CS-E4650 – Homework 3

STEFANO BOSOPPI
ID: 103370763
stefano.bosoppi@aalto.fi

BÁLINT JÁNOS PRÁGAI
ID: 103390468
balintjanos.pragai@aalto.fi

BENEDEK GABOR KISS
ID: 103574543
benedekgabor.kiss@aalto.fi

November 10, 2025

Contents

1 Methods

We implemented the data mining pipeline using a shell script. We performed feature extraction from the provided CSV file using a custom program written in Python 3, shown in Appendix A. The `namescodes` utility was used to convert the extracted text-based features into a numerical format for mining and to translate the resulting rules back into a human-readable format.

We conducted association rule discovery, considering both positive and negative ones, using the Kingfisher program. We searched for the top 300 rules. We used Kingfisher's default goodness measure, $ln(p_F)$, as the mutual information measure yielded almost the same results. The other parameter settings included an upper bound for the item number of 170, a lower frequency threshold of 20%, a confidence lower threshold of at least 90%, and an initial goodness measure threshold of -5.

We used the following constraints, as suggested by the task description file:

```
eats_insects eats_invertebrates
eats_worms eats_invertebrates
eats_snails eats_invertebrates
eats_larvae eats_invertebrates
eats_invertebrates eats_insects eats_worms eats_snails eats_larvae
```

We found no other set of rules that would make sense to exclude and would make a significant difference on the output.

2 Feature extraction

We extracted features from the `birds2025ext.csv` file using a Python script, reported in section A, to generate a space-separated transactional file. The extraction logic, detailed below, was designed to convert numerical, categorical, and text-based data into a binary attribute format suitable for association rule mining.

2.1 Numerical features

Numerical features were processed to capture the most informative extremes, rather than using the full range of values.

Intervals Columns containing ranges (e.g., "10-12") or single numbers, such as `length`, `wspan`, `weight`, `eggs`, `arrives`, and `leaves`, were first converted to a single floating-point number representing their mean value.

Derived indices Two indices, Body Mass Index (BMI) and Wing Span Index (WSI), were calculated from these mean values:

$$BMI = \frac{\texttt{weight}/1000}{(\texttt{length}/100)^2} \tag{1}$$

$$WSI = \frac{\texttt{wspan}}{\texttt{length}} \tag{2}$$

We performed unit conversion for the BMI to respect its definition.

Discretization The `mean_eggs`, `BMI`, and `WSI` features were discretized using their first (Q1) and third (Q3) quartiles. Values in the bottom quartile ($\leq$ Q1) were assigned a `low_<name>` attribute (e.g., `low_eggs`, `low_wsi`), and values in the top quartile ($\geq$ Q3) were assigned a `high_<name>` attribute (e.g., `high_eggs`, `high_bmi`).

2.2 Categorical features

Categorical features were converted into binary attributes. The specific logic depended on the feature type, as summarized in Table 1.

Feature type	Original columns	Extraction logic
Taxonomy	`group`	The group name was added with a `group_` prefix (e.g., `group_laridae`). A hard-coded mapping was used to also add the more general parent group (e.g., `group_lari`) based on the taxonomy.
Binary (Yes)	`diver, long-billed, webbed-feet, long-feet, wading-bird, plunge-dives`	The feature name was added to the transaction *only* if its value was "Yes". (e.g., `webbed_feet`).
Binary (Both)	`sim`	Both "Yes" and "No" values were considered informative. "Yes" was converted to `genders_similar` and "No" to `genders_dissimilar`.
Multi-element	`diet`, `biotope`	The text was split, and each element was added with a prefix. (e.g., `eats_fish`, `livesin_lakes`).
Multi-valued	`back, belly, ftype, billcol, legcol, incub, ccare`	Each value was split and added with a prefix. (e.g., `back_brown`, `ftype_a`, `incub_f`, `ccare_both`).

Table 1: Summary of categorical feature extraction logic.

2.3 Custom migration features

To capture migration timing, two binary features were created based on the mean arrival and departure months:

- `migrates_arrives_early`: Added if the mean arrival month was ≤ 4.0, meaning April or earlier.
- `migrates_leaves_late`: Added if the mean departure month was ≥ 10.0, meaning October or later.

3 Results

From the script output we have identified 4 groups outputs, that are likely to correlate to real-life observations.

3.1 Childcare responsibility division

```
incub_both -> genders_similar fr=31 (0.6200), cf=1.000, gamma=1.250,
 ↪  delta=0.124, M=-1.162e+01
incub\_both -> ccare_both fr=28 (0.5600), cf=0.903, gamma=1.328, delta=0.138,
 ↪  M=-1.056e+01
```

Based on these two rules we can conclude that if both parents attend the incubation period, they are rather likely to continue with the upbringing of the offspring. In this case the two genders are also similar, likely to camouflage themselves and the nest while attending the eggs.

```
ccare_f -> incub_f fr=12 (0.2400), cf=1.000, gamma=2.778, delta=0.154,
 ↪  M=-1.569e+01
genders_dissimilar -> incub_f fr=10 (0.2000), cf=1.000, gamma=2.778,
 ↪  delta=0.128, M=-1.237e+01}
```

Based on the rule "female parent cares for chicks → female parent incubates" is a likely and biologically supported rule, however it would be expected for the reasoning to be the other way around, as the incubating parent would be the likely one to also continue with the upbringing. The other rule is similar in the sense, that it has support in biology, but one would expect reasoning to be in the other direction, however when the two genders are visually distinct it is usually observed that females (who lay the eggs) are the one who incubate them. For this of course they need camouflage colours, while in species like these the males oftentimes have very bright and outstanding colours in order to attract mates.

3.2 Physical attributes as predictors for lifestyle

```
webbed_feet -> ~wading_bird fr=21 (0.4200), cf=0.955, gamma=1.646, delta=0.165,
 ↪  M=-1.378e+01
webbed_feet -> ~eats_small-rodents fr=22 (0.4400), cf=1.000, gamma=1.250,
 ↪  delta=0.088, M=-6.663e+00
```

The first rule highlights a clear functional divergence. Webbed feet are a specific adaptation for propulsion in the water. This is functionally incompatible with the "wading" lifestyle, which typically requires long, un-webbed toes to spread weight on soft mud or move through shallow water with minimal drag. The rule confirms that these two adaptations are mutually exclusive.

The second rule shows that the webbed feet rules out a terrestrial predatory niche. A foot structure optimized for swimming is morphologically unsuited for grasping and killing of land-based prey like rodents. The 100% confidence shows that this ecological boundary is, within this dataset, a strict and unbreakable rule.

3.3 Biological groups identification

For this experiment, we lowered the frequency threshold to 10% to have a higher species granularity in the hopes of catching rules defining biological groups.

In the following paragraphs, we present the rules that in our opinion best described three biological groups.

Accipitridae

This group, diurnal birds of prey, is defined by a very strong set of interconnected features:

```
group_accipitridae -> ftype_a fr=8 (0.1600), cf=1.000, gamma=5.556, M=-1.790e+01
group_accipitridae -> legcol_yellow fr=8 (0.1600), cf=1.000, gamma=4.545,
 ↪  M=-1.500e+01
ccare_both legcol_yellow -> group_accipitridae fr=8 (0.1600), cf=1.000,
 ↪  gamma=6.250, M=-2.010e+01
```

They show that in this dataset, every bird in the `group_accipitridae` has flying type A, flapping and soaring flight, and yellow legs. The third rule is even more interesting, showing that the combination of shared parental care and yellow legs is a perfect predictor for this group.

Anatinae

This group, ducks, is also defined by a strong set of rules, particularly related to appearance and breeding.

```
genders_dissimilar high_eggs -> group_anatinae fr=7 (0.1400), cf=1.000,
 ↪  gamma=7.143, M=-1.842e+01
webbed_feet ccare_f -> group_anatinae fr=7 (0.1400), cf=1.000, gamma=7.143,
 ↪  M=-1.842e+01
group_anatinae -> genders_dissimilar fr=7 (0.1400), cf=1.000, gamma=5.000,
 ↪  M=-1.363e+01
group_anatinae -> ccare_f fr=7 (0.1400), cf=1.000, gamma=4.167, M=-1.174e+01
```

These rules have extremely high confidence and lift, linking the biological group to observable differences between the genders, high egg count, and female-only chick care.

It is also interesting to see how a sub-group is distinguished:

```
    back_dappled genders_dissimilar high_eggs -> group_dabbling_ducks fr=5
     ↪  (0.1000), cf=1.000, gamma=10.000, M=-1.457e+01
```

By adding the `back_dappled` feature to the existing Antinae rules, like `genders_dissimilar` and `high_eggs`, this rule separates dabbling ducks from other Anatinae, getting a perfect predictor for this subgroup. The extremely high lift, 10.0, confirms this is a very strong, specific association.

Scolopacidae

This group, waders, is clearly defined by its feeding habits and related physical traits.

```
    eats_invertebrates long_billed -> group_scolopacidae fr=5 (0.1000),
     ↪  cf=1.000, gamma=6.250, M=-1.054e+01
    group_scolopacidae -> eats_invertebrates fr=8 (0.1600), cf=1.000,
     ↪  gamma=2.778, M=-9.415e+00
    belly_white eats_invertebrates wading_bird -> group_scolopacidae fr=7
     ↪  (0.1400), cf=0.875, gamma=5.469, M=-1.428e+01
```

These rules strongly link the group to its ecological niche. The first rule is a perfect (cf=1.000) predictor: a diet of invertebrates, combined with a long bill, is a defining characteristic. The second rule confirms that all birds in this group eat invertebrates. The third rule adds appearance, white belly, and behavior, wading bird, to create a strong, high-lift predictor for the group.

A Code

Source Code 1: Python code written for the feature extraction.

```python
from pathlib import Path
from typing import Any

import numpy as np
import pandas as pd

```

```python
INPUT_FILE = "./birds2025ext.csv"
OUTPUT_FILE = "birdstrans.txt"

TAXONOMY_MAP = {
    "Laridae": "Lari",
    "Sternidae": "Lari",
    "Scolopacidae": "Charadrii",
    "Charadriidae": "Charadrii",
    "dabbling ducks": "Anatinae",
    "diving ducks": "Anatinae",
}

BINARY_COLS = [
    "diver",
    "long-billed",
    "webbed-feet",
    "long-feet",
    "wading-bird",
    "plunge-dives",
]
MULTI_VALUE_COLS = ["back", "belly", "ftype", "billcol", "legcol", "incub",
 ↪  "ccare"]


def parse_interval(val: Any) -> float:
    """Convert interval string 'X-Y' to mean, or return float value."""
    if pd.isna(val):
        return np.nan
    s = str(val)
    if "-" in s:
        try:
            parts = s.split("-")
            return (float(parts[0]) + float(parts[1])) / 2
        except (ValueError, IndexError):
            pass
    try:
        return float(s)
    except ValueError:
        return np.nan


def extract_prefixed_features(text: str, prefix: str) -> list[str]:
    """Split multi-element field and return prefixed feature list."""
    if pd.isna(text):
        return []
    elements = str(text).replace(",", " ").split()
    return [f"{prefix}_{e.lower().replace(' ', '_')}" for e in elements if e]

```

```python

def discretize_to_extremes(df: pd.DataFrame, col: str, feature_name: str) ->
↪ pd.Series:
    """Return discretized features for extreme values (Q1/Q3)."""
    q1, q3 = df[col].quantile([0.25, 0.75])

    def categorize(val):
        if pd.isna(val):
            return None
        if val <= q1:
            return f"low_{feature_name}"
        elif val >= q3:
            return f"high_{feature_name}"
        return None

    return df[col].apply(categorize)


def extract_row_features(row: pd.Series, discretized_features: list[str]) ->
↪ list[str]:
    """Extract all features for a single bird species."""
    features = []

    for feat in discretized_features:
        if pd.notna(row[feat]):
            features.append(row[feat])

    group = row.get("group")
    if pd.notna(group):
        group_clean = str(group).lower().replace(" ", "_")
        features.append(f"group_{group_clean}")
        if group in TAXONOMY_MAP:
            features.append(f"group_{TAXONOMY_MAP[group].lower()}")

    features.extend(extract_prefixed_features(row.get("diet"), "eats"))
    features.extend(extract_prefixed_features(row.get("biotope"), "livesin"))

    for col in BINARY_COLS:
        if row.get(col) == "Yes":
            features.append(col.replace("-", "_"))

    sim = row.get("sim")
    if sim == "Yes":
        features.append("genders_similar")
    elif sim == "No":
        features.append("genders_dissimilar")

    for col in MULTI_VALUE_COLS:
        val = row.get(col)
        if pd.notna(val):
            parts = str(val).lower().replace("-", " ").split()
            features.extend(f"{col}_{p}" for p in parts)

    arrives = row.get("mean_arrives")
    leaves = row.get("mean_leaves")

    if pd.notna(arrives) and arrives <= 4:
        features.append("migrates_arrives_early")
    if pd.notna(leaves) and leaves >= 10:
        features.append("migrates_leaves_late")

    return sorted(set(features))


def main():
    if not Path(INPUT_FILE).exists():
        print(f"Error: Could not find {INPUT_FILE}")
        return

    df = pd.read_csv(INPUT_FILE, sep=";", index_col="species")

    interval_cols = ["length", "wspan", "weight", "eggs", "arrives", "leaves"]
    for col in interval_cols:
        if col in df.columns:
            df[f"mean_{col}"] = df[col].apply(parse_interval)

    df["BMI"] = (df["mean_weight"] / 1000) / (df["mean_length"] / 100) ** 2
    df["WSI"] = df["mean_wspan"] / df["mean_length"]

    discretized_cols = []
    for col, name in [("mean_eggs", "eggs"), ("BMI", "bmi"), ("WSI", "wsi")]:
        if col in df.columns:
            feat_col = f"disc_{name}"
            df[feat_col] = discretize_to_extremes(df, col, name)
            discretized_cols.append(feat_col)

    transactions = [
        extract_row_features(row, discretized_cols) for _, row in df.iterrows()
    ]

    with open(OUTPUT_FILE, "w") as f:
        for features in transactions:
            f.write(" ".join(features) + "\n")

    print(f"  Feature extraction complete. Saved to {OUTPUT_FILE}")
    print(
        f"  Processed {len(transactions)} species with {sum(len(t) for t in
         ↪  transactions)} total features"
    )


if __name__ == "__main__":
    main()
```