The University of North Carolina at Charlotte

Hit or Miss: Game Edition

Nam Pham

ITCS 3156

Dr. Jake Lee

May 7, 2025

1 Introduction

1.1 Problem Statement

Over the past few decades, the video game industry has steadily evolved, profoundly shaping popular culture, entertainment, and consumer behavior. This transformation has created lucrative opportunities for developers and publishers within a rapidly expanding market. According to Newzoo, a market analytics firm focused on games and esports, the video game industry generated $174 billion in global revenue in 2020, reflecting a 19.6% increase from the previous year (qtd. in Ullmann et al.). This growth was fueled in part by the steady release of hundreds of new titles on platforms such as Steam, which continually expanded the volume of content available to consumers. However, this surge in activity also intensified competition, making it increasingly difficult for developers to distinguish their games in an oversaturated market. As a result, many titles underperform commercially due to limited visibility and weak consumer engagement, with only a small fraction reaching major revenue milestones. A game's success or failure often depends on a complex interplay of many factors, including its features, market trends, consumer preferences, and the release timing of competing titles. Understanding these dynamics is critical for maximizing the chances of success in a market that continues to grow more competitive.

1.2 Motivation

Given that game development typically involves substantial capital and resource commitments, timely and accurate insights are invaluable for developers, publishers, and investors. Li et al. noted that traditional sales forecasting methods, relying on historical trends or expert intuition, lack objectivity and struggle to capture complex linear and non-linear relationships (497). In contrast, data-driven approaches powered by machine learning can uncover deeper patterns within the multifaceted factors influencing a game's performance.

These insights can also guide both design and marketing decisions, ultimately enhancing a game's commercial viability.

1.3 Summary of Approach

This project developed and evaluated two machine learning models, logistic regression and a neural network, to predict whether a video game's global sales would exceed one million units. Both models were trained and validated on a curated set of features. Complex relationships between features were accounted for in the analysis, along with an assessment of each feature's relative influence. By identifying the features most strongly associated with successful titles, the study provides actionable insights for optimizing both game design and marketing. Additionally, employing two distinct models enabled an impartial assessment of their respective predictive power, accuracy, and interpretability.

2 Data Preparation

The dataset used in this research was obtained from Kaggle, where it was published by Rush Kirubi. It contained 16,719 records on video game titles, each characterized by sixteen features such as name, platform, release year, genre, publisher, critic and user scores, regional and global sales, developer, and rating. All analysis was conducted in Python to leverage open-source libraries, including Scikit-learn, to support model development, training, and evaluation. These toolsets provided essential modules that streamlined data processing and reduced computational requirements. Initial exploration included inspecting dataset dimensions using the shape attribute.

2.1 Exploratory Visualization

A basic visual analysis, utilizing a pair plot, was conducted to assess correlations among key features. This multivariate plot type was selected for its ability to simultaneously display pairwise relationships and potential class separation. The pair plot (Figure 1) revealed how combinations such as critic score versus user score, and year of release versus critic

score, exhibited visible separation between hit and non-hit games. This supported the expectation that higher review scores and more recent releases were associated with commercial success. In contrast, pairings of genre, platform, and rating showed substantial overlap between classes, with no clear clustering patterns. The diagonal trend between critic and user scores indicated a strong linear correlation, raising concerns about potential multicollinearity that could affect the performance of models like logistic regression.

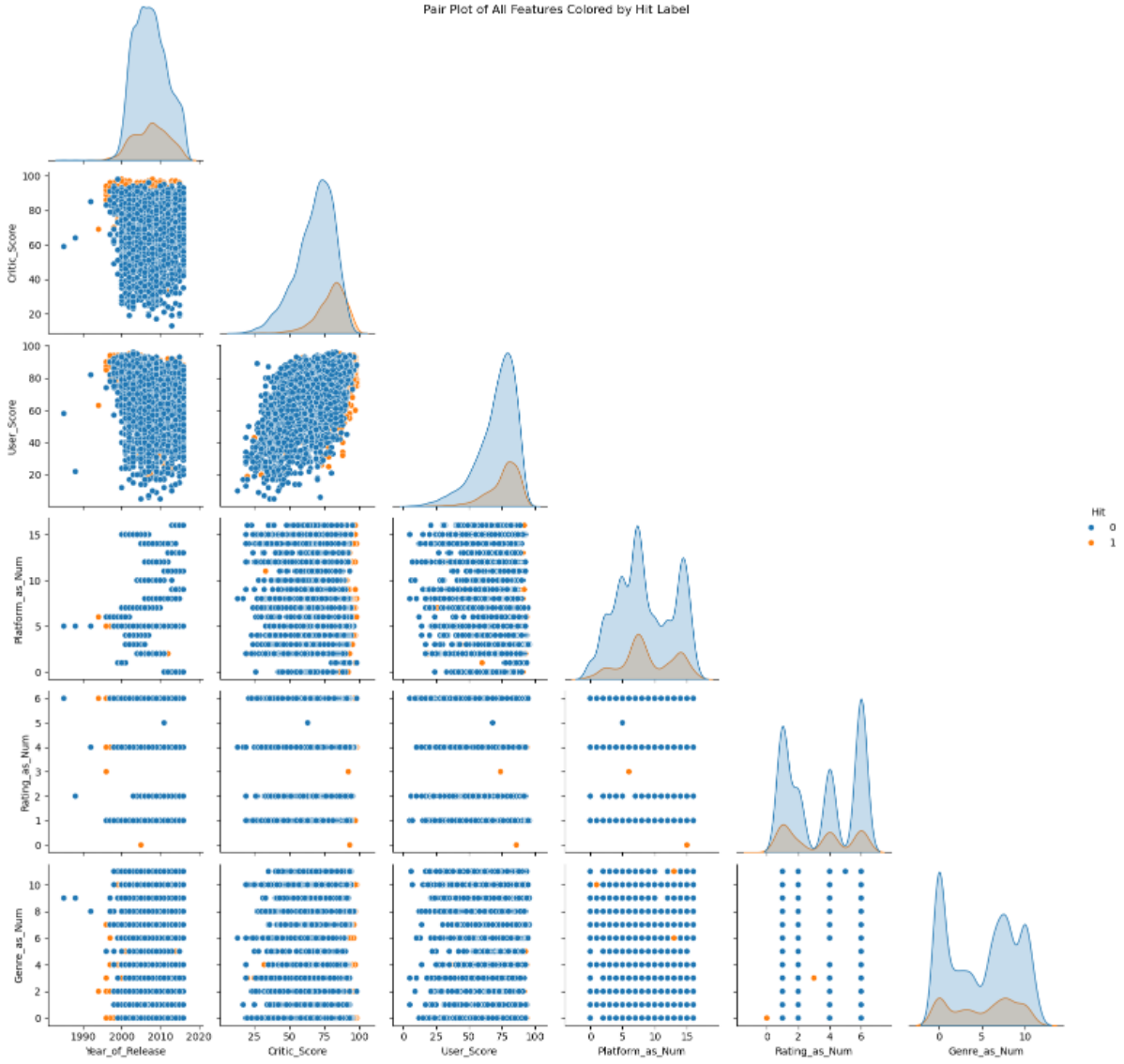

Fig. 1. Pair plot showing relationships among numerical features.

As a complement to the multivariate analysis, a bar chart (Figure 2) showing the distribution of the binary target variable revealed a clear class imbalance, with non-hit titles comprising the majority of the dataset.

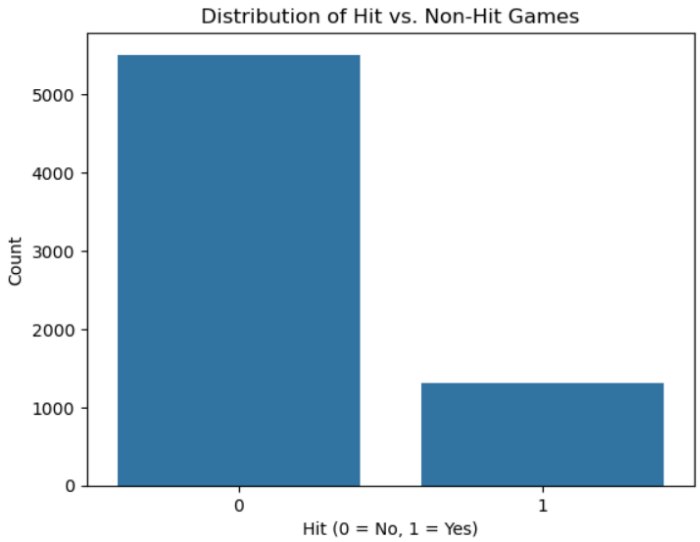

Fig. 2. Frequency distribution plot of hit and non-hit games.

2.2 Data Cleaning and Preprocessing

Data preprocessing is a crucial phase in machine learning modeling that addresses inconsistencies, noise, and scale variance. In this project, preprocessing began with cleaning the dataset by removing rows with missing or invalid entries from important columns. For instance, rows containing the value ‘tbd' in the user score column were excluded, as they could not be used during training. User scores were multiplied by 10 to align their scale with that of critic scores. A new binary classification target variable, Hit, was created to represent commercial success, with games labeled as hits if their global sales exceeded one million units. Several columns deemed unsuitable for modeling, including name, publisher, developer, and regional sales, were dropped to reduce noise. For visualization purposes, categorical features were temporarily encoded, and the encodings were removed after plotting.

To preserve the distribution of the Hit class across subsets, the clean dataset was stratified and split into training (64%), validation (16%), and test (20%) sets. Within each split, categorical and numerical features were transformed using one-hot encoding and standardization, respectively. One-hot encoding generated a new binary column for each category, marking the presence of a specific value with a 1 in the corresponding row.

Standardization, by contrast, scaled numerical features to a mean of 0 and a standard deviation of 1. These transformations ensured consistent input formats for both the logistic regression and neural network models. After preprocessing, the resulting splits contained 4,368 training records, 1,092 validation records, and 1,366 test records, each with 39 features. The Hit class was relatively rare across all splits, emphasizing the importance of strategies to address class imbalance during model training.

3 Methods

Two machine learning models were selected for this study: binary logistic regression and a neural network. Both were used to classify whether a video game would achieve commercial success, defined as exceeding one million units in global sales. The following subsections detail each model's implementation and evaluation procedures, including the model architecture, training strategies, and performance assessment.

3.1 Logistic Regression Model

The logistic regression model, implemented using the Scikit-learn library, estimates the probability of a binary outcome. In this context, it was used to predict whether a video game would be a commercial hit (coded as 1) or not (coded as 0), a technique commonly applied to classification problems involving two distinct outcomes (IBM). The model applies the sigmoid function to map a linear combination of input features to a probability between 0 and 1. This function is defined as

$$\pi(x) = \frac{1}{1 + e^{(-z)}}, \qquad where\ z = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n.$$

Here, $\pi(x)$ represents the predicted probability of success, x_i are the input features, and β_i are the regression coefficients learned during training.

To interpret the relationship between features and the predicted outcome, logistic regression employs the logit transformation, which expresses the natural logarithm of the odds (i.e., the ratio of success to failure) as a linear function of the predictors:

$$ln\left(\frac{\pi(x)}{1-\pi(x)}\right) = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n.$$

This transformation allows the model to establish a linear relationship between the input variables and the log-odds of the target outcome.

Parameter estimation was performed using maximum likelihood estimation (MLE), which identifies the set of coefficients that maximize the likelihood of the observed data. To address class imbalance in the training set, balanced class weights were applied, increasing the penalty for misclassifying the minority Hit class. This adjustment was particularly important given the skewed distribution observed during visual analysis. Model performance was evaluated using accuracy and recall on the training, validation, and test sets. Additionally, a learning curve was generated to assess the model's sensitivity to training size and to detect signs of underfitting or overfitting.

3.2 Neural Network Model

A deep learning neural network was implemented using the TensorFlow and Keras libraries. Neural networks are computational models inspired by the human brain, composed of layers of interconnected nodes that process input data (IBM). Each node computes a weighted sum of its inputs, adds a bias term, and passes the result through an activation function to produce an output. Formally, this computation can be represented as

$$z = \sum_{i=1}^{n} w_i x_i + b,$$

where w_i are the learned weights, x_i are the input values, and b is the bias term. The result z is then passed through an activation function $f(x)$, such as the step function or sigmoid function. In a basic binary threshold setting, the output is defined as

$$f(x) = \begin{cases} 1, & if\ z \geq 0 \\ 0, & if\ z < 0. \end{cases}$$

In this study, the network architecture consisted of three fully connected hidden layers with 128, 64, and 32 neurons, respectively. Each hidden layer used the rectified linear unit (ReLU) activation function to introduce non-linearity, allowing the model to learn complex patterns in the data. The output layer used a sigmoid activation function, suitable for binary classification tasks. Binary cross-entropy was used as the loss function, and the model was optimized using the Adam optimizer with a learning rate of 0.0002.

To address class imbalance, manual class weights were applied during training by assigning a weight of 1.2 to the minority Hit class and 1.0 to the majority class, increasing the penalty for misclassifying successful games. An early stopping mechanism monitored validation loss to prevent overfitting by halting training when improvement plateaued. Additionally, the default classification threshold of 0.5 was manually lowered to 0.35 to improve recall for the minority Hit class. Model performance was visualized across epochs by plotting both accuracy and loss curves, enabling evaluation of both convergence and generalization.

4 Results

The experiment was conducted in a Jupyter Notebook environment, enabling interactive coding, visualization, and iterative testing. The goal was to evaluate two machine learning models for classifying whether a video game would be commercially successful. Model performance was assessed using accuracy, recall, confusion matrices (on training, validation, and test sets), and learning curves.

4.1 Logistic Regression Performance

The logistic regression model demonstrated moderately strong performance across all data splits. The training set confusion matrix (Figure 3) shows that the model correctly identified 643 games as commercial hits and 2,515 as non-hits but also misclassified 1,013

non-hits as hits and failed to detect 197 actual hits. This performance indicates a high false positive rate and a challenge in distinguishing true hits from the dominant non-hit class.

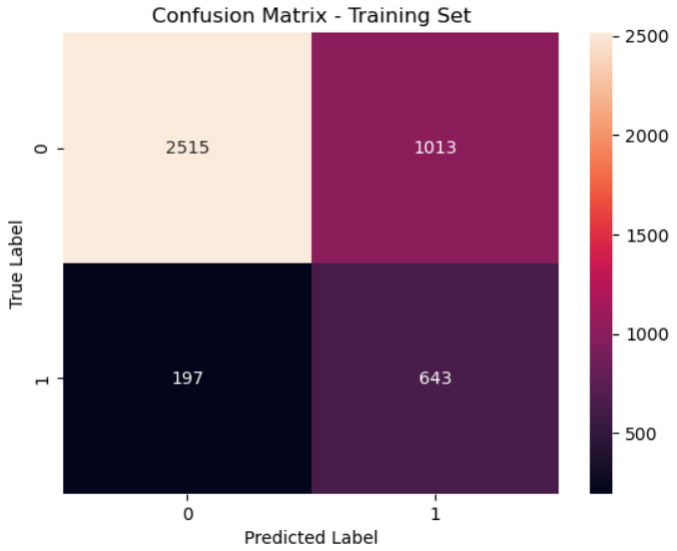

Fig. 3. Confusion matrix for logistic regression on the training set.

Validation results (Figure 4) revealed similar patterns: 268 false positives and 66 false negatives. Although the model tended to overpredict hits, its recall (the ability to identify actual hits) was noticeably higher than its precision, suggesting some success in retrieving hit titles despite frequent misclassification.

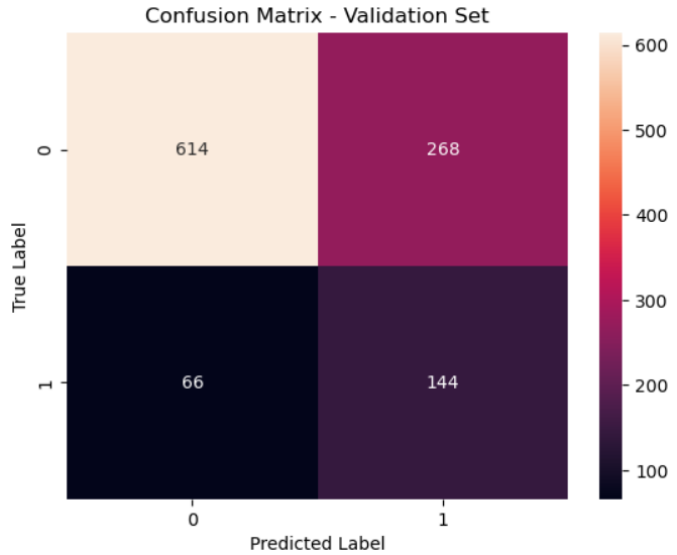

Fig. 4. Confusion matrix for logistic regression on the validation set.

Consistent trends emerged in the test set results (Figure 5), where the model identified 198 true positives and 798 true negatives, but also produced 305 false positives and 65 false

negatives. The similar false negative count across validation and test sets reflects stable recall. Nevertheless, the high number of false positives reinforces the model's tendency to overpredict hits, a pattern likely driven by the class imbalance.

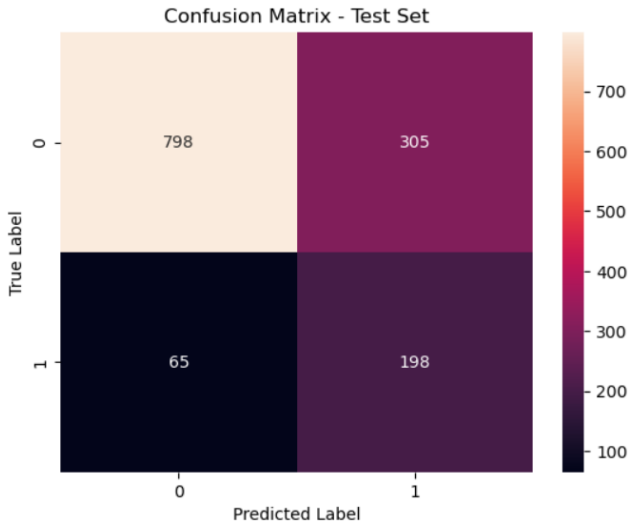

Fig. 5. Confusion matrix for logistic regression on the test set.

The learning curves (Figure 6) showed the convergence between training and validation accuracy as the training size increased. Validation accuracy rose steadily and stabilized around 72%. Based on Ibrahim's interpretation, such convergence indicates that the model effectively learned from the training data and reached near-optimal performance. This outcome implies logistic regression captured sufficient feature signals to generalize, although its sensitivity to the minority Hit class remained imperfect.

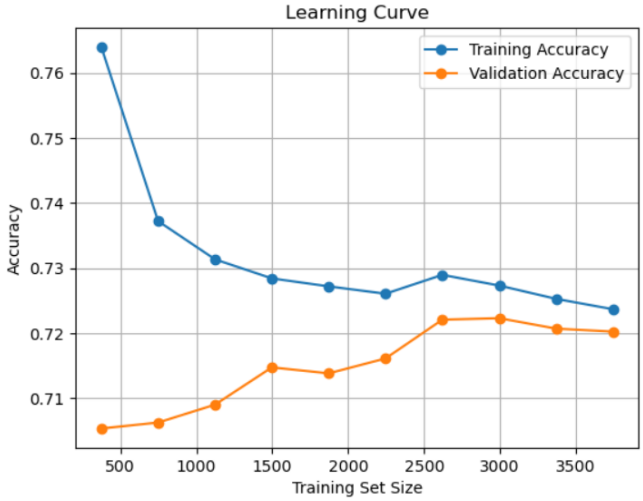

Fig. 6. Learning curves for logistic regression showing training and validation accuracy.

4.2 Neural Network Performance

The neural network exhibited relatively stable performance across all three splits, adopting a more cautious prediction strategy compared to logistic regression. The training set results (Figure 7) show that the model correctly classified 536 hit games and 3,162 non-hit games, while also producing 366 false positives and 304 false negatives. This indicates that the neural network was more selective when predicting hits and avoided labeling a game as successful without strong evidence. However, this conservatism came at a cost, as the model missed more actual hits, reflected in its higher false negative count.

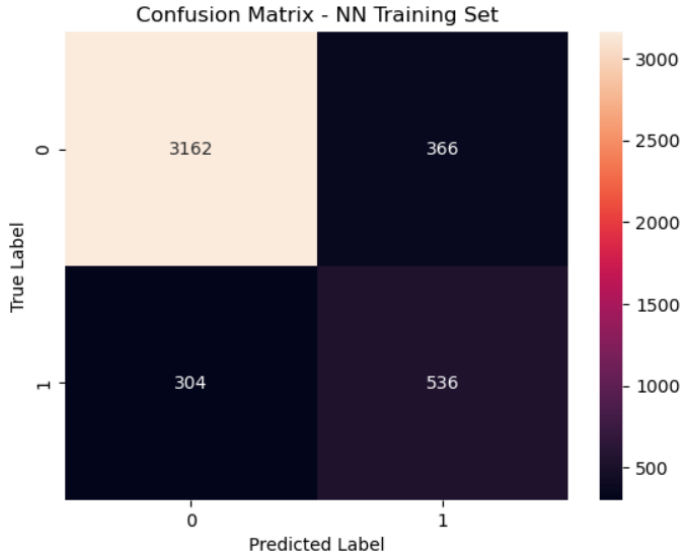

Fig. 7. Confusion matrix for the neural network on the training set.

The validation confusion matrix (Figure 8) displays a similar pattern. The model identified 117 true positives and 772 true negatives, alongside 110 false positives and 93 false negatives. The near one-to-one ratio between true and false positives suggests that the model required strong evidence to label a game as a hit. While this approach helped reduce false positives, it also caused the model to overlook less obvious hits.

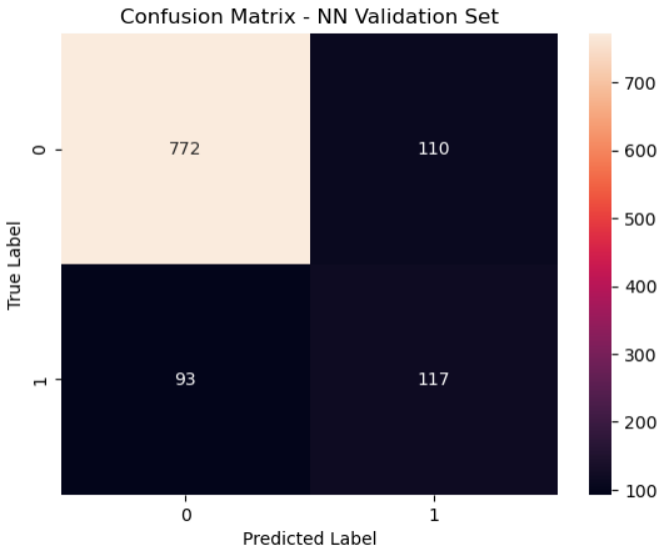

Fig. 8. Confusion matrix for the neural network on the validation set.

Test results (Figure 9) show that the model correctly classified 162 hit games and 984 non-hit games but also generated 119 false positives and 101 false negatives. Compared to logistic regression, which produced 305 false positives on the test set, the neural network was more precise in identifying non-hit games. However, it recalled fewer hits, indicating a trade-off between precision and recall. This reinforces the model's overall tendency to prioritize certainty over sensitivity to the minority class.

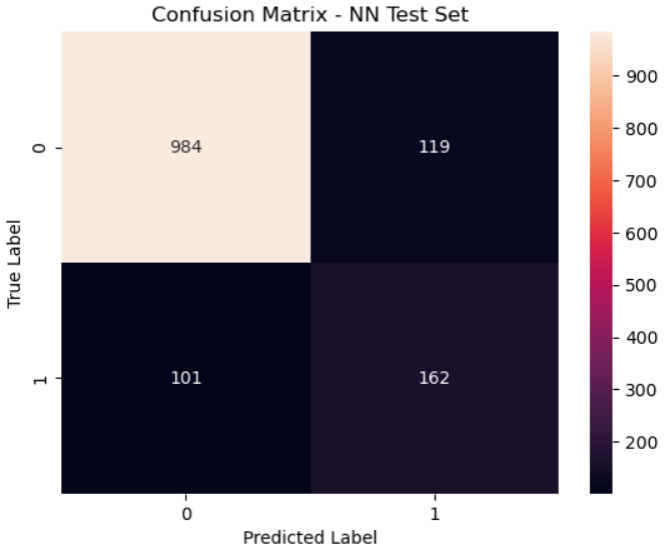

Fig. 9. Confusion matrix for the neural network on the test set.

The learning curves (Figure 10) offer additional insight into the model's behavior. Training accuracy increased steadily, surpassing 0.88 by epoch 33. Validation accuracy

initially improved, but eventually plateaued and then declined slightly to approximately 0.83. The widening gap between training and validation accuracy indicates overfitting, implying that the model learned the training data well but struggled to generalize to unseen examples (Ibrahim). Training loss decreased consistently, while validation loss dropped early but began to rise after epoch 10. Such a trend is indicative of overfitting, where the model memorizes the training data at the expense of generalization.

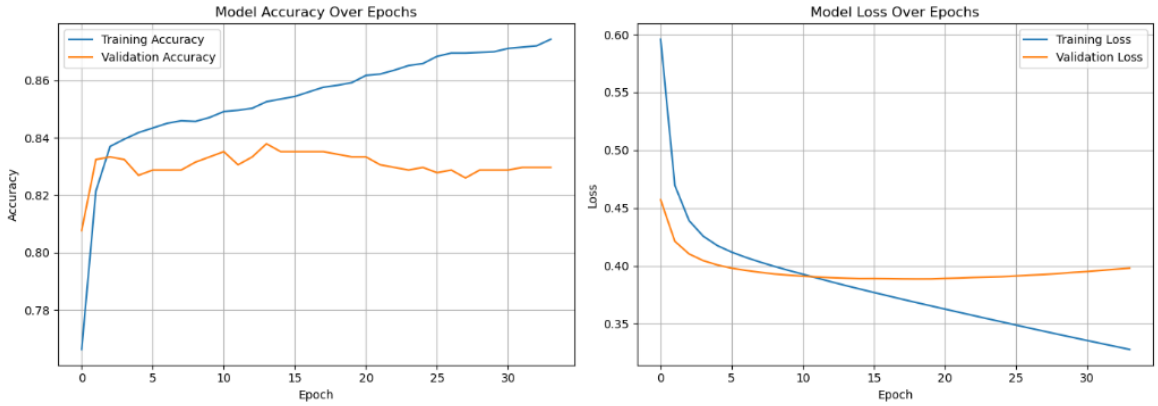

Fig. 10. Learning curves for the neural network showing accuracy and loss.

4.3 Model Performance Comparison

The two models were evaluated on the test set using accuracy and recall. As shown in Table 1, logistic regression achieved 72.9% accuracy and 75.3% recall, performing reasonably well at flagging hits, though it tended to overpredict them. In contrast, the neural network reached a higher accuracy of 83.9% but a lower recall of 61.6%, reflecting a more cautious prediction style that prioritized certainty and resulted in fewer detected hits. This trade-off highlights the model's conservative bias, which may have been shaped by the class imbalance and its internal confidence thresholds.

As illustrated in the learning curves, the logistic model benefited from class weighting and generalized more consistently. The neural network, despite early stopping and threshold tuning, showed signs of overfitting as validation performance declined in later epochs.

Table 1. Accuracy and recall on the test set for both models.

Metric	Logistic Regression	Neural Network
Test Accuracy	0.729	0.839
Test Recall	0.753	0.616

5 Conclusion

This study explored the use of supervised learning models to predict the commercial success of video games based on pre-release features. Logistic regression and a neural network were trained on a curated dataset and evaluated using accuracy and recall. While logistic regression performed reliably and captured useful patterns despite the class imbalance, the neural network achieved higher overall accuracy but recalled fewer hits due to its more conservative predictions.

These results highlight a fundamental trade-off in predictive modeling: optimizing for accuracy does not always align with detecting minority class outcomes such as commercial hits. Feature quality, class distribution, and model complexity each influence performance. Future work may benefit from incorporating additional features, such as social media sentiment or publisher reputation, or from exploring ensemble methods to balance accuracy and recall more effectively.

5.1 Limitations and Challenges

Class imbalance was a significant limitation in this project due to the relatively small proportion of hit games. However, both models incorporated strategies to address this issue: logistic regression used class weights during training, while the neural network relied on a manually adjusted classification threshold of 0.35. These adjustments improved hit recall but also led to an increase in false positives.

Works Cited

IBM. “What is a Neural Network?” *IBM,* https://www.ibm.com/think/topics/neural-networks. Accessed 4 May 2025.

IBM. “What is logistic regression?” *IBM,* https://www.ibm.com/think/topics/logistic-regression. Accessed 4 May 2025.

Ibrahim, Mostafa. “A Deep Dive into Learning Curves in Machine Learning.” *Weights & Biases,* https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0. Accessed 4 May 2025.

Kirubi, Rush. “Video Game Sales with Ratings.” *Kaggle,* https://www.kaggle.com/datasets/rush4ratio/video-game-sales-with-ratings. Accessed 3 May 2025.

Li, Jianbin, et al. "Predicting video game sales based on machine learning and hybrid feature selection method." *2021 16th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*. IEEE, 2021.

Ullmann, Gabriel C., et al. "What makes a game high-rated? Towards factors of video game success." *Proceedings of the 6th International ICSE Workshop on Games and Software Engineering: Engineering Fun, Inspiration, and Motivation*. 2022.

Acknowledgement

I, Nam Pham, acknowledge the use of online sources listed in the works cited section to justify model design choices and support the insights drawn from the results. I also used machine learning packages to implement the models and reduce the likelihood of coding errors.

Furthermore, I acknowledge the use of AI in completing this assignment and would like to provide a brief explanation of how I utilized AI, specifically ChatGPT, as a tool to support my work. Using GPT as a virtual conceptual tutor allowed me to clarify misunderstandings, identify more efficient solutions, and ensure my code correctly implemented the concepts taught in class. To do so, I crafted the following prompts:

1. "Why does my learning curve only show two data points when using Logistic Regression?" *Screenshot of code*
    1.1. When I first tried plotting the learning curve for my Logistic Regression model, the graph only displayed two points instead of a full curve. I asked GPT for clarification and learned that the issue was due to the model using only a single epoch by default. I then set max_iter to 1000 per GPT's suggestion, which allowed me to properly visualize training and validation accuracy across the full range of sample sizes.
2. "How should I handle class imbalance in my dataset?" *Screenshot of code*
    2.1. My dataset had far fewer 'hit' games compared to 'non-hit' ones, which led to biased predictions in early model versions. I asked GPT how to address this imbalance, and it recommended two changes: 1) using class_weight = 'balanced' in Logistic Regression to give more importance to the minority class, and 2) using the stratify = y parameter when splitting the data to ensure both classes remain proportionally represented in each subset.

3. “My neural network keeps favoring the majority class. How can I improve detection of minority (hit) class?” *Screenshot of code + notebook*

    3.1. Initially, my neural network was heavily biased toward predicting the majority class (non-hits), even when training accuracy seemed decent. GPT suggested adjusting the classification threshold used to convert predicted probabilities into binary labels. Instead of using the default 0.5 threshold, I experimented with lowering it to 0.35. This shift led to an increase in recall for the hit class, helping the model make better tradeoffs between precision and sensitivity.

4. “Why do I still see the very last progress bar in Jupyter even though I set verbose = 0?” *Screenshot of code*

    4.1. I initially set verbose = 0 in the .fit() method to suppress the training output, but the progress bar still appeared in my notebook. I asked GPT for help, and it explained that verbose = 0 doesn’t always suppress all logs in Jupyter environments due to backend behavior. GPT suggested a more robust solution: setting os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' to silence TensorFlow backend logs and using disable_interactive_logging() from tensorflow.keras.utils. After adding these, the progress bar and logs were fully suppressed, cleaning up the notebook output.

Project Repository

https://github.com/n7vpham/Hit-or-Miss-ML

(Includes source notebook and final project report.)