Credit Scoring Algorithm

Giorgio Micaletto, Sofia Congdon, Pietro Maran, Davide Danioni

Andrea Celli | Bocconi University

Contents

1 Introduction 1

2 Data Exploration 1

3 Data Cleaning 1

4 Methods used 1

4.1 Neural Networks 1

4.2 XGBoost 2

4.3 Sparse Projection Oblique Randomer Forest 2

5 Results 2

5.1 Model Interpretability 2

5.2 Model Performance + Choice 3

References 3

1. Introduction

This report presents a comprehensive overview of how we developed a credit scoring algorithm for banks to predict the likelihood of financial distress in clients over the next two years.

Before constructing our model, we explored and cleaned the data handed to us, to facilitate the fitting process and prevent error while fitting or doing inference with our model. We then experimented with Neural Networks, eXtreme Gradient Boosting (XGBoost) and Sparse Projection Oblique Randomer Forest (SPORF) as improvements to our baseline model, a logistic regression. Our findings reveal superior performance in training and validation phases for both XGBoost and SPORF.

2. Data Exploration

For data exploration, we used the `AutoViz` library, which automatically checks for outliers, correlation and gives an overall summary of the data we have. Below, in Figure 1, we have reported the output on our training dataset. As we can see, the data has a lot of outliers, with some correlation between variables, namely `NT3059`, `NT6089`, `NT90DL` (references of shortcuts listed in the appendix), some NaNs in `MonthlyIncome` and `NumberOfDependents`, and one meaningless observation where $\texttt{Age} = -1$. We also note that the data is highly unbalanced, where 0 is the majority class for the target variable with a disproportionately large number of observations compared to class 1. Training our model on this data could distort the model's learning process and lead to biased predictions, thus we want to clean it.

3. Data Cleaning

To address the problems of our dataset, we first started by removing the only observation where $\texttt{Age} = -1$. We decided on this as its target variable was 0 (the majority class) so removing it wouldn't make much difference.

Then, to remove correlation we first set `NT90DL` to be the residuals of the following regression:

$$\texttt{NT90DL} = \texttt{NT3059}, \tag{1}$$

and then set `NT6089` to be the residuals of the regression:

$$\texttt{NT6089} = \texttt{NT90DL} + \texttt{NT3059}. \tag{2}$$

From Figure 2, we can see that we don't have correlation between the variables anymore.

Later, to remove the NaNs, we use SKLEARN's `IterativeImputer` [10], a strategy for imputing missing values, using a Random Forest as a baseline estimator since it is shown to be the best possible estimator [1]. The idea behind `IterativeImputer` is to repeatedly select the best combination columns of the dataset in a round-robin fashion, feed it to the estimator as a predictor, and assign those predictions to the missing values within the other columns.

Lastly, we used SKLEARN's `train_test_split` to split our data into training and validation sets, and use $\texttt{stratify} = \texttt{y}$ to ensure that the distribution of 0s and 1s in our target variable is proportionate between both sets. To balance our training set, we used Synthetic Minority Oversampling Technique (SMOTE) [2] by generating synthetic samples for the minority class.

4. Methods used

Base Model: Logistic Regression

To establish a reliable benchmark for the performance of more advanced models, we chose Logistic Regression as a baseline, due to its simplicity and interpretability (more on its results in the Result section) in binary classification tasks. We implemented Logistic Regression with ℓ_2 regularization and the $\texttt{Newton} - \texttt{Cholesky}$ solver to optimize our solution, which, instead of the standard $\texttt{L} - \texttt{BFGS}$, uses the Cholensky decomposition to approximate the Hessian matrix, thus requiring less memory and computational resources. This model achieved a ROC-AUC score of 0.800, an F1 score of 0.3084 and a Recall score of 0.6473, serving as our baseline for subsequent comparisons with more complex algorithms.

4.1. Neural Networks

As Neural Networks are very sensitive to the scale of the data they are fed, before starting on training our Neural Network, we used SKLEARN's `StandardScaler`, as it has been shown to be the best scaler for balanced data [9], which transforms our data to be

$$z = \frac{x - \mu}{\sigma} \tag{3}$$

where x is our datapoint, μ is the mean and σ is the standard deviation. We then created a custom dataset class and a collation function and trained our model. This model consists of a network with three linear layers, each followed by an `ELU`[3] activation function and a dropout layer, with the number of neurons decreasing progressively across the layers (a detailed graph can be found in Figure 4).

In an effort to enhance performance, we experimented with adding additional layers of different types, such as `Conv1d`, which could have theoretically improved prediction by capturing more complex features and enhancing the model's ability to recognize patterns over different scales [7], as well as increasing the network's depth and width. However, these approaches proved counterproductive, leading to increased complexity without a corresponding improvement in results. Ultimately, the model's ROC-AUC was slightly higher than the baseline, at 0.8177, however the F1 score was much lower at 0.2130, mostly due to the fact that the model had a tendency of predicting 1, as evidenced by its recall score of 0.8441.

4.2. XGBoost

We then transitioned to an eXtreme Gradient Boosting (XGBoost) model. XGBoost is recognized for its efficiency and effectiveness in a variety of machine learning competitions and real-world applications, particularly due to its ability to manage underfitting and overfitting, making it highly suitable for our project [8].

XGBoost is an enhancement of the gradient boosting machine (GBM) framework. It integrates advanced computational techniques including efficient tree pruning, handling of missing values, and built-in regularization which prevents the model from becoming overly complex and overfitting the data. Unlike traditional boosting, where the ensemble's prediction is a simple sum of the predictions from individual models weighted by their accuracy, XGBoost incorporates a more nuanced approach. In traditional boosting methods we have:

$$\hat{f}(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) \tag{4}$$

where:

- $\hat{f}(\mathbf{x})$ denotes the final prediction made by the ensemble.
- $h_t(\mathbf{x})$ indicates the prediction of the t-th model.
- α_t is the weight assigned to the t-th model, typically proportional to its prediction accuracy.

However, XGBoost modifies this approach:

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{T} \gamma_i g_i(\mathbf{x}) \tag{5}$$

with:

- $g_i(\mathbf{x})$ being the contribution of the i-th tree.
- γ_i being the learning rate associated with the i-th tree, moderating the impact each tree has on the final prediction.

The XGBoost model is refined by minimizing an objective function that includes both a gradient-based component and a regularization component:

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} l\left((y_i, \hat{y}^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)\right) + \Omega(f_t) \tag{6}$$

where $\Omega(f_t)$ represents the regularization term that controls overfitting by penalizing the complexity of the model. The notation is explained further in the appendix.

To optimize the hyperparameters of our XGBoost model, we employed Optuna [6], a state-of-the-art hyperparameter optimization framework that uses a Bayesian optimization technique known as a Tree-structured Parzen Estimator (TPE) [11] to explore the given hyperparameter space. After extensive optimization, our model achieved a ROC-AUC score of 0.8655 and an F1 score of 0.3526, but a lower Recall score at 0.2590.

4.3. Sparse Projection Oblique Randomer Forest

We further expanded our exploration by implementing a Sparse Projection Oblique Randomer Forest (SPORF) model. SPORF is an extension of the traditional Random Forest (RF) algorithm that enhances the decision boundary flexibility by using oblique splits instead of the axis-aligned splits typically used in standard RFs.

The key difference in SPORF compared to traditional RF is in how the splits are made:

$$\theta^* = \arg\min_{\theta} \left\{ \min_{\tau} [H(Q_{left}(\tau, \theta)) + H(Q_{right}(\tau, \theta))] \right\} \tag{7}$$

where $Q_{left}(\tau, \theta)$ and $Q_{right}(\tau, \theta)$ are the partitions of the data on the left and right of the split respectively; H denotes the impurity measure used (in our case, the Gini coefficient) and θ represents the oblique projection parameters, making the split decision boundary non-axis-aligned.

SPORF was chosen for its potential to improve performance over XGBoost due to its ability to create more complex decision boundaries and utilize interactions between variables more effectively [4]. The model was fitted with the following parameters:

- `n_estimators`: 1000,
- `max_depth`: 22,
- `'min_samples_split'`: 50,

Although the ROC-AUC score was marginally lower than XGBoost at 0.8566, SPORF achieved a higher Recall score than XGboost at 0.4372 and F1 score at 0.4331. As a side note, the results were obtained from fitting SPORF on the dataset augmented by SMOTE (without correlation effects). If SPORF were to be fitted on the original correlated dataset instead, it would actually perform better than XGBoost would on that same dataset. However, for our task we are only interested in results without correlation.

5. Results

5.1. Model Interpretability

In the previous section, we presented various performance metrics obtained by our models on the validation dataset. Building upon these metrics, this section further explores the interpretability of each model using SHAP (SHapley Additive exPlanations) values [5]. Values are assigned to each predictive outcome to represent the contribution of each feature to that single prediction's log odds.

Figure 3 arranges features in decreasing order based on their influence on predictions for both Logistic Regression and XGBoost. The x-axis plots the features' mean absolute contributions (values between 0 and 1), which disregard directionality. The y-axis feature names are reported in the appendix with details about their contribution values. Unfortunately, SHAP lacks an implementation for PyTorch's Neural Networks and the SPORF model, so we were not able to construct their Global Feature Importance bar plots.

In both models, the most important feature is NT3059, with a high contribution of 0.76 and 0.97 for Logistic Regression and XGBoost respectively. However, the order of importance of the remaining features is completely different between both models. In fact, as we go down Logistic Regression's hierarchy, its feature contributions are relatively small in magnitude (lower than 0.4) and reduce greatly. For XGBoost, RevolvingUtilizationOfUnsecuredLines also plays a substantial role in contributing to its predictive outcomes (0.84), but its other features (different order than Logistic Regression's hierarchy) decrease in importance.

5.2. Model Performance + Choice

Finally, going back to the performance metrics to evaluate which model to choose, we can say that Logistic Regression and Neural Networks exhibit ROC-AUC scores that are not high enough compared to the other two models. Since our main objective is to achieve the highest possible ROC-AUC, we chose XGBoost as our model, making it the best at distinguishing between borrowers who are likely to experience financial distress and those who are not. Although XGBoost scores marginally higher on the ROC-AUC compared to SPORF, the decision between using SPORF or XGBoost is not straightforward. We can see that for a very small trade-off in the ROC-AUC (by approximately 0.01), we gain a substantially higher Recall score (by approximately 0.13) when using SPORF. SPORF's higher F1 and Recall imply a stronger predictive precision (minimising false positives) as well as a superiority in minimising false negatives. For the credit scoring task at hand, minimising false negatives is critical because of the severe financial consequences of failing to identify borrowers who are at high risk of experiencing financial distress (and possibly defaulting). Thus, choosing a model with a higher Recall, like SPORF, would be more appropriate, even if it means sacrificing some precision and potentially lowering the ROC-AUC.

Appendix

Shortcuts for the variable names:

- NT3059 = NumberOfTime30 − 59DaysPastDueNotWorse
- NT6089 = NumberOfTime60 − 89DaysPastDueNotWorse
- NT90DL = NumberOfTimes90DaysLate

Explanation of the objective function of XGBoost:

- y_i is the true label for the i-th instance.
- $\hat{y}^{(t-1)}$ is the prediction from the model for the i-th instance at iteration $t-1$.
- g_i and h_i are the first and second order gradients of the loss function with respect to $\hat{y}^{(t-1)}$, respectively.
- $f_t(x_i)$ is the output of the t-th tree for the i-th instance.
- $\Omega(f_t)$ is the regularization term, defined as:

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{8}$$

- T is the number of leaves in the tree f_t.
- w_j are the weights associated with each leaf of the tree.
- γ and λ are parameters that control the complexity of the tree structure and the magnitude of the weights, respectively.

Feature names in the SHAP Global Feature Importance bar plots:

- Feature 0 = RevolvingUtilizationOfUnsecuredLines
- Feature 1 = age
- Feature 2 = NT3059
- Feature 3 = DebtRatio
- Feature 4 = MonthlyIncome
- Feature 5 = NumberOfOpenCreditLinesAndLoans
- Feature 6 = NT90DL
- Feature 7 = NumberRealEstateLoansOrLines
- Feature 8 = NT6089
- Feature 9 = NumberOfDependents

Mean absolute SHAP values (contributions) of the 3 most important features:

- For Logistic Regression: 0.76 for NT3059; 0.37 for age; 0.23 for NT90DL.
- For XGBoost: 0.97 for NT3059; 0.84 for RevolvingUtilizationOfUnsecuredLines; 0.59 for NumberOfDependents.

References

[1] Pantanowitz, "Missing data imputation through the use of the random forest algorithm", *https://arxiv.org/abs/0812.2412*, 2009.

[2] Chawla, "Smote: Synthetic minority over-sampling technique", *https://arxiv.org/abs/1106.1813*, 2011.

[3] Clevert, "Fast and accurate deep network learning by exponential linear units (elus)", *https://arxiv.org/abs/1511.07289*, 2015.

[4] Tomita, "Sparse projection oblique randomer forests", *https://arxiv.org/abs/1506.03410*, 2015.

[5] Lundberg, "A unified approach to interpreting model predictions", *https://arxiv.org/abs/1705.07874*, 2017.

[6] Akiba, "Optuna: A next-generation hyperparameter optimization framework", *http://arxiv.org/abs/1907.10902*, 2019.

[7] Kiranyaz, "1d convolutional neural networks and applications: A survey", *https://arxiv.org/abs/1905.03554*, 2019.

[8] Shwartz-Ziv, "Tabular data: Deep learning is not all you need", *https://arxiv.org/abs/2106.03253*, 2021.

[9] de Amorim, "The choice of scaling technique matters for classification performance", *https://arxiv.org/abs/2212.12343*, 2022.

[10] Jarret, "Hyperimpute: Generalized iterative imputation with automatic model selection", *https://arxiv.org/abs/2206.07769*, 2022.

[11] Watanabe, "Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance", *https://arxiv.org/abs/2304.11127*, 2023.

Figures

	Data Type	Missing Values%	Unique Values%	Minimum Value	Maximum Value	DQ Issue
SeriousDlqin2yrs	float64	0.000000	0	0.000000	1.000000	No issue
RevolvingUtilizationOfUnsecuredLines	float64	0.000000	NA	0.000000	50708.000000	Column has 595 outliers greater than upper bound (1.35) or lower than lower bound(-0.76). Cap them or remove them.
age	float64	0.000000	NA	-1.000000	103.000000	Column has 37 outliers greater than upper bound (96.00) or lower than lower bound(8.00). Cap them or remove them.
NumberOfTime30-59DaysPastDueNotWorse	float64	0.000000	NA	0.000000	98.000000	Column has 17994 outliers greater than upper bound (0.00) or lower than lower bound(0.00). Cap them or remove them.
DebtRatio	float64	0.000000	NA	0.000000	329664.000000	Column has 23330 outliers greater than upper bound (1.89) or lower than lower bound(-0.85). Cap them or remove them.
MonthlyIncome	float64	19.479790	NA	0.000000	2866005.615861	21846 missing values. Impute them with mean, median, mode, or a constant value such as 123., Column has 3741 outliers greater than upper bound (14690.31) or lower than lower bound(-3649.00). Cap them or remove them.
NumberOfOpenCreditLinesAndLoans	float64	0.000000	NA	0.000000	58.000000	Column has 2986 outliers greater than upper bound (20.00) or lower than lower bound(-4.00). Cap them or remove them.
NumberOfTimes90DaysLate	float64	0.000000	NA	0.000000	98.000000	Column has 6257 outliers greater than upper bound (0.00) or lower than lower bound(0.00). Cap them or remove them., Column has a high correlation with ['NumberOfTime30-59DaysPastDueNotWorse']. Consider dropping one of them.
NumberRealEstateLoansOrLines	float64	0.000000	NA	0.000000	54.000000	Column has 595 outliers greater than upper bound (5.00) or lower than lower bound(-3.00). Cap them or remove them.
NumberOfTime60-89DaysPastDueNotWorse	float64	0.000000	NA	0.000000	98.000000	Column has 5693 outliers greater than upper bound (0.00) or lower than lower bound(0.00). Cap them or remove them., Column has a high correlation with ['NumberOfTime30-59DaysPastDueNotWorse', 'NumberOfTimes90DaysLate']. Consider dropping one of them.
NumberOfDependents	float64	2.560925	NA	0.000000	20.000000	2872 missing values. Impute them with mean, median, mode, or a constant value such as 123., Column has 9987 outliers greater than upper bound (2.50) or lower than lower bound(-1.50). Cap them or remove them.

Figure 1. AutoViz table before (with correlation)

	Data Type	Missing Values%	Unique Values%	Minimum Value	Maximum Value	DQ Issue
SeriousDlqin2yrs	float64	0.000000	0	0.000000	1.000000	No issue
RevolvingUtilizationOfUnsecuredLines	float64	0.000000	NA	0.000000	50708.000000	Column has 595 outliers greater than upper bound (1.35) or lower than lower bound(-0.76). Cap them or remove them.
age	float64	0.000000	NA	20.000000	103.000000	Column has 36 outliers greater than upper bound (96.00) or lower than lower bound(8.00). Cap them or remove them.
NumberOfTime3059DaysPastDueNotWorse	float64	0.000000	NA	0.000000	98.000000	Column has 17993 outliers greater than upper bound (0.00) or lower than lower bound(0.00). Cap them or remove them.
DebtRatio	float64	0.000000	NA	0.000000	329664.000000	Column has 23330 outliers greater than upper bound (1.89) or lower than lower bound(-0.85). Cap them or remove them.
MonthlyIncome	float64	19.479964	NA	0.000000	2866005.615861	21846 missing values. Impute them with mean, median, mode, or a constant value such as 123., Column has 3741 outliers greater than upper bound (14690.62) or lower than lower bound(-3649.22). Cap them or remove them.
NumberOfOpenCreditLinesAndLoans	float64	0.000000	NA	0.000000	58.000000	Column has 2986 outliers greater than upper bound (20.00) or lower than lower bound(-4.00). Cap them or remove them.
NumberOfTimes90DaysLate	float64	0.000000	NA	-12.570443	17.145368	Column has 20997 outliers greater than upper bound (0.15) or lower than lower bound(0.15). Cap them or remove them.
NumberRealEstateLoansOrLines	float64	0.000000	NA	0.000000	54.000000	Column has 595 outliers greater than upper bound (5.00) or lower than lower bound(-3.00). Cap them or remove them.
NumberOfTime6089DaysPastDueNotWorse	float64	0.000000	NA	-11.366346	9.727752	Column has 22780 outliers greater than upper bound (0.07) or lower than lower bound(0.07). Cap them or remove them.
NumberOfDependents	float64	2.560947	NA	0.000000	20.000000	2872 missing values. Impute them with mean, median, mode, or a constant value such as 123., Column has 9987 outliers greater than upper bound (2.50) or lower than lower bound(-1.50). Cap them or remove them.

Figure 2. AutoViz table after removing correlation

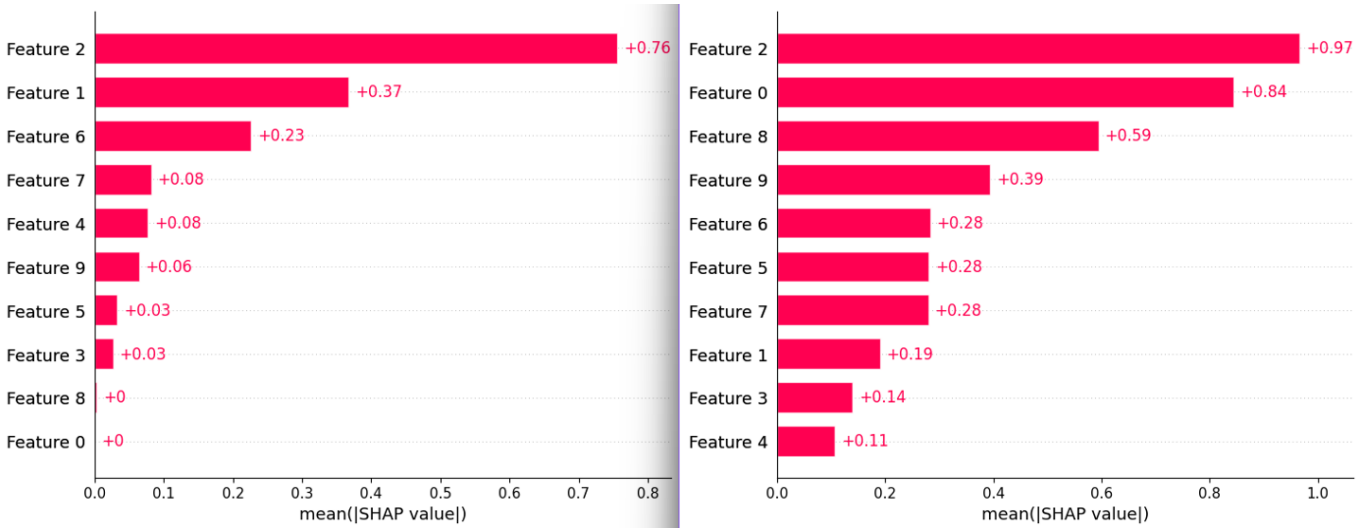

Figure 3. SHAP Global Feature Importance Bar Plots for Logistic Regression (left) and XGBoost (right)

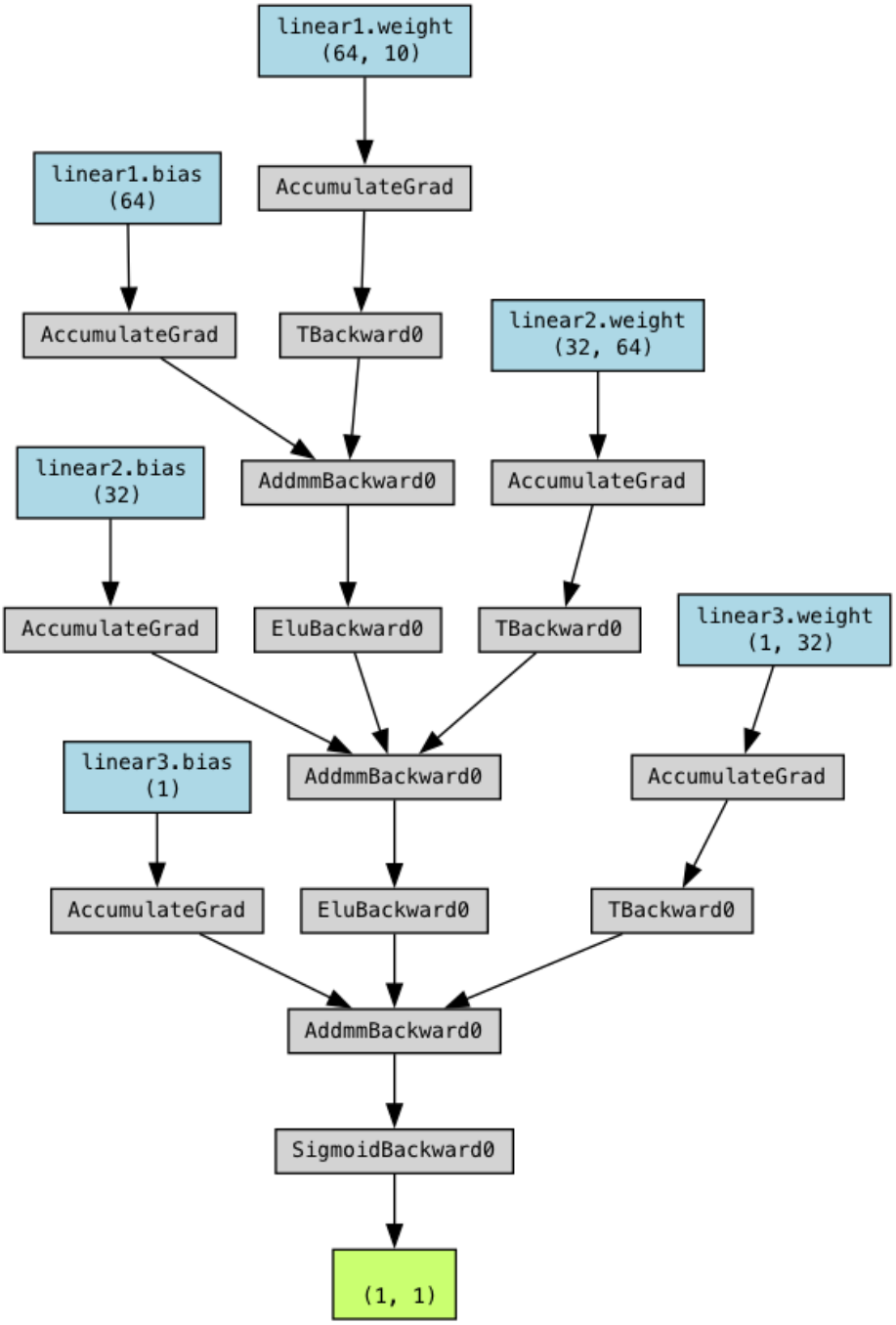

Figure 4. Neural Network Model