Master Degree Program in

Data Science and Advanced Analytics

Neural and Evolutionary Learning

Crude Protein Estimation in Poultry

Carcass Composition Prediction Problem
Utilizing Neuroevolutionary Algorithms to Infer Protein Content

---

Group 22:

Catarina Nunes, number: 20230083
Sofia Jacinto, number: 20240598
José Cavaco, number: 20240414

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

8th June 2025

This project used ChatGPT (free version) and DeepSeek Chat for minor code refinements and adapting classroom examples, but all core research, pipeline development, and implementation were independently executed. AI served only as a supplementary aid for debugging and adjustments.

1. Introduction

Using computational models to analyse biological data is increasingly important in the pursuit of efficient agricultural practices. This project focuses on predicting the crude protein content of chicken carcasses using machine learning algorithms, based on measurements from a controlled dataset. The algorithms that we are going to test are Genetic Programming (GP), Geometric Semantic Genetic Programming (GSGP), Semantic Learning Algorithm based on Inflate and Deflate Mutations (SLIM), Neural Networks (NN) and NeuroEvolution of Augmenting Topologies (NEAT). The data, obtained from carcass components such as liver, breast, spleen and others, provides a detailed view of each chicken's composition after slaughter. Currently, crude protein values are measured directly, which can be time-consuming or require specialized equipment. The objective of this project is to develop predictive models that estimate crude protein content based on easily obtainable features. If successful, this approach could offer a simpler and faster way to estimate protein levels, contributing to better-informed decisions in poultry nutrition and processing.

2. Data Exploration and Preprocessing

The dataset provided (*sustavianfeed.xlsx*) consists of **96 records** and **15 features**. The *wing_tag* variable, which uniquely identifies each chicken, was excluded as a predictive feature and instead set as the index to maintain traceability. No missing values or duplicate entries were found, allowing us to proceed directly to further preprocessing.

A few potential outliers were observed but given the expected biological variability in chickens and the small dataset size, we opted **not to remove them**. Instead, we integrated the **Robust Scaler** into the modeling pipeline to mitigate its influence.

We also evaluated correlations (Pearson and Spearman) between features and the target variable, ***crude_protein***:

- **Correlation with the target:** Features such as *cold_carcass_weight, hot_carcass_weight, ether_extract, weight, carcass_weight_with_head_and_legs* and *thigh_weight* showed the strongest correlation with the target, making them good candidates for model inclusion. Despite their lower individual correlations, other variables like *heart, liver,* and *breast_weight* were kept in the dataset, as they may still contribute to the models when combined with other features, especially in non-linear approaches or through interaction effects.
- **Correlation between features:** The correlation matrix also revealed **high multicollinearity (0.94–0.98)** among some weight-related features (e.g. *weight, cold_carcass_weight, hot_carcass_weight*), suggesting possible redundancy. However, since the project explicitly requires using **all remaining features** (except the animal ID) for prediction, we deferred feature selection.

For preprocessing, the binary feature *empty_muscular_stomach* was was transforme into a binary indicator, and all other features in the dataset were numerical by nature, eliminating the need for any additional encoding transformations. This straightforward preprocessing approach preserved the data's integrity while ensuring it was properly formatted for subsequent modeling stages.

3. Monte Carlo vs Nested Cross-Validation

In this project, nested cross-validation was chosen as the strategy for both model tuning and performance evaluation. This approach was adopted to ensure fair and consistent comparisons across all algorithms studied. By using the same data partitions throughout, nested CV helps eliminate biases that could arise from uneven splits. Another important reason for this choice was the separation between tuning and evaluation. With nested CV, hyperparameter optimisation is performed in the inner loop, while performance is assessed in the outer loop, preventing information leakage and helping avoid overfitting, which is particularly important for

algorithms like GP that are prone to premature convergence. Although Monte Carlo cross-validation could also be used, it introduces randomness in the splits as some samples may **appear more than once in the test set**, while others may be **excluded entirely**. This lack of control makes it less suitable for structured comparisons across methods. Nested CV, on the other hand, ensures that **each data point is used for testing exactly once**, allowing for more reliable error estimates and statistical tests. Despite the small size of the dataset (96 records), nested CV was still appropriate, as each fold retained enough data to train and evaluate the models. The added computational cost was negligible and the benefits in terms of robustness and fairness made it the preferred option. We used a consistent seed (random_state = 42) for comparability across models Folds performance.

4. Estimating the predictive error

The RMSE was chosen as the metric to estimate predictive error because it assigns greater weight to larger errors, given that errors are squared before being averaged. This characteristic makes RMSE especially valuable when large errors occur, as they can significantly affect the model's overall performance.

5. Evolutionary Algorithms

Three evolutionary algorithms were implemented and tested. Since the overall structure of these algorithms is quite similar, we developed a structured function that adapts to each algorithm simply by passing the specific algorithm name. All three algorithms shared several hyperparameters, such as the initial size of the trees (*init_depth*), population size (*pop_size*), crossover probability (*p_xo*), number of iterations (*n_iter*), the initialization strategy (initializer), among others. These parameters formed a base configuration, but each algorithm also introduced specific parameters that required individual tuning. Given the stochastic nature of evolutionary algorithms, which means that the same configuration can yield different results across runs, we executed each model three times using different random seeds. This strategy allowed us to gain a better understanding of the average behavior of each algorithm and to mitigate the impact of randomness in the final performance analysis.

5.1. Genetic Programming (GP)

For GP, an additional *max_depth* parameter was used to control the maximum size of the trees during evolution, which helped balance model complexity and avoid overfitting. Although the best-performing GP configuration was selected through tuning, we re-ran it with verbosity enabled using 5-fold outer cross-validation to better observe fold behaviour. Across folds, GP showed common issues such as early convergence, bloat and overfitting. Folds 1 and 2 attempted to escape local optima by increasing complexity, with limited success. Fold 3 performed the worst, with persistent stagnation and poor generalisation. Fold 4 showed no evolution at all, indicating underfitting. Only Fold 5 achieved stable improvement with minimal bloat, highlighting GP's sensitivity to diversity and complexity control.

5.2. Geometric Semantic Genetic Programming (GSGP)

In GSGP, the *reconstruct* parameter was added to manage how semantic trees are rebuilt during training, impacting both performance and solution growth. As with GP, the best-performing GSGP configuration was re-tested with verbosity using 5-fold outer cross-validation. Folds 2 and 5 showed early convergence followed by overfitting and bloat, with improvement in test performance. Fold 3 performed the worst, with aggressive complexity growth, poor generalisation, and overfitting. Folds 1 and 4 followed a similar pattern but achieved a better balance between accuracy and model size. Fold 5 successfully escaped a local optimum and improved performance, though still with notable bloat. Overall, GSGP proved prone to overfitting and complexity growth when not properly controlled.

5.3. Semantic Learning Algorithm based on Inflate and Deflate Mutations (SLIM)

SLIM introduced three additional parameters: *slim_version*, which defines the variant of the algorithm to be used, *p_inflate*, which controls the probability of applying inflate mutations, and *copy_parent*, which determines whether the parent should be preserved during mutation. Across all folds, the model showed early convergence followed by increased complexity in attempts to escape local optima, often leading to overfitting and bloat. Folds 1 and 2 exhibited significant complexity growth with no gains in test performance. Fold 3 started well but degraded after trying to escape a local minimum. Folds 4 and 5 followed the same pattern: initial good results followed by overfitting and unnecessary model growth. Overall, SLIM struggled to control complexity and maintain generalisation.

6. Neural Networks (NN)

In addition to the evolutionary approaches, a neural network was also implemented using PyTorch. The network was designed with three linear layers. The first layer upscales the 2 input features to 3 neurons, the second expands to 4 neurons and the final output layer compresses the result down to a single neuron. ReLU activation function was applied between the hidden layers to introduce non-linearity, while a sigmoid activation function was used in the output layer to ensure bounded output values suitable for regression with scaled targets. Weights were initialised using Xavier Uniform and biases with zeros. The backpropagation and weight updates were handled within the training loop using backward() and step() methods. The backward pass computed the gradients of the loss function with respect to each parameter and the optimizer applied the corresponding updates to minimise the error.

The network was trained and evaluated using 5-fold cross-validation, following the same methodology as the evolutionary algorithms. Across all folds, the model showed a consistent decrease in loss, confirming effective learning. Fold 3 performed the best, with the loss dropping from 1.1 to 0.5, using a configuration with one hidden layer of 4 neurons, learning rate of 0.001, batch size of 8, 100 epochs and SGD as the optimizer. Fold 4 also showed strong early convergence, stabilising around 0.7. Fold 2 improved from 1.3 to 0.7 also, but more gradually, while Folds 1 and 5 showed slower convergence, ending near 0.8.

7. NeuroEvolution of Augmenting Topologies (NEAT)

NEAT was implemented as our final approach, evolving both the weights and topology of neural networks to solve the target task (XOR classification). Across all five folds, the model exhibited unstable and inconsistent learning behavior, marked by limited training optimization and poor generalization. Most folds showed either stagnant fitness or divergent training and validation errors, with signs of premature convergence, overfitting, or loss of beneficial structural innovations. While some folds (notably Folds 2 and 4) achieved temporary gains in validation performance, these improvements were short-lived.

Structurally, in Folds 1 and 5, the NEAT algorithm produced networks with complex topologies, characterized by multiple hidden nodes and a dense web of connections. This suggests that the algorithm explored more elaborate architectures to solve the XOR task under those data splits. In contrast, the networks from Folds 2 and 4 were less complex, containing fewer hidden neurons and more selective connectivity, indicating that simpler architectures were sufficient to achieve good performance on those particular training sets. Fold 3 exhibited the most minimal topology, with very few hidden nodes and more direct input-output connections. These differences highlight NEAT's capacity to evolve diverse solutions depending on the training data distribution, while still converging on functional networks capable of learning the XOR task.

8. Model Comparison and Results

SLIM emerges as the most promising model, demonstrating the best overall performance, even though it loses in Fold 1 to GP and in Fold 4 to GSGP. It boasts the lowest mean RMSE (1.331), median RMSE (1.011), and maximum RMSE (2.658). Although statistical tests did not reveal significant differences due to the limited sample size, SLIM's consistent advantages suggest practical superiority.

Table 1. *Models Comparison (RMSE Scores)*

Fold	GP	GSGP	SLIM	NN	NEAT
1	1.383	1.452	1.406	1.384	1.712
2	1.525	1.031	0.906	1.338	1.075
3	2.832	2.830	2.658	3.566	3.037
4	1.157	0.996	1.011	1.310	1.511
5	0.723	0.854	0.676	0.893	0.847
Mean	1.524	1.433	**1.331**	1.698	1.636

In contrast, the neural network (NN) performed the worst, exhibiting the highest mean RMSE (1.698) and maximum RMSE (3.566), along with the largest standard deviation (0.950), indicating unreliable performance. The **Friedman test** ($p = 0.0805$) and pairwise **Wilcoxon tests with Holm correction**—where all corrected p-values are 1.000, except for SLIM vs. NEAT at 0.625—showed **no statistically significant differences between models**. This lack of significance is likely due to the small number of folds ($n = 5$), which we anticipated as a limitation given our small dataset.

This means we cannot confidently claim that any model outperforms another based on this data. The observed differences in mean RMSE (e.g., SLIM's lower error) can be due to random variation rather than true superiority. The effect sizes (Wilcoxon r) indicate modest performance gaps, with the largest gap observed between NN and NEAT ($r = 0.400$). SLIM's effect size versus NEAT is negligible ($r = 0.000$), suggesting comparable performance in certain folds. However, SLIM's lower error metrics and narrower bootstrap confidence interval (95% CI: [0.835, 2.057]) reinforce its reliability. Conversely, NN's wider confidence interval ([1.149, 2.597]) reinforces its instability.

For real-world applications, **SLIM is the recommended choice due to its balanced performance, despite the lack of statistical significance**. The NN model should be avoided due to its high variability. Future work should involve increasing the number of folds or repetitions to enhance statistical power, as the current analysis indicates SLIM's potential superiority, which larger samples might confirm.

9. Conclusion

- Was hyperparameter tuning easy or difficult? Why?
- How well do your models generalize?
- How strong is the model's learning performance?
- Are the final models transparent or opaque? If they are transparent, what is the model revealing?

Hyperparameter tuning proved challenging due to our small dataset size and the use of nested cross-validation (5 outer folds, 3 inner folds), which limited the reliability of validation scores and introduced artifacts. While our models showed some learning capability, their generalization performance could be improved through additional data, feature selection to address multicollinearity, and outlier treatment, though we retained all features as required by the problem description. The models exhibited varying degrees of transparency: GP, GSGP, and SLIM produced more interpretable models (showing their "working steps", node evolution, trees, etc.), while neural networks (NN, NEAT) operated as black boxes. Overall, while the current implementations show promise, their performance could be strengthened with larger datasets and additional preprocessing to enhance learning robustness and generalization capability.