CS-E4650 Methods of Data Mining

Exercise 3.4 Bird associations

:

Group members

Nguyen Xuan Binh (887799)

Erald Shahinas (906845)

Alexander Pavlyuk (906829)

Table of Contents

1. Methods
2. Feature extraction
3. Results
4. Appendix

1. Methods

All the calculations have been perfomed on JypyterHub (https://jupyter.cs.aalto.fi) in the Python notebook. Additionally, numpy (https://numpy.org/), matplotlib (https://matplotlib.org/), pandas (https://pandas.pydata.org/), and scikit-learn (https://scikit-learn.org/stable/index.html) libraries have been imported to handle specific functions.

In [ ]:
```
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans, SpectralClustering
from sklearn.metrics import silhouette_score, calinski_harabasz_score, normalize
from sklearn.metrics.pairwise import rbf_kernel
```

Learning goals: Mining association rules in practice; making efficient data mining pipelines

This is an explorative task, where you should invent good features to extract from the extended bird data and then search and analyze association rules. The pattern discovery process is iterative, and you will very likely experiment with multiple versions of feature extraction. Therefore, it is recommended to do the preparations well and make a shell script that speeds up the process. You can find instructions and hints in MyCourses (instructionsforkingfisher.pdf). You can find an extended version of the bird species data, birdspeciesv2.csv, and its description in MyCourses.

2. Feature extraction

In [ ]:
```
birddf = pd.read_csv('birdspeciesv2.csv', sep=';')
birddf.head()
```

Out[ ]:

	species	group	length	wspan	weight	back	belly	sim	billcol	legcol	..
0	pikkulokki	laridae	25-27	62-69	100-150	light-grey	white	Yes	black	red	..
1	naurulokki	laridae	34-38	86-99	200-350	light-grey	white	Yes	red	red	..
2	kalalokki	laridae	40-44	99-108	350-500	bluish-grey	white	Yes	yellow	greenish-yellow	..
3	selkälokki	laridae	52-60	117-134	700-800	black	white	Yes	yellow	yellow	..
4	harmaalokki	laridae	55-65	123-148	800-1300	bluish-grey	white	Yes	yellow	reddish	..

5 rows × 23 columns

Describe compactly but carefully what features you extracted. You can, e.g., use a list or a table that tells the original feature, new attributes, and how they were extracted. Describe carefully non-trivial extraction (like handling numerical values or migration month ranges).

(a) Extract good features for association discovery from the bird data. You can find interesting associations only, if the involved properties are captured by features! It is suggested to proceed iteratively, from easier to more difficult features:

– Group, habitat and diet can be used as such (just list group and all elements of habitat and diet in the transaction).

In [ ]:
```python
group = birddf["group"]
group.drop_duplicates(inplace=True)
group_vals = ["group_"+g for g in birddf["group"].to_list()]
group_ft = list(set(["group_"+item for item in group]))
print(f"The {len(group_ft)} bird groups are: ")
print(group_ft)
print()

diet = birddf["diet"]
diet = diet.to_numpy()
diet = [line.split(",") for line in diet]
diet_ft = list(set(["diet_"+item for sublist in diet for item in sublist]))
print(f"The {len(diet_ft)} diet are: ")
print(diet_ft)
print()
diet_vals = [["diet_"+item for item in sublist] for sublist in diet]
# print(diet_vals)

habitat = birddf["biotope"]
habitat = habitat.to_numpy()
habitat = [line.split(",") for line in habitat]
habitat_ft = list(set(["habitat_"+item for sublist in habitat for item in sublis
print(f"The {len(diet_ft)} habitat are: ")
print(habitat_ft)
habitat_vals = [["habitat_"+item for item in sublist] for sublist in habitat]
# print(habitat_vals)
```

```
The 15 bird groups are:
['group_gruifores', 'group_rallidae', 'group_ardeidae', 'group_dabbling-ducks',
'group_diving-ducks', 'group_phalacrocoracidae', 'group_laridae', 'group_cygnin
i', 'group_anserini', 'group_podicipedidae', 'group_gaviidae', 'group_scolopacida
e', 'group_haematopodidae', 'group_charadriidae', 'group_sternidae']

The 23 diet are:
['diet_snails', 'diet_garbage', 'diet_lizards', 'diet_shellfish', 'diet_seeds',
'diet_snakes', 'diet_berries', 'diet_invertebrates', 'diet_frogs', 'diet_chicks',
'diet_algae', 'diet_fish', 'diet_worms', 'diet_vertebrae', 'diet_insects', 'diet_
plankton', 'diet_grass', 'diet_plants', 'diet_clams', 'diet_grain', 'diet_larva
e', 'diet_small-rodents', 'diet_molluscs']

The 23 habitat are:
['habitat_pastures', 'habitat_nutrient-rich-lakes', 'habitat_reedbeds', 'habitat_
fells', 'habitat_marshland', 'habitat_coastal-meadows', 'habitat_archipelago', 'h
abitat_wetlands', 'habitat_lakes', 'habitat_meadows', 'habitat_seashores', 'habit
at_fields', 'habitat_streams', 'habitat_shores', 'habitat_forests', 'habitat_sea-
bays', 'habitat_islets', 'habitat_sea-coast', 'habitat_ponds']
```

– For most binary features (like long-billed), you can use only the Yes-values (list attribute “long-billed” in the transaction, but forget its opposite, “non-long-billed”). The only exception is field sim, where both values are interesting (if genders look similar or different).

In [ ]:
```
# Binary features
binary_ft = [col for col in birddf.columns if birddf[col].isin(['Yes', 'No']).al

binary_vals = []
for i in range(len(group_vals)):
    binary_vals.append([])
    for ft in binary_ft:
        if birddf[ft][i] == "Yes":
            binary_vals[i].append(ft)


# Adding non similarity feature
binary_ft += ["non-sim"]
for i in range(len(group_vals)):
    if birddf["sim"][i] == "No":
        binary_vals[i].append("non-sim")

print("The binary features are:\n")
print(binary_ft)
```

```
The binary features are:

['sim', 'diver', 'long-billed', 'webbed-feet', 'long-legs', 'wading-bird', 'plung
e-dives', 'non-sim']
```

– For multi-valued categorical features, you can create one attribute for each value.

In [ ]:
```
# Each value of the category is its own feature
# Color features only apply to the female.
# They apply to male also only if sim is True
```

```
color_cols = ["back","belly","billcol","legcol"]
cat_columns = color_cols + ["incub" ,"ccare"]
print(f"The {len(cat_columns)} multivalued categorical features are\n")
print(birddf[cat_columns].head())

cat_ft = []

for col in cat_columns:
    cat_ft += [col+"_"+item for item in birddf[col].unique()]
cat_ft
# print(cat_ft)

cat_vals = []
for i in range(len(group_vals)):
    cat_vals.append([])
    for ft in color_cols:
        if birddf["sim"][i] == "Yes":
            cat_vals[i].append(ft+"_"+birddf[ft][i])

for i in range(len(group_vals)):
    for ft in ["incub" ,"ccare"]:
        cat_vals[i].append(ft+"_"+birddf[ft][i])
# print(cat_vals)
```

```
The 6 multivalued categorical features are

          back  belly billcol           legcol incub ccare
0   light-grey  white   black              red  both  both
1   light-grey  white     red              red  both  both
2  bluish-grey  white  yellow  greenish-yellow  both  both
3        black  white  yellow           yellow  both  both
4  bluish-grey  white  yellow          reddish  both  both
```

– Invent some informative features from the spring and autumn migrations times (fields "arrives" and "leaves"), e.g., describing that migration starts early or ends late.

In [ ]:
```
#Features for bird if it arrives early or late

# Check each row if the period starts at March or before
def is_march_or_before(row):
    # List of months to check
    months_before_april = ['January', 'February', 'March']

    return any(month in row for month in months_before_april)

# Check each row if the period end at October or after
def is_october_or_after(row):
    # List of months to check
    months_after_september = ['October', 'November', 'December']

    return any(month in row for month in months_after_september)

arrives = birddf["arrives"]
leaves = birddf["leaves"]
arrives_e = arrives.apply(is_march_or_before)
early_vals = []
for i in range(len(group_vals)):
    early_vals.append([])
    if arrives_e[i]:
        early_vals[i].append("arrives_early")
leaves_l = leaves.apply(is_october_or_after)
late_vals = []
for i in range(len(group_vals)):
    late_vals.append([])
    if leaves_l[i]:
        late_vals[i].append("leaves_late")


migration_ft = ["arrives_early", "leaves_late"]
print("The two migration features are")
print(migration_ft)
```

```
The two migration features are
['arrives_early', 'leaves_late']
```

– Invent how to handle numerical features. Usually, only the extremes are interesting, like laying relatively few eggs or many eggs.

In [ ]:
```
#Numerical features
num_columns = ["length", "wspan", "weight", "eggs"]
num_ft = ["extreme_"+col for col in num_columns]
print("The numerical features are:")
print(num_ft)

num_values = birddf[num_columns]

#find the mean of each range
for col in num_columns:
    num_values[col] = num_values[col].apply(lambda x: np.mean([float(item) for i


# find the standard deviation of each column
std = num_values.std(axis=0)


num_vals = []
# extreme feature is true only if the range mean is more than 2 standard deviati
for i in range(len(group_vals)):
    num_vals.append([])
    for col in num_columns:
        extremes = num_values[col].apply(lambda x: True if x > std[col]*2 else '
        if extremes[i]:
            num_vals[i].append("extreme_"+col)
```

```
The numerical features are:
['extreme_length', 'extreme_wspan', 'extreme_weight', 'extreme_eggs']
```

```
<ipython-input-47-a45ea5826824>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  num_values[col] = num_values[col].apply(lambda x: np.mean([float(item) for item
in x.split("-")]))
```

In [ ]:
```
# New feature - BMI

# Mean weight values
```

```
weight = birddf["weight"].to_numpy()
wight_mean = np.array([])
for interval in weight:
    lower, upper = map(int, interval.split('-'))
    mean = (lower + upper) / 2
    wight_mean = np.append(wight_mean, mean)

# Mean length values
length = birddf["length"].to_numpy()
length_mean = np.array([])
for interval in length:
    lower, upper = map(int, interval.split('-'))
    mean = (lower + upper) / 2
    length_mean = np.append(length_mean, mean)

# Mean BMI index
bmi_vals = (wight_mean/1000) / (length_mean/100)**2

# Binary BMI feature
bmi_ft = ["high_BMI", "small_BMI"]
high_bmi_vals = np.where(bmi_vals > bmi_vals.mean(), "high_BMI", "small_BMI")

# New feature - WSI

# Mean wing span values
wspan = birddf["wspan"].to_numpy()
wspan_mean = np.array([])
for interval in wspan:
    lower, upper = map(int, interval.split('-'))
    mean = (lower + upper) / 2
    wspan_mean = np.append(wspan_mean, mean)

# Mean WSI index
wsi_vals = wspan_mean / length_mean

# Binary WSI feature
wsi_ft = ["high_WSI", "small_WSI"]
high_WSI_vals = np.where(wsi_vals > wsi_vals.mean(), "high_WSI", "small_WSI")
```

In [ ]:
```
#adding all the features together
features =  group_ft + diet_ft + habitat_ft + binary_ft + cat_ft + migration_ft
features_df = pd.DataFrame(features)

#adding space in the end for namescodes program to work
features_df['Space'] = ''

features_df.to_csv("birdtable.txt", sep=" ", header=False)
```

In [ ]:
```
values = [[group_vals[i], *diet_vals[i], *habitat_vals[i], *binary_vals[i],*cat_


values_df = pd.DataFrame(values)
print(values_df.head())
#adding space in the end for namescodes program to work
values_df['Space'] = ''

values_df.to_csv("birdstrans.txt", sep=" ", header=False, index=False)
```

```
                0            1                    2              3   \
0  group_laridae  diet_insects            diet_fish  diet_plankton
1  group_laridae     diet_fish  diet_invertebrates   diet_garbage
2  group_laridae    diet_worms            diet_fish   diet_insects
3  group_laridae     diet_fish   habitat_sea-coast  habitat_lakes
4  group_laridae     diet_fish         diet_garbage    diet_chicks

                4                            5                  6   \
0     diet_worms  habitat_nutrient-rich-lakes      habitat_ponds
1  habitat_lakes             habitat_sea-bays                sim
2  habitat_lakes         habitat_archipelago  habitat_marshland
3            sim                  webbed-feet       plunge-dives
4     diet_grain               habitat_lakes  habitat_sea-coast

                   7             8                9  ...           18  \
0   habitat_sea-bays           sim      webbed-feet  ...     high_WSI
1        webbed-feet  plunge-dives  back_light-grey  ...         None
2                sim   webbed-feet     plunge-dives  ...     high_WSI
3         back_black   belly_white   billcol_yellow  ...         None
4  habitat_marshland           sim      webbed-feet  ...  leaves_late

                 19              20        21        22    23    24    25    26  \
0              None            None      None      None  None  None  None  None
1              None            None      None      None  None  None  None  None
2              None            None      None      None  None  None  None  None
3              None            None      None      None  None  None  None  None
4  extreme_length  extreme_wspan  high_BMI  high_WSI  None  None  None  None

     27
0  None
1  None
2  None
3  None
4  None

[5 rows x 28 columns]
```

(b) Search association rules with Kingfisher. You may need to search quite many rules (e.g., 300) to find more versatile rules, since there will be many variants of similar associations. Try to find rules that describe different aspects of the data, like different groups, appearance, diet, habits, environment, etc, but remember that all attributes do not necessarily participate any significant associations.

Step 1: We need to convert the birdspeciesv2.csv into the encoded version like the mushroom.dat file

After running the previous code, we obtained the transformed features file **birdstrans.txt**, which contains 64 rows for 64 birds, each row contains the interesting features filtered from part (a). The next text file is **birdtable.txt**, which has only two columns separated by a white space. The first column is the code index and the second column is the feature name. The number of features in "birdtable.txt" corresponds to the number of unique entry values in the file "birdstrans.txt".

Suppose that the current working directory contains the kingfisher and namescodes folder, we run this command to transform labels to numerical codes

After running this command, namescodes create a new file called **birdstrans.txt.codes**, which contains encoded numbers for the features since kingfisher can only work with numeric values.

```
$ ./namescodes/namescodes -n birdstrans.txt -tbirdtable.txt -L
```

Step 2: After that, we define the constraints in the file **constraint.txt** as follows

```
diet_insects diet_invertebrates
diet_worms diet_invertebrates
diet_snails diet_invertebrates
diet_larvae diet_invertebrates
diet_invertebrates diet_insects diet_worms diet_snails
diet_larvae

diet_snakes diet_vertebrae
diet_chicks diet_vertebrae
diet_fish diet_vertebrae
diet_frogs diet_vertebrae
diet_lizards diet_vertebrae
diet_vertebrae diet_lizards diet_frogs diet_fish diet_chicks
diet_snakes

diet_grass diet_plants
diet_grain diet_plants
diet_plants diet_grain diet_grass

diet_algae diet_plankton
diet_plankton diet_algae

diet_clams diet_shellfish
diet_molluscs diet_shellfish
diet_shellfish diet_clams diet_molluscs
diet_clams diet_molluscs
diet_molluscs diet_clams
```

Then, the constraints are compiled by namescodes using information from birdtable.txt

```
$ ./namescodes/namescodes -n constraint.txt -t birdtable.txt
```

The encoded file for constraints is created with the name **constraint.txt.codes**

Step 3: Kingfisher is run to find the most significant associating rules, given the constraints.

In this context, we choose these parameters *k = 140*, which corresponds to the upper bound of number of features in birdtable.txt

q = 300, which means the program outputs max 100 positive association.

M = -5, which means the program uses the default values, and outputs 100 top rules above based on the criteria $ln(p_F) \leq -5$

Note is that these are just placeholder parameters to help us build the pipeline, as we have not tried out kingfisher thoroughly yet.

The running command of Kingfisher is

```
$ ./kingfisher/kingfisher -i birdstrans.txt.codes -k 152 -M -5 -q300 -o birdrules.txt -b constraint.txt.codes
```

The rules found by Kingfisher is then printed to the output file **birdrules.txt**. The first few lines show the associating patterns between the features. However, we cannot understand the encoded version, so we need to translate it.

```
2 9 18 -> 10 fr=10 (0.1562), cf=1.000, gamma=6.400, delta=0.132,
M=-2.574e+01
9 12 18 -> 10 fr=10 (0.1562), cf=1.000, gamma=6.400,
delta=0.132, M=-2.574e+01
9 51 -> 49 fr=13 (0.2031), cf=1.000, gamma=4.267, delta=0.156,
M=-2.555e+01
etc ...
```

Step 4: We translate the numbers back to original features in string format.

The command to do this is

```
$ ./namescodes/namescodes -c birdrules.txt -t birdtable.txt
```

which outputs the final translation for association result file named **birdrules.txt.names**. Here are the first few lines in that file showing the associating patterns between the features

```
diet_fish webbed-feet high_WSI -> plunge-dives fr=10 (0.1562),
cf=1.000, gamma=6.400, delta=0.132, M=-2.574e+01
webbed-feet belly_white high_WSI -> plunge-dives fr=10 (0.1562),
cf=1.000, gamma=6.400, delta=0.132, M=-2.574e+01
webbed-feet ccare_F -> non-sim fr=13 (0.2031), cf=1.000,
gamma=4.267, delta=0.156, M=-2.555e+01
etc...
```

How to interpret the rules in **birdrules.txt.names**

Take for example the first line:

diet_fish webbed-feet high_WSI -> plunge-dives fr=10 (0.1562), cf=1.000, gamma=6.400, delta=0.132, M=-2.574e+01

Antecedent (Left-hand side): diet_fish webbed-feet high_WSI: This part of the rule represents a combination of conditions for association.

Consequent (Right-hand side): plunge-dives: This is the outcome or behavior associated with the conditions on the left-hand side

Metrics:

fr=10: This is the absolute frequency, indicating that there are 10 rows in the dataset where the combination of diet_fish, webbed-feet, and high_WSI is associated with plunge-dives.

(0.1562): This is the support of the rule, calculated as the absolute frequency divided by the total number of rows. As the dataset has 64 rows in total, then 0.1562 = 10/64.

cf=1.000: This is the confidence of the rule, calculated as the absolute frequency divided by the number of instances where the antecedent is true. In this case, it suggests that whenever a bird has a diet of fish, webbed feet, and a high WSI, it always (100% of the time) engages in plunge-diving.

gamma=6.400: This is the lift of the rule, calculated as the confidence of the rule divided by the overall probability of the consequent in the dataset. A lift value greater than 1 indicates a positive statistical association between the antecedent and consequent.

delta=0.132: This is the leverage of the rule, calculated as the support of the rule minus the product of the support of the antecedent and the support of the consequent. It measures the difference between the observed frequency of the antecedent and consequent appearing together and what would be expected if they were independent. A positive value indicates a positive association.

M=-2.574e+01: This is the Mutual Information for the rule, which measures the amount of information gained about the consequent (plunge-dives) given the antecedent (diet_fish, webbed-feet, high_WSI). A higher MI value indicates a stronger association between the antecedent and consequent.

The shell script **runbirds.sh** that we used is the combination of previous commands:

```
./namescodes/namescodes -n birdstrans.txt -tbirdtable.txt -L
./namescodes/namescodes -n constraint.txt -tbirdtable.txt
./kingfisher/kingfisher -i birdstrans.txt.codes -k152 -M-5 -q300
-o birdrules.txt -b constraint.txt.codes
./namescodes/namescodes -c birdrules.txt -tbirdtable.txt
```

3. Results

(c) Report the most significant and interesting rules. The idea is not to list all rules, but group rules and describe the information they reveal (e.g., what things are associated to scolopacidae or plunge-divers). Tell also which features seem to be irrelevant (did not occur in any rules). Describe the most significant and interesting associations you discovered

After tuning the results, here is the final version shell script that we haved used to uncover the patterns of bird species

```
./namescodes/namescodes -n birdstrans.txt -t birdtable.txt -L
./namescodes/namescodes -n constraint.txt -t birdtable.txt
./kingfisher/kingfisher -i birdstrans.txt.codes -k 140 -t1 -M -5
-q 50 -o birdrules.txt - b constraint.txt.codes
./namescodes/namescodes -c birdrules.txt -t birdtable.txt
```

t = 3 means both positive and negative rules are considered.

q = 50 as we only considers the top 50 most significant associations.

1. What things are associated to scolopacidae?

To know the rules that are associated with scolopacidae, we can search all lines occurring in **birdrules.txt.names** using the grep command:

```
$ grep -e 'scolopacidae' birdrules.txt.names >
scolopacidae_rules.txt
```

From the output file scolopacidae_rules.txt, we can infer that the Scolopacidae bird group has the following common associations:

- small BMI
- diet of invertebrates
- wading bird
- long billed

2. What things are associated to plunge-divers?

To know the rules that are associated with plunge-divers, we can search all lines occurring in **birdrules.txt.names** like before

```
$ grep -e 'plunge-divers' birdrules.txt.names >
plunge_divers_rules.txt
```

From the output file scolopacidae_rules.txt, we can infer that the Scolopacidae bird group has the following common associations:

- webbed-feet

- high WSI
- white belly
- diet of fish

Tell also which features seem to be irrelevant (did not occur in any rules).

We use this shell scripts to find irrelevant rules:

```
# Extract feature names from birdtable.txt, ignoring the index
numbers
cut -d ' ' -f 2- birdtable.txt > feature_names.txt

# Check each feature in feature_names.txt against
birdrules.txt.names
while IFS= read -r feature; do
  if ! grep -q "$feature" birdrules.txt.names; then
    echo "$feature"
  fi
done < feature_names.txt > irrelevant_features.txt
```

There are lots of irrelevant features and we cant list all. Some of them are:

```
diet_insects
diet_plankton
diet_worms
habitat_nutrient-rich-lakes
habitat_ponds
habitat_sea-bays
back_light-grey
billcol_black
legcol_red
diet_garbage
habitat_lakes
```

Unfortunately, due time restrictions, we cannot afford to uncover more interesting patterns. Given more time, we can definitely try to mine more associating patterns in the future

Appendix

All the code for this exercise has been added with respect to each part for closest referencing. Therefore, we do not attach any more code here in the Appendix section