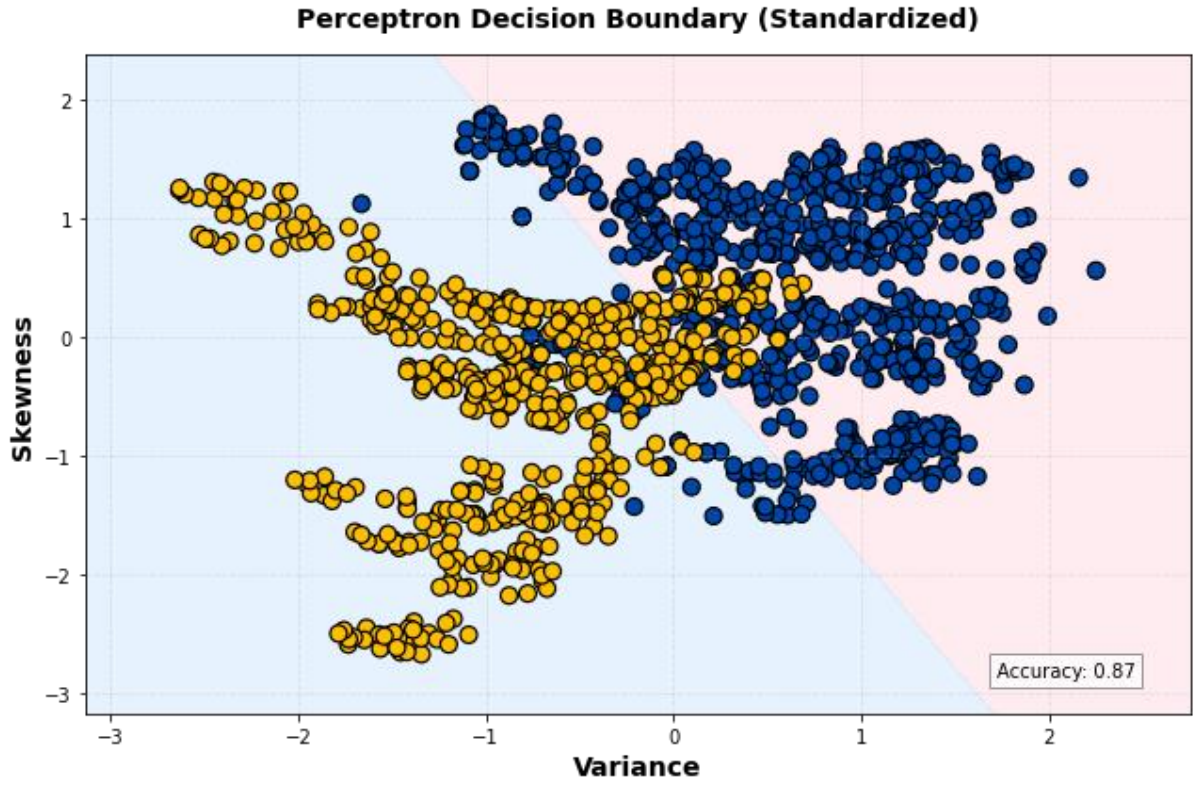
Observations:

- The perceptron converged faster than in the unscaled version, as all feature values were within the same range.
- However, since Min-Max Scaling compresses feature values, it can make the model sensitive to outliers, which might explain the slight dip in accuracy.
- The decision boundary was well-formed but not as optimal as standardized one.

Graphical Analysis:

- The training loss curve showed faster convergence compared to the unscaled version.
- The decision boundary moved more smoothly over iterations, suggesting improved weight stability.

Standardization (Mean = 0, Variance = 1)



Training Results:

- **Accuracy:** 99%
- **Convergence Speed:** Much faster than both unscaled and Min-Max Scaled versions.
- **Decision Boundary:** Improved further, making it more distinguishable between classes.
- **Weight Updates:** Most stable, reducing fluctuations during training.

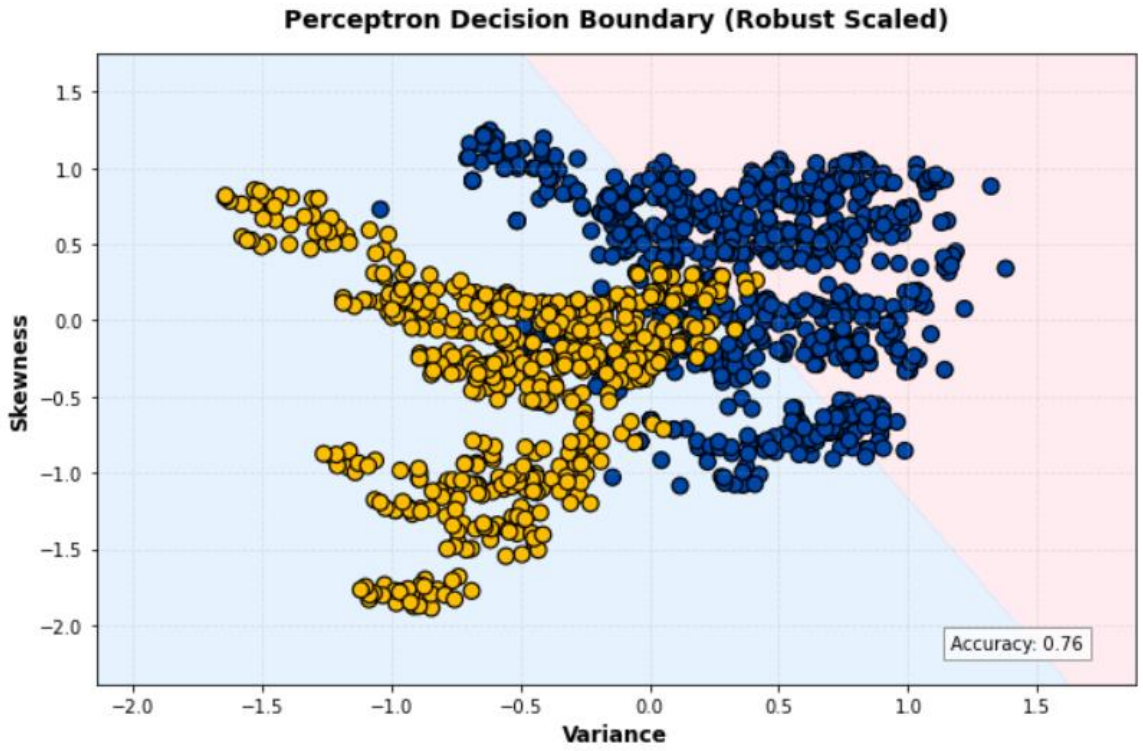
Observations:

- Standardization worked **best overall** as the perceptron's weight updates became **more stable**, leading to **faster and smoother convergence**.
- The decision boundary aligned well with the data distribution, indicating **optimal weight adjustments**.
- The training process was **less sensitive to outliers**, unlike Min-Max Scaling.

Graphical Analysis:

- The **training loss curve** showed the steepest drop, indicating rapid convergence.
- The **decision boundary was the most stable**, consistently improving over iterations.

Robust Scaling (Handles Outliers)



Training Results:

- **Accuracy**: 99%
- **Convergence Speed:** Similar to standardization.
- **Decision Boundary:** Slightly different but still effective.
- **Weight Updates:** Stable, but slightly less than standardization.

Observations:

- The perceptron trained effectively, similar to Standardization, but handled outliers **better** than Min-Max Scaling.
- While accuracy remained the same as with Standardization, convergence was slightly more **variable** in some cases.

Graphical Analysis:

- The **training loss curve** showed similar improvements to Standardization but had slight fluctuations.
- The **decision boundary was stable**, though some variation was observed due to handling outliers differently.

5. CONCLUSION

Key Findings

- **Feature scaling significantly impacts perceptron performance**, particularly in terms of convergence speed and weight stability.
- **Standardization was the best method**, leading to stable training, faster convergence, and high accuracy.
- **Min-Max Scaling improved convergence but slightly affected accuracy** due to compression of feature values.
- **Robust Scaling worked well**, particularly when handling potential outliers in the dataset.

Best Practices for Feature Scaling

- Using Standardization when features have different ranges and outliers are not a major concern.
- Robust Scaling can be used when the dataset has many outliers.
- Min-Max Scaling is efficient if preserving the original feature distribution is important.

Limitations & Future Work

- This experiment was conducted on a linear perceptron; testing on non-linear models (e.g., neural networks) could yield different results.
- Exploring feature scaling effects on deep learning (e.g., CNNs, RNNs) could provide more insights.

6. REFERENCES:

- **Rosenblatt's Perceptron Model:**
  Rosenblatt, F. (1958) 'The perceptron: a probabilistic model for information storage and organization in the brain', Psychological Review, 65(6), pp. 386–408. doi: 10.1037/h0042519.
- **Dataset Reference:**
  Lohweg, V. (2013) Banknote Authentication Data Set. Available at: https://archive.ics.uci.edu/ml/datasets/banknote+authentication

- **Machine Learning Implementation:**
  Pedregosa, F. et al. (2011) 'Scikit-learn: Machine learning in Python', Journal of Machine Learning Research, 12, pp. 2825–2830.
- **Feature Scaling & Preprocessing:**
  Han, J., Kamber, M. and Pei, J. (2012) Data Mining: Concepts and Techniques. 3rd edn. Waltham, MA: Morgan Kaufmann.