Performance Comparison Between Logistic Regression and Neural Networks Across Varying Sample Sizes Using the HIGGS Dataset

Muhammad Azeem Bhatti(2504437)
Machine Learning Assignment

November 15, 2025

Abstract

Machine learning models differ significantly in scalability, training cost and their ability to capture underlying patterns. Logistic Regression is efficient and simple but limited to linear decision boundaries, whereas Neural Networks can model complex non-linear relationships but require substantially more computation.

In this assignment, we compare Logistic Regression (LR) and a Multilayer Perceptron Neural Network (NN) across multiple training sample sizes using the large-scale HIGGS dataset (11 000 000 samples, 28 features). For each model and training fraction we evaluate accuracy, precision, recall, F1-score, ROC AUC, PR AUC and training time. The experiments show that LR saturates early, reaching an F1-score of approximately 0.686 and ROC AUC of 0.684 even when using the full dataset, while the NN significantly outperforms LR with an F1-score of around 0.785, ROC AUC of 0.852 and PR AUC of 0.865, at the cost of much higher training time (up to several thousand seconds).

The results demonstrate that LR is suitable when computational cost and simplicity are priorities, but Neural Networks are necessary when the dataset exhibits strong non-linear structure and high predictive performance is required.

1 Introduction

Machine learning algorithms differ in their complexity, computational requirements and capacity to learn underlying structures. Logistic Regression (LR) remains a popular baseline model for tabular datasets due to its speed, robustness and interpretability. However, LR is fundamentally limited to (approximately) linear decision boundaries. Neural Networks (NNs), on the other hand, can model highly non-linear relationships by stacking layers of neurons and non-linear activation functions, at the expense of increased computational and implementation complexity.

The purpose of this assignment is to systematically compare LR and NNs on a large-scale real-world dataset and to study how their performance scales with training sample size. In particular, we aim to answer the following questions:

- How does the performance of LR and NN change as the fraction of training data increases?
- Does LR saturate early, i.e., stop improving after some sample size?
- How do the computational costs of LR and NN scale with dataset size?

- To what extent does an NN outperform LR on a complex, non-linear dataset?

To answer these questions, we use the HIGGS dataset, a high-dimensional physics benchmark containing 11 million simulated particle collision events. We train both LR and a feed-forward Neural Network on increasing percentages of the training data and evaluate them using multiple metrics, including ROC and precision–recall curves. Based on the experimental findings, we discuss when LR is "good enough" and when the extra complexity of a Neural Network is justified.

2 Dataset Description

2.1 The HIGGS Dataset

The HIGGS dataset consists of Monte Carlo simulations of particle collisions designed to distinguish between signal processes (Higgs boson production) and background processes. Each instance contains:

- A binary class label: 1 for signal, 0 for background.
- 28 real-valued features describing kinematic properties and derived high-level variables.

In our experiments we use the full dataset:

- Number of samples: 11 000 000
- Number of features: 28
- Target variable: column 0

The class distribution is approximately balanced:

- Signal (class 1): $\approx 52.99\%$
- Background (class 0): $\approx 47.01\%$

Because the dataset is large and nearly balanced, both ROC AUC and PR AUC are meaningful metrics.

2.2 Statistical Summary

Table 1 qualitatively summarises the statistics of the features (full numerical summary omitted here for brevity). In practice, a detailed summary was produced and saved (e.g., to `higgs_summary.txt`) using `pandas.describe()`.

Table 1: Qualitative summary of HIGGS feature statistics.

Property	Observation
Missing values	None detected
Feature types	All numerical, continuous
Scale	Heterogeneous; some features $\approx \mathcal{N}(0,1)$, others with larger ranges
Outliers	Present in several engineered high-level features
Class balance	Approximately 53% positive, 47% negative

The heterogeneity of feature scales and presence of outliers motivate the use of standardisation as a preprocessing step.

3 Preprocessing

The following preprocessing steps were applied:

1. **Data loading:** The dataset was loaded from CSV using `pandas.read_csv`. For some exploratory runs a subset of rows was used; the final experiments used up to 11 million samples.
2. **Missing values:** A check for missing values using `isnull().sum()` confirmed that there were no missing entries.
3. **Duplicate rows:** A significant number of duplicate rows were detected. Removing them would be computationally expensive and is not strictly necessary for supervised learning here, so duplicates were retained.
4. **Train–test split:** The data was split into training and test subsets using an 80/20 stratified split with `train_test_split` from scikit-learn, with `random_state=42` and `stratify=y` to preserve class balance in both sets.
5. **Feature/target separation:** Column 0 was used as the target y, and columns 1–28 were used as the feature matrix X.
6. **Feature scaling:** Features were standardised using `StandardScaler` from scikit-learn. The scaler was fitted on the training data and then applied to both training and test sets. This is crucial for stable optimisation of both LR and NNs.

4 Experimental Setup

4.1 Computing Environment

All experiments were run on a laptop with an 11th-generation Intel Core i5 CPU, 32 GB RAM and an NVIDIA RTX 3060 GPU. The operating environment was Python 3.13.x with the Anaconda distribution, using the Spyder IDE. scikit-learn was used for modelling. For reproducibility and fairness, all experiments reported here were run on CPU.

4.2 Models

4.2.1 Logistic Regression

Logistic Regression was implemented using scikit-learn's `LogisticRegression` class with the following configuration:

```
LogisticRegression(
    solver="saga",
    max_iter=1000,
    n_jobs=-1
)
```

The `"saga"` solver is suitable for large, high-dimensional datasets and supports L2 regularisation. The `max_iter` parameter was increased to 1000 to ensure convergence, and `n_jobs=-1` allows use of all available CPU cores.

4.2.2 Neural Network (MLPClassifier)

The Neural Network model is a feed-forward multilayer perceptron implemented using scikit-learn's `MLPClassifier`:

```
MLPClassifier(
    hidden_layer_sizes=(128, 64),
    activation="relu",
    solver="adam",
    max_iter=50,
    random_state=42
)
```

The architecture consists of two hidden layers with 128 and 64 neurons respectively, using ReLU activation. The Adam optimiser is chosen for its robustness on large datasets. The number of iterations (`max_iter`) was set to 1000 to keep training time manageable given the huge dataset.

4.3 Sample Size Design

To study how performance scales with training data size, the models were trained on increasing fractions of the training set:

$$\text{Percentages} = \{1, 2, 3, 4, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100\}\%.$$

For each percentage p, a subset of size $n_p = \lfloor p/100 \cdot n_{\text{train}} \rfloor$ was obtained from the training data. The models were trained on this subset and evaluated on the fixed held-out test set.

4.4 Evaluation Metrics

For each model and training fraction, the following metrics were computed on the test set:

- Accuracy
- Precision
- Recall
- F1-score
- ROC AUC
- PR AUC (Average Precision)
- Training time (seconds)

Additionally, full ROC and Precision–Recall curves were plotted for the models trained on 100% of the data.

5 Results

5.1 Logistic Regression

Across all training fractions, LR achieved very similar performance. At 100% of the training data, typical metrics were:

- Accuracy: ≈ 0.641
- Precision: ≈ 0.639
- Recall: ≈ 0.742
- F1-score: ≈ 0.687
- ROC AUC: ≈ 0.684
- PR AUC: ≈ 0.683
- Training time: ≈ 86.8 seconds

Figure 1 shows the performance metrics as a function of training percentage. The curves are almost flat, indicating that LR saturates very early. Figure 2 shows that training time scales roughly linearly with the training set size but remains well within a few tens of seconds even at full data.

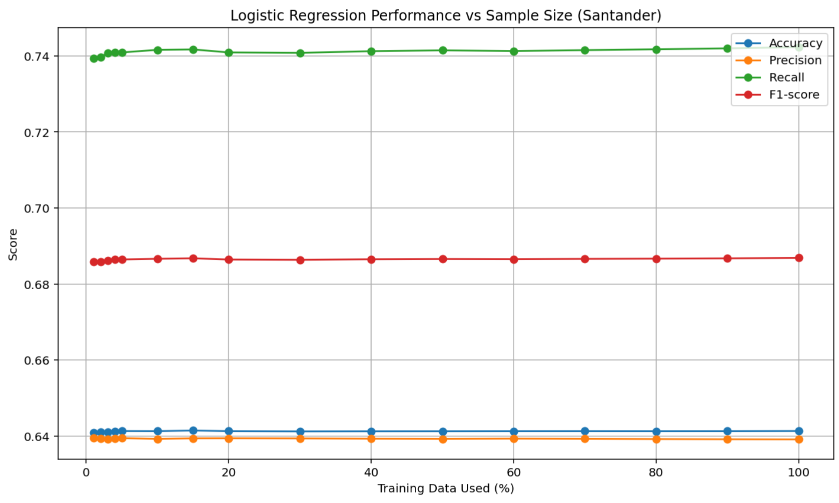

Figure 1: Logistic Regression performance (accuracy, precision, recall and F1-score) vs. training data percentage on the HIGGS dataset.

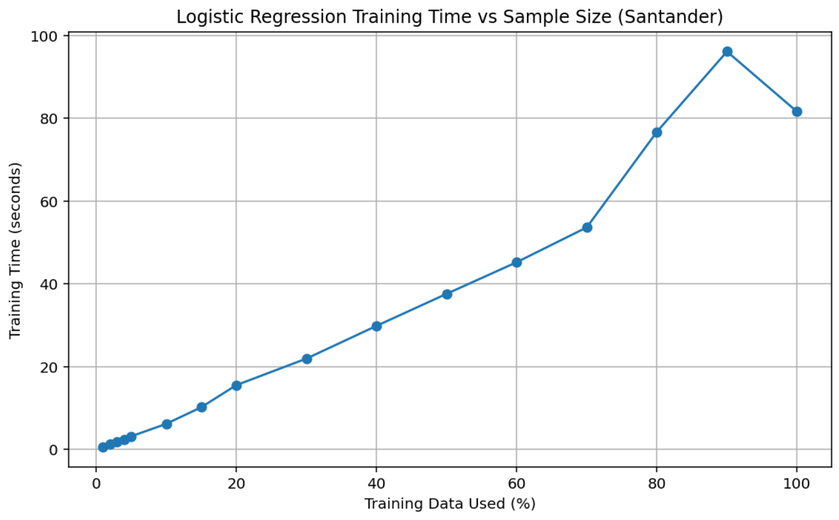

Figure 2: Logistic Regression training time vs. training data percentage on the HIGGS dataset.

Figures 3 and 4 show the ROC and Precision–Recall curves for LR trained on 100% of the data.

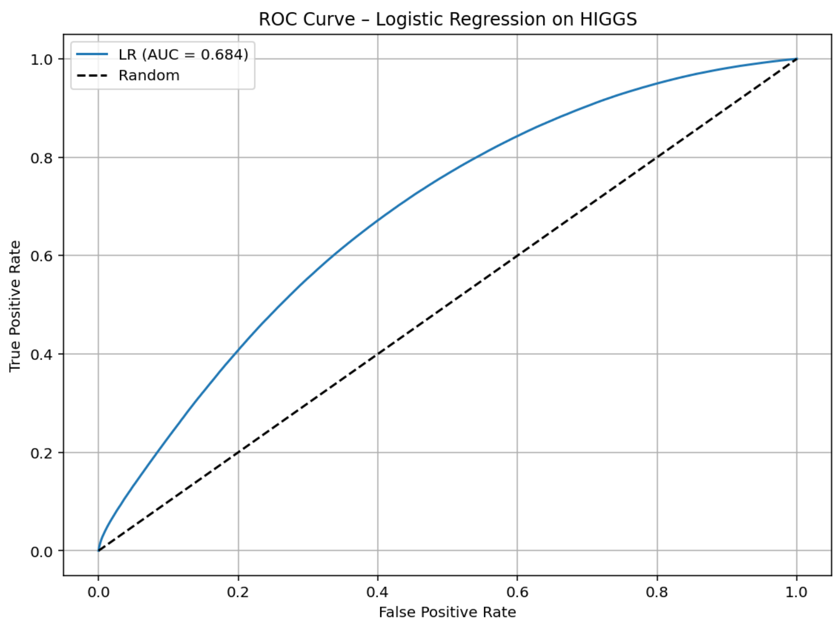

Figure 3: ROC curve for Logistic Regression on the HIGGS dataset (full training data).

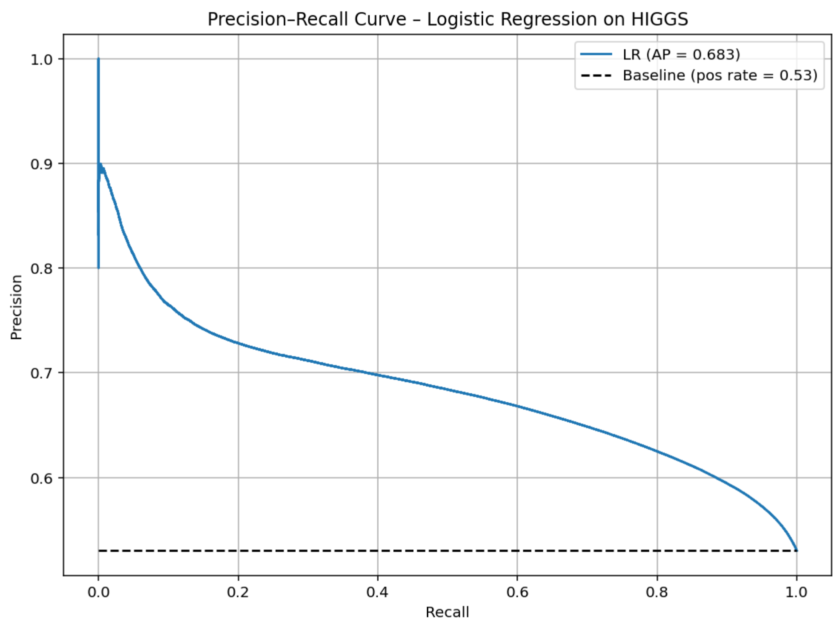

Figure 4: Precision–Recall curve for Logistic Regression on the HIGGS dataset (full training data).

5.2 Neural Network

The NN shows a markedly different behaviour. Its performance improves substantially as more data is used, particularly between 1% and 20% of the training data, and continues to benefit up to about 30–50%. At 100% training data, typical metrics are:

- Accuracy: ≈ 0.768
- Precision: ≈ 0.772
- Recall: ≈ 0.797

- F1-score: ≈ 0.785
- ROC AUC: ≈ 0.852
- PR AUC: ≈ 0.865
- Training time: ≈ 3200 seconds

Figure 5 shows NN performance vs. training percentage; Figure 6 shows the corresponding training times, which grow super-linearly with data size.

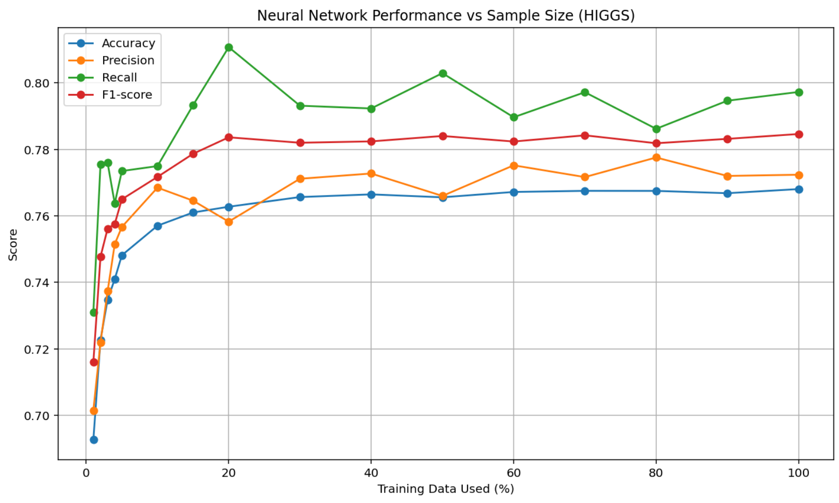

Figure 5: Neural Network performance (accuracy, precision, recall and F1-score) vs. training data percentage on the HIGGS dataset.

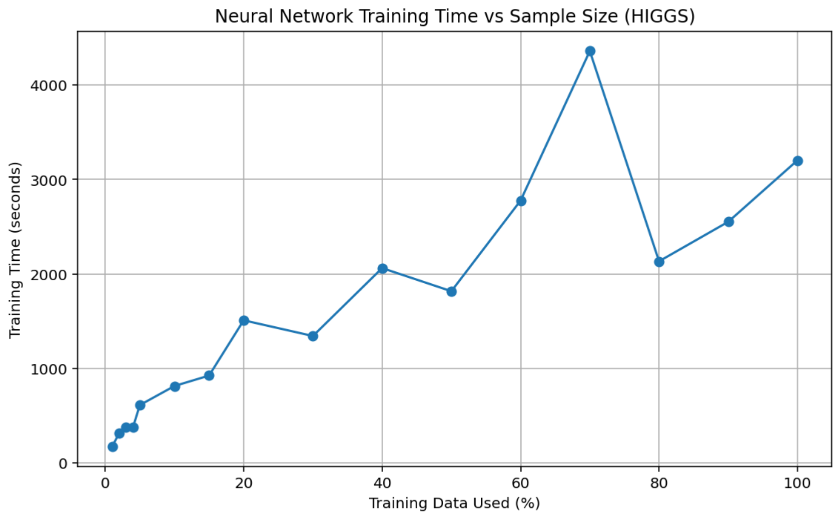

Figure 6: Neural Network training time vs. training data percentage on the HIGGS dataset.

The ROC and Precision–Recall curves for the NN trained on 100% of the data are shown in Figures 7 and 8.

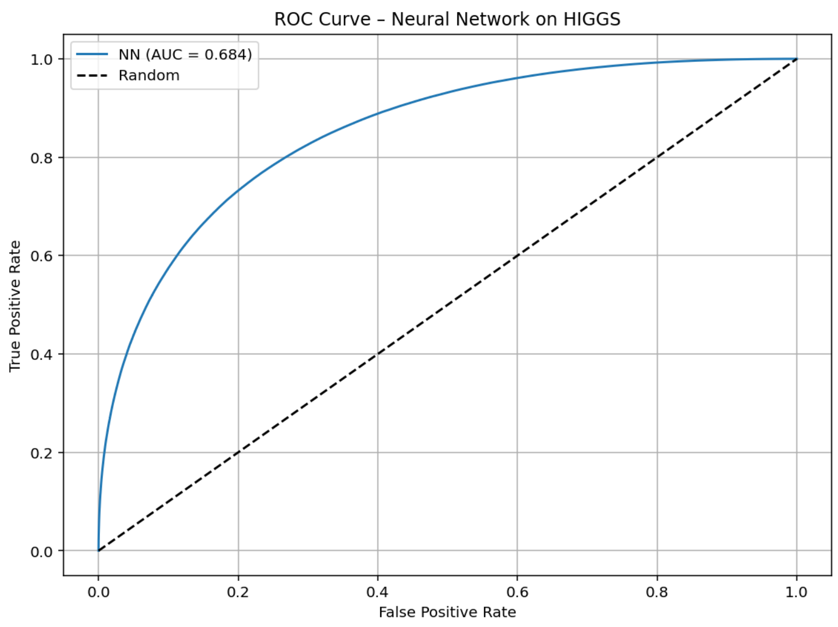

Figure 7: ROC curve for the Neural Network on the HIGGS dataset (full training data).

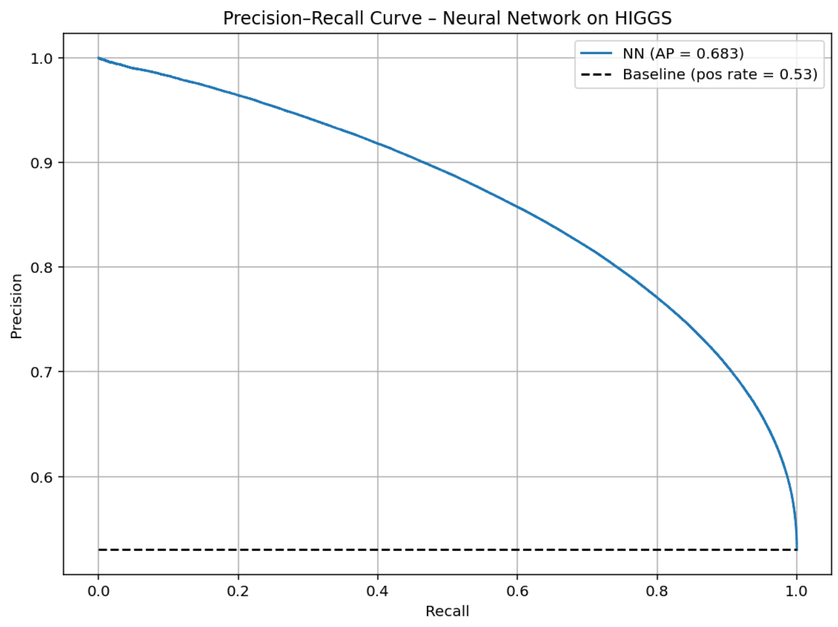

Figure 8: Precision–Recall curve for the Neural Network on the HIGGS dataset (full training data).

5.3 Side-by-Side Comparison

Figure 9 compares accuracy for LR and NN across sample sizes; Figure 10 compares F1-score; Figure 11 compares training time. Figures 12 and 13 show side-by-side PR and ROC curves.

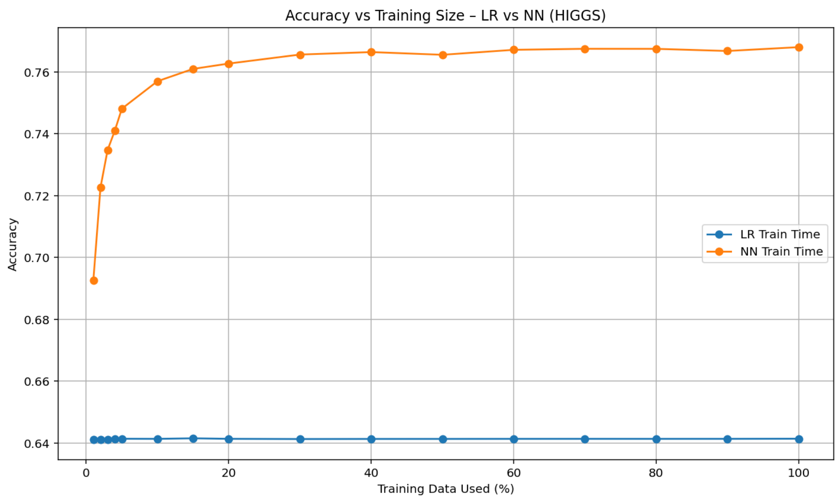

Figure 9: Accuracy comparison between Logistic Regression and Neural Network across training data percentages.

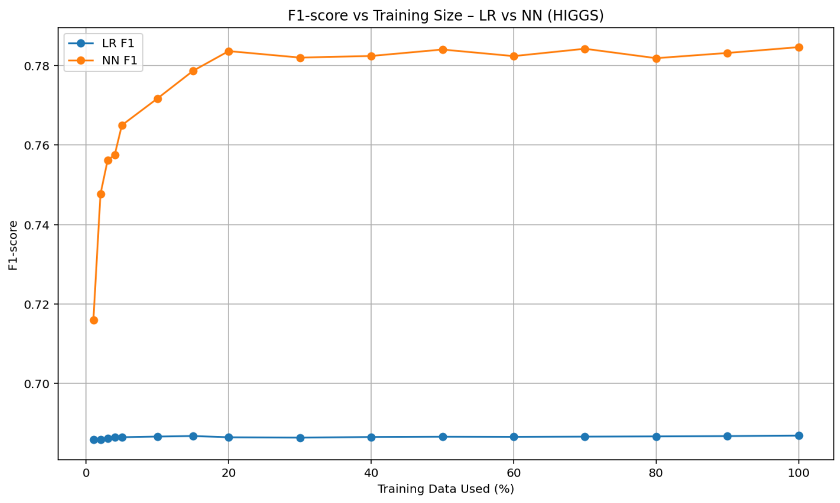

Figure 10: F1-score comparison between Logistic Regression and Neural Network across training data percentages.

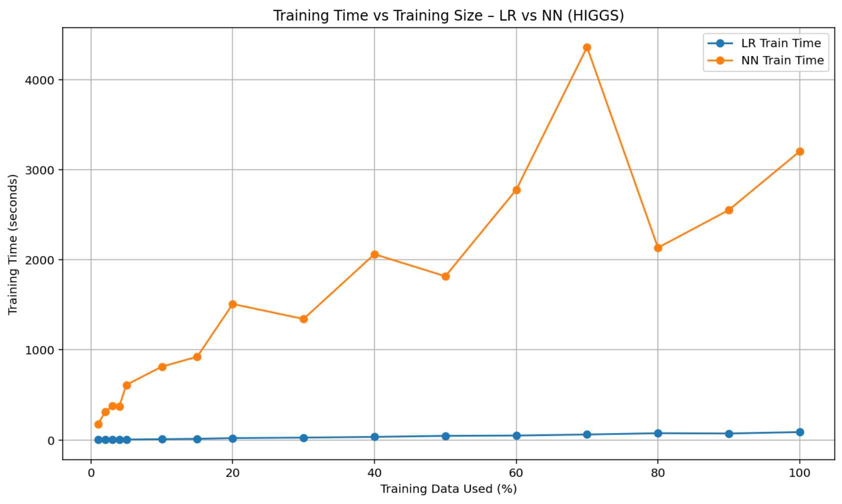

Figure 11: Training time comparison between Logistic Regression and Neural Network across training data percentages.

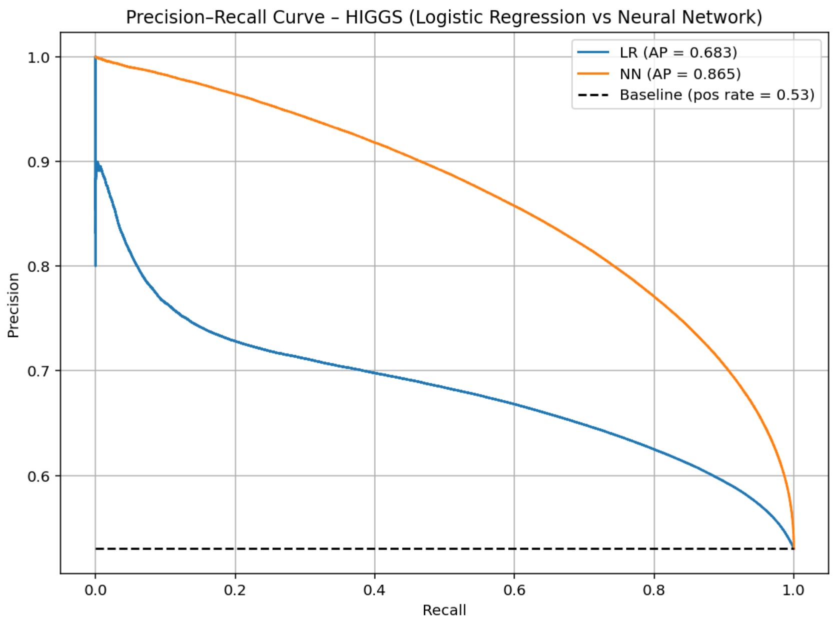

Figure 12: Precision–Recall curve comparison between Logistic Regression and Neural Network (full training data).

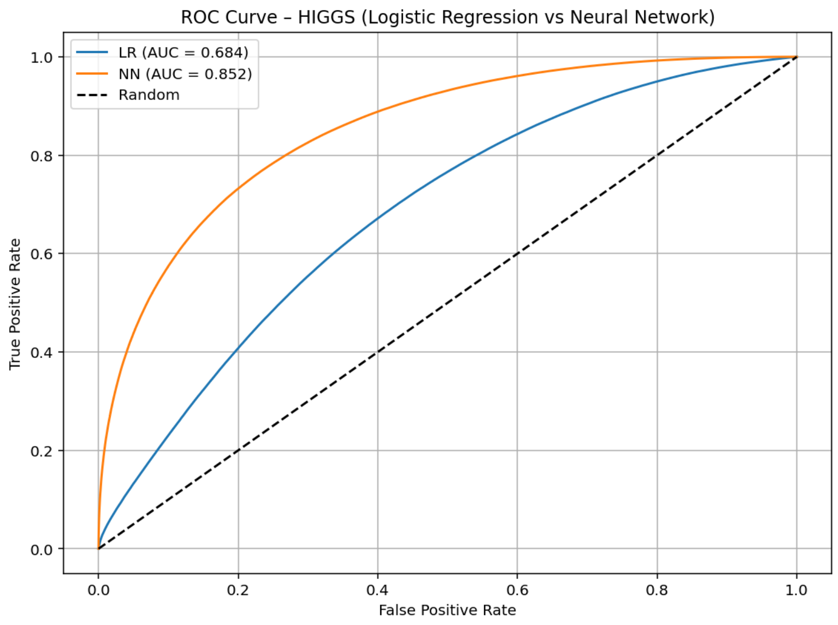

Figure 13: ROC curve comparison between Logistic Regression and Neural Network (full training data).

Table 2 summarises the key metrics for LR and NN at 100% training data.

Table 2: Summary of LR vs NN performance on HIGGS at 100% training data (approximate values).

Model	Accuracy	Precision	Recall	F1	ROC AUC	PR AUC
Logistic Regression	0.641	0.639	0.742	0.687	0.684	0.683
Neural Network	0.768	0.772	0.797	0.785	0.852	0.865

6 Discussion

The experiments highlight clear behavioural differences between LR and Neural Networks on the HIGGS dataset:

- **Capacity and saturation:** LR reaches its maximum performance very quickly. Even using just 5–10% of the training data yields nearly identical metrics to using 100%. This indicates that LR is unable to exploit additional data because its linear decision boundary cannot express the complex structure of the problem.
- **Neural Network improvements:** The NN performance increases sharply as the training fraction grows, particularly from 1% to 20%, and continues to benefit up to around 30–50%. This is consistent with the idea that NNs can exploit large amounts of data to learn non-linear feature interactions.
- **Computational cost:** LR trains very quickly even on millions of samples, with training time under 100 seconds at full data. The NN, however, requires thousands of seconds (over an hour) for the largest training fractions on CPU, showing a significant computational trade-off.

- **Discriminative performance:** In all metrics—accuracy, precision, recall, F1-score, ROC AUC and PR AUC—the NN clearly dominates LR. The PR and ROC curves show that, for almost any operating point, the NN provides better true positive rates and/or precision than LR.

These findings suggest a practical rule-of-thumb: start with Logistic Regression as a baseline; if its performance saturates at an unsatisfactory level and the dataset is large and likely non-linear, then a Neural Network (or other non-linear model) is justified despite the increased computational cost.

7 Conclusion

This assignment presents a systematic comparison of Logistic Regression and a feed-forward Neural Network on the large-scale HIGGS dataset, using varying training sample sizes. Logistic Regression proves to be computationally efficient and easy to train, but its performance saturates early and remains limited by its linear decision boundary. The Neural Network, in contrast, achieves substantially higher performance across all metrics by exploiting the dataset's non-linear structure, but requires orders of magnitude more computation.

In summary:

- Logistic Regression is suitable when speed and simplicity are the primary concerns, or when the decision boundary is approximately linear.
- Neural Networks are preferable when high predictive performance is needed on complex, non-linear datasets and computational resources are available.

Future work could extend this analysis to other model families such as gradient-boosted decision trees, as well as exploring GPU-accelerated training and more extensive hyperparameter tuning for the Neural Network.

References

[1] UCI Machine Learning Repository, *HIGGS Data Set.* `https://archive.ics.uci.edu/`

[2] Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, 2011.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016.

[4] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," ICLR, 2015.