Predictive model to aid environmental impact assessments

April 6, 2021

1 Introduction

An environmental impact assessment (EIA) is one of the tools being used worldwide to provide decision-makers and the concerned public with essential information to plan for environmentally sustainable economic development (1). The goal of economic development is to invest in projects or programmes - such as the construction of mines, reservoirs, highways and housing to enhance the prosperity and quality of life of all residents. Sustainable economic development aims to find a compromise between economic development and environmental protection.

The responsible development of spaces requires an assessment of the value of natural spaces. Spaces with low natural value can be developed as investments. **The presence of amphibians (frogs, toads and salamanders) in natural spaces contributes to the value of natural spaces**. Amphibians are a critical part of nature as both predator and prey. They eat insect pests which a is benefit to argriculture and help control mosquitos which benefits human health. They're are one of the main links in many ecosystem food webs (2).

Amphibians are almost entirely dependent on water for reproduction. Their eggs have no water-proof covering and their larvae (tadpoles) are quite fish like. Amphibians also tend to return to the same water reservoir where they were born to breed. They also have a land habitat on which they depend to feed, hide and survive winter.

As part of an EIA, field work is required to estimate amphibian populations in and near water reservoirs where economic development projects are planned. Determining the contribution of amphibian populations to the value of a natural space is difficult for several reasons. In the case of road infrastructure projects, the area that needs to be assessed can be very large. Certain areas of the terrain can be difficult to reach. The number of people available to do field work can be limited. **An initial assessment of the natural space could help to better plan field work**

which should result in time and cost savings. This can all contribute to helping reduce the underestimation of amphibian populations and the value of natural spaces.

A geographic information system (GIS) can provide information about the habitats of amphibians. A GIS integrates many different kinds of information (3). It can include information about landscape, such as the location of stream, different kinds of vegetation, and different kinds of soil. It can also include information about the sites of factories, farms, schools and houses. (4) **As amphibians are highly dependant on their habitat, the assessment of their habitats using features obtained from a GIS, can potentially be used to predict their presence.**

2 The Dataset

The data used in this project can be obtained from the UCI Machine Learning Repository. The data was derived from GIS and sattelite images, as well as from information gathered about the size of amphibians populations that formed part of EIA reports for two planned road projects (Road A and Road B) in Poland.

Marcin Blachnik, Marek SoÅ‚tysiak, Dominika DÄ…browska Predicting presence of amphibian species using features obtained from GIS and satellite images. ISPRS International Journal of Geo-Information 8 (3) pp. 123. MDPI. 2019

2.1 Data Description

The data consists of 15 input variables and a multi-label target variable with 7 possible values indicating the presence of a certain type of frog. Multiple species of frog can be present in the same location.

2.1.1 Attribute Information - Features

	Description	Type
MV	Motorway	categorical
SR	Surface of water reservoir (m2)	numerical
NR	Number of water reservoirs in habitat	numerical
TR	Type of water reservoirs	categorical
VR	Presence of vegetation within the reservoirs	categorical
SUR1	Surrounding types of land cover (primary)	categorical
SUR2	Surrounding types of land cover (secondary)	categorical
SUR3	Surrounding types of land cover (tertiary)	categorical
FR	The presence of fishing	categorical
OR	Percentage access to undeveloped areas	categorical
BR	Minimum distance to buildings	ordinal
MR	Maintenance status of the reservoir	categorical
CR	Type of shore	categorical

2.1.2 Attribute Information - Target

	Description
Label 1	The presence of Green frogs
Label 2	The presence of Brown frogs
Label 3	The presence of Common toad
Label 4	The presence of Fire-bellied toad
Label 5	The presence of Tree frog
Label 6	The presence of Common newt
Label 7	The presence of Great Crested newt

3 Problem Statement

Ideally, the estimation of amphibian populations that forms part of an environmental impact assessment should be completed in the most cost effective manner without underestimation. In reality, the field work required to estimate amphibian populations can be very difficult and time consuming as a result of the terrain conditions and the vast terrain that needs to be assessed. Underestimation can lead to an underevaluation of natural spaces where road infrastructure or other economic development projects will happen. This can have devastating consequences on the future prosperity and quality of life of all natural beings.

An initial estimate of amphibian populations in sub-locations of the field work study area can help to better plan the area of focus for the field work teams and aid the efficient allocation of resources. This project aims to create a predictive model model, trained on features obtained from publically available GIS data and satellite images, to predict the presence of different species of frogs in study area sub-locations.

4 Problem Approach

This is a supervised, multi-label classification problem. This is not a multi-class classification problem as the classes are not mutually exclusive. Multiple species of amphibians can be found at a site.

4.1 Classification Algorithms

Several classification algorithms exist and have an implementation in scikit-learn. The algorithms are:

Algorithm	Name of implementation in scikit-learn
K-Nearest Neighbours	KNeighboursClassifier
	RadiusNeighborsClassifier
Logistic Regression	LogisticRegression
Support Vector Machines	SVC
	NuSVC
	LinearSVC
Stochastic Gradient Descent	SGDClassifier
Decision Trees	DecisionTreeClassifier
Randomized Forests	RandomForestClassifier

Algorithm	Name of implementation in scikit-learn
AdaBoost	AdaBoostClassifier
Gradient Tree Boosting	GradientBoostingClassifier
Neural Network	MLPClassifier

Keep in mind that the above list is not exhaustive and many more variants of these algorithms exist.

K-Nearest Neighbours works well for small datasets and can be a good baseline model to compare other algorithms against. It is important to ensure that features are standardized for K-Nearest Neighbours.

A decision tree could be interesting for this problem. Decision trees are interpretable and the ideas behind a decision tree is easily understood by nonexperts. They also work well for a mixture of binary and continuous features which is the case for this project's feature space. The disadvantage of decision trees is that they tend to overfit and provide poor generalization performance.

Randomized Forests, AdaBoost and Gradient Tree Boosting are all ensemble methods. Ensemble methods combine multiple machine learning models to create better performing models. Random Forests is one way to address to the overfitting problem of decision trees. Ramdom Forests often work well out of the box and don't require scaling of the data. They also work well with a mixture of binary and continuous variables.

Like random forests, gradient boosted trees are widely used in industry. They can be more sensitive to parameter settings than random forests but provide better accuracy if the parameters are set correctly. AdaBoost works by combining multiple weak learners into a single strong classifier.

A neural network or more specific a mulitlayer perceptron (MLP) can also be used for classification. An MLP is sensitive to feature scaling and has multiple hyperparameters to tune.

A final decision will made following an initial exploration of the data to decide which alogrithms are suitable for the problem.

4.2 Multi-label problem approach

This is a multi-label classification problem as the predictive model should be able to predict whether multiple amphibian species are present in a location i.e. In location with ID 2, there were brown frogs, common toads and common newts present.

scikit-learn has functionality for meta-estimators. In scikit-learn an estimator can be a predictive model of some specific algorithm. Base estimators work for binary classification problems. Meta-estimators extend the functionality of of the base estimator to support multi-learning problems like multi-class, multi-label and multi-output classification problems. The scikit-learn multioutput module, and specifically the MultiOutputClassifier can be used for multi-label classification problems. The MultiOutputClassifier makes it possible to fit one classifier per target. For this particular problem, this approach would divide the problem into 7 binary classification tasks, one for each species of amphibians.

Some classifiers are actually capable of natively supporting multi-label classification. Native multi-label classifiers may treat the multiple classes simultaneously and can therefore account for corre-

lated behaviour among them. In scikit-learn, the following algorithms natively support multilabel classification:

- tree.DecisionTreeClassifier
- tree.ExtraTreeClassifier
- ensemble.ExtraTreesClassifier
- neighbors.KNeighborsClassifier
- neural_network.MLPClassifier
- neighbors.RadiusNeighborsClassifier
- ensemble.RandomForestClassifier
- linear_model.RidgeClassifierCV

AdaBoost and GBT are not natively multi-label classifiers.

5 Performance Metrics

The information included in the description of the dataset does not give any specifics regarding the evaluation of the predictive model's performance like whether the correct prediction of all species is equally important.

A final decision will made following an initial exploration of the data to decide which performance metric is suitable.

Accuracy

The accuracy of a classifier is the number of correct predictions made as a ratio of all predictions made. Accuracy is a very common evaluation metric for classification problems but is not suitable when their are an unequal number of observations in each class. We will check for class imbalance during the exploratory data analysis section of this project. In scikit-learn, the accuracy_score function returns the subset accuracy in the case of multi-label classification. The subset accuracy per sample will be 1.0 if the entire set of predicted labels for a sample match the true set of labels otherwise it would be 0. Consider the example of subset accuracy for a multi-label classification. The classifier incorrectly predicted Label 1 for Sample 0. It correctly predicted for Sample 2. The classifier achieved a subset accuracy of 50%.

[91]:
```
# An example demonstrating subset accuracy

import pandas as pd

y_true = pd.DataFrame({'Label 1': [0, 1], 'Label 2':[1, 1]})
y_pred = pd.DataFrame({'Label 1': [1, 1], 'Label 2':[1, 1]})
y_true
```

[91]:

	Label 1	Label 2
0	0	1
1	1	1

[92]:
```
y_pred
```

[92]:

	Label 1	Label 2
0	1	1
1	1	1

[93]:
```
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score

accuracy_score(y_true, y_pred)
0.5
```

[93]: 0.5

Confusion Matrix

A confusion matrix is another popular method for evaluating the performance of a classifier. scikit-learn provides an implementation of a multilabel_confusion_matrix. The multilabel_confusion_matrix outputs a confusion matrix per label and the true negatives (TN), false negatives (FN), false positives (FP) and true positives (TP) for each label can be read from each matrix in a similar fashion to a binary classification confusion matrix. The dataframe `MCM` below is a reminder of the location of each value in the matrix.

As the underestimation of amphibian populations could mean that natural spaces are under-valued. An under-estimation would be the result of too many false negatives i.e. Falsely deciding a site would not have amphibians present. Conversely, too many false positives could increase costs of field work because sites that had no potential to host amphibians would be included in the field work study.

[94]:
```
import pandas as pd

MCM = pd.DataFrame({'0': ['true negatives', 'false negatives'], '1': ['false␣
↪positives', 'true positives']})
MCM
```

[94]:

	0	1
0	true negatives	false positives
1	false negatives	true positives

[95]:
```
# An example demonstrating the use of a multilabel_confusion_matrix taken from␣
↪the scikit-learn documentation
# The target has 3 labels
import numpy as np
from sklearn.metrics import multilabel_confusion_matrix

y_true = np.array([[1, 0, 1],
                   [0, 1, 0]])
y_pred = np.array([[1, 0, 0],
                   [0, 1, 1]])
multilabel_confusion_matrix(y_true, y_pred)
```

[95]:
```
array([[[1, 0],
        [0, 1]],

       [[1, 0],
        [0, 1]],

       [[0, 1],
        [1, 0]]])
```

In the example above, the first matrix is the confusion matrix for Label 1. The classifier correctly predicted Label 1 for Sample 0 and 1. The number of TN is 1 and TP is 1. The last matrix is the confusion matrix for Label 3. The classifier incorrectly predicted Label 3 for both samples. The number of FP and FN is 1.

ROC AUC

An ROC Curve has true postive rate on the Y axis and false positive rate on the X axis. The ROC summarizes the performance of a binary classification model on the positive class. The ideal point is in the top left corner with a false positive rate of zero and a true positive rate of one. In the real world, this would rarely be the case but it does mean that a larger area under the curve (AUC) is usually better. A perfect classifier would have a ROC AUC equal to 1 and a purely random classifier will have a ROC AUC equal to 0.5.

The ROC is a popular metric for evaluating classifiers in balanced and unbalanced data as it is not biased towards the majority or minority class and the ROC AUC score is a single value that can be used to compare binary classifiers directly [5]. ROC AUC can also be misleading if there is severe skew and very few examples of the minority class. The reason is that a small number of correct or incorrect predictions can result in a large change in the ROC curve or ROC AUC score.

6 Exploratory Data Analysis

In this section, an initial exploratory analysis of the dataset will be done. The dataset will be evaluated for any data quality problems that have to be resolved.

[96]:
```
# Import modules needed exploratory analysis
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

%matplotlib inline
```

The data for both road projects is contained in a single csv file. The repository claims that there are no missing values.

[97]:
```
# View the frist 5 rows of the dataset
amphibians = pd.read_csv('amphibians.csv', sep=';', header=1, index_col='ID')
amphibians.head()
```

[97]:
```
   Motorway   SR  NR  TR  VR  SUR1  SUR2  SUR3  UR  FR  …  BR  MR  CR  \
ID                                                        …
1         A1  600   1   1   4     6     2    10   0   0  …   0   0   1
2         A1  700   1   5   1    10     6    10   3   1  …   1   0   1
3         A1  200   1   5   1    10     6    10   3   4  …   1   0   1
4         A1  300   1   5   0     6    10     2   3   4  …   0   0   1
5         A1  600   2   1   4    10     2     6   0   0  …   5   0   1

    Green frogs  Brown frogs  Common toad  Fire-bellied toad  Tree frog  \
ID
1             0            0            0                  0          0
2             0            1            1                  0          0
3             0            1            1                  0          0
4             0            0            1                  0          0
5             0            1            1                  1          0

    Common newt  Great crested newt
ID
1             0                   0
2             1                   0
3             1                   0
4             0                   0
5             1                   1

[5 rows x 22 columns]
```

The first 5 rows of the dataset does not reveal anything unusual. The feature names and target labels are consistent with the dataset description. The values for the categorical features seem to be numerical which is different to what was expected based on the dataset description.

[98]:
```
# Check the automatically assigned data types
amphibians.dtypes
```

[98]:
```
Motorway     object
SR            int64
NR            int64
TR            int64
VR            int64
SUR1          int64
SUR2          int64
SUR3          int64
UR            int64
FR            int64
OR            int64
RR            int64
BR            int64
MR            int64
```

```
CR                      int64
Green frogs             int64
Brown frogs             int64
Common toad             int64
Fire-bellied toad       int64
Tree frog               int64
Common newt             int64
Great crested newt      int64
dtype: object
```

Note that the 11 categorical features and 2 ordinal features are integer values. This is interesting because the description of the dataset describes the features has having text or character levels such a-j for the TR feature.

[99]:
```
# Check for missing values
print("Count of missing values in each column of the dataset:")
amphibians.isna().sum()
```

```
Count of missing values in each column of the dataset:
```

[99]:
```
Motorway              0
SR                    0
NR                    0
TR                    0
VR                    0
SUR1                  0
SUR2                  0
SUR3                  0
UR                    0
FR                    0
OR                    0
RR                    0
BR                    0
MR                    0
CR                    0
Green frogs           0
Brown frogs           0
Common toad           0
Fire-bellied toad     0
Tree frog             0
Common newt           0
Great crested newt    0
dtype: int64
```

[100]:
```
# Determine the number of observations in the dataset
amphibians.shape
```

[100]:
```
(189, 22)
```

The dataset is quite small with only a 189 observations. A small dataset increases the risk of overfitting.

The dataset will be split into train and test set. Cross validation should be used to evaluate the model/s on the training set.

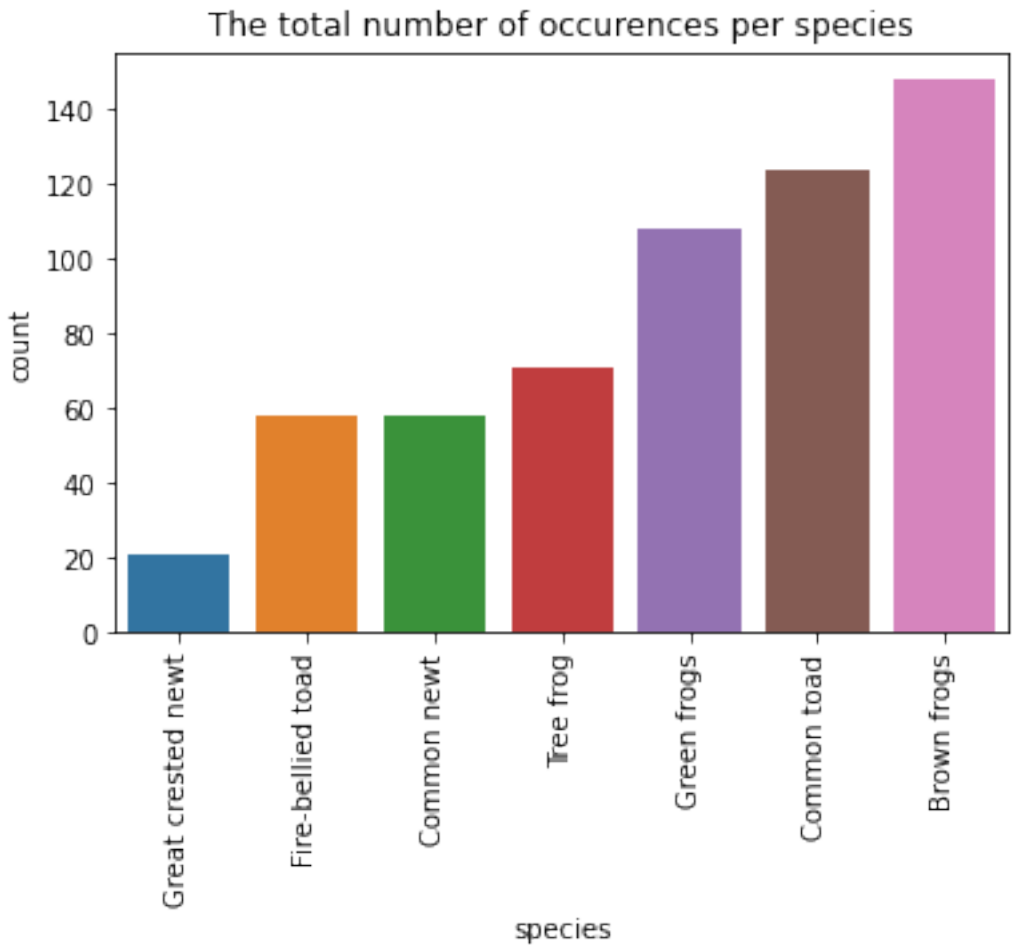
[101]:
```
# View the number of frogs per species
species = ['Green frogs', 'Brown frogs', 'Common toad', 'Fire-bellied toad',␣
↪'Tree frog', 'Common newt', 'Great crested newt']
species_count = amphibians[species].sum().to_frame('count').reset_index()
species_count.columns = ['species', 'count']
species_count = species_count.sort_values('count')
species_count['percentage'] = round(species_count['count']/189 * 100)
species_count
```

[101]:

	species	count	percentage
6	Great crested newt	21	11.0
3	Fire-bellied toad	58	31.0
5	Common newt	58	31.0
4	Tree frog	71	38.0
0	Green frogs	108	57.0
2	Common toad	124	66.0
1	Brown frogs	148	78.0

[102]:
```
# Create a barplot of the counts per species
sns.barplot(data=species_count, x='species', y='count')
plt.xticks(rotation=90)
plt.title('The total number of occurences per species')
```

[102]:
```
Text(0.5, 1.0, 'The total number of occurences per species')
```



This figure shows the total number of occurences per species. Brown frogs were present in 148 of the sites observed, followed by common toads and green frogs. The Great Crested Newt was present at only 21 of the 189 sites observed.

This a form of an unbalanced classification problem because some of the classes are underrepresented. This rules out accuracy as metric to assess the performance of the classifier.

The data was collected for two road development projects. Let's compare the distribution of labels between the two road projects.

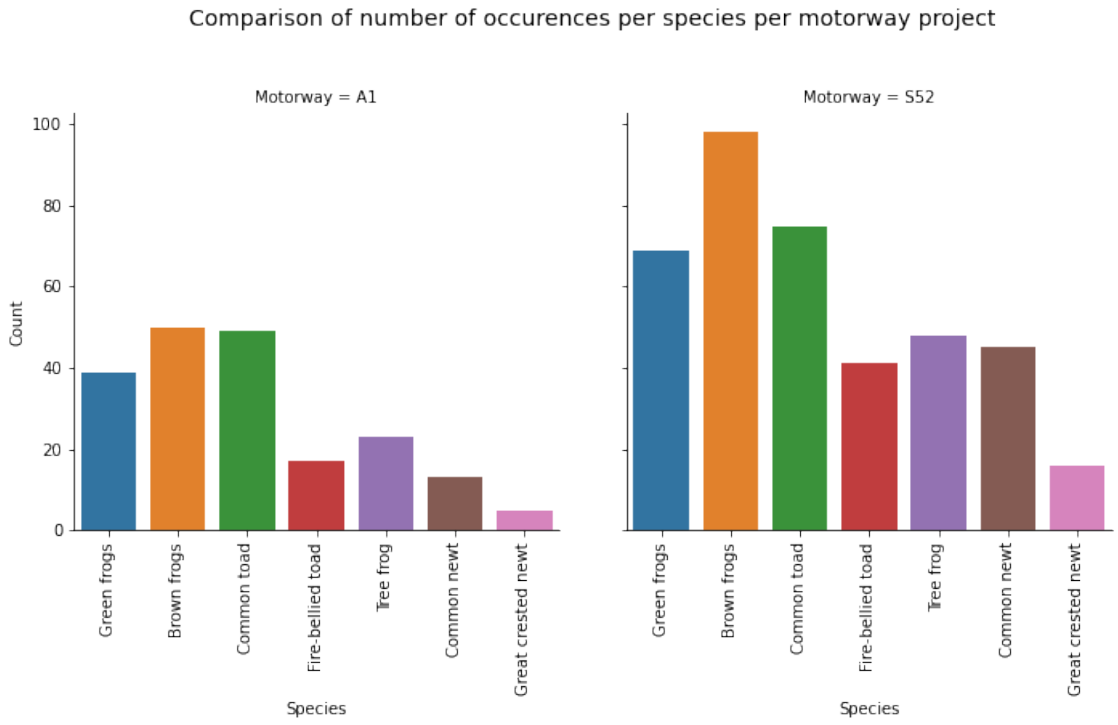
[103]:
```
# Group the values according to the motorway to determine if their are any␣
↪differences between the two motorways
species_count_motorway = amphibians.groupby('Motorway')[species].apply(lambda x␣
↪: x.astype(int).sum())
species_count_motorway.reset_index(inplace=True)
species_count_motorway
```

[103]:

	Motorway	Green frogs	Brown frogs	Common toad	Fire-bellied toad	Tree frog	Common newt	Great crested newt
0	A1	39	50	49	17	23	13	5
1	S52	69	98	75	41	48	45	16

[104]:
```
# Create barplots for each road/motorway and plot side by side for comparison
g = sns.catplot(data=species_count_motorway, kind='bar', col='Motorway')
g.fig.suptitle('Comparison of number of occurences per species per motorway␣
 ↪project', size=14)
g.fig.subplots_adjust(top=0.8)
g.set_xticklabels(rotation=90)
g.set_axis_labels("Species", "Count")
```

[104]: <seaborn.axisgrid.FacetGrid at 0x7ff378f51c40>



The figure above shows that more sample were collected for the S52 motorway project. The A1 motorway samples have an almost even number of observations of brown frogs and common toads whereas samples from the S52 motorway seem to indicate that brown frogs were present more frequently than common toads. The relatively rarer species are Fire-bellied toads, Tree frogs, Common Newst and Great crested newts for both motorways.

[105]:
```
# Are there any observations with no species present?
frogs_per_observation = amphibians[species].sum(axis=1).
 ↪to_frame('species_present')
frogs_per_observation[frogs_per_observation['species_present'] == 0]
```

[105]:

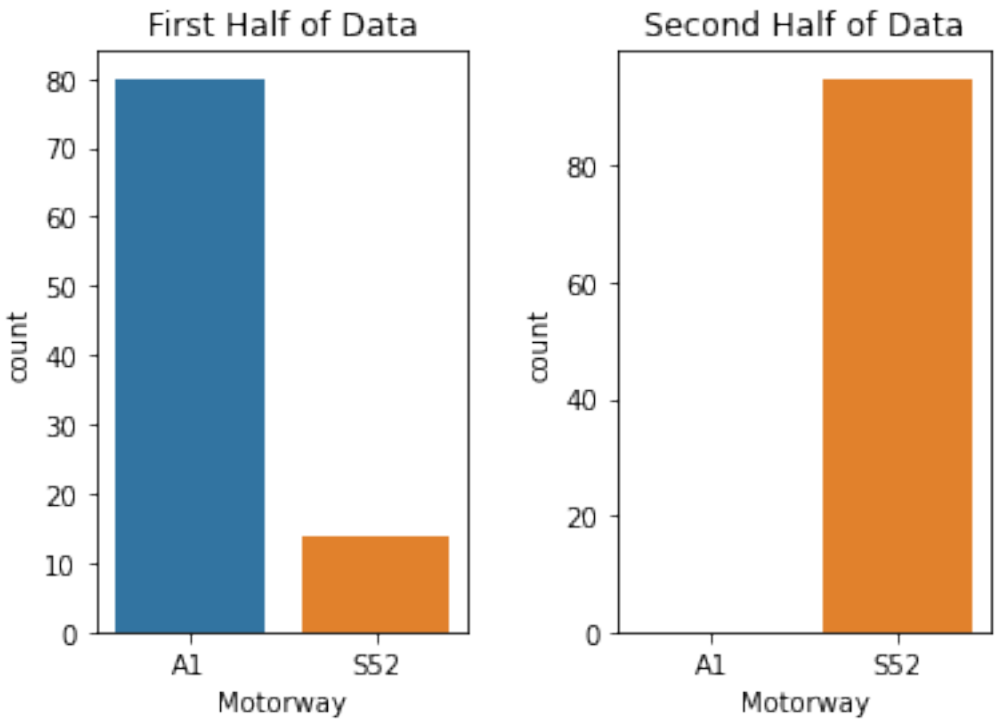
ID	species_present
1	0
6	0
125	0
129	0
132	0
179	0

There are 6 observations where there were no observations of any species of amphibians.

[106]:
```
# Check whether the data is ordered by splitting the data in half and comparing␣
 ↪the motorway projects from which the sample were collected
first_half = amphibians.iloc[0:int(189/2)]
second_half = amphibians.iloc[int(189/2):]
```

[107]:
```
# Create barplot to compare the distribution of samples from each motorway␣
 ↪project in the first half and the second half of the dataset
fig, ax = plt.subplots(1, 2)
fig.subplots_adjust(hspace=0.4, wspace=0.4)
sns.countplot(ax=ax[0],data=first_half, x='Motorway', order=['A1', 'S52'])
sns.countplot(ax=ax[1],data=second_half, x='Motorway', order=['A1', 'S52'])
ax[0].set_title('First Half of Data')
ax[1].set_title('Second Half of Data')
```

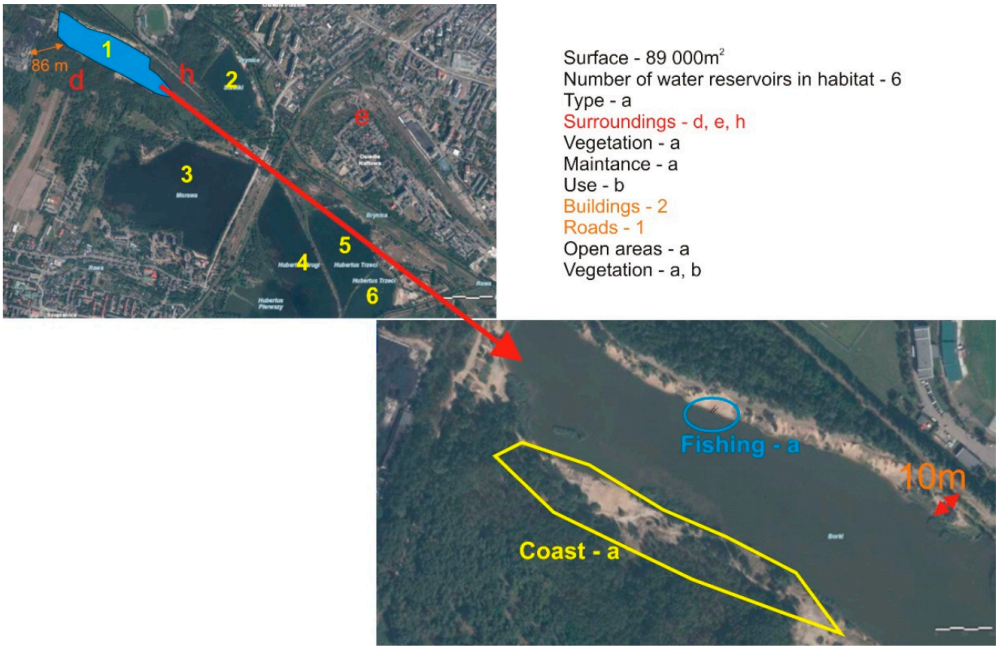
[107]:
```
Text(0.5, 1.0, 'Second Half of Data')
```



The observations are organized according to the location from where the data was collected (Motorway A1 or S52). We will need to shuffle the data before splitting it so that the model works well for any motorway.

6.0.1 Exploration of the numerical features

In this section, we will explore the two numerical features in the dataset. In order to better explore and understand the numerical features as well as the categorical features, we need to understand how the feature data was collected. The figure below shows how the researchers that compiled the dataset assessed a sample to determine the values for the various features.



The figure is shows the assessment of a water reservoir (labelled 1). This is the Borki water reservoir is located near Sosnowiec. It was unclear wether the surface area (SR) measurement is the total some of all water reservoirs in the habitat. Using the 'measure distance' feature, it was confirmed that this is not the total area of the water reservoirs in the habitat but the surface area of the specific water reservoir.

The habitat containing the Borki reservoir contains 5 other water reservoirs. Therefore the value for number of reservoirs in habitate (NR) is 6. It is not clear how a habitat is defined. The reservoir was classified as type a (TR) which means that the reservoir is a reservoir with natural features. The surroundings of the reservoir are classified into 3 categories (dominant (SR1), 2nd most dominant (SR2), 3rd most dominant (SR3) Borki's most dominant surroundings can be classified as being parks and green areas (level d). The 2nd most dominant surroundings are dense building development (level e) and the 3rd most are roads and streets (h).

The presence of vegetation in this reservoir (VR) is classfied has no vegetation (level a). The maintenance status (MR) of the reservoir is classified as clean (level a) and the use of the reservoir (UR) as recreational and scenic (level b). The level of fishing present (FR) is classified as lack of or occasional fishing (level a). The minimum distance to buildings (BR) is between 50m and 100m (level b) and the minimum distance to roads (RR) is less than 50m (level a). The Borki shore type (CR) is not shown in the figure but I would say it can be classified as natural (level a).

SR - Surface area of the water reservoir (m2)

This is the first numerical feature in the dataset.

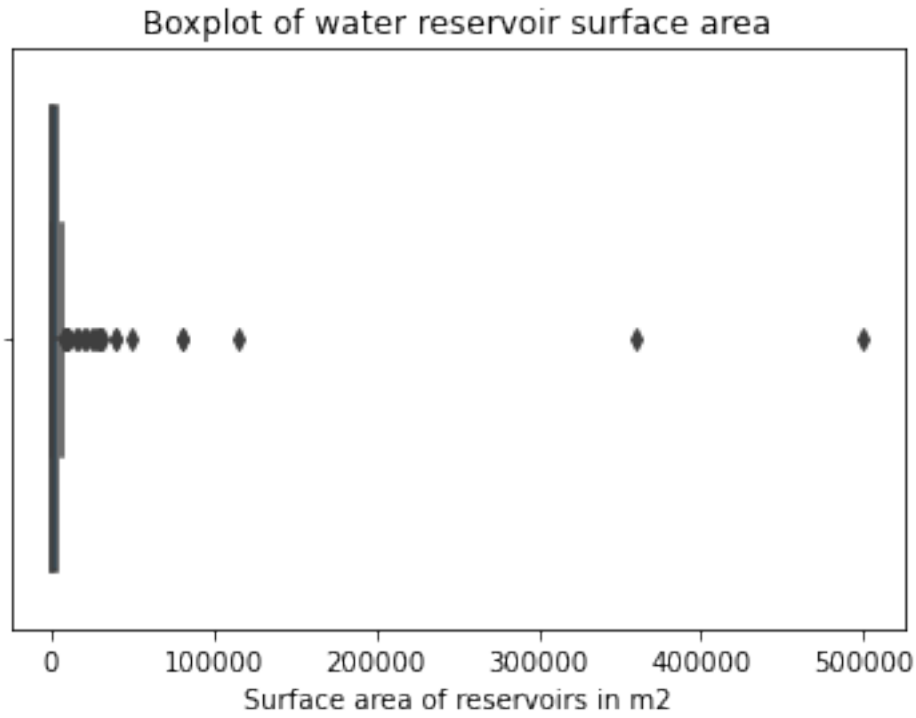
[108]:
```
# Get a summary for the numerical feature
amphibians['SR'].describe()
```

[108]:
```
count       189.000000
mean       9633.227513
std       46256.078309
min          30.000000
25%         300.000000
50%         700.000000
75%        3300.000000
max      500000.000000
Name: SR, dtype: float64
```

The SR feature has a large range from 30 to 500000 m2. The large mean compared to the median suggests that there might be outlier in the datasets.

[109]:
```
# Create a boxplot to visualize the distribution of the feature
sns.boxplot(data=amphibians, x='SR')
plt.title('Boxplot of water reservoir surface area')
plt.xlabel('Surface area of reservoirs in m2')
```

[109]:
```
Text(0.5, 0, 'Surface area of reservoirs in m2')
```



The boxplot for water reservoir surface area shows a few very extreme values in the approximate range of 100000 to 500000 square metres or 0.1 to 0.5 square kilometres. This may seem like very extreme values compared to the rest of the dataset but to perhaps put it in persepctive, some lakes in Poland range from between 9.8 to 113.8 square kilometres.

[110]:
```
# Determine the number of outliers for reservoir surface area
Q3 = amphibians['SR'].quantile(q=0.75)
Q1 = amphibians['SR'].quantile(q=0.25)
sr_outlier = len(amphibians[amphibians['SR'] > (1.5 * (Q3 - Q1))].
 ↪sort_values(by='SR', ascending=False))
```

[111]:
```
# Determine what percentage of the observations, the outlier present
sr_outlier/len(amphibians) * 100
```

[111]:
```
18.51851851851852
```

There are 35 samples that have a surface area larger than 1.5 times the IQR. This is 18.5% of the dataset. We would lose too much data by removing these values.

As a result of the large range of values that this feature can take, it will be important to standardize this feature if algorithms like K-Nearest Neighbours is selected.

NR - Number of water reservoirs in habitat

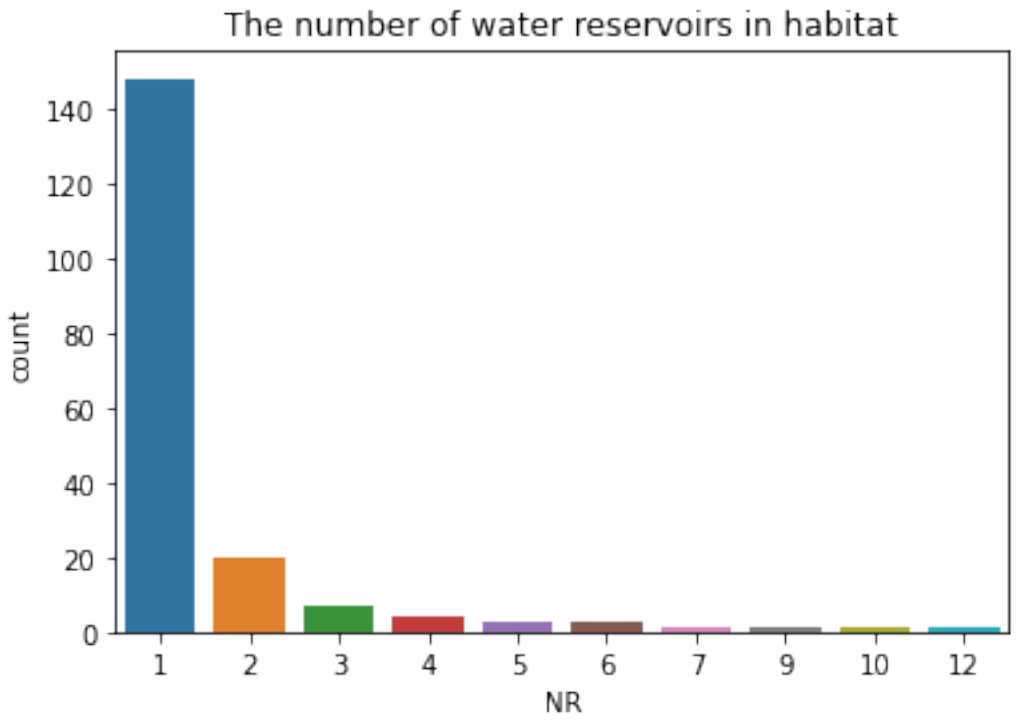
This is the second numerical feature in the dataset. This feature is not a continuous numerical feature but a discrete numerical feature.

[112]:
```
# Determine the number of unique value for this feature.
amphibians['NR'].value_counts().sort_index()
```

[112]:
```
1     148
2      20
3       7
4       4
5       3
6       3
7       1
9       1
10      1
12      1
Name: NR, dtype: int64
```

[113]:
```
# Create a barplot of the feature
sns.countplot(data=amphibians, x='NR')
plt.title("The number of water reservoirs in habitat")
```

[113]:
```
Text(0.5, 1.0, 'The number of water reservoirs in habitat')
```



[114]:
```
# Determine percentage of observations that have 2 or more water reservoirs
len(amphibians[amphibians['NR'] > 1])/len(amphibians) * 100
```

[114]: 21.693121693121693

The count plot above shows that average value for the number of reservoirs in a habitate is 1.0. As mentioned earlier, it is unclear from the dataset description how the perimeter of a habitat was defined. 21 % of the observations have 2 or more reservoirs in the habitat. The dataset description does mention that it is believed that the greater the number of reservoirs in habitat, the more likely it is that some of them will be suitable for amphibian breeding.

6.0.2 Exploration of the categorical features

In this section, we will explore the 12 categorical features of in the dataset. We have excluded the Motorway (MR) variable as it should not be as input into a model.

TR - Type of water reservoirs

According to the dataset description, there are 10 unique values for this variable

5) TR -> Type of water reservoirs:
a. reservoirs with natural features that are natural or anthropogenic water reservoirs (e.g., subsidence post-exploited water reservoirs), not subjected to naturalization
b. recently formed reservoirs, not subjected to naturalization
c. settling ponds
d. water reservoirs located near houses
e. technological water reservoirs
f. water reservoirs in allotment gardens
g. trenches
h. wet meadows, flood plains, marshes
i. river valleys
j. streams and very small watercourses

[115]:
```
# Determine the number of unique values for this feature
amphibians['TR'].value_counts().sort_index()
```

[115]:
```
1     116
2       4
5      12
7       1
11      4
12     23
14     10
15     19
Name: TR, dtype: int64
```

SUR1 - Surroundings 1

According to the dataset description, there 9 unique values for this variable

7) SUR1 - Surroundings 1â€"the dominant types of land cover surrounding the water reservoir
8) SUR2 - Surroundings 2â€"the second most dominant types of land cover surrounding the water reservoir
9) SUR3 - Surroundings 3â€"the third most dominant types of land cover surrounding the water reservoir
Comment: The â€œsurroundingsâ€ feature was designated in three stages. First, the dominant surroundings were selected. Then, two secondary types were chosen.
a. forest areas (with meadows) and densely wooded areas
b. areas of wasteland and meadows
c. allotment gardens
d. parks and green areas
e. dense building development, industrial areas
f. dispersed habitation, orchards, gardens
g. river valleys
h. roads, streets
i. agricultural land
The most valuable surroundings of water reservoirs for amphibians are areas with the least anthropopressure and proper moisture.

[116]:
```
# Determine the number of unique values for this feature
amphibians['SUR1'].value_counts().sort_index()
```

[116]:
```
1     43
2     70
4      1
6     19
7     20
9      5
10    30
14     1
Name: SUR1, dtype: int64
```

There are levels of this category that are missing.

SUR2 - Surroundings 2

According to the dataset description, there 9 unique values for this variable.

[117]:
```
# Determine the number of unique values for this feature
amphibians['SUR2'].value_counts().sort_index()
```

[117]:
```
1     36
2     41
```

```
6     39
7     18
9     10
10    44
11     1
Name: SUR2, dtype: int64
```

There are levels of this category that are missing.

SUR3 - Surroundings 3¶

According to the dataset description, there 9 unique values for this variable

[118]:
```
# Determine the number of unique values for this feature
amphibians['SUR3'].value_counts().sort_index()
```

[118]:
```
1     29
2     29
5      2
6     55
7     18
9     10
10    45
11     1
Name: SUR3, dtype: int64
```

CR - Type of shore

According to the dataset description, there are 2 unique values for this variable

[119]:
```
# Determine the number of unique values for this feature
amphibians['CR'].value_counts().sort_index()
```

[119]:
```
1    186
2      3
Name: CR, dtype: int64
```

There are only 3 observations where the shore is concrete. Could it be strong feature for there won't be any amphibians? Or not informative at all?

VR - Intensity of vegetation development

According to the dataset description, there should be 5 unique values for this variable

6) VR - Presence of vegetation within the reservoirs:
a. no vegetation
b. narrow patches at the edges
c. areas heavily overgrown
d. lush vegetation within the reservoir with some part devoid of vegetation
e. reservoirs completely overgrown with a disappearing water table
Comment: The vegetation in the reservoir favors amphibians, facilitates breeding, and allows the larvae to feed and give shelter. However, excess vegetation can lead to the overgrowth of the pond and water shortages.

[120]:
```
# Determine the number of unique values for this feature
amphibians['VR'].value_counts().sort_index()
```

[120]:
```
0    30
1    55
2    35
3    41
4    28
Name: VR, dtype: int64
```

All the levels of the variable appear in the dataset

MR - Maintenance status of the reservoir

According to the dataset description, there should be 3 unique values for this variable

> 15) MR - Maintenance status of the reservoir:
> a. Clean
> b. slightly littered
> c. reservoirs heavily or very heavily littered
> Comment: Trash causes devastation of the reservoir ecosystem. Backfilling and leveling of water reservoirs with ground and debris should also be considered.

[121]:
```
# Determine the number of unique values for this feature
amphibians['MR'].value_counts().sort_index()
```

[121]:
```
0    184
1      1
2      4
Name: MR, dtype: int64
```

My gut says this feature won't be informative.

FR - The presence of fishing

According to the dataset description, there should be 3 unique values for this variable

> 11) FR - The presence of fishing:
> a. lack of or occasional fishing
> b. intense fishing
> c. breeding reservoirs
> Comment: The presence of a large amount of fishing, in particular predatory and intense fishing, is not conducive to the presence of amphibians.

[122]:
```
# Determine the number of unique values for this feature
amphibians['FR'].value_counts().sort_index()
```

[122]:
```
0    125
1     16
2     15
3     18
4     15
Name: FR, dtype: int64
```

BR - Building development variable

There 6 levels of this variable in the dataset description.

14) BR - Building development - Minimum distance to buildings:
a. <50 m
b. 50â€“100 m
c. 100â€“200 m
d. 200â€“500 m
e. 500â€“1000 m
f. >1000 m
Comment: The more distant the buildings, the more favorable the conditions for the occurrence of amphibians.

[123]:
```
# Determine the number of unique values for this feature
amphibians['BR'].value_counts().sort_index()
```

[123]:
```
0     39
1     62
2     29
5     46
9      7
10     6
Name: BR, dtype: int64
```

The values of the two ordinal variables does not correspond to the descriptions of the variables but there are also 6 unique values.

RR - Minimum distance from the water reservoir to roads

According to the dataset description, there should be 6 categories or levels for this variable.

13) RR Minimum distance from the water reservoir to roads:
a. <50 m
b. 50â€“100 m
c. 100â€“200 m
d. 200â€“500 m
e. 500â€“1000 m
f. >1000 m
Comment: The greater the distance between the reservoir and the road, the more safety for amphibians.

[124]:
```
# Determine the number of unique values for this feature
amphibians['RR'].value_counts().sort_index()
```

[124]:
```
0     47
1     50
2     39
5     42
9      7
10     4
Name: RR, dtype: int64
```

The values of the two ordinal variables does not correspond to the descriptions of the variables but there are also 6 unique values.

OR - Access from water table to land habitats

According to the dataset description, there should be 4 unique values for this variable.

12) OR - Percentage access from the edges of the reservoir to undeveloped areas (the proposed percentage ranges are a numerical reflection of the phrases: lack of access, low access, medium access, large access to free space):
a. 0â€“25%â€lack of access or poor access
b. 25â€“50%â€low access
c. 50â€“75%â€medium access,
d. 75â€“100%â€large access to terrestrial habitats of the shoreline is in contact with the terrestrial habitat of amphibians.

[125]:
```
# Determine the number of unique values for this feature
amphibians['OR'].value_counts().sort_index()
```

[125]:
```
25       7
50      16
75      22
80       1
99       2
100    141
Name: OR, dtype: int64
```

The categories of the OR feature is different from the values described by the dataset description. Does 25 mean 0 to 25 or above 25 and there were no values below 25? It is most likely 0 to 25 because this feature is a percentage and therefore the 100 category cannot mean, greater than a 100. There are also two odd values: 80 and 90.

[126]:
```
# View samples that contain odd values for OR
amphibians[amphibians['OR'].isin([80, 99])]
```

[126]:

ID	Motorway	SR	NR	TR	VR	SUR1	SUR2	SUR3	UR	FR	…	BR	MR	CR	\
											…				
5	A1	600	2	1	4	10	2	6	0	0	…	5	0	1	
53	A1	300	1	1	4	1	2	6	0	0	…	2	0	1	
181	S52	4000	2	1	1	4	6	9	3	2	…	1	0	1	

ID	Green frogs	Brown frogs	Common toad	Fire-bellied toad	Tree frog	\
5	0	1	1	1	0	
53	0	1	0	0	0	
181	1	1	1	0	1	

ID	Common newt	Great crested newt
5	1	1
53	0	0
181	1	1

[3 rows x 22 columns]

There is no quick visible indication of why these samples may have odd values for OR. We will assume that these variables were incorrectly encoded and actually belong to the 75 to 100% category.

[127]:
```
# Replace samples that have OR value equal to 80 or 99
amphibians['OR'].replace([80, 99], [100, 100], inplace=True)
```

[128]:
```
# Determine the number of unique values for this feature
amphibians['OR'].value_counts().sort_index()
```

[128]:
```
25       7
50      16
75      22
100    144
Name: OR, dtype: int64
```

The dataset description lists the OR variable as a nominal categorical variable but it is a percentage value which does represent a rank ordering between the values and could be considered to be a ordinal categorical variable.

6.1 Exploratory Data Analysis Summary

The dataset is very small with only a 189 samples. There are no missing values but there are issues with how the categorical features have been encoded. The descriptions of the categories of many of the features is inconsistent with the description of the dataset so it is unclear how these features should be cleaned or engineered. It is possible that the categorical features in the dataset provided have already been encoded.

No outliers will be removed from the dataset as it would reduce the size of the available data for model development too much.

The labels of the target are unbalanced.

Some of the amphibian species are rare. Only 6 of the 189 sites observed had no presence of amphibians. As the dataset is unbalanced, accuracy cannot be used to evaluate the models.

[129]:
```
# Drop the Motorway column from the dataset as this is not used
amphibians = amphibians.drop(columns='Motorway')
```

[130]:
```
# Split the data into features and target
amphibians_feat = amphibians.drop(species, axis=1)
```

6.2 Model Development

6.2.1 Fine tuning of the problem approach

A random forest model is a logically choice for this problem. In the early discussion regarding suitable classification algorithms for this problem, it was highlighted that a random forest is suitable for a mix of categorical and numerical features. A random forest is insensitive to the scaling of numerical features which is good because preprocessing can introduce an extra source of variance. An added benefit is that the random forest implementation in scikit-learn natively support multi-label classification.

Following the same reasoning as above, the Gradient Boosted Trees and AdaBoost algorithm will also be good choices. The discussion on multi-label problem approaches, it was noted that the implementations of these two algorithms in scikit-learn do not natively support mulit-label classification. To use these algorithms, the problem will be effectively divided into 7 binary classifiers, one

for each label, and together their outputs provide the multi-label output required. We will make use MultiOutputClassifier from scikit-learn to achieve this.

The ROC AUC score will be used to evaluate the performance of the classifiers. The data will be randomly shuffled and split into a training and testing set. As the dataset is so small, we will make sure to evaluate the generalization performance of the models using cross validation and we will find the best model parameters using grid search.

It also seems that the categorical features have already undergone some form of encoding. As the categorical features are already numerical values and can be directly fed to a machine learning algorithm, we will assume they have been encoded and will not perform any further encoding as part of preprocessing.

[41]:
```
# Import required packages
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.multioutput import MultiOutputClassifier
from sklearn.metrics import roc_auc_score
```

The original dataset is split into a train and test set.

[42]:
```
# Shuffle and split the data into train, test and validations sets
X_train, X_test, y_train, y_test = train_test_split(amphibians_feat,␣
 ↪amphibians[species], test_size=0.20, random_state=42, shuffle=True)
```

6.2.2 Random Forest (RF) Classifier

[43]:
```
# Initialize the random forest classifier and set a random state for␣
 ↪reproducibility
rf = RandomForestClassifier(random_state=42)
```

RF Hyperparamerter Tuning

A grid of parameters needs to be created to use with grid search to find the optimal tuning parameters for the model. The most important parameters of a random forest are `n_estimators`, `max_features` and `max_depth`. `n_estimators` refers to the number of trees and more trees are better. Averaging more trees will lead to a more robust ensemble by reducing overfitting. `max_features` determines how random each tree is. If the value `max_features` is 'sqrt' then `max_features=sqrt(n_features)`.

[44]:
```
# Define the hyperparamter grid
param_grid = {'n_estimators': [5, 10, 15, 20, 30, 40],
              'max_features': ['sqrt', 'log2'],
              'max_depth' : [4,5,6,7,8,9],
              'criterion' :['gini', 'entropy']}
```

[45]:
```
# Initialize and fit GridSearchCV
gridsearch_rf = GridSearchCV(rf, param_grid, cv=5, scoring='roc_auc',␣
↪return_train_score=True)
gridsearch_rf.fit(X_train, y_train)
print("Best parameters for a random forest model:")
gridsearch_rf.best_params_
```

```
Best parameters for a random forest model:
```

[45]:
```
{'criterion': 'gini',
 'max_depth': 8,
 'max_features': 'sqrt',
 'n_estimators': 20}
```

[46]:
```
# View the best score achieved
print("ROC AUC score of the best estimator")
round(gridsearch_rf.best_score_,2)
```

```
ROC AUC score of the best estimator
```

[46]:
```
0.68
```

The grid search process determined that a random forest with 20 trees with max depth of 8 achieved the best roc auc score. The ROC AUC score is 0.68. Remember that a purely random classifier would achive a roc auc score of 0.5. The higher the ROC AUC, the better the classifier is at predicting 1s and 1s and 0s as 0s.

A great benefit of a Random Forest is that determining feature importances is inherent to the model.

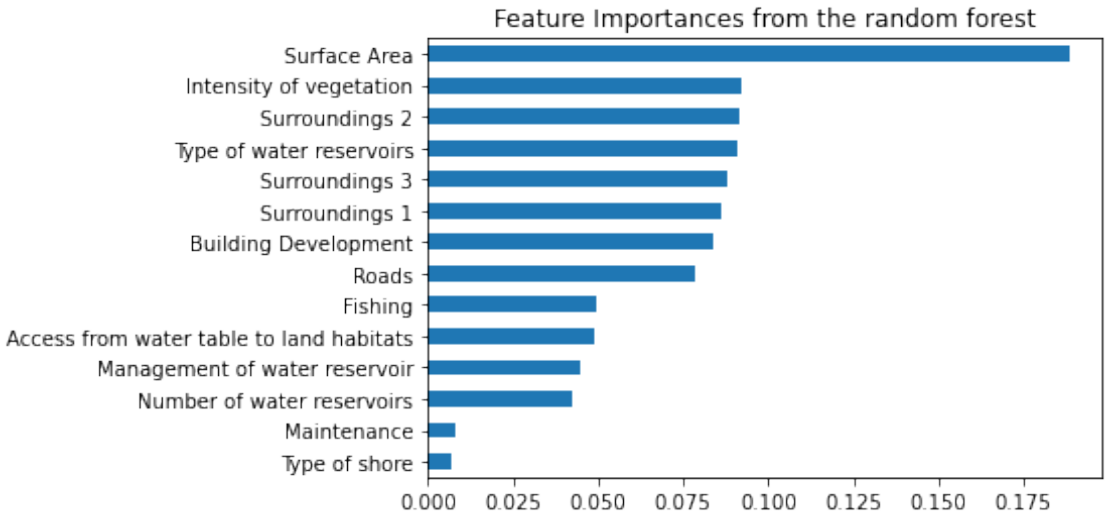
[47]:
```
importances_rf = pd.Series(gridsearch_rf.best_estimator_.feature_importances_,␣
↪amphibians_feat.columns)
importances_rf = importances_rf.sort_values(ascending=False)
full_feature_names = ['Surface Area', 'Intensity of vegetation', 'Surroundings␣
↪2', 'Type of water reservoirs', 'Surroundings 3',
                      'Surroundings 1', 'Building Development', 'Roads',␣
↪'Fishing',
                      'Access from water table to land habitats', 'Management␣
↪of water reservoir',
                      'Number of water reservoirs', 'Maintenance', 'Type of␣
↪shore']
importances_rf.index = full_feature_names
importances_rf
```

[47]:
```
Surface Area                          0.188689
Intensity of vegetation               0.092102
Surroundings 2                        0.091735
Type of water reservoirs              0.090907
```

```
Surroundings 3                              0.088142
Surroundings 1                              0.086380
Building Development                        0.083713
Roads                                       0.078468
Fishing                                     0.049482
Access from water table to land habitats    0.048591
Management of water reservoir               0.044480
Number of water reservoirs                  0.042203
Maintenance                                 0.008044
Type of shore                               0.007063
dtype: float64
```

[48]:
```
# Plot the impurity based feature importances
importances_rf.sort_values(ascending=True).plot(kind='barh')
plt.title('Feature Importances from the random forest')
```

[48]:
```
Text(0.5, 1.0, 'Feature Importances from the random forest')
```



According to the random forest model, surface area of the water reservoir (SR) is most important. After the SR features, there appear to three groups of mean descrease in impurity. The first group includes the intensity of the vegetation, the primary, secondary and tertiary surroundings, building development and roads. The maintenance of the water reservoir and the type of shore are the least informative features.

RF Model Evaluation

The best estimator or model will be evaluated using ROC AUC on the testing set. The ROC AUC will be determined for each class as well as macro average for over all labels will be determined.

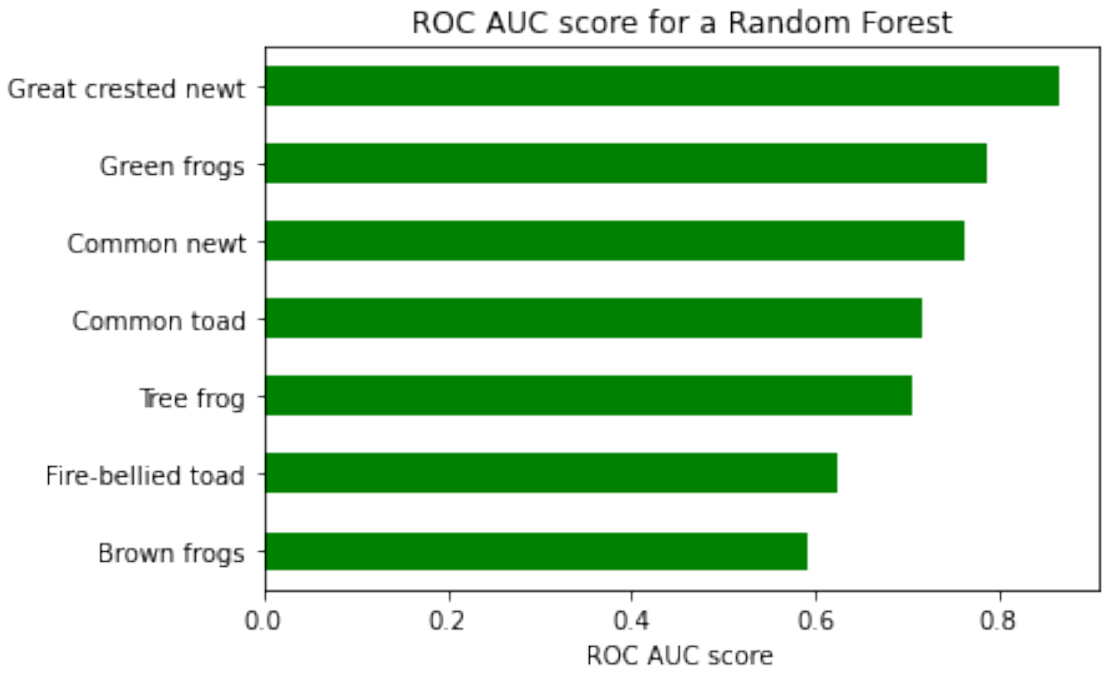
[49]:
```
# Use the best estimator to predict labels for the test set
y_pred_rf = gridsearch_rf.predict_proba(X_test)
y_pred_rf = np.transpose([pred[:, 1] for pred in y_pred_rf])
```

[50]:
```
# Calculate the ROC AUC scores for each of the labels and store in a Pandas␣
↪series
auc_scores_classes_rf = pd.Series(roc_auc_score(y_test, y_pred_rf,␣
↪average=None), amphibians[species].columns)
auc_scores_classes_rf.sort_values()
```

[50]:
```
Brown frogs           0.591667
Fire-bellied toad     0.624521
Tree frog             0.704545
Common toad           0.715942
Common newt           0.763636
Green frogs           0.785714
Great crested newt    0.864865
dtype: float64
```

[51]:
```
# Plot the ROC AUC scores for each target label
auc_scores_classes_rf.sort_values(ascending=True).plot(kind='barh' ,␣
↪color='green')
plt.title('ROC AUC score for a Random Forest')
plt.xlabel('ROC AUC score')
```

[51]:
```
Text(0.5, 0, 'ROC AUC score')
```



[52]:
```
# Calculate the unweighted mean AUC which does not take label imbalance into␣
↪account
auc_scores_macro_rf = roc_auc_score(y_test, y_pred_rf, average='macro')
round(auc_scores_macro_rf,2)
```

[52]: 0.72

[53]:
```
# Calculate the AUC for each label, find their average, weighted by support␣
↪(the number of true instances for each label)
auc_scores_weighted_rf = roc_auc_score(y_test, y_pred_rf, average='weighted')
round(auc_scores_weighted_rf,2)
```

[53]: 0.69

The RF classifier achieves an unweighted average AUC of 0.72 on the test data which is slightly higher than the 0.68 achieved on the training data. The classifier achieves a weighted average AUC of 0.69. The weights are determined by the number of true instances for each label. The RF classifier achieves the highest AUC score for Great Crested Newts and the lowest score for Brown Frogs. Great Crested newts are the most rare species in the dataset and Brown Frogs were present most frequently. The rest of the AUC scores have no relation to the number of occurences of each species. In simple terms, the RF classifier is better at predicting whether Great Crested Newts will be present in a location compared than brown frogs. Collecting more samples where brown frogs were not present will help to improve this classifier.

6.2.3 Gradient Boosting Tree (GBT) Classifier

A GBT classifier will be trained for each class or label. The outputs from individual GBT classifiers are combined to produce a multi-label output.

[54]:
```
# Initialize the random forest classifier and set a random state for␣
↪reproducibility
gbt = MultiOutputClassifier(GradientBoostingClassifier(random_state=42))
```

GBT Hyperparameter Tuning

[55]:
```
# Define the hyperparameter grid
param_grid_gbt = {'estimator__n_estimators': [50, 60, 70, 80, 90, 100, 110,␣
↪120, 130],
                  'estimator__learning_rate': [0.08, 0.09, 0.1]}
```

[56]:
```
# Initialize and fit GridSearchCV
gridsearch_gbt = GridSearchCV(gbt, param_grid_gbt, cv=5, scoring='roc_auc',␣
↪return_train_score=True)
gridsearch_gbt.fit(X_train, y_train)
print("Best parameters for a GBT model:")
```

```
gridsearch_gbt.best_params_
```

Best parameters for a GBT model:

[56]:
```
{'estimator__learning_rate': 0.1, 'estimator__n_estimators': 60}
```

[57]:
```
# View the best score achieved
print("ROC AUC score of the best estimator")
round(gridsearch_gbt.best_score_,2)
```

```
ROC AUC score of the best estimator
```

[57]:
```
0.63
```

To develop a multi-label classifier using the GBT algorithm, 7 binary classifiers are created and their output is combined. This is makes it complex to extract the overall features importances and we will leave that future work on this problem.

GBT Model Evaluation

The best estimator or model will be evaluated using ROC AUC on the testing set. The ROC AUC will be determined for each class as well as macro average for over all labels will be determined.

[58]:
```
# Use the best estimator to predict labels for the test set
y_pred_gbt = gridsearch_gbt.predict_proba(X_test)
y_pred_gbt = np.transpose([pred[:, 1] for pred in y_pred_gbt])
```

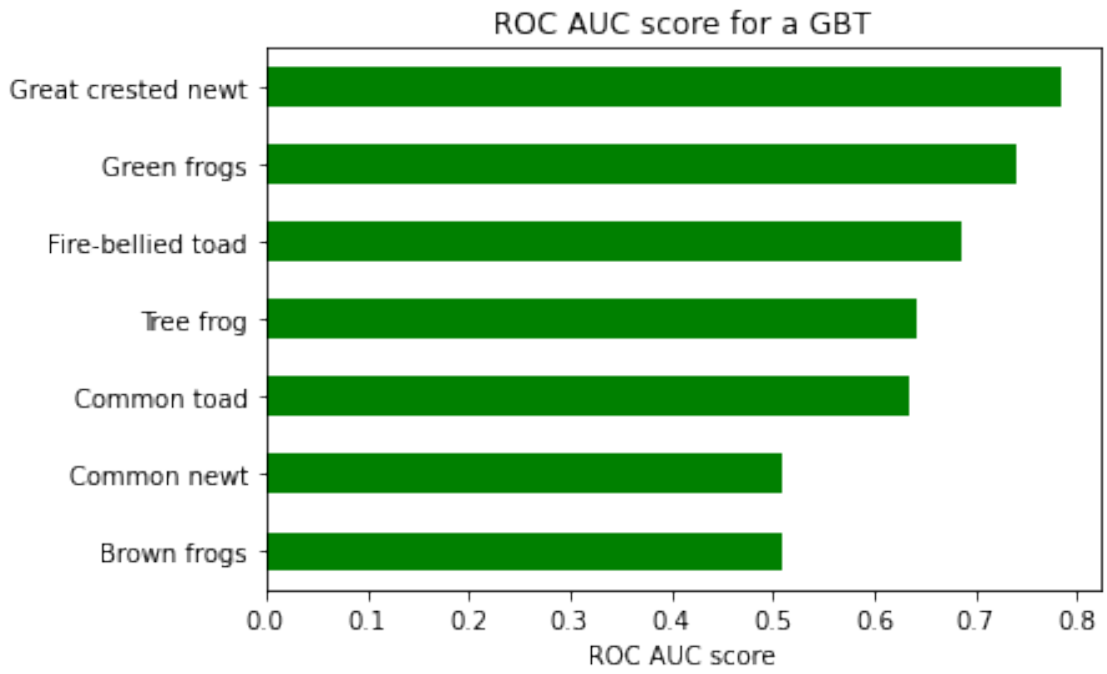
[59]:
```
# Calculate the ROC AUC scores for each of the labels and store in a Pandas␣
↪series

auc_scores_classes_gbt = pd.Series(roc_auc_score(y_test, y_pred_gbt,␣
↪average=None), amphibians[species].columns)
auc_scores_classes_gbt.sort_values()
```

[59]:
```
Brown frogs           0.508333
Common newt           0.509091
Common toad           0.634783
Tree frog             0.642045
Fire-bellied toad     0.685824
Green frogs           0.739496
Great crested newt    0.783784
dtype: float64
```

[60]:
```
# Plot the ROC AUC scores for each target label
auc_scores_classes_gbt.sort_values(ascending=True).plot(kind='barh' ,␣
↪color='green')
plt.title('ROC AUC score for a GBT')
plt.xlabel('ROC AUC score')
```

[60]: Text(0.5, 0, 'ROC AUC score')



[61]:
```
# Calculate the unweighted mean AUC which does not take label imbalance into␣
↪account

auc_scores_macro_gbt = roc_auc_score(y_test, y_pred_gbt, average='macro')
round(auc_scores_macro_gbt,2)
```

[61]: 0.64

[62]:
```
# Calculate the AUC for each label, find their average, weighted by support␣
↪(the number of true instances for each label)

auc_scores_weighted_gbt = roc_auc_score(y_test, y_pred_gbt, average='weighted')
round(auc_scores_weighted_gbt,2)
```

[62]: 0.62

The GBT classifier achieves an unweighted average AUC of 0.64 on the test data which is similar to the AUC achieved in training of 0.63. The classifier achieves a weighted average AUC of 0.62. The weights are determinee by the number of true instances for each label. Similar to RF classifier, the GBT classifier achieves the highest AUC score for Great Crested Newts and the lowest score for Brown Frogs.

6.2.4 AdaBoost Classifier

A AdaBoost classifier will be trained for each class or label. The outputs from individual GBT classifiers are combined to produce a multi-label output.

[63]:
```
# Initialize the random forest classifier and set a random state for␣
↪reproducibility
ada = MultiOutputClassifier(AdaBoostClassifier(random_state=42))
```

AdaBoost Hyperparameter Tuning

[64]:
```
# Define the hyperparameter grid
param_grid_ada = {'estimator__n_estimators': [50, 60, 70, 80, 90, 100, 110,␣
↪120, 130],
                  'estimator__learning_rate': [0.08, 0.09, 0.1]}
```

[65]:
```
# Initialize and fit GridSearchCV
gridsearch_ada = GridSearchCV(ada, param_grid_ada, cv=5, scoring='roc_auc',␣
↪return_train_score=True)
gridsearch_ada.fit(X_train, y_train)
print("Best parameters for a AdaBoost model:")
gridsearch_ada.best_params_
```

```
Best parameters for a AdaBoost model:
```

[65]:
```
{'estimator__learning_rate': 0.09, 'estimator__n_estimators': 50}
```

[66]:
```
# View the best score achieved
print("ROC AUC score of the best estimator")
round(gridsearch_ada.best_score_,2)
```

```
ROC AUC score of the best estimator
```

[66]:
```
0.64
```

As with the GBT, the extraction of the overall features importances will be left for future work on this problem.

AdaBoost Model Evaluation

The best estimator or model will be evaluated using ROC AUC on the testing set. The ROC AUC will be determined for each class as well as macro average for over all labels will be determined.

[67]:
```
# Use the best estimator to predict labels for the test set
y_pred_ada = gridsearch_ada.predict_proba(X_test)
y_pred_ada = np.transpose([pred[:, 1] for pred in y_pred_ada])
```

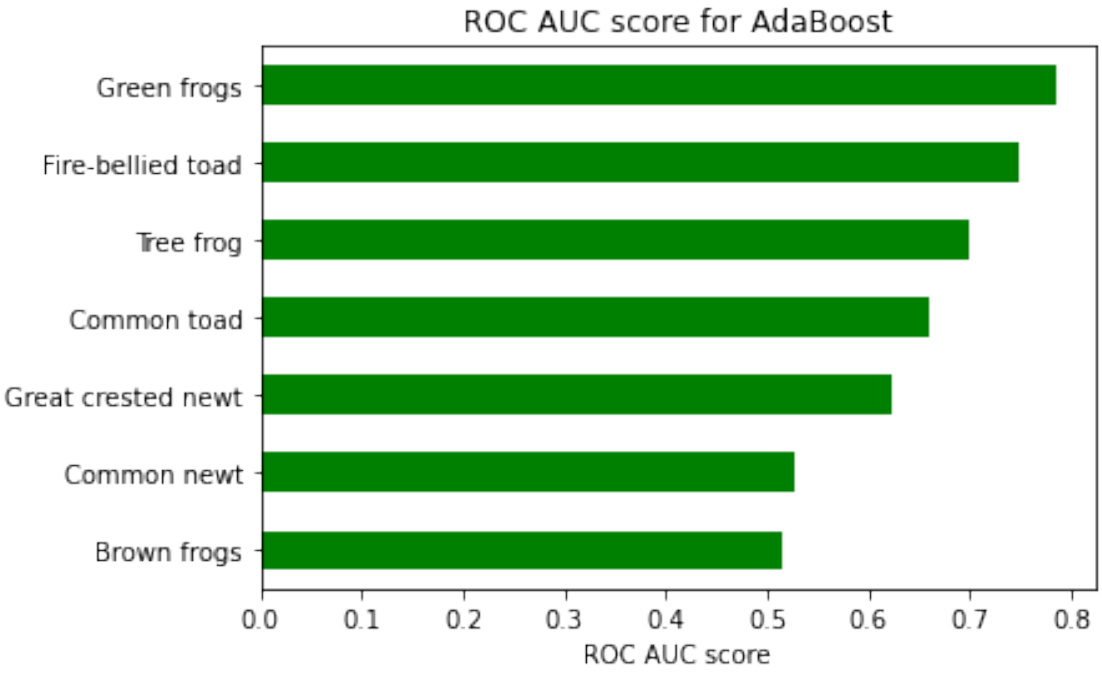
[68]:
```
# Calculate the ROC AUC scores for each of the labels and store in a Pandas␣
↪series
```

```
auc_scores_classes_ada = pd.Series(roc_auc_score(y_test, y_pred_ada,␣
 ↪average=None), amphibians[species].columns)
auc_scores_classes_ada.sort_values()
```

[68]:
```
Brown frogs           0.514583
Common newt           0.527273
Great crested newt    0.621622
Common toad           0.659420
Tree frog             0.698864
Fire-bellied toad     0.749042
Green frogs           0.784314
dtype: float64
```

[69]:
```
# Plot the ROC AUC scores for each target label
auc_scores_classes_ada.sort_values(ascending=True).plot(kind='barh' ,␣
 ↪color='green')
plt.title('ROC AUC score for AdaBoost')
plt.xlabel('ROC AUC score')
```

[69]:
```
Text(0.5, 0, 'ROC AUC score')
```



[70]:
```
# Calculate the unweighted mean AUC which does not take label imbalance into␣
 ↪account

auc_scores_macro_ada = roc_auc_score(y_test, y_pred_ada, average='macro')
```

```
round(auc_scores_macro_ada,2)
```

[70]: 0.65

[71]:
```
# Calculate the AUC for each label, find their average, weighted by support␣
↪(the number of true instances for each label)

auc_scores_weighted_ada = roc_auc_score(y_test, y_pred_ada, average='weighted')
round(auc_scores_weighted_ada,2)
```

[71]: 0.65

The AdaBoost classifier achieves an unweighted average AUC of 0.65 on the test data which is similar to the AUC achieved in training of 0.64. The classifier achieves a weighted average AUC of 0.65. Similar to both the GBT and RF classifiers, the AdaBoost classifier achieves the highest AUC score for Great Crested Newts and the lowest score for Brown Frogs.

7 Results Comparison

[72]:
```
# Combine the auc scores per label for each of the three algorithms into a␣
↪single dataframe
auc_results = pd.concat([auc_scores_classes_rf, auc_scores_classes_gbt,␣
↪auc_scores_classes_ada], axis=1)
auc_results.columns = ['RF', 'GBT', 'Ada']
```

[73]:
```
# Define a custom function to highlight the maximum value in yellow
def highlight_max(s):
    '''
    highlight the maximum in a Series yellow.
    '''
    is_max = s == s.max()
    return ['background-color: yellow' if v else '' for v in is_max]
```

[74]:
```
# Apply the custom function to the results dataframe to highlight the highest␣
↪value per row
auc_results.style.apply(highlight_max, axis=1)
```

[74]: <pandas.io.formats.style.Styler at 0x7ff37075cdc0>

The table above shows a comparison of the AUC scores achieved by each algorithm per class. The random forest classifier achieves the highest AUC score for 6 out of the 7 labels. The AdaBoost algorithm performs better than a random forest for Fire-bellied toads. Fire-bellied toads were the 2nd most rare species along with the common newt in the dataset.

The GBT and AdaBoost algorithms do not do better than a purely random classifier for Brown frogs.

[75]:
```
print('Unweighted average AUC for RF:', round(auc_scores_macro_rf,2))
print('Unweighted average AUC for GBT:', round(auc_scores_macro_gbt,2))
print('Unweighted average AUC for AdaBoost:', round(auc_scores_macro_ada,2))
```

```
Unweighted average AUC for RF: 0.72
Unweighted average AUC for GBT: 0.64
Unweighted average AUC for AdaBoost: 0.65
```

[76]:
```
print('Weighted average AUC for RF:', round(auc_scores_weighted_rf,2))
print('Weighted average AUC for GBT:', round(auc_scores_weighted_gbt,2))
print('Weighted average AUC for AdaBoost:', round(auc_scores_weighted_ada,2))
```

```
Weighted average AUC for RF: 0.69
Weighted average AUC for GBT: 0.62
Weighted average AUC for AdaBoost: 0.65
```

The unweighted average AUC and weighted average AUC scores for each classifier is shown above. According the to macro average results, the GBT and AdaBoost algorithms perfrom sligtly better than a random forest. However, if the AUC score is weighted to take into account the unbalanced target labels, then the random forest algorithm performs a bit better than the other two algorithms.

The reason for the random forest performing better could be related to the scikit-learn implementation of the algorithm which natively supports multi-label classification which treats the target labels simultaneously and can account for correlations among the the labels.

8 Conclusion and Recommendations

The project aimed to investigate the potential to develop a predictive model to predict the presence of 7 species of frogs in a sub-location of a environmental impact assessment study area using features derived from publicly available GIS data and satellite images. The results from the model can then be used to help identify sub-locations with high natural value and to help optimize the allocation of field work resources.

Using the available dataset, an random forest model was developed that achieved a weighted ROC AUC Score in testing of 0.69. This model has potential. The dataset is small and the predictive model would benefit from more training data. The ROC AUC score for all three algorithm were fairly close (RF: 0.69, GBT: 0.62 and AdaBoost: 0.65). These results should be evaluated using a suitable statistical test to determine if the results are truly different.

The values of the categorical variables did not always match the data dictionary and should be followed up on with the data collection team. For example, the categorical variable for the presence of fishing should have 3 levels according to the data dictionary but the variable has 5 levels in the dataset.

It was unclear from the dataset description how the perimeter of a habitat is determined. The rules for the definition of a habitat should be determined and recorded.

It is not clear if it is of necessity to an environmental impact assessment to have counts for each species or if the presence of any type of amphibian is important. If it is not important to distinguish between species, then this problem can be reformulated a binary classification problem with the

goal of predicting whether any species of amphibians is present or not. This will be a simpler problem compared to multi-label classification.

As mentioned already, more data should be collected. In this dataset, only 6 observations had no species of amphibians present and 48 (22%) of the observations had no brown frogs. Any new data collection should focus on collecting data for sub-locations that do not have any amphibians or that have no brown frogs. This will help to build classifiers that are better at either predicting the presence of brown frogs or amphibians in general.

The importances of the features were determined using information extracted from the random forest model. The surface area of the water reservoirs was by far the most important and the type of reservoir and the maintenance status of the reservoir were the least important features. This method determines feature importance using the Gini importance (or mean decrease in impurity). This method has the advantage of speed as the necessary values are calculated during training but the method can have a tendency to select numerical features and high cardinality categorical features (categorical features with many values). Alternative methods like mean decrease in accuracy or a permutation method should be investigated.

9 References

- (1) Environmental Impact Assessment in Kenya
- (2) Endangered Species International - Amphibians
- (3) What is GIS?
- (4) GIS (Geographic Information System)
- (5) ROC Curves and Precision Recall Curves for Imbalanced Classification