Classification of Swarm Behaviour in Boids

Khanh Pham[1], Huy Vu[2]

[1]tpc.khanhpham83@gmail.com

[2]hdvflss@gmail.com

1 Introduction

Flocking is the coordinated movement of a group of animals, such as birds, fish, or insects. Boids (*Figure 1*, source: Rodseng (n.d.)) are computer-generated agents designed to mimic the collective behavior of those animals and their real-life patterns such as flocking, grouping, and aligning. Humans can often recognise flocking, even though it is not always easy for them to explain why. Meanwhile, computers can simulate flocking relatively easily, but have significant difficulty in recognising it—whether in real-world settings or in simulations (UNSW Swarm Survey, n.d.).

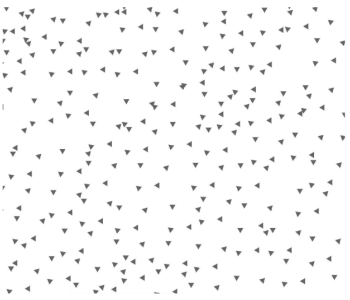

Figure 1. A random boid simulation with individual agents as black triangles, scattered across the environment, with no visible flocking behavior at this moment.

The goal of this machine learning (ML) project is to see if specific characteristics of boids can be correctly classified as flocking (*"showing swarm behaviour"*) or not. Some examples of application domain are swarm robotics, autonomous systems, and biological modeling, in which we need to understand and simulate collective behavior.

This report is structured as follows: Section 1 provides the background of ML applications and an overview of the report's structure. Section 2 give details of the dataset and a summary of the problem. Section 3 describes the methods used. Section 4 presents results of these methods, a summary and suggestions for further work.

2 Problem Formulation

2.1 Dataset

The dataset can be downloaded from UCI Machine Learning Repository (n.d.) or Kaggle by Deep Contractor (2024). This dataset achieved from an online survey, which was run by University of New South Wales, Australia, where participants were asked to identify when they believed a group of boids was exhibiting flocking behavior.

The total number of data points in the dataset is 23,309, representing the number of entries. Each data point represents a group of boids. For each group, the attributes are repeated for all m boids, where $m = 1, ..., 200$. Details about the attributes of each boid is presented in *Figure 2*. The dataset is multi-dimensional, with multiple boids per set and multiple attributes for each boid. The ML model also use all of these feature variables in classification.

Attribute	Data Type	Description
x_m, y_m	Double	(X,Y) position of each boid
$xVel_n$, $yVel_n$	Double	Velocity vector
xA_m, yA_m	Double	Alignment vector
xS_m, yS_m	Double	Separation vector
xC_m, yS_m	Double	Cohesion vector
nAC_m	Double	Number of boids in the radius of Alignment/Cohesion
nS_m	Double	Number of boids in the radius of Separation
Class Label (y)	Binary	1 = Flocking, grouped, and aligned ("Showing Swarm Behaviour") 0 = Not flocking, not grouped, and not aligned (" Not Showing Swarm Behaviour")

Figure 2. Data type and description of attributes of each boid

2.2 Summary of the Problem

With existing curiosity about the question *"What makes a group of animals appear to be flocking to humans? What elements lie behind this perception?"*, this project aims to determine whether a set of boids, which are computer simulations, is flocking or not. The project uses Supervised Learning, in which the model is trained on labeled data to predict the target class (*Swarm_Behaviour*), with labels of 1 (*"showing swarm behaviour"*) or 0 (*"not showing swarm behaviour"*).

3 Methods

3.1 Data Pre-processing

3.1.1 Feature Selection

In our project, each of the 200 boids is represented by 12 key features (e.g., position, velocity, alignment), resulting in 2400 features in total. We chose to keep all features because each boid plays a unique role in the swarm. Boids can be at the edge, in special positions, or have unique velocities, all of which significantly affect swarm dynamics. Removing features could change the swarm's structure, omit important details about individual boids' effects, and lead to misclassification. Keeping all features means we capture the full range of interactions and allow the model to learn from the entire dataset more accurately.

3.1.2 Contruction of Training, Validation and Test Sets

From the original dataset of 23,309 rows, we first checked for any NULL values and confirmed that the dataset had none, so we kept all rows to use in the model. Next, we split the dataset into training, validation and test sets with a 70/15/15 ratio. We chose this ratio because we wanted the model to have enough data for training, while still keeping enough data available for model tuning and evaluating its performance on unseen data. Then, we performed random shuffling during the split to make sure the data points were randomly distributed between the training and test sets. Furthermore, we applied stratified splitting to maintain the same proportion of *"swarming"* and *"not swarming"* boids in both sets, as the original dataset was imbalanced. After splitting, the current training set contains 16,316 rows, validation set has 3,496 rows, and the test set has 3,497 rows.

Downsampling on Training Set

After further examining the training set of 16,316 rows, we found that it contains 10,748 rows with a value of 0 (*"not showing swarm behavior"*) in the 'Swarm_Behaviour' column, while only 5,568 rows have a value of 1 (*"showing swarm behavior"*). This shows a significant imbalance between the two classes, with the majority class (class 0) being much larger than the minority class (class 1). To solve this imbalance, we decided to downsample the majority class (Jihenbelhoudi, 2022) to match the size of the minority class to prevent the model from becoming biased toward predicting the majority class.

Feature Scaling on Training Set *(only applied for Logistic Regression)*

Since one of our methods is Logistic Regression, and the boids' behavioral data contains features with varying scales, such as:

- Position coordinates (x, y) having large values (e.g., hundreds or thousands),
- Velocity, alignment, and cohesion vectors, having smaller ranges (e.g., -1 to 1),

we decided to use feature scaling for the data of the training set, to make sure all features contribute equally to the model and to prevent bias toward data with larger magnitudes. By applying StandardScaler (which standardizes features to have a mean of 0 and a standard deviation of 1), Logistic Regression can converge faster and perform better (Müller & Guido, 2016).

The final size of the training set is 11,136, with each class having 5,568 rows.

3.2 Logistic Regression (LoR)

We chose LoR because it is common for binary classification tasks, regarding the goal of our project is to calculate the probability of a set of boids exhibiting flocking behavior. We acknowledge that LoR may struggle with a complex, high-dimensional data as our dataset, but we still chose it because we want to compare the results with Random Forest. Moreover, LoR provides quite clear, interpretable results.

For each observation in our dataset, we calculate the predicted output using the linear hypothesis map $h(x) = w^T x$ (Jung, 2022, p. 89), where w is some parameter vector $w \in \mathbb{R}^n$. This produces a score that shows the connection between the input features and the predicted behavior of the boids. The model first calculates a weighted sum of all the features in the input data, where each feature—such as the position, speed, or direction of the boids—is assigned a weight. These weights combines the features into a single score. Then, this score is processed using a logistic function (often called a sigmoid function) into a probability value between 0 and 1. If the probability score is high (typically above 0.5), the model predicts that the boids are flocking; if it is low (below 0.5), the model predicts that they are not flocking.

The loss function we chose is the Logistic Loss, because it is used by LoR (Jung, 2022, p. 90) and already implemented as the log loss function in the Scikit-learn library. It is mathematically defined as follows.

$$L_{\log}(y, p) = -\log \Pr(y|p) = -(y \log(p) + (1 - y) \log(1 - p))$$

The Logistic Loss function is minimized during the training of the model. By adjusting the weights w to reduce the loss, the model learns to make better predictions about whether the boids are flocking or not.

Our training set has 11,136 samples and 2,400 features, which makes traditional k-fold cross-validation like GridSearchCV quite computationally expensive and time-consuming. Instead, we decided to do manual hyperparameter tuning, which allowed us to focus on key hyperparameters, such as regularization strength (C) and penalty type (l1 or l2), without overloading our computing resources. Additionally, we supposed that by using our dedicated validation set for tuning, our model would generalize well without the need for extensive re-training over multiple folds.

3.3 Random Forest (RF)

The second method we chose to compare against Logistic Regression (LoR) is Random Forest (RF). With such a complex multi-dimensional dataset, we consider RF to be more suitable than LoR. Unlike LoR which assumes a linear relationship between features and the target, RF can capture non-linear relationships (Breiman, 2001), which is crucial for identifying swarm behavior with complex, non-linear interactions between features. Moreover, RF can automatically model these interactions, saving us significant time compared to the manual feature engineering required for LoR. Also, RF is robust against overfitting compared to single decision trees (Breiman, 2001), since it combines predictions from many trees that can improve the performance on new data.

The hypothesis space of a RF for our dataset includes all possible combinations of decision trees that can be built using the 2,400 features (attributes of 200 boids). Each tree is trained on a random subset of the data, created by a technique called bootstrapping (Scikit-learn, n.d.). Since each tree is trained on a bootstrap sample, the remaining data points (those not

included in the bootstrap sample) can be used as a kind of validation set for that specific tree. This is the reason why RF model does not require a separate validation set.

We also applied RandomizedSearchCV in order to optimize the hyperparameters (Scikit-learn, n.d.). We chose it because it is computationally more efficient compared to GridSearchCV, especially for high-dimensional datasets like ours.

Regarding loss function, note that in Scikit-learn's RF, the concept of a loss function as seen in traditional gradient-based methods is not explicitly used in the same way. Instead, RF uses criteria to assess the quality of the splits. Among the criteria, we chose Gini Impurity since it is easier to compute than the cross-entropy (Breiman, L. et al., 1986). According to Zhou V. (2019), Gini Impurity measures the probability of incorrectly classifying a randomly chosen element in the dataset if it were randomly labeled according to the class distribution. Gini Impurity is defined as $1 - \sum_i p_i^2$, where p_i is the proportion of samples in a dataset that belong to class i. A Gini Impurity of 0 is the lowest and best possible impurity.

4 Results

4.1 Comparison

For the LoR model, we got a training accuracy of 0.919 and a validation accuracy of 0.906. We can say the model is learning effectively from the training data and generalizing well to unseen data, with little overfitting. For the RF model, we got a training accuracy of 0.973 and a validation accuracy of 0.895. Because the model has high training accuracy but drops significantly in validation accuracy, it likely means RF is overfitting to the training data and picking up noise instead of finding general patterns.

When we compare the two models, RF has a higher accuracy on the training set but shows a bigger gap between training and validation accuracy, while LoR strikes a good balance between training and validation accuracy, generalizing better to new data. Therefore, we decided to go with Logistic Regression as our final chosen method. See *Figure 3* for a comparison of the two models on the validation set. The result of each model on the training set is included in Appendix B. Code part.

	Logistic Regression Model				Random Forest Model			
class	precision	recall	f1-score	support	precision	recall	f1-score	support
0	0.97	0.88	0.93	2303	0.98	0.86	0.92	2303
1	0.81	0.95	0.87	1193	0.78	0.97	0.86	1193
accuracy			0.91	3496			0.90	3496

Figure 3. Classification report of two models on validation set

4.2 Test Error of Chosen Method - Logistic Regression

As mentioned in section 3.1.2, the test set is made up of 15% of the original dataset. It is entirely separate from the training and validation sets, so the model is tested on data it has not seen before. The test accuracy for our LoR model is 0.9071, which means it correctly classified about 91% of the test samples, with a test error of 9.29%. *Figure 4* shows the confusion matrix for the LoR model on the test set.

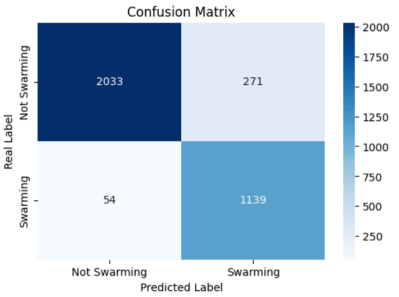

Figure 4. Confusion Matrix of the Logistic Regression Model on the test set

4.3 Conclusion

In this report, we aimed to classify swarm behavior in a dataset of boids using LoR and RF models. We split the dataset into training, validation, and test sets (70/15/15) and preprocessed the data by scaling features and solving class imbalance using downsampling. Surprisingly, the RF model did not perform as well as we expected. Wee selected Logistic Regression as our final chosen method because on the test set, this model has a test error of 0.0929 (or 9.29%), meaning it achieved low

misclassification on unseen data. From this result, combined with its balance between training and validation accuracy (0.919 and 0.906, respectively), we conclude that the model generalizes well to new samples while avoiding overfitting.

However, there is still room for improvement with the RF model. The precision score for class 1 in the RF model was low (a high rate of false positives). To improve the performance of our RF model, we have some suggestions. First, we can adjust the decision threshold for class 1, for example by increasing it from 0.5 to a higher value, and the model would be more cautious in predicting *"showing swarm behavior"*. Second, we can do some manual parameters tuning, for example limit the tree depth (reduce max_depth parameter to prevent trees from becoming too deep), increase min_samples_split parameter, reduce number of trees (n_estimators) in the forest, or collect more training data (Raj, 2021).

Lastly, in this project, we chose to keep all the features; however, in the future, we could remove some features for each boid or reduce the number of boids in each group, then apply various ML models to see how the results turn out. The large number of features could also be one reason for introducing noise into the RF model.

5 References

1. UNSW Swarm Survey. (n.d.). Screenshots. https://unsw-swarm-survey.netlify.app/Files/Screenshots.pdf
2. Rodseng, B. (n.d.). Geometric tiles with triangles background. Vecteezy. https://www.vecteezy.com/vector-art/2556753-geometric-tiles-with-triangles-background
3. UCI Machine Learning Repository. (n.d.). Swarm behaviour. https://archive.ics.uci.edu/dataset/524/swarm+behaviour
4. Contractor, D. (2024). Swarm behaviour classification. Kaggle. https://www.kaggle.com/datasets/deepcontractor/swarm-behaviour-classification/data
5. Jihenbelhoudi. (2022, January 7). 10 techniques to deal with imbalanced classes. Kaggle. https://www.kaggle.com/code/jihenbelhoudi/10-techniques-to-deal-with-imbalanced-classes
6. Jung, A. (2022). Machine learning: The basics. Springer. https://alexjungaalto.github.io/MLBasicsBook.pdf
7. Scikit-learn. (n.d.). Metrics and scoring: Quantifying the quality of predictions. https://scikit-learn.org/stable/modules/model_evaluation.html#log-loss
8. Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5-32. https://doi.org/10.1023/A:1010933404324
9. Müller, A. C., & Guido, S. (2016). Introduction to machine learning with Python. O'Reilly Media.
10. Scikit-learn. (n.d.). sklearn.ensemble.RandomForestClassifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
11. Scikit-learn. (n.d.). sklearn.model_selection.RandomizedSearchCV. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
12. Raj, M. (2021, December 2). In-depth parameter tuning for random forest. Medium. https://medium.com/all-things-ai/in-depth-parameter-tuning-for-random-forest-d67bb7e920d
13. StatQuest with Josh Starmer. (2018, December 8). StatQuest: Random forests part 1 - Building, using and evaluating [Video]. YouTube. https://www.youtube.com/watch?v=J4Wdy0Wc_xQ
14. Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1986). Classification and regression trees. Wadsworth and Brooks/Cole Advanced Books & Software.
15. Zhou, V. (2019, March 29). A simple explanation of Gini impurity. Victor Zhou. https://victorzhou.com/blog/gini-impurity/#:~:text=Gini%20Impurity%20is%20the%20probability%20of%20incorrectly%20classifying%20a%20randomly#:~:text=Gini%20Impurity%20is%20the%20probability%20of%20incorrectly%20classifying%20a%20randomly

Originality Declaration

1) In Kaggle, there are several projects using the same dataset and feature selection. To make our project original, we chose to pre-process the data differently. We referred to Jihen Belhoudi's ideas from 10 Techniques to deal with Imbalanced Classes to learn how to handle the imbalanced dataset. However, we did not use their code for downsampling, and other preprocessing steps—such as checking for NULL values, random shuffling, and feature scaling—were entirely our own.
2) About ML methods, other projects applied techniques such as PCA and Logistic Regression. We also used Logistic Regression but since our data pre-processing was unique, our results turned out different from theirs. Moreover, the application of Random Forest is original.

6 Appendices

A. Graphs and Figures

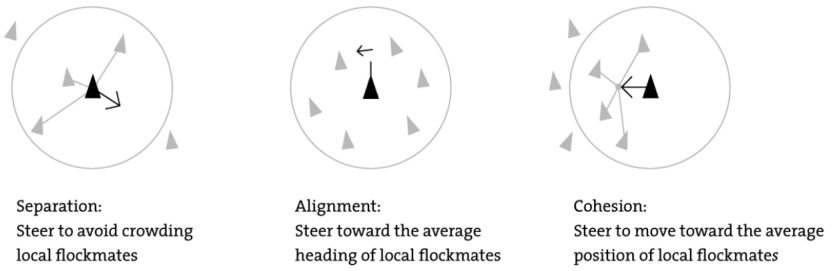

Figure A1. Visual Representation of the Alignment, Cohesion, and Separation Rules in Swarm Behavior (Source: https://parametricmonkey.com/2018/06/20/glossary-of-computational-terminology/)