Level 1: Local Frog Discovery Tool (EY)

Semi-finalist support document

Table of Contents

Introduction

The aim of this document is to describe the approach, assumptions and methods used for the Level 1: Local Frog Discovery Tool. The goal is t to build a species distribution model (SDM) for one Australian frog species using only variables from the TerraClimate dataset. The frog species identified for this challenge is the eastern dwarf tree frog Litoria fallax. Such models have become increasingly important in conservation efforts globally to better understand and map the habitats of species of interest, particularly threatened or endangered species.

Data Preparation

This first part of the document will focus on the data preparation along with the assumptions methods and techniques that resulted in improvements for the model accuracy and in a better representation of the distribution model for the target species.

Frog Data

The first task, before building the model data using only variables from the TerraClimate dataset, was to extract and prepare the frog occurrences that will be used to represent the frog distribution for the target.

Geo-spatial sampling

For this task, different methods were tested in an attempt to find a good sample representing the locations of the target species with a varied landscape of bushland, plains, rivers, and urban areas. The best results were obtained using the test regions as a starting point to create extended regions around. Using this approach, I sampled both the east coast, which is representative of Litoria fallax habitat, and the western region of Australia, where it is not found. The resulting distribution can be found in Figure 1.

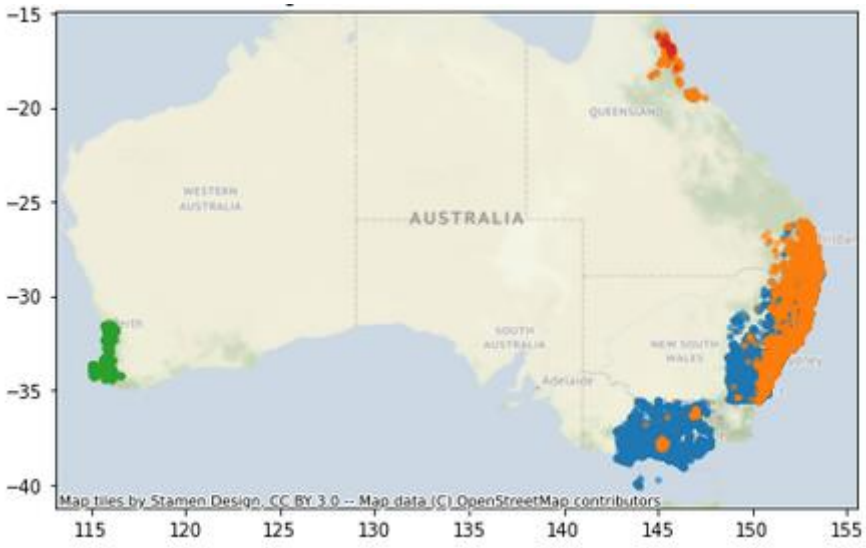

Figure 1 – Frog occurrences map

Year range

Looking at the frog dataset, it is possible to find data from 1900 to 2020 from many different regions around the world. Since this is a large amount of data and does not always provide information about the target frog, it is necessary to carefully filter this data set. After many iterations and tests, I decided to remove all data prior to 2009 as it might be too old and no longer representative. Finally, I removed the latest year (2020) to get a more balanced dataset as there is a large peak of the non-target species is added (Figure 2).

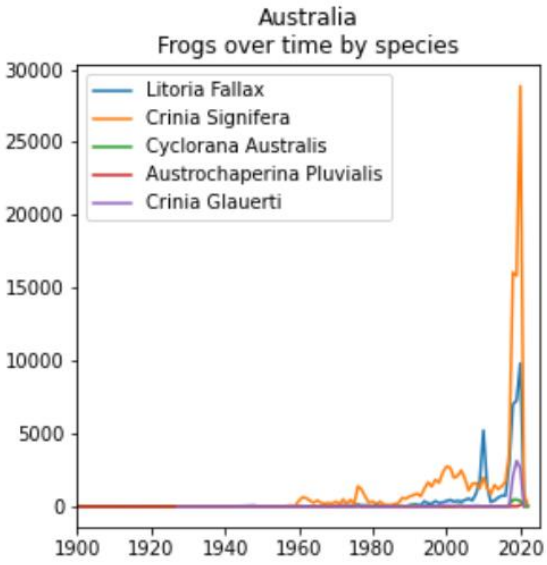

Figure 2 – Frog occurrences by eventDate

Class Balancing

One bias shown in the below visualizations is the class imbalance (Figure 3). To handle this, I down-samples the absence points so that their numbers match that of the target species. Random sampling may cause isolated frog occurrences to be lost, while clusters of occurrences are more likely to persist. Then, I've decided to test different approaches to find a smarter sampling method to improve the model.

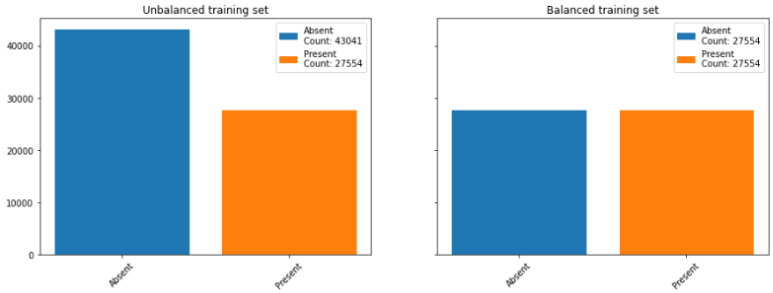

Figure 3 – Class Balancing

First, I calculated the Euclidean distance between each data point. With this information it is possible to create representations of how isolated a point is from the others. For this, several configurations were tested, of which the most efficient was the sum of the distances to the 3 closest points, as a representation of "isolation".

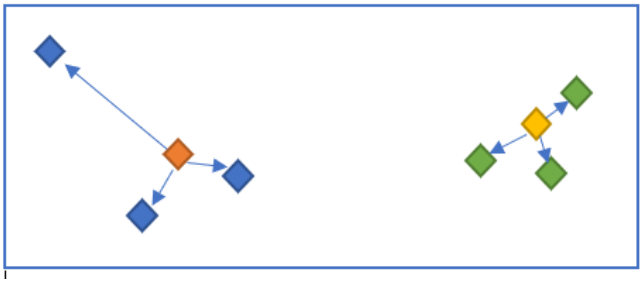

Figure 4 – Estimate isolation of an occurrence

With this information, it is possible to get an idea of the "isolation" of each data point. At this point it is assumed that more isolated occurrences provide more information on the location

where the target species is absent, while data points that are too close may result in redundant information for the model. I decided to keep a percentage of the most isolated and then use random sampling for the remaining data points, until the number of non-target data points is balanced by the number of target data points.

Using this method resulted in a large improvement in model accuracy, as the most isolated data points are included in the model data, while the rest are randomly sampled, which helps maintain representation. statistic of the original sample. The final map it is shown in Figure 5.

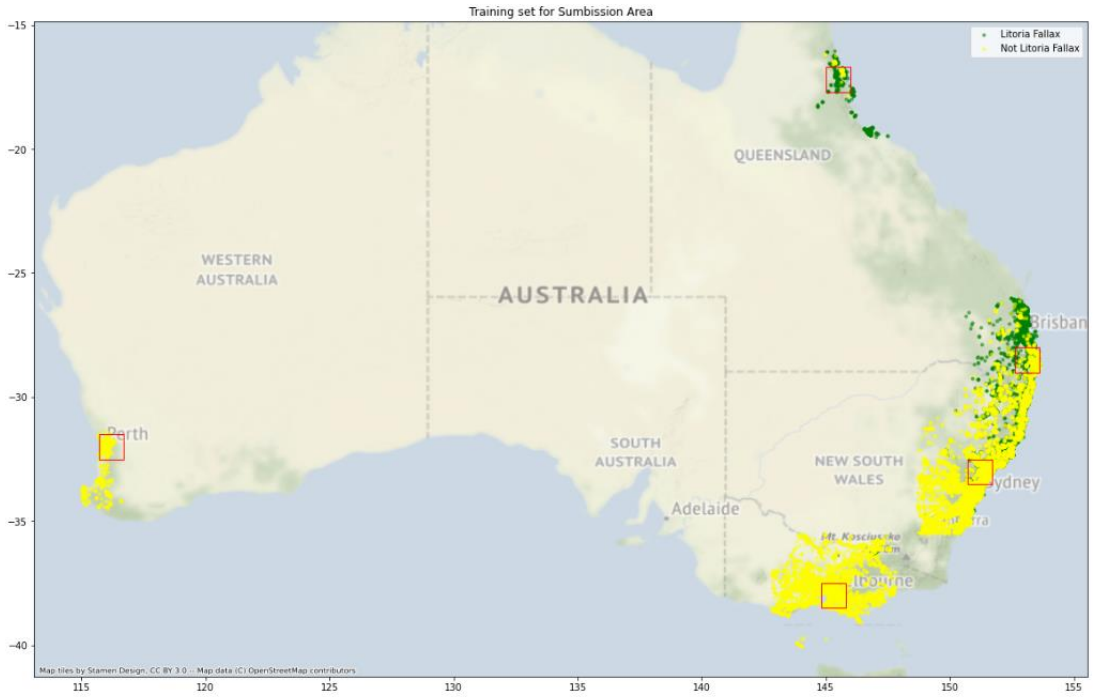

Figure 5 – Frog occurrences map

TerraClimate Data

For the weather to closely match the occurrences on the dataset, I've extracted the TerraClimate predictor variables for every year instead of binning the data as proposed in the benchmark. This way I was able to get a more accurate climate representation of every occurrence, including only data from the same year and a time slice of 4 years for the calculations. So, for a specific occurrence in 2018-05-31 the TerraClimate features will take in consideration only from 2015-01-01 to 2018-12-31. Using 4 years span instead of 5 was also an improvement since allowed to better identify the occurrences with the assumption that frog occurrences within that period are representative of the entire time.

```
[('2006-01-01', '2009-12-31'),
 ('2007-01-01', '2010-12-31'),
 ('2008-01-01', '2011-12-31'),
 ('2009-01-01', '2012-12-31'),
 ('2010-01-01', '2013-12-31'),
 ('2011-01-01', '2014-12-31'),
 ('2012-01-01', '2015-12-31'),
 ('2013-01-01', '2016-12-31'),
 ('2014-01-01', '2017-12-31'),
 ('2015-01-01', '2018-12-31'),
 ('2016-01-01', '2019-12-31')]
```

Besides the metrics already included in the benchmark, some other were considered for the inclusion and added to the data mode such as: (q) Surface runoff which is the flow of water occurring on the ground surface when excess rainwater which could be an indicator of vegetation, (def) climatic water deficit, (ws) 10-m wind speed, etc.

Feature importance

During early stages of the data preparation, many models were trained and an intensive feature importance analysis was done in order to select the TerraClimate features that are present in the final data model. Here is one of the feature importance plots resulting from this analysis:

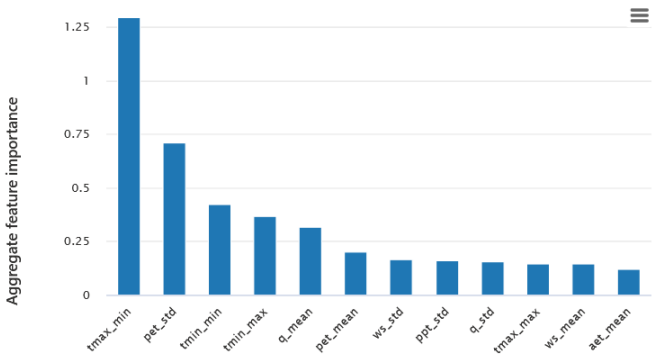

Figure 6 – Feature importance analysis

Model data

Now that I have our predictor variables, I join them onto the response variable of frogs. To do this, we loop through the frog occurrence data, select the weather data for the "eventDate" attribute, and assign each frog occurrence the closest predictor pixel value from each of the predictor variables based on the geo-coordinates.

Feature selection

This the process of selecting the attributes that can make the predicted variable more accurate or eliminating those attributes that are irrelevant and can decrease the model accuracy and quality. In this case we can observe that there are some features that are highly correlated (Figure 7).

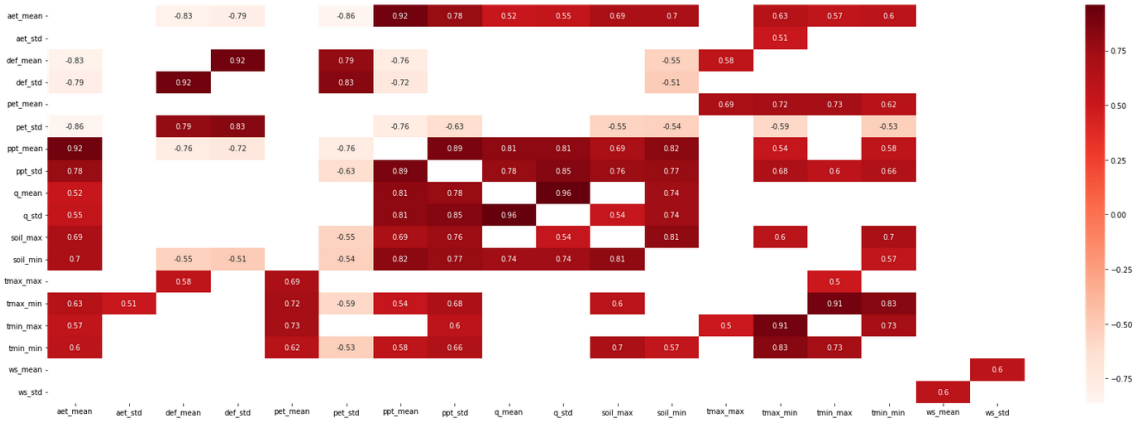

Figure 7 – Pearson correlation coefficients

From this analysis, and as it is mentioned in this study, the central hypothesis is that good feature sets contain features that are highly correlated with the (predictive of) class, yet uncorrelated

with each other. Then, I've selected features to keep a set of columns that are not highly correlated to each other. A sample of the final dataset is shown in Figure 8.

	occurrenceStatus	aet_std	pet_mean	pet_std	soil_min	tmax_max	tmin_max	tmin_min	ws_mean	ws_std
39257	0	25.650665	108.020833	55.896664	1.0	30.800003	15.900002	-0.899998	3.245833	0.641599
22189	1	31.606954	116.895833	36.692551	37.0	30.200005	20.300003	6.200005	3.643750	0.446237
31693	0	28.821891	112.375000	41.286915	8.0	31.099998	18.200005	3.299999	3.156250	0.465433
4294	1	27.771568	110.520833	36.237636	19.0	29.400002	19.400002	5.599998	4.079167	0.578777
44798	0	23.024293	104.750000	53.217126	7.0	26.599998	13.099998	1.200001	4.800000	0.692820

Figure 8 – Sample dataset with feature selection

Split dataset

For training and testing purposes of the model, I have our data broken down into three distinct dataset splits: **training, validation, testing**. The training set is the set of data that is used to train and make the model learn the hidden features/patterns in the data. The validation set is a set of data, separate from the training set, that is used to validate our model performance during training. The test set is a separate set of data used to test the model after completing the training.

```
# Use a utility from sklearn to split and shuffle your dataset.
train_df, test_df = train_test_split(model_data, test_size=0.2)
train_df, val_df = train_test_split(train_df, test_size=0.2)

# Form np arrays of labels and features.
train_labels = np.array(train_df.pop('occurrenceStatus'))
bool_train_labels = train_labels != 0
val_labels = np.array(val_df.pop('occurrenceStatus'))
test_labels = np.array(test_df.pop('occurrenceStatus'))

train_features = np.array(train_df)
val_features = np.array(val_df)
test_features = np.array(test_df)
```

Figure 9 – Split dataset

Scale features

Feature scaling in machine learning is one of the most critical steps during the pre-processing of data before creating a machine learning model. Scaling can make a difference between a weak machine learning model and a better one. After testing many scaling techniques and libraries, the best results were found with sklearn **MaxAbsScaler**.

Model Building

Now that I have the data in a format appropriate for machine learning, I can begin training a model.

Model selection

In this stage many different models were trained and tested such as: LogisticRegression, XGBClassifier, LightGBM, Keras Classifier, etc. Running multiple tests and fine tuning the hyperparameters resulted in XGBClassifier and Keras Classifier as the best performing models for this task. In the last stage of the model selection, an intensive grid search, to find the best hyperparemters on these last two models, resulted in the Keras DNN as the selected model.

Define Model

After many attempts, testing with the use of grid search and try different configurations and scalers, the best results were obtained with a Keras Sequential model with 4 Dense layers and Dropout as shown in Figure 10.

```
def make_model(metrics=METRICS):
    def build_model():
        model = Sequential()
        model.add(Dense(500, activation='relu', input_dim=input_dim))
        model.add(Dropout(0.2))
        model.add(Dense(500, activation='relu'))
        model.add(Dropout(0.2))
        model.add(Dense(500, activation='relu'))
        model.add(Dropout(0.2))
        model.add(Dense(500, activation='relu'))
        model.add(Dense(1, activation='sigmoid'))
        return model
    model = build_model()
    model.compile(loss='binary_crossentropy', optimizer="adam", metrics=metrics)
    return model

model = make_model()
model.summary()
```

Figure 10 – Define Model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_160 (Dense)            (None, 500)               6500

dropout_96 (Dropout)         (None, 500)               0

dense_161 (Dense)            (None, 500)               250500

dropout_97 (Dropout)         (None, 500)               0

dense_162 (Dense)            (None, 500)               250500

dropout_98 (Dropout)         (None, 500)               0

dense_163 (Dense)            (None, 500)               250500

dense_164 (Dense)            (None, 1)                 501

=================================================================
Total params: 758,501
Trainable params: 758,501
Non-trainable params: 0
```

Figure 11 – Model definition

Train Model

The model was trained with the definitions shown in the Figure 12, using EarlyStopping with the validation F1 score as monitor. This way we can be sure that the model is not overfitting and will get the best results for the F1 score.

```
EPOCHS = 500
BATCH_SIZE = 500

early_stopping = EarlyStopping(
    monitor='val_f1',
    verbose=1,
    patience=50,
    mode='max',
    restore_best_weights=True)

# Evaluate Model
model = make_model()
model_history = model.fit(
    train_features,
    train_labels,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    callbacks=[early_stopping],
    shuffle=True,
    validation_data=(val_features, val_labels),
    verbose=0)

```

Figure 12 – Train Model

Looking at the training history, we can see the importance of the EarlyStopping to prevent the model to keep training when the validation score is not improving anymore, and the model might be overfitting.

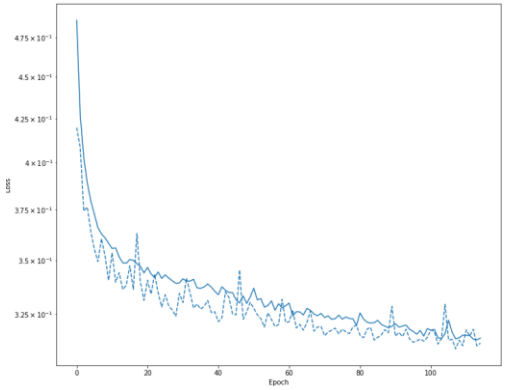

Figure 13 – Training: Loss vs Epoch

Evaluate Model

Evaluation metrics

For the model evaluation I used the test dataset that was not used before when training the model. Here evaluation metrics are shown in Figure 14, but it is important to highlight the **F1 score result of 0.85** (baseline model was 0.68) which is a very good result.

```
loss :  0.32206082344055176
tp :  4972.0
fp :  1227.0
tn :  4280.0
fn :  488.0
accuracy :  0.8436217904090881
precision :  0.802064836025238
recall :  0.9106227159500122
auc :  0.9328649044036865
prc :  0.9306991100311279
f1 :  0.8526038527488708
```

Figure 14 – Evaluation metrics (test dataset)

Confusion matrix

From the confusion matrix in Figure 15, we can see that the model seems to confuse absent points with present points aka false positives, as shown in the top right corner. One of the reasons I found for this problem was the differences in the density of occurrences depending on the area. In the next section I will explain the use of a dynamic threshold to mitigate this issue.

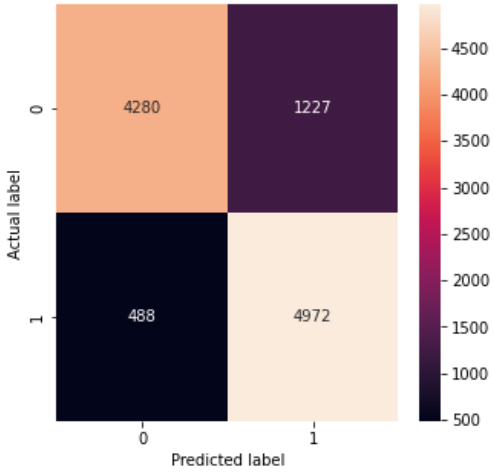

Figure 15 – Confusion matrix

ROC curve

A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The ROC curve for the trained model is shown in Figure 16. The best possible prediction method would yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing 100% sensitivity (no false negatives) and 100% specificity (no false positives). In this case we can see that the model is doing a fairly good job with a curve near that ideal corner.

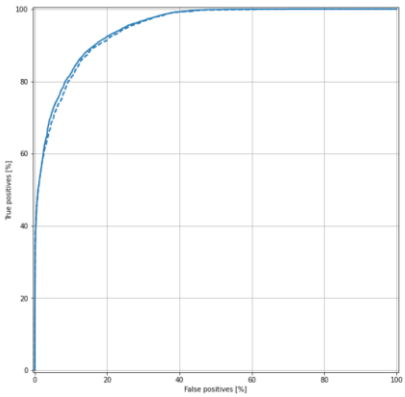

Figure 16 – ROC curve

Dynamic Threshold

The model predictions return a probability value between 0 and 1 and it is necessary to establish a threshold to decide if the frog is present or not. The simplest approach is just to set 0.5 as a threshold and transform all the values to 1 or 0.

As suggested in the "**Geo-spatial modelling with unbalanced data**" document published in this study, the use of **Dynamic Threshold** can help to mitigate this issue and improve the results when using the model. Using a dynamic threshold allows to mitigate the false positive issue presented in the previous section and maximize the F1 score, which is the metric used in this

challenge to measure the quality of the model. To do this, a dynamic threshold is used depending on the region that we are trying to predict. This technique allows to improve the model usage, adapting the threshold to regions that contains different characteristics.

Submission

The model was tested with the predictions on the unseen data (submission csv file) resulting in a F1 score, generated on the ranking board on the EY challenge portal, of **0.77**. A sample resulting map of the target species point, the probabilities and the classification is shown in Figure 17.

IMPORTANT: Due to the lack of experience on these challenges (this is my first time participating) I did not save the trained model with that resulting score. The attached trained model in the assessment is the same model, trained with the same dataset, but the results on the submits may differ.

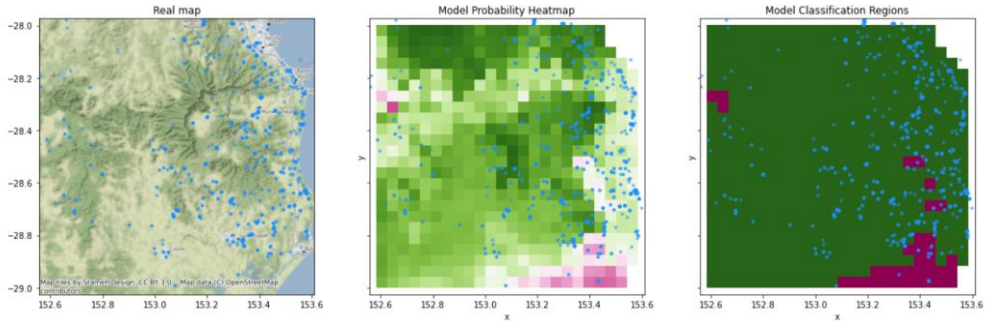

Figure 17 – Sample submission maps

Conclusions and Improvements

Some key elements of the presented approach:

- Deep study and statistical evaluation of the problem and the data set
- Smart down-sampling based on the isolation of each occurrence
- Annual climate data selected to best represent each occurrence
- Dataset split in train, test, and validation dataset for better evaluation
- Usage of dynamic threshold based on the density of the test region
- Usage of Grid Search in the early iterations of the model selection
- Multiple models were tested until the selection of the DNN model
- Inference time: `13.54 seconds`

Many improvements can be implemented to the present data preparation and model, unfortunately I discovered the existence of the challenge a little bit late, and the lack of time was a big impediment. Some features to explore will be the skewness of the features, the statistical representations, and the removal of outliers, also it is important to mention some unexplored techniques such as dedicated scalers for each feature, etc.

I hope that this work will prove useful for the identification of the eastern dwarf tree frog Litoria Fallax, since as stated in the challenge introduction a successful frog SDM will have broader implications in quantifying biodiversity, gaining insight into ecosystems for better learn the impact of human and climate changes, and in the end build a better world.