المعهد الهندي للتكنولوجيا دلهي
أبوظبي

INDIAN INSTITUTE OF TECHNOLOGY DELHI
ABU DHABI

Bird Mortality in Windfarms

A Comparative Analysis Using Classification Models

Course: AI for Energy Transition (AENL338)
Date: December 18, 2025

Submitted By:
Sunaina Reji Baker (24A1EENB0072)
Tanvi Parsam (24A1CSEB0022)

Contents

1 Introduction

Wind energy is a critical component of renewable energy strategies globally. However, windfarms pose significant risks to avian populations, primarily through collisions with turbine blades. Understanding the factors that lead to high-risk behaviors, such as flying at turbine height, is essential for mitigation.

1.1 Problem Statement

The core problem addressed in this project is the binary classification of bird activity. By predicting whether a bird is "Flying" (1) or performing other activities like perching or feeding (0) based on environmental and biological parameters, we can assess the potential risk of mortality.

1.2 Dataset Description

The analysis utilizes the dataset from the study *'Bird mortality at wind farms in a tropical desert'* conducted by Roy et al. (2025). The target variable `Activity` was transformed into a binary target.

2 Methodology

The project was implemented using Python, using the `pandas` library for data manipulation and `scikit-learn` for modeling.

2.1 Data Preprocessing

1. **Target Engineering:** The `Activity` column was binarized (Flying=1, Others=0).
2. **Feature Selection:** Irrelevant identifiers were dropped.
3. **Split:** 70% Training, 30% Testing.

2.2 Models Implemented

Three distinct supervised learning algorithms were trained:

2.2.1 Logistic Regression

Logistic Regression is a linear classification algorithm used to predict the probability of a binary outcome (e.g., Flying vs. Not Flying). It works by fitting a sigmoid function to the data, which maps input features to a probability value between 0 and 1. If the probability is greater than a threshold (usually 0.5), the instance is classified as positive. It is highly interpretable, allowing us to understand the influence of each feature through its weight coefficient.

2.2.2 Decision Tree

A Decision Tree is a non-linear model that splits the data into subsets based on feature values, forming a tree-like structure of decisions. It starts at a root node and asks a series of questions (e.g., "Is wind speed > 10?"). Each answer leads to a branch and eventually a leaf node, which represents the final class prediction. Decision Trees are powerful because they can capture complex interactions between variables and are easy to visualize and understand.

2.2.3 Neural Network (MLP Classifier)

A Multi-Layer Perceptron (MLP) is a type of artificial neural network inspired by the human brain. It consists of layers of interconnected nodes (neurons): an input layer, one or more hidden layers, and an output layer. Each connection has a weight that adjusts during training to minimize error. MLPs are capable of learning highly complex, non-linear patterns in data that simpler models might miss, though they often require more data to perform well and are less interpretable than trees or linear models.

3 Results

3.1 Performance Metrics

Table 1 summarizes the performance. The Decision Tree outperformed the other models.

Table 1: Model Performance Metrics

Model	**Accuracy**	**AUC Score**	**Weighted F1-Score**
Decision Tree	**0.79**	**0.84**	**0.78**
Logistic Regression	0.76	0.83	0.76
Neural Network	0.69	0.77	0.69

3.2 Visual Analysis

3.2.1 ROC Curve Comparison

The ROC curve (Figure 1) illustrates the trade-off between sensitivity and specificity. The Decision Tree and Logistic Regression show similar areas under the curve, significantly better than the Neural Network.

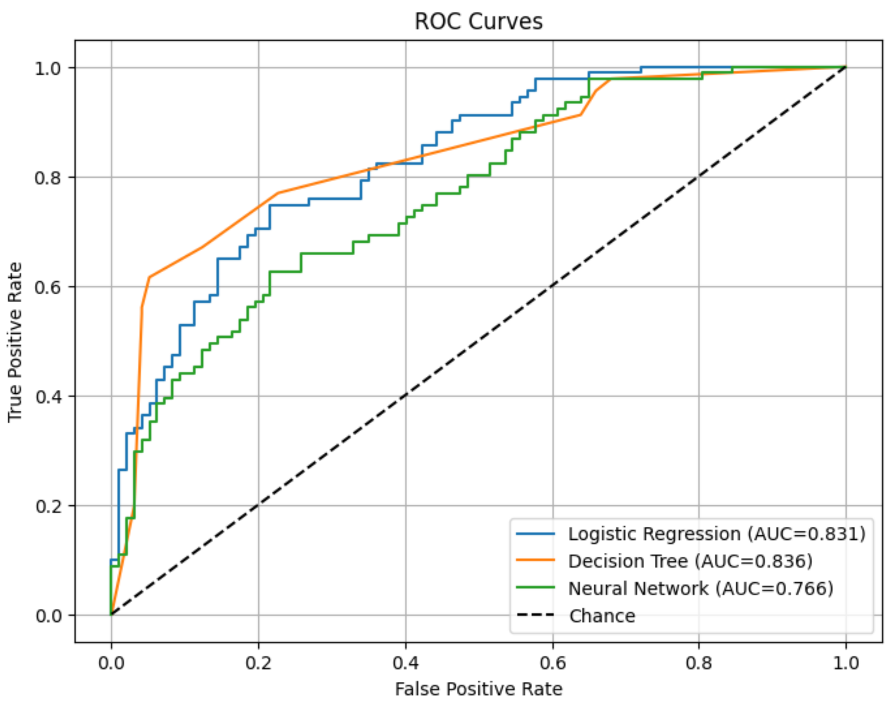


Figure 1: ROC Curve Comparison of all three models

3.2.2 Decision Tree Analysis

The Decision Tree achieved the best accuracy. Figure 2 shows its confusion matrix, highlighting its ability to correctly classify both flying and non-flying birds with reasonable precision.

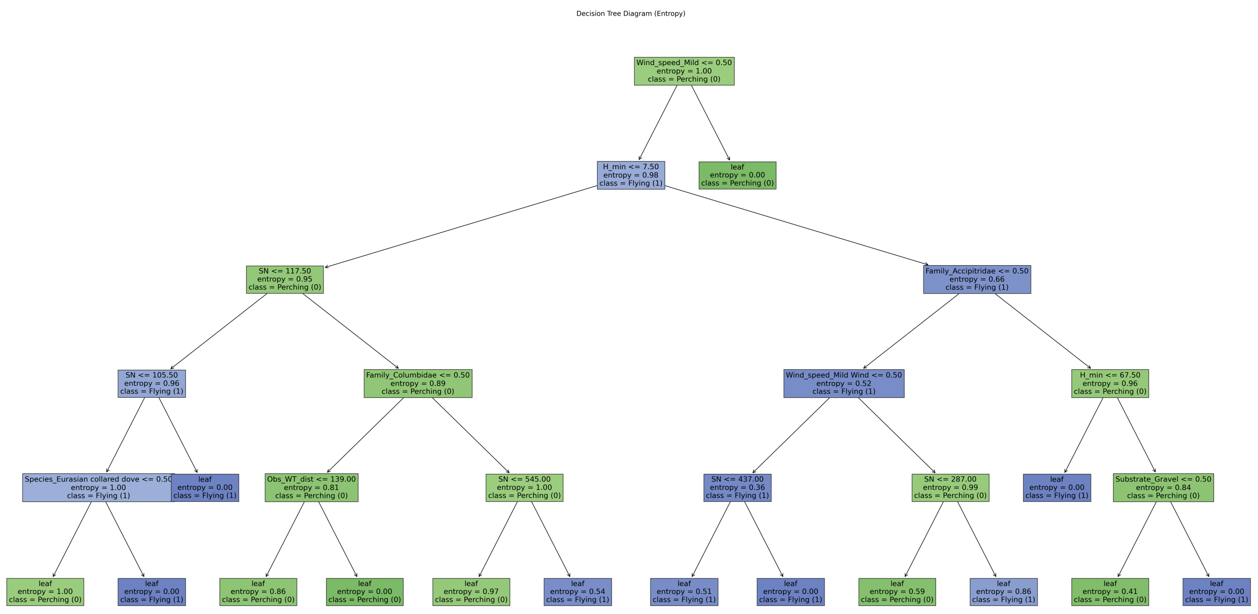

Decision Tree Diagram (Entropy)
Wind_speed_Mild <= 0.50 entropy = 1.00 class = Perching (0)
H_min <= 7.50 entropy = 0.98 class = Flying (1)
leaf entropy = 0.00 class = Perching (0)
SN <= 117.50 entropy = 0.95 class = Perching (0)
Family_Accipitridae <= 0.50 entropy = 0.66 class = Flying (1)
SN <= 105.50 entropy = 0.96 class = Flying (1)
Family_Columbidae <= 0.50 entropy = 0.89 class = Perching (0)
Wind_speed_Mild Wind <= 0.50 entropy = 0.52 class = Flying (1)
H_min <= 67.50 entropy = 0.96 class = Perching (0)
Species_Eurasian collared dove <= 0.50 entropy = 1.00 class = Flying (1)
leaf entropy = 0.00 class = Flying (1)
Obs_WT_dist <= 139.00 entropy = 0.81 class = Perching (0)
SN <= 545.00 entropy = 1.00 class = Perching (0)
SN <= 437.00 entropy = 0.36 class = Flying (1)
SN <= 287.00 entropy = 0.99 class = Perching (0)
leaf entropy = 0.00 class = Flying (1)
Substrate_Gravel <= 0.50 entropy = 0.84 class = Perching (0)
leaf entropy = 1.00 class = Perching (0)
leaf entropy = 0.00 class = Flying (1)
leaf entropy = 0.86 class = Perching (0)
leaf entropy = 0.00 class = Perching (0)
leaf entropy = 0.97 class = Perching (0)
leaf entropy = 0.54 class = Flying (1)
leaf entropy = 0.51 class = Flying (1)
leaf entropy = 0.00 class = Flying (1)
leaf entropy = 0.59 class = Perching (0)
leaf entropy = 0.86 class = Flying (1)
leaf entropy = 0.41 class = Perching (0)
leaf entropy = 0.00 class = Flying (1)

Figure 2: Decision Tree

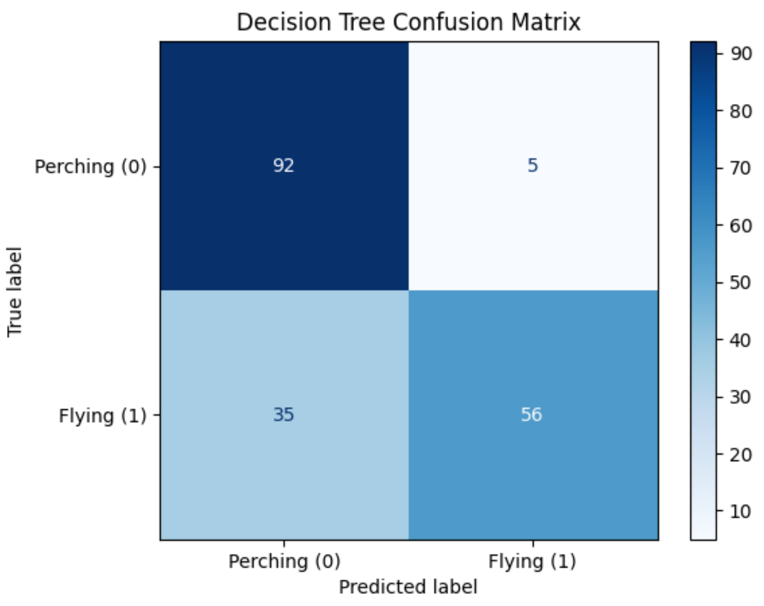


Figure 3: Confusion Matrix for Decision Tree Classifier

3.2.3 Logistic Regression Analysis

Logistic Regression performed robustly with 76% accuracy. Figure 4 shows its confusion matrix, while Figure 5 illustrates the top 10 feature weights. This graph reveals that specific species families (like Accipitridae) and wind conditions are the strongest predictors, with negative weights indicating a lower likelihood of flying.

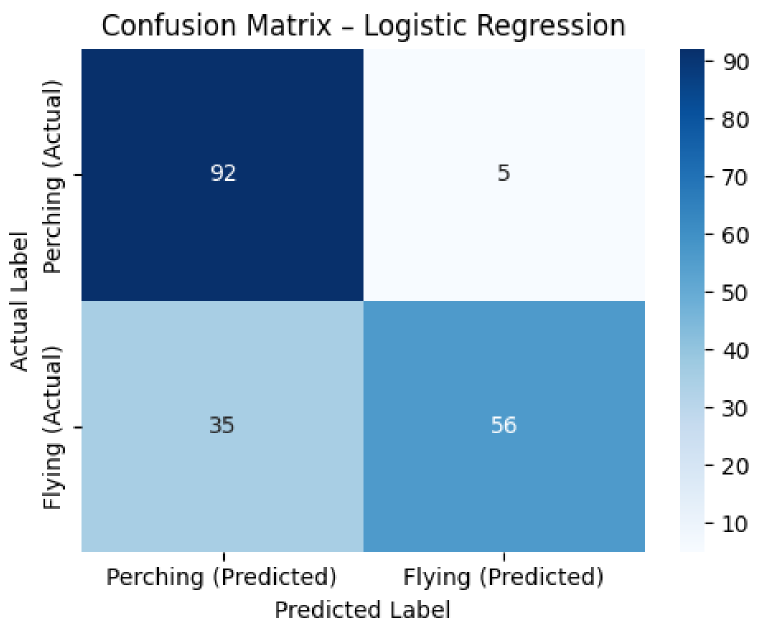


Figure 4: Confusion Matrix for Logistic Regression

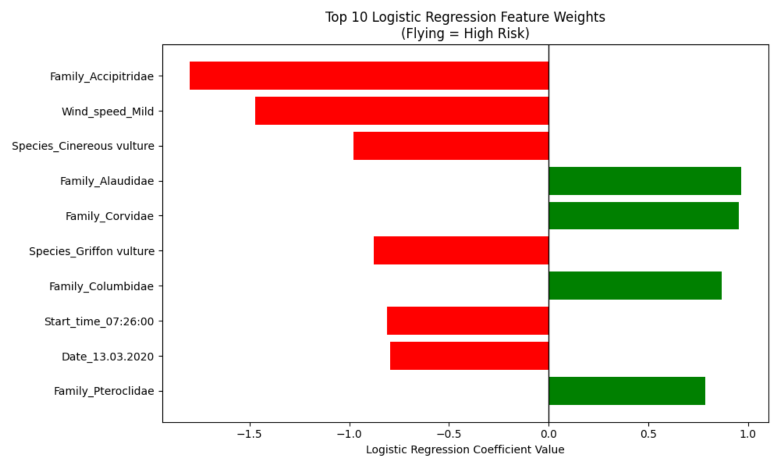
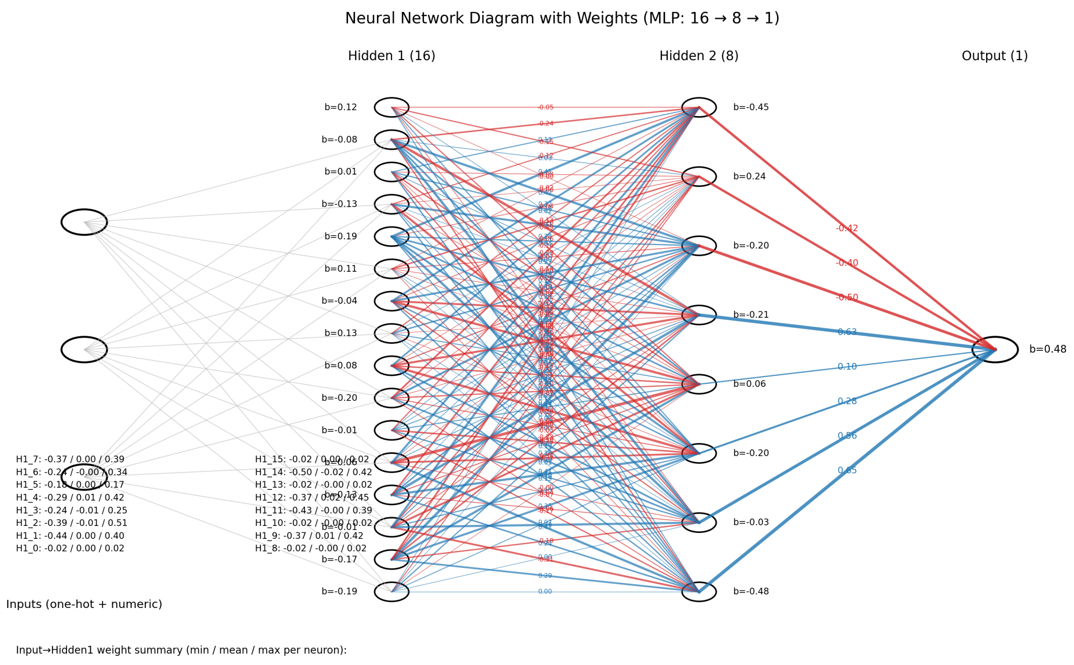


Figure 5: Feature Weights in Logistic Regression

3.2.4 Neural Network Analysis

The Neural Network showed the lowest performance (69%). Due to the complexity and size of the network visualization, the full architecture and weight details are provided in the attached SMT file. This file contains the complete model structure (Input → 16 Neurons → 8 Neurons → Output) used to capture non-linear patterns in the data.



Input→Hidden1 weight summary (min / mean / max per neuron):

```
H1_7: -0.37 / 0.00 / 0.39
H1_6: -0.24 / -0.00 / 0.34
H1_5: -0.18 / 0.00 / 0.17
H1_4: -0.29 / 0.01 / 0.42
H1_3: -0.24 / -0.01 / 0.25
H1_2: -0.39 / -0.01 / 0.51
H1_1: -0.44 / 0.00 / 0.40
H1_0: -0.02 / 0.00 / 0.02
```

```
H1_15: -0.02 / 0.00 / 0.02
H1_14: -0.50 / -0.02 / 0.42
H1_13: -0.02 / -0.00 / 0.02
H1_12: -0.37 / 0.02 / 0.45
H1_11: -0.43 / -0.00 / 0.39
H1_10: -0.02 / -0.00 / 0.02
H1_9: -0.37 / 0.01 / 0.42
H1_8: -0.02 / -0.00 / 0.02
```

Figure 6: Neural Network Structure

```

; Neural Network Verification (Flying vs Perching)
; SMT-LIB v2 format


(set-logic QF_LRA)

; INPUT FEATURES
(declare-fun x1 () Real) ; feature 1 (e.g., WindSpeed)
(declare-fun x2 () Real) ; feature 2 (e.g., WT_Distance)
```

```

; HIDDEN LAYER 1 (2 neurons example)
(declare-fun h1_1 () Real)
(declare-fun h1_2 () Real)

; Weights from input -> hidden1
(define-fun w11 () Real  0.42)
(define-fun w12 () Real -0.18)
(define-fun w21 () Real  0.33)
(define-fun w22 () Real  0.71)

; Biases
(define-fun b1_1 () Real 0.12)
(define-fun b1_2 () Real -0.09)

; ReLU activation
(assert (= h1_1 (ite (>= (+ (* w11 x1) (* w21 x2) b1_1) 0)
                     (+ (* w11 x1) (* w21 x2) b1_1)
                     0)))

(assert (= h1_2 (ite (>= (+ (* w12 x1) (* w22 x2) b1_2) 0)
                     (+ (* w12 x1) (* w22 x2) b1_2)
                     0)))

; OUTPUT LAYER
(declare-fun out () Real)

; output weights
(define-fun wo1 () Real  1.15)
(define-fun wo2 () Real -0.77)
(define-fun bo  () Real -0.30)

(assert (= out (+ (* wo1 h1_1) (* wo2 h1_2) bo)))

; CLASSIFICATION RULE
; Flying if output >= 0
(declare-fun Flying () Bool)
(assert (= Flying (>= out 0)))

; QUERY
; Example: is there an input where Flying is true?
(check-sat)
(get-model)
```

Listing 1: Neural Network smt2 code

4 Conclusion

The **Decision Tree Classifier** proved to be the most effective model (79% accuracy). Key factors influencing prediction included wind speed and specific species families (e.g., Accipitridae). Future work should focus on expanding the dataset to improve Neural Network convergence.

5 Python Code

The following code was used to preprocess data, train models, and generate the results presented in this report.

```python
# LOGISTIC REGRESSION
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score,
    classification_report, roc_curve


# 1. Target: Flying=1, Perching=0
df["Target"] = df["Activity"].apply(
    lambda x: 1 if str(x).lower()=="flying" else 0
)

# 2. Remove ID-type columns
drop_cols = ["Activity","Target","sn","id","date"]
drop_cols = [c for c in drop_cols if c in df.columns]
X = df[[c for c in df.columns if c not in drop_cols]]
y = df["Target"]


# 3. 70/30 split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30, random_state=42, stratify=y
)


# 4. Preprocessing
numeric = X_train.select_dtypes(include=["int64","float64"]).columns.tolist
    ()
categorical = X_train.select_dtypes(include=["object"]).columns.tolist()

preprocess = ColumnTransformer([
    ("num", Pipeline([
        ("imputer", SimpleImputer(strategy="median")),
        ("scaler", StandardScaler())
    ]), numeric),
    ("cat", OneHotEncoder(handle_unknown="ignore"), categorical),
])

pipe_lr = Pipeline([
    ("preprocess", preprocess),
    ("lr", LogisticRegression(max_iter=5000, solver="lbfgs"))
])



# 5. Logistic Regression pipeline
pipe_lr = Pipeline([
    ("preprocess", preprocess),
    ("lr", LogisticRegression(max_iter=1000))
])

pipe_lr.fit(X_train, y_train)


```

```python
# 6. Evaluation
y_pred = pipe_lr.predict(X_test)
y_prob = pipe_lr.predict_proba(X_test)[:,1]

print("\n=== Logistic Regression ===")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("AUC:", roc_auc_score(y_test, y_prob))
print(classification_report(y_test, y_pred))


# 7. Overfitting check
print("Train Accuracy:", pipe_lr.score(X_train, y_train))
print("Test Accuracy :", pipe_lr.score(X_test, y_test))

# ROC curve data for later plotting
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_prob)
auc_lr = roc_auc_score(y_test, y_prob)


from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# 8. Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# 9. Plot heatmap
plt.figure(figsize=(5, 4))
sns.heatmap(
    cm,
    annot=True,
    fmt="d",
    cmap="Blues",
    xticklabels=["Perching (Predicted)", "Flying (Predicted)"],
    yticklabels=["Perching (Actual)", "Flying (Actual)"]
)

plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
plt.title("Confusion Matrix     Logistic Regression")
plt.tight_layout()
plt.show()
```

Listing 2: Logistic Regression

```python
# DECISION TREE CLASSIFIER (Flying = High Risk, Perching=Low)

import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.tree import DecisionTreeClassifier, plot_tree

from sklearn.metrics import accuracy_score, roc_auc_score,
    classification_report, roc_curve, confusion_matrix,
    ConfusionMatrixDisplay

```

```python
import matplotlib.pyplot as plt

# 1. Target = Flying (1) vs Perching (0)
df["Target"] = df["Activity"].apply(
    lambda x: 1 if str(x).lower()=="flying" else 0
)


# 2. Drop columns not used
drop_cols = ["Activity","Target","sn","id","date"]
drop_cols = [c for c in drop_cols if c in df.columns]

X = df[[c for c in df.columns if c not in drop_cols]]
y = df["Target"]


# 3. 70/30 train test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.30,
    random_state=42,
    stratify=y
)


# 4. Preprocessing
numeric = X_train.select_dtypes(include=["int64","float64"]).columns.tolist
    ()
categorical = X_train.select_dtypes(include=["object"]).columns.tolist()

preprocess = ColumnTransformer([
    ("num", SimpleImputer(strategy="median"), numeric),
    ("cat", OneHotEncoder(handle_unknown="ignore"), categorical)
])


# 5. Decision Tree model
pipe_dt = Pipeline([
    ("preprocess", preprocess),
    ("dt", DecisionTreeClassifier(
        criterion="entropy",
        max_depth=5,
        random_state=42
    ))
])


# 6. Fit
pipe_dt.fit(X_train, y_train)


# 7. Evaluate
y_pred = pipe_dt.predict(X_test)
y_prob = pipe_dt.predict_proba(X_test)[:,1]

print("\n=== Decision Tree Results ===")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("AUC:", roc_auc_score(y_test, y_prob))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

```

```
# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", cm)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Perching
    (0)", "Flying (1)"])
disp.plot(cmap="Blues")
plt.title("Decision Tree Confusion Matrix")
plt.show()


# 8. Overfitting check
print("\n=== Overfitting Check ===")
print("Train Accuracy:", pipe_dt.score(X_train, y_train))
print("Test Accuracy :", pipe_dt.score(X_test, y_test))

# ROC curve data for later plotting
fpr_dt, tpr_dt, _ = roc_curve(y_test, y_prob)
auc_dt = roc_auc_score(y_test, y_prob)


# 9. DISPLAY THE TREE DIAGRAM
dt_model = pipe_dt.named_steps["dt"]

plt.figure(figsize=(20,10))
plot_tree(
    dt_model,
    filled=True,
    feature_names=list(numeric)+list(pipe_dt.named_steps["preprocess"]
                                    .named_transformers_["cat"]
                                    .get_feature_names_out(categorical)),
    class_names=["Perching (0)", "Flying (1)"]
)
plt.title("Decision Tree Diagram (Entropy)")
plt.show()
```

Listing 3: Decision Trees

```
# NEURAL NETWORK (Flying vs Perching)

import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, roc_auc_score,
    classification_report, roc_curve

# 1. Target
df["Target"] = df["Activity"].apply(
    lambda x: 1 if str(x).lower()=="flying" else 0
)


# 2. Drop columns
drop_cols = ["Activity","Target","sn","id","date"]
drop_cols = [c for c in drop_cols if c in df.columns]

```

```python
X = df[[c for c in df.columns if c not in drop_cols]]
y = df["Target"]


# 3. Train-test split (70/30)
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.30,
    random_state=42,
    stratify=y
)


# 4. Preprocessing
numeric = X_train.select_dtypes(include=["int64","float64"]).columns.tolist()
categorical = X_train.select_dtypes(include=["object"]).columns.tolist()

preprocess = ColumnTransformer([
    ("num", SimpleImputer(strategy="median"), numeric),
    ("cat", OneHotEncoder(handle_unknown="ignore"), categorical)
])


# 5. Neural Network model
pipe_nn = Pipeline([
    ("preprocess", preprocess),
    ("nn", MLPClassifier(
        hidden_layer_sizes=(16,8),   # 2 hidden layers: 16     8 neurons
        activation="relu",
        solver="adam",
        max_iter=1000,
        random_state=42
    ))
])

pipe_nn.fit(X_train, y_train)


# 6. Evaluation
y_pred = pipe_nn.predict(X_test)
y_prob = pipe_nn.predict_proba(X_test)[:,1]

print("\n=== Neural Network Results ===")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("AUC:", roc_auc_score(y_test, y_prob))
print(classification_report(y_test, y_pred))


# 7. Overfitting check
print("\n=== Overfitting Check ===")
print("Train Accuracy:", pipe_nn.score(X_train, y_train))
print("Test Accuracy :", pipe_nn.score(X_test, y_test))

# ROC curve data for later plotting
fpr_nn, tpr_nn, _ = roc_curve(y_test, y_prob)
auc_nn = roc_auc_score(y_test, y_prob)


# 8. PRINT WEIGHTS FOR EVERY LAYER

# Extract fitted model (after pipeline)
nn_model = pipe_nn.named_steps["nn"]

print("\n=== Neural Network Architecture ===")
print("Hidden layers:", nn_model.hidden_layer_sizes)
print("Activation   :", nn_model.activation)

# Get transformed feature names:
ohe = pipe_nn.named_steps["preprocess"].named_transformers_["cat"]
cat_names = ohe.get_feature_names_out(categorical)
input_feature_names = list(numeric) + list(cat_names)

# Weight matrices
W_input_hidden1 = nn_model.coefs_[0]   # input -> hidden1
W_hidden1_hidden2 = nn_model.coefs_[1] # hidden1 -> hidden2
W_hidden2_output = nn_model.coefs_[2]  # hidden2 -> output

print("\n=== WEIGHTS: input -> hidden layer 1 ===")
print("Shape:", W_input_hidden1.shape)
print("Each column is a neuron in layer 1")
print(pd.DataFrame(W_input_hidden1, index=input_feature_names))

print("\n=== WEIGHTS: hidden layer 1 -> hidden layer 2 ===")
print("Shape:", W_hidden1_hidden2.shape)
print(pd.DataFrame(W_hidden1_hidden2))

print("\n=== WEIGHTS: hidden layer 2 -> output ===")
print("Shape:", W_hidden2_output.shape)
print(pd.DataFrame(W_hidden2_output))
```
Listing 4: Neural Network

6 References

- IBM. (n.d.). *What is logistic regression?* https://www.ibm.com/think/topics/logistic-regression
- IBM. (n.d.). *What is a decision tree?* https://www.ibm.com/think/topics/decision-trees
- Wikipedia. (n.d.). *Neural network (machine learning)*. https://en.wikipedia.org/wiki/Neural_network_(machine_learning)
- Roy, A., Banerjee, S., Uddin, M. et al. *Bird mortality at wind farms in a tropical desert.* Sci Rep 15, 19221 (2025). https://doi.org/10.1038/s41598-025-03407-8