Impact of Gaussian Noise on Machine Learning Models

Harrison Tolley, 47447842

Report Outline

# Introduction

## Background

For this report, we will be using the provided Drosophila dataset to implement and investigate machine learning models/algorithms/techniques. This dataset came from a study which investigated variation in body size and life history traits in Drosophila aldrichi and Drosophila buzzatii in eastern Australia published in July, 2000. This dataset provides a rich source of information for exploring the robustness of various machine learning models trained with noisy data.

## Research Question

The research question formulated for this report is:

Can other machine learning models outperform neural networks in this noisy data context?

## Objectives

The objective of this study is to compare the performance of different machine learning models when trained on data with various levels of Gaussian noise. Specifically, we will:

* Add Gaussian noise to the dataset at levels of 1%, 2.5%, 5%, 10%, and 20%
* Train several machine learning models on this noisy data. These models include neural networks, decision trees, random forests, and k-nearest neighbours.
* Evaluate and compare the performance of these models using relevant performance metrics.
* Analyse the robustness of each model to noise and discuss our findings.

By systematically investigating these aspects, we aim to determine which model/s are more robust and resilient to noise and provide insight into improving model performance when trained on noisy data.

# Data Description and Preprocessing

With the provided dataset, we have 3 comma-separated value sheets. The first 5 rows of each datasheet can be seen in the following images:

A screen shot of a computer

Description automatically generated*83\_Loeschcke\_et\_al\_2000\_Thorax\_&\_wing\_traits\_lab pops.csv:*

A screenshot of a computer

Description automatically generated*84\_Loeschcke\_et\_al\_2000\_Wing\_traits\_&\_asymmetry\_lab pops.csv:*

A screenshot of a computer screen

Description automatically generated*85\_Loeschcke\_et\_al\_2000\_Wing\_asymmetry\_lab\_pops.csv:*

These tables are generated using Python’s pandas library by reading the csv into dataframes and printing the dataframe’s head. The code for this can be seen in the appendix.

For this report, we need to determine which dataset would be most beneficial to train classification models on.

The first dataset (*83\_Loeschcke\_et\_al\_2000\_Thorax\_&\_wing\_traits\_lab pops.csv*) provides a diverse range of features, including morphological measurements which could prove to be important for classifying species and populations. As well as this, it provides a combination of geographic, and temporal data.

The second dataset (*84\_Loeschcke\_et\_al\_2000\_Wing\_traits\_&\_asymmetry\_lab pops.csv*) focuses on wing measurements and asymmetry, which can also be critical for species differentiation. This also includes geographic and temporal data.

The third dataset (*85\_Loeschcke\_et\_al\_2000\_Wing\_asymmetry\_lab\_pops.csv*) focused on asymmetry in various measurements. This *may* reveal subtle differences between species/populations that other features may not capture.

For our classification task, its most likely best to use a dataset which provides a diverse range of features, which, in this case, would be the first dataset. This provides a variety of morphological measurements, which are likely to be highly relevant for distinguishing between various species/populations. Furthermore, the inclusion of temporal, geographical, and sex data may provide more accurate classification.

Overall, the first dataset offers a rich dataset for training robust machine learning models, and thus will be used for training the models.

## Data Preprocessing

Various steps must be taken with this data to convert it into a usable form to train models on.

Firstly, we must drop the data that will not be useful for the classification task. We can use the pandas library to count how many unique values are in each column. The code and output from this can be seen in the appendix.

There are several important things to note from this table. Firstly, there are 2 unique species, with 5 unique populations, which results in 10 unique classes that our models will classify into. Secondly, the starting and ending year only provide one unique value, and thus offer no differentiating features between species/population. These columns will be dropped (to be clear, the Year\_start and Year\_end columns). Thirdly, we will also drop the *Replicate* column, as it only has 3 unique values, which arguably will not provide much extra differentiation between species/population, and thus is redundant.

To reiterate what will be done:

* The species and population columns will be combined.
* The columns *Year\_start, Year\_end*, and *Replicate* will all be dropped from the dataset.

The code to achieve this can be seen in the appendix.

The next step for preprocessing the data is to encode numerical values for the combined species and population so that they can be used for training an evaluation metrics. For example, encoding the species of *D.\_aldrichi* with a population *Binjour* to a value of 0, meaning that it now becomes the class of ‘0’. This will be repeated for all unique combinations of species and populations, which as mentioned is a total of 10 unique classes (0 through to 9). This will be done as well on the sex column, converting the ‘female’ value to 0, and ‘male’ to 1. This can be done by using the library scikit-learn’s Label Encoder. The code for this can be seen in the appendix.

The distribution of these classes can be seen in the following graph:A graph of a number of species

Description automatically generated

This shows that there is a near equal distribution of the classes, which means that the provided data is not biased towards one class or another. This ensures that the training data for all classes is near equal. The code to generate this graph can be seen in the appendix.

Next, there were issues identified with the data that the column ‘*wing\_loading*’ and ‘*Thorax\_length*’ were of type object instead of the desired float or integer data types. The origin of these errors is unknown but are suspected to be caused by missing values in the dataset. To rectify these errors, we dropped all rows which didn’t have a number as a value. This resulted in a total of 2 rows being dropped. The code to perform this can be seen in the appendix.

The next precaution taken was to drop all rows which contained a 0 in a measurement field. If a 0 is measured, that likely meant that the dragonfly was deformed or missing a limb/wing/etc (besides human error), which negatively affects the quality of the training data. To rectify this, all rows which have 0 as a measurement are dropped. The code to perform this can be seen in the appendix.

## Adding Gaussian Noise

The next step is to create a series of CSV files that contain various percentages of added noise; for this report the percentages of added noise will be 1%, 2.5%, 5%, 10%, 20%. The process for adding noise will be to calculate the standard deviation of a column’s values. We multiply the standard deviation by the percentage of noise for that iteration (1%, 10%, etc.).

We then generate a 1-dimensional vector the size of the column, filled with noise sampled from a random normal distribution with a mean of 0 and a standard deviation equal to the column’s standard deviation multiplied by the noise percentage.

This process will be repeated for all numerical columns (excluding categorical columns) and saved to separate CSV files with their respective noise percentage.

The code used to calculate this can be seen in the appendix.

### Example

For the thorax length column, if the standard deviation is 0.1 mm and the added noise is 10%, the standard deviation for sampling will be 0.01 (0.1 mm × 10%). A 1D vector of the column's length will be generated with values from the normal distribution centred around 0 with the calculated standard deviation.

## Model Investigation

3. Machine Learning Models

Model Selection:

Neural Networks (baseline)

Decision Trees

Random Forests

Support Vector Machines (SVM)

K-Nearest Neighbors (KNN)

Gradient Boosting Machines (e.g., XGBoost, LightGBM)

Model Architecture and Hyperparameters:

Briefly describe the architecture of the neural network and any changes made.

List the hyperparameters for each model and the ranges considered for tuning.

4. Experimental Setup

Noise Addition: Explain how noise is added to the dataset and why different levels are chosen.

Training and Evaluation: Describe the training process for each model and how you evaluate their performance.

Metrics: Define the performance metrics used for comparison (accuracy, precision, recall, F1-score, ROC-AUC).

5. Results and Analysis

Performance Comparison:

Present the performance of each model on the original and noisy datasets.

Use tables and plots (e.g., bar charts, line plots) to compare the performance metrics across models and noise levels.

Error Analysis:

Confusion matrices for the best-performing models.

Analysis of where models make errors and how noise affects these errors.

Hyperparameter Tuning Results:

Discuss the results of hyperparameter tuning and its impact on model performance.

6. Discussion

Interpretation of Results:

Discuss why certain models perform better or worse in the presence of noise.

Explain any surprising findings or deviations from expected results.

Model Robustness:

Evaluate the robustness of each model to increasing levels of noise.

Discuss any techniques used to enhance robustness (e.g., regularization, dropout).

Real-World Implications:

Consider the implications of your findings for real-world applications where data may be noisy.

7. Conclusion

Summary of Findings: Recap the key findings of your study.

Answer to Research Question: Address whether other models can outperform neural networks in noisy data contexts.

Future Work: Suggest areas for further research or potential improvements to the models.

8. Appendix

Code Snippets: Include important code snippets that highlight key parts of your implementation.

Additional Figures/Tables: Provide any additional figures or tables that support your analysis.

Investigations to Conduct

Baseline Performance:

Evaluate the performance of a simple neural network on the clean dataset.

Use this as a benchmark to compare other models.

Noise Impact Analysis:

Add Gaussian noise at different levels to the training data.

Train and evaluate the neural network and other selected models on the noisy datasets.

Hyperparameter Tuning:

Perform hyperparameter tuning for each model using techniques like Grid Search or Random Search.

Document the best hyperparameters and their impact on model performance.

Model Comparison:

Compare the performance of neural networks with other models (Decision Trees, Random Forests, SVM, KNN, Gradient Boosting Machines) on noisy data.

Use multiple metrics (accuracy, precision, recall, F1-score, ROC-AUC) for a comprehensive comparison.

Robustness Evaluation:

Analyze the robustness of each model to increasing levels of noise.

Investigate techniques to improve model robustness, such as data augmentation, regularization, or ensemble methods.

Appendix

A screen shot of a computer code

Description automatically generatedCode to read a CSV file and print the first 5 rows of it. The code snippet also includes commented out lines which provide further information about that nature of the data.

A screenshot of a computer

Description automatically generatedCode and output from printing the number of unique values in each column of the original dataset.

Code to combine species and population as well as drop columns not being used is as follows:

A computer screen with text on it

Description automatically generated

A screen shot of a computer code

Description automatically generatedCode to encode every variation of species/population and gender into a unique class:

Code to drop NaN values within the dataset.

A screen shot of a computer code

Description automatically generated

A screen shot of a computer program

Description automatically generatedCode to remove any rows which contained 0 as a measurement.

The function to add noise to the dataset can be seen below.

A screen shot of a computer code

Description automatically generated

The code to generate the class distribution graph.

A screen shot of a computer code

Description automatically generated