

AHooper ML Assignment 1

FINAL

by Austin Hooper

FILE	936_AUSTIN_HOOPER_AHOOPER_ML_ASSIGNMENT_1_FINAL_309231_235050845.PDF (3.42M)		
TIME SUBMITTED	03-DEC-2020 03:18PM (UTC+0000)	WORD COUNT	12034
SUBMISSION ID	138202783	CHARACTER COUNT	69253



L7: Application of Machine Learning (ACCA7025)

Machine Learning Assignment 1

Student: Austin Hooper (1700481)
MSc Software Engineering and Artificial Intelligence

Wales Institute for Science and Art (WISA)
University of Wales Trinity Saint David

Lecturer: Gordon Dickers

30th November 2020

Table of Contents

1. Introduction	1
2. Dataset Choice and Justification	2
2.1. Dataset Context	2
2.2. Dataset Purpose	3
2.3. Data Pre-processing and Exploration	3
2.3.1. Data Cleaning	3
2.3.2. Feature Selection and Augmentation	4
2.3.3. Exploring Correlations	5
3. Machine Learning Algorithms	7
3.1. Random Forest Classifiers	8
3.1.1. Classifying Genus	8
3.1.2. Classifying Family	12
3.2. Regression	15
3.2.1. Predicting Mean Clutch Size and SVL based on Normalised SVL Range	16
3.2.2. Predicting Largest Clutch Size from the Mean	19
3.2.3. Average SVL to Clutch Size	19
3.2.4. Mean Body Temperature to Predict Range and Maximum	21
4. Conclusion and Future Work	23
References	24
Background Reading	28
Appendices	29
Appendix A	29
Appendix B	57
Appendix C	58
Appendix D	59

Table of Figures

Figure 1: Numerical value imputation code	3
Figure 2: Feature normalisation code	4
Figure 3: Correlation matrix for all features	5
Figure 4: Pruned correlation matrix	6
Figure 5: Scikit-Learn machine learning program flow	7
Figure 6: Genus classification RF confusion matrix	9
Figure 7: Genus RF classification probability ROC curve	10
Figure 8: Genus RF classification OneVsRestClassifier ROC curve	10
Figure 9: Genus classification performance visualisation	11
Figure 10: Family classification RF confusion matrix	12
Figure 11: Family RF classification OneVsRestClassifier ROC curve	13
Figure 12: Family classification performance visualisation	14
Figure 13: SVL range to mean clutch size polynomial orders	16
Figure 14: Normalised SVL range to mean clutch size, regressor comparison	17
Figure 15: Normalised SVL range to average SVL regression	18
Figure 16: Mean clutch size to largest clutch size regression	19
Figure 17: Clutch size to SVL - high alpha	20
Figure 18: Clutch size to SVL - low alpha	20
Figure 19: Range to mean body temperature regression	21
Figure 20: Mean to maximum body temperature regression	22

Table of Tables

Table 1: Useful dataset features	2
Table 2: Derived features	4
Table 3: Classifier performance comparison for genus	11
Table 4: Classifier performance comparison for family	14

Abstract

Machine Learning is a rapidly expanding field within Artificial Intelligence, describing the ability for a computer system to learn without being specifically programmed to do so. This report applied Machine Learning concepts and algorithms to a dataset of lizard traits using Python's Pandas and Scikit-Learn libraries. Data exploration and cleaning were undertaken, including deriving new features and imputing missing values. Classification and regression algorithms were explored, with a focus on classification. Classification on the top 15 classes for lizard genus and family was successful, using Receiver Operating Characteristic's Area Under the Curve score as the primary evaluation metric. Random Forest classification performed best on lizard genus, with a small decrease in performance on lizard family. A number of regression algorithms were explored, including linear, polynomial, Support Vector Regression, and Random Forest regression. These regression algorithms provided insight into the relationships between variables, particularly lizard size (snout-vent length), and number of eggs laid per clutch. The exploration of Machine Learning techniques was successful, having identified strengths and weaknesses of a range of algorithms. Future work could entail classification of additional features such as future extinction, as well as exploring multiple regression.

1. Introduction

Machine learning (ML), a subset of Artificial Intelligence, is the *“Field of study that gives computers the ability to learn without being explicitly programmed”* [1, p. 283], a frequently used paraphrase of leading author Arthur Samuel's quote *“Programming computers to learn from experience should eventually eliminate the need for much of this detailed programming effort”* [2].

This report details and justifies the choice a lizard traits dataset. This dataset is used to demonstrate data cleaning, pre-processing, feature selection and ML algorithm techniques in Python using the Pandas [3] and Scikit-Learn [4] libraries. Classification and regression were explored, described, justified, and evaluated. The code is provided as a Jupyter Notebook, which has been included in Appendix A, along with a list of library requirements.

2. Dataset Choice and Justification

This section justifies and explores the chosen dataset, provided in S. Meiri's paper [5] discussing lizard traits and correlations, though not utilising ML techniques. The dataset was chosen as it lies within the author's interests, also providing many opportunities to explore ML techniques. Understanding herpetology allows for understanding the dataset at a subject level and allowing for identification of any false or invalid data.

2.1. Dataset Context

Application of statistical analysis and ML techniques on reptile data has been extensively explored [6]–[15]. This assignment aims to further explore application of ML techniques to zoological data. The dataset has a range of categorical and numerical features. Potentially useful features are listed in Table 1. The full table of features is provided in Appendix B. The dataset contains approximately 6,500 records.

Table 1: Useful dataset features

Feature Name	Data Type
Binomial	Categorical
Genus	Categorical
Epithet	Categorical
Year of description	Integer
Country described from	Categorical
main biogeographic Realm	Categorical
Latitude centroid (from Roll et al. 2017)	Decimal
Longitude centroid (from Roll et al. 2017)	Decimal
Insular endemic	Boolean (yes / no)
Maximum SVL	Decimal
Female SVL	Decimal
Hatchling/neonate SVL	Decimal
Leg development	Categorical
Activity time	Categorical
Substrate	Categorical
Diet	Categorical
Foraging mode	Categorical
Reproductive mode	Categorical
Smallest mean clutch size	Decimal
Largest mean clutch size	Decimal
Youngest age at first breeding (months)	Integer
Oldest age at first breeding (months)	Integer
Minimum mean Tb	Decimal
Maximum mean Tb	Decimal
Family	Categorical
IUCN redlist assessment	Categorical
IUCN population trend	Categorical
Extant/Extinct	Categorical
Geographic Range	Categorical, list

2.2. Dataset Purpose

Using this dataset aims to demonstrate classification and regression algorithms. Classification was the primary focus, used to classify genus and family. Regression was used to predict feature values. These algorithms could be used on larger datasets, aiding in making new classifications, and validating known classifications.

2.3. Data Pre-processing and Exploration

Prior to utilising ML algorithms, the dataset was explored, pre-processed, and new features were derived from existing ones.

Principle Component Analysis (PCA) was not explored, as data exploration and ML algorithms had already been programmed before PCA was discussed in class.

2.3.1. Data Cleaning

Where appropriate, one-hot encoding was performed on categorical features to enable using them in ML algorithms [16]–[18], along with converting “yes” / “no” features to Booleans.

Imputation was performed on features which consisted of at least 70% valid values - a mid-point between retaining features and a low proportion of imputed data [19]–[23], though this would ideally be higher. Figure 1 shows developed code to impute values of numerical features. The median value was chosen for numerical data over the mean as it reduces the impact of outlier data points. A similar approach was used to impute categorical values, instead using a mode imputer.

```
# Features to impute medians of
numerical_features = ["year of description", "latitude centroid", "longitude centroid", "distance from equator",
                     "maximum svl", "female svl", "hatchling/neonate svl", "smallest clutch", "largest clutch",
                     "mean clutch size", "mean body temperature"]

# A list of the features which were kept after imputing, i.e. have enough non-null values
# This is used for normalisation later
numerical_features_kept = []

# Empty dataframe to add to
imputed_rep_traits = pd.DataFrame()

# Columns less than this proportion of values intact will be ignored
NON_MISSING_THRESHOLD = 0.70

# For every numerical feature
for feature_name in numerical_features:
    # If more than a specified amount of the data is not null, impute and then add it to a new dataset
    if GetCountNumericalVals(rep_traits, feature_name) > (len(rep_traits.index) * NON_MISSING_THRESHOLD):
        imputed_rep_traits[feature_name] = ImputeMedian(rep_traits, feature_name).reshape(-1, 1)
        numerical_features_kept.append(feature_name)
```

Figure 1: Numerical value imputation code

Numerical features of the dataset were normalised to improve algorithm performance, using the code in Figure 2.

```
Normalise all of the values that are numerical in our imputed dataframe
n sklearn import preprocessing
_min_max_scaler = preprocessing.MinMaxScaler()

For each feature, normalise it and replace the original with the normalised
feature_name in numerical_features_kept:
    column_values = imputed_rep_traits[feature_name].values.reshape(-1, 1)
    column_values_scaled = min_max_scaler.fit_transform(column_values)
    imputed_rep_traits[feature_name] = pd.DataFrame(column_values_scaled)
```

Figure 2: Feature normalisation code

2.3.2. Feature Selection and Augmentation

Some features contained few valid values, ranging from 700 to 1,200. These were omitted from classification models as they were unlikely to affect classification results while allowing for classification with less imputed values – increasing model performance.

Algorithm-specific feature selection is discussed in each algorithm's Section (3.1 and 3.2). New features were derived from existing ones – described in Table 2.

Table 2: Derived features

Feature Name	Calculation Method	Data Type
Mean breeding age (months)	Mean of oldest age at first breeding (months) and youngest age at first breeding (months)	Integer
Mean clutch size	Mean of minimum and maximum clutch sizes	Float
Clutch size range	Largest clutch - smallest clutch	Integer
Mean body temperature	Mean of minimum and maximum body temperatures	Float
Body temperature range	Maximum body temperature - Minimum body temperature	Float
Average SVL	Mean of hatchling/neonate SVL and maximum SVL	Float
Neonate to adult SVL range	Maximum SVL - neonate/hatchling SVL	Float
Normalised SVL range	Neonate to adult SVL range ÷ average SVL	Float
Distance from equator	Absolute value of latitude centroid	Float
Number of countries present in	Number of commas in geographic range + 1, e.g. ("Cyprus, Lebanon, Syria, Turkey") has 3 commas, thus 4 items	Integer

2.3.3. Exploring Correlations

A correlation matrix heatmap evaluated correlations between appropriate features, shown in Figure 3. A trend in this matrix is blocks of correlations (white), where related features correlate strongly (0.7 to 1.0). Not all calculated columns had meaningful correlations.

From these findings, correlations were further explored for relevant features, including scatter plots and correlation coefficients, shown in Figure 4.

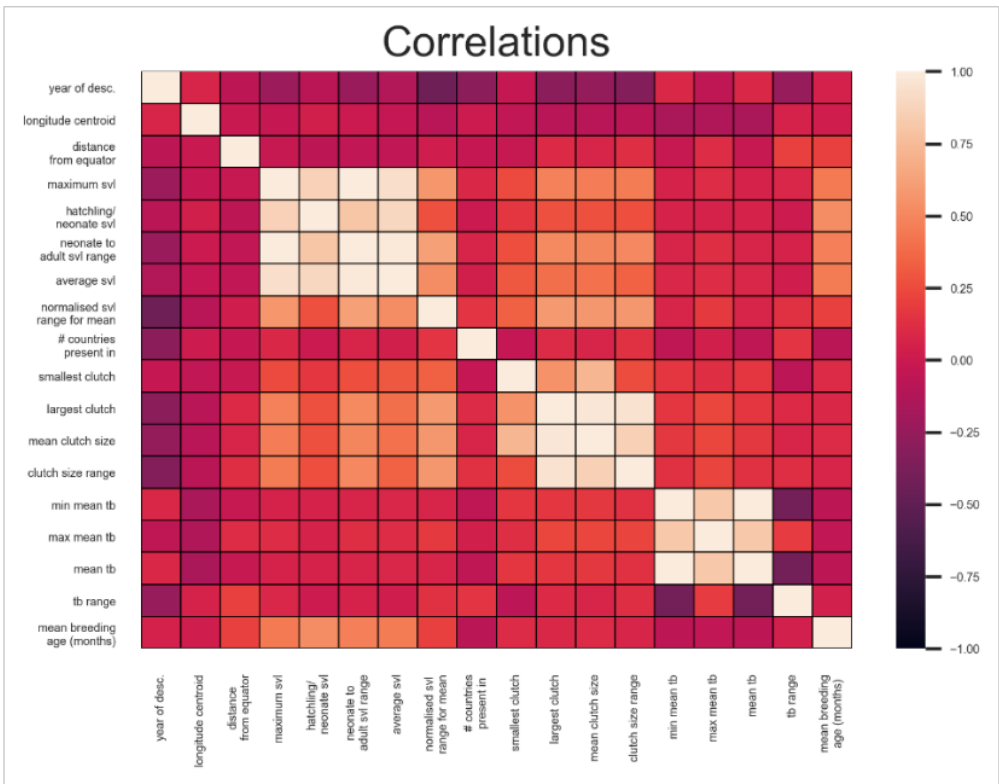


Figure 3: Correlation matrix for all features

After removing uninformative features, remaining correlations were mostly weak to moderate (~ 0.4 to ~ 0.6), with some strong correlations, above 0.8. These correlations appear to have a mix of linear and polynomial trends, although some are noisy.

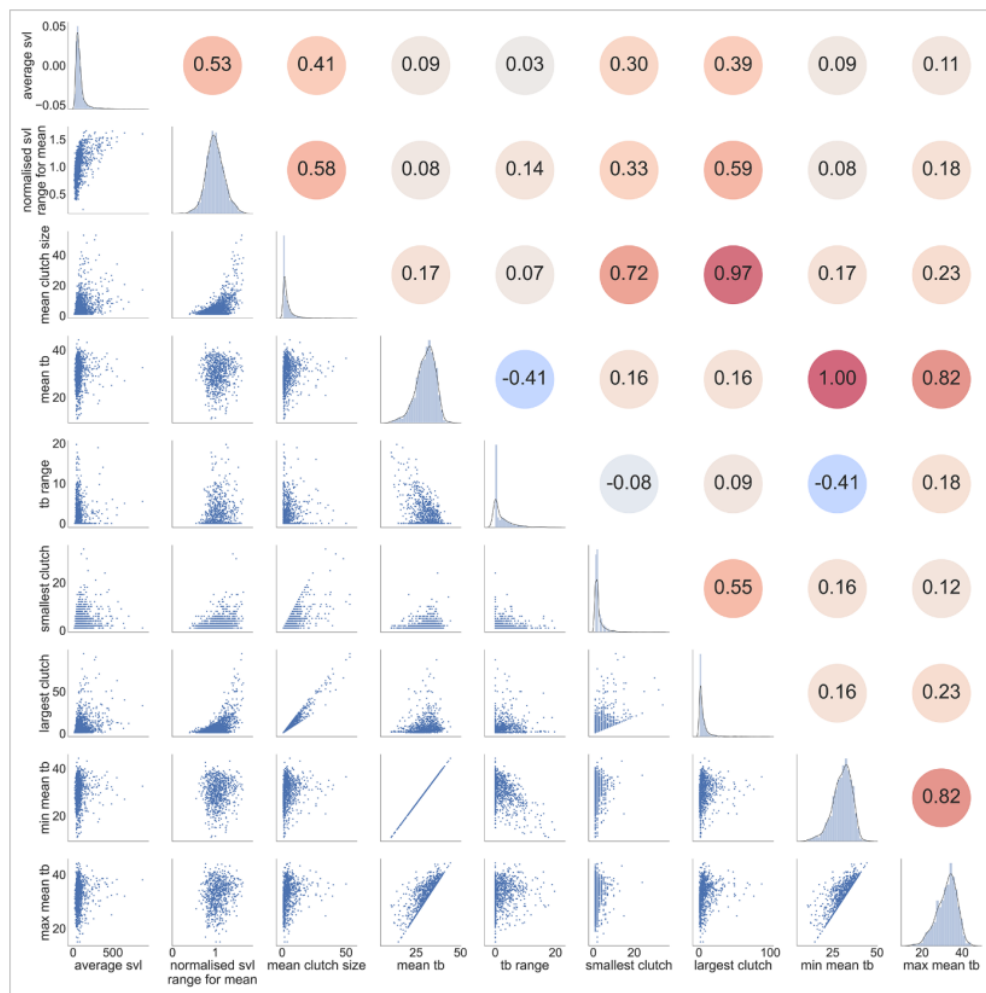


Figure 4: Pruned correlation matrix

3. Machine Learning Algorithms

This section describes classification using Random Forests (RF), and regression using a variety of algorithms, evaluating their suitability and performance given specific goals as discussed in Section 2.2. Common Scikit-Learn ML workflow code is shown in Figure 5.

```
# Get the features to be used
X = exampleDataFrame.iloc[:, :-1].values # X contains the independent variable
y = exampleDataFrame.iloc[:, 1].values   # Y contains the dependent variable

# Split the data 80:20, train:test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Select the model
from sklearn.linear_model import LinearRegression
lin_reg_model = sklearn.linear_model.LinearRegression()

# Train the model
lin_reg_model.fit(X_train, y_train)
print(lin_reg_model.intercept_)
print(lin_reg_model.coef_)

# Predict for the test set
y_pred = lin_reg_model.predict(X_test)
```

Figure 5: Scikit-Learn machine learning program flow

3.1. Random Forest Classifiers

Random Forests, proposed by Tin Kam Ho in 1995 [24], output the mode classification result of a multitude of decision trees (DT) [25], [26], [27, p. 189], [28]. DTs perform sequential splits of data in a tree-like structure, commonly binary trees [25], [27, pp. 175–179], [29]. RF is a supervised learning algorithm; it trains a model using correct labels to classify new data points. Random Forests were chosen due to their high performance compared to alternative algorithms, despite their tendency for overfitting [25], [26], [29], [30].

Lizards' genera and families were classified. Animal genus is more specific than family as shown by Encyclopaedia Britannica [31], included in Appendix C, which may affect classification performance.

The following features were used for classification, determined through background knowledge and experimentation:

- Leg development
- Activity time
- Substrate
- Maximum Snout-Vent Length (SVL)
- Latitude centroid
- Longitude centroid
- Insular endemic
- Reproductive mode
- Diet

The most frequent 15 classes were used for classification. This increases validity by ensuring only classes with high sample counts were used.

3.1.1. Classifying Genus

For classifying the lizards' genus, a default, un-pruned DT achieved accuracy of 0.89, precision, recall and F1 scores of 0.87. Pruning had little effect – the only impact came from setting the maximum number of leaf nodes to 150, providing a negligible improvement of 0.01 to all scores. Any other pruning reduced performance – suggesting that even such a complex tree is still underfitting the test data. A default RF classifier performed identically to the pruned decision tree, though limiting the number of features per decision, and limiting the maximum depth increased performance as shown in Table 3. If this dataset were larger, the performance impact of RFs may outweigh the rate of improved classification, but more data may improve performance and efficacy of RFs. A sample of one of the RFs trees is included in Appendix D.

See in Figure 6 the confusion matrix for the genus classifier, showing very few confusions between genera.

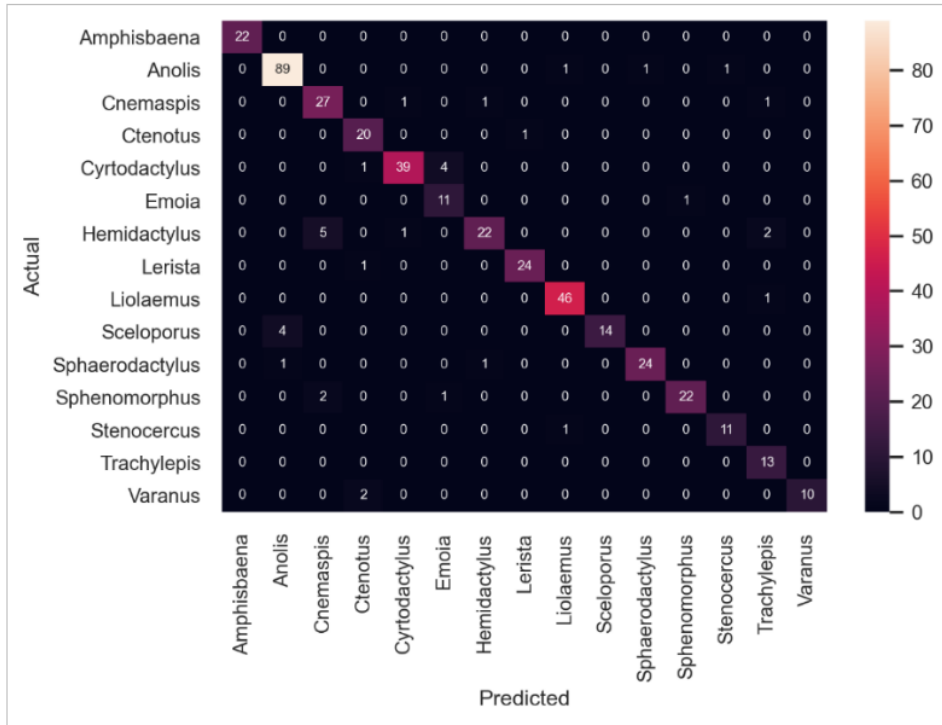


Figure 6: Genus classification RF confusion matrix

A Receiver Operating Characteristic (ROC) curve was used to evaluate the classifier's performance along with previously discussed metrics. Two approaches were tested for this:

- Predicting the probability (continuous) of a lizard being a member of each class, versus being in any of the others, as discussed by Géron [27, pp. 98–100]. This is shown in Figure 7 (zoomed in).
- Using the scikit-learn OneVsRestClassifier [32], predicting the genus class (discrete) of each lizard, shown in Figure 8.

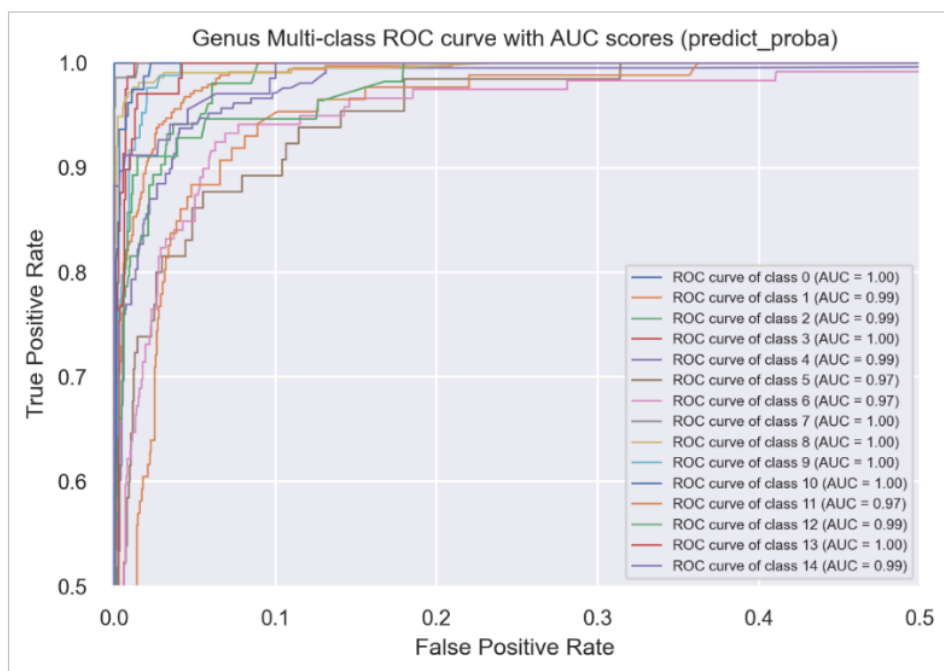


Figure 7: Genus RF classification probability ROC curve

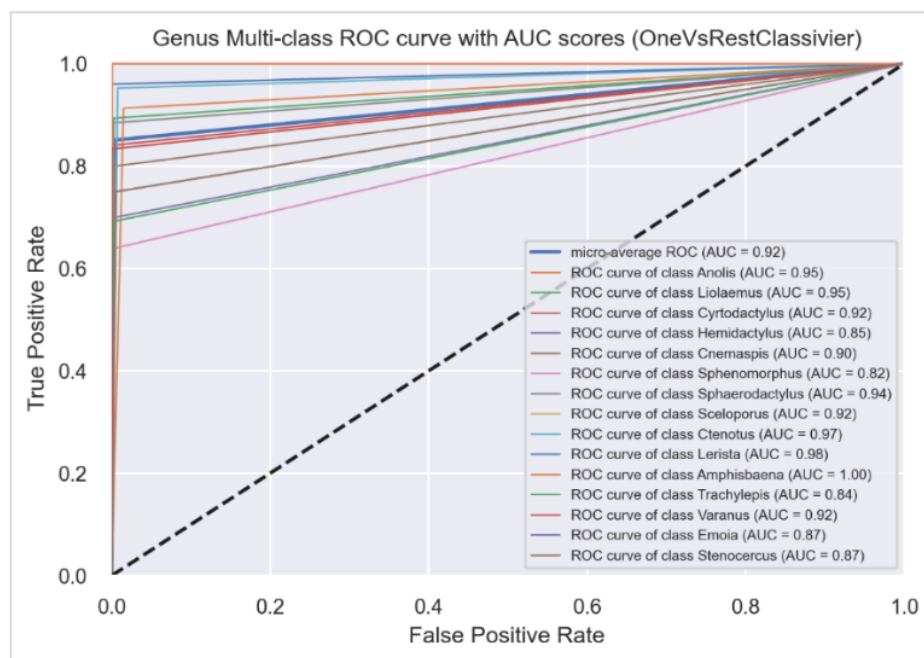


Figure 8: Genus RF classification OneVsRestClassifier ROC curve

The probability method overestimates the classifier's performance as it does not predict discrete values, as discussed in [33], thus not meeting the goal of using this algorithm. The OneVsRestClassifier's Area Under the Curve (AUC) scores ranged from 0.84 to 1.0, being more representative of the classifier's performance, agreeing with its confusion matrix. Therefore, the OneVsRestClassifier method was used for ROC curves hereafter.

Table 3 compares classifiers for genus prediction, visualised in Figure 9. RFs considerably outperform Stochastic Gradient Descent (SGD) and marginally outperform DTs and even SVMs.

Table 3: Classifier performance comparison for genus

Classifier Performance for Genus Prediction				
Classifier	Decision Trees	Random Forests	Stochastic Grad. Descent Classifier	Support Vector Machines
Best Hyper-parameters	max_leaf_nodes: 150	n_estimators: 100 max_features: 6 max_depth: 13	penalty: "l1" max_iter: 50	C: 1000
Accuracy	0.90	0.92	0.77	0.88
Precision	0.88	0.91	0.78	0.86
Recall	0.88	0.91	0.75	0.87
F1	0.88	0.90	0.75	0.86
AUC Min.	0.79	0.82	0.50	0.70
AUC Max.	0.98	1.00	1.00	1.00
AUC Mean	0.92	0.97	0.75	0.93

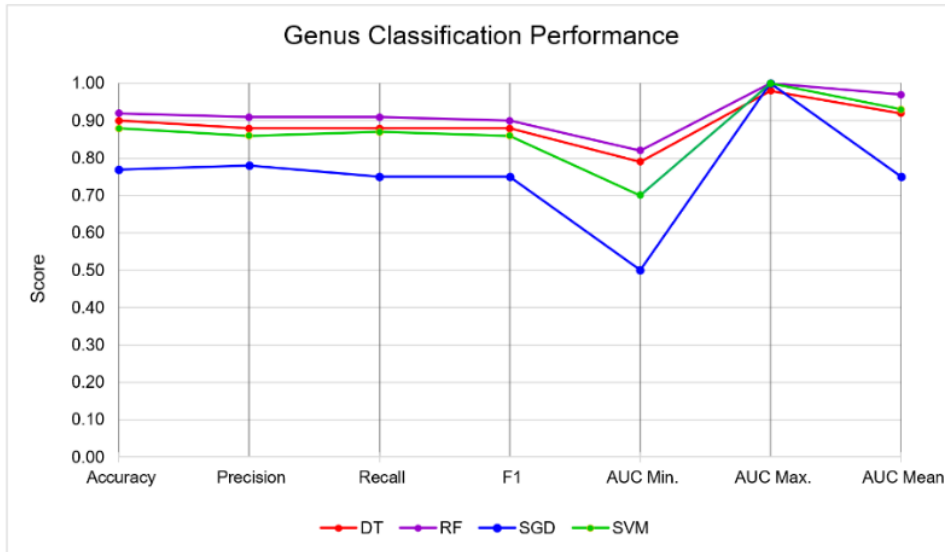


Figure 9: Genus classification performance visualisation

3.1.2. Classifying Family

Classifying family showed similar trends to classifying genus. RFs performed marginally better than DTs, with unpruned RFs achieving accuracy of 0.78, precision of 0.75, recall of 0.72, and F1 of 0.73. Pruning the RF trees marginally improved performance.

As seen in the confusion matrix in Figure 10, some lizard families were classified far better than others. Classes with more distinct features were classified better, for example, Amphisbaenidae (Worm Lizards) are distinct in their lack of legs. Classes with many similar features were often confused, for example Agamidae and Scincidae. This indicates that a larger range of features could be beneficial.

Family classification performed poorer than genus classification. Each family consists of many genera, and thus there is more variance in a family than genus – in an ML context, family data is noisier than genus data. No DT visualisation is provided for family classification due to its complexity.

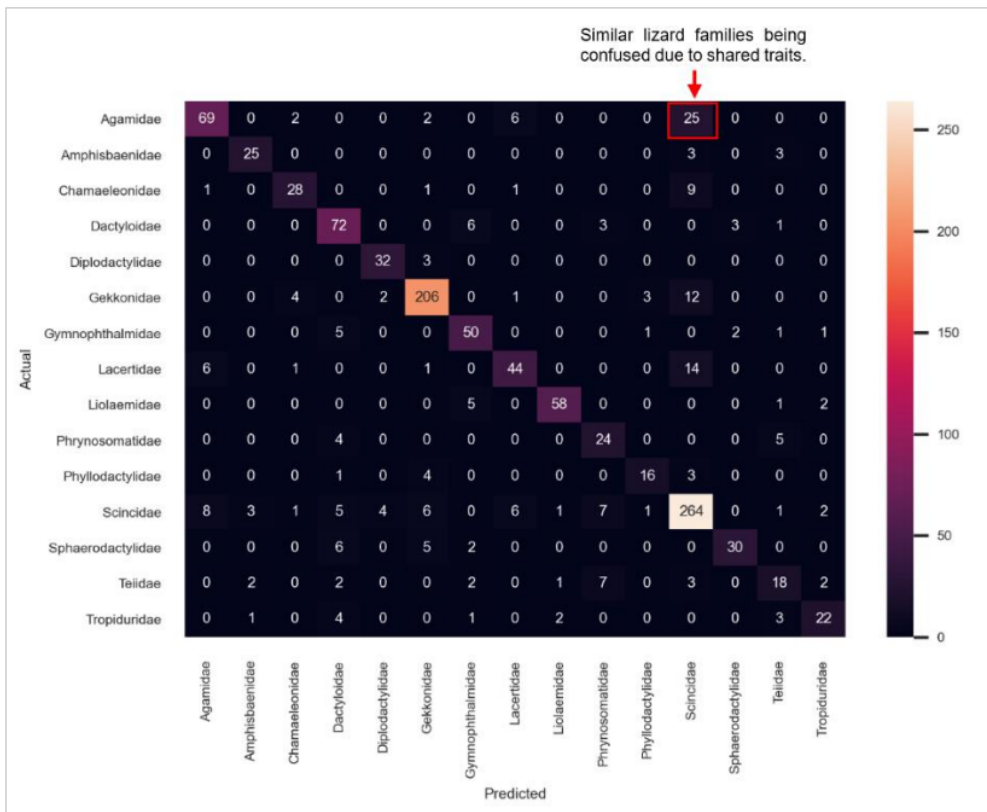


Figure 10: Family classification RF confusion matrix

As shown in the ROC curve in Figure 11, RF classification of lizard family performed well. Most classes were predicted with around 0.8 AUC score, ranging from 0.70 to 0.95. The ROC curve highlights a poorly classified family (Teiidae, AUC 0.70), which was not shown in the confusion matrix.

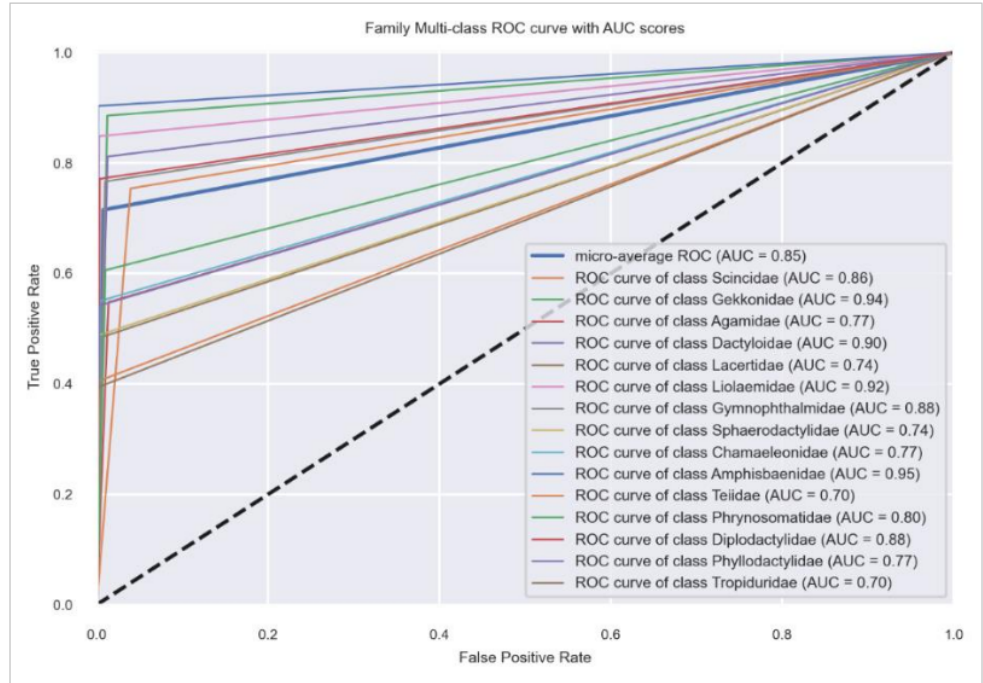


Figure 11: Family RF classification OneVsRestClassifier ROC curve

Table 4 compares classifiers for family prediction, visualised in Figure 12. Similarly to genus prediction, the SGD classifier performed worst, with RFs being slightly ahead of DTs and SVMs.

Table 4: Classifier performance comparison for family

Classifier Performance for Family Prediction				
Classifier	Decision Trees	Random Forests	Stochastic Grad. Descent Classifier	Support Vector Machines
Best Hyper-parameters	max_depth: 11	max_depth: 16 max_features: 7	penalty: "l1" max_iter: 50	C: 1000
Accuracy	0.74	0.80	0.65	0.77
Precision	0.72	0.78	0.67	0.76
Recall	0.70	0.75	0.50	0.72
F1	0.70	0.76	0.51	0.73
AUC Min.	0.74	0.70	0.50	0.55
AUC Max.	0.94	0.95	0.90	0.98
AUC Mean	0.89	0.87	0.75	0.84

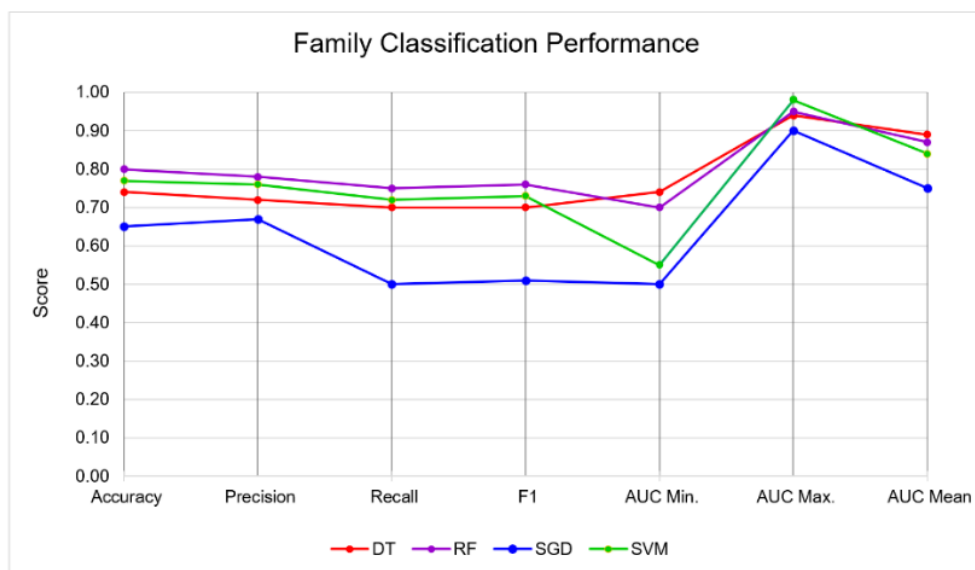


Figure 12: Family classification performance visualisation

3.2. Regression

Regression is a supervised ML method consisting of many algorithms and approaches. Regression determines the correlation between an independent and dependent variable, allowing for prediction of the dependent variable [34]–[37]. Linear Regression (LR) is a basic and early method which remains extremely useful, as discussed by leading authors [27], [38]. Polynomial Regression (PR) allows for prediction on data with more complex correlations, by modelling relationships as n th degree polynomial features [36], [39], [40]. Both LR and PR were used to predict feature values.

Due to the low count of values, imputation was not performed. The following features were used for regression due to their correlations:

- Average SVL
- normalised SVL range for mean
- Mean clutch size
- Mean TB
- TB range
- minimum mean TB
- Maximum mean TB
- Smallest clutch
- Largest clutch

3.2.1. Predicting Mean Clutch Size and SVL based on Normalised SVL Range

SVL, a measure of lizard size, was used to calculate the normalised SVL range feature. The relationship between normalised SVL range and mean clutch size appears polynomial. Regression was explored using several polynomial degrees as shown in Figure 13. Performance metrics and overlapping regression lines show that performance improved slowly until order 4, reaching diminishing returns.

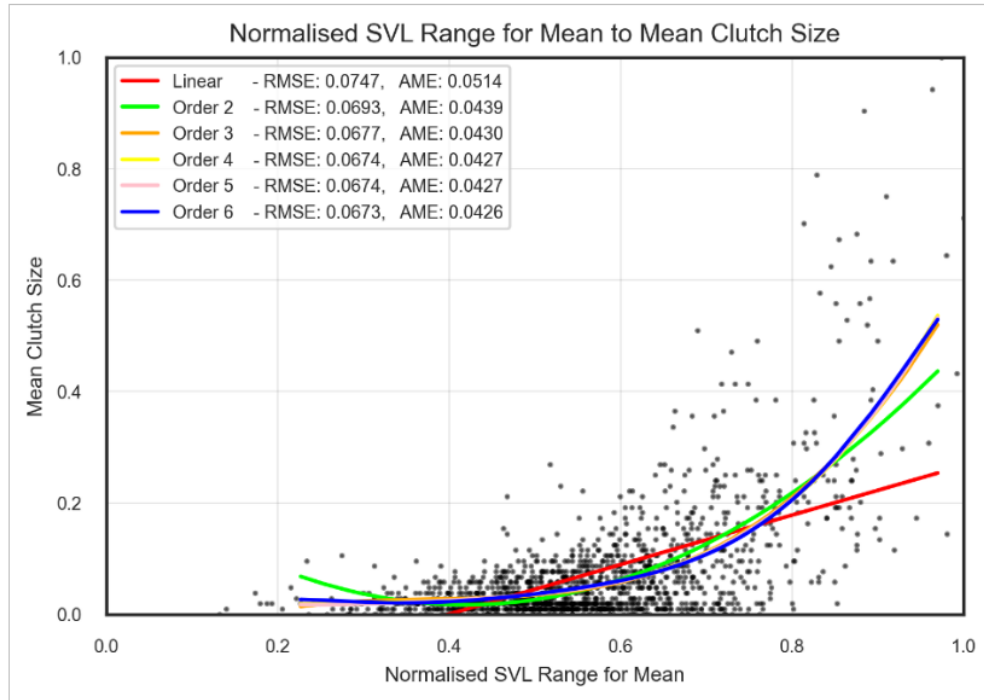


Figure 13: SVL range to mean clutch size polynomial orders

The relationship was further explored using RF regression and Support Vector Regression (SVR). The regressors performed similarly with their best hyperparameters, shown in Figure 14. An overfitting regressor example is also shown. RF regression achieved the lowest RMSE, while SVR achieved the lowest AME despite slow performance, though both were within margin of error.

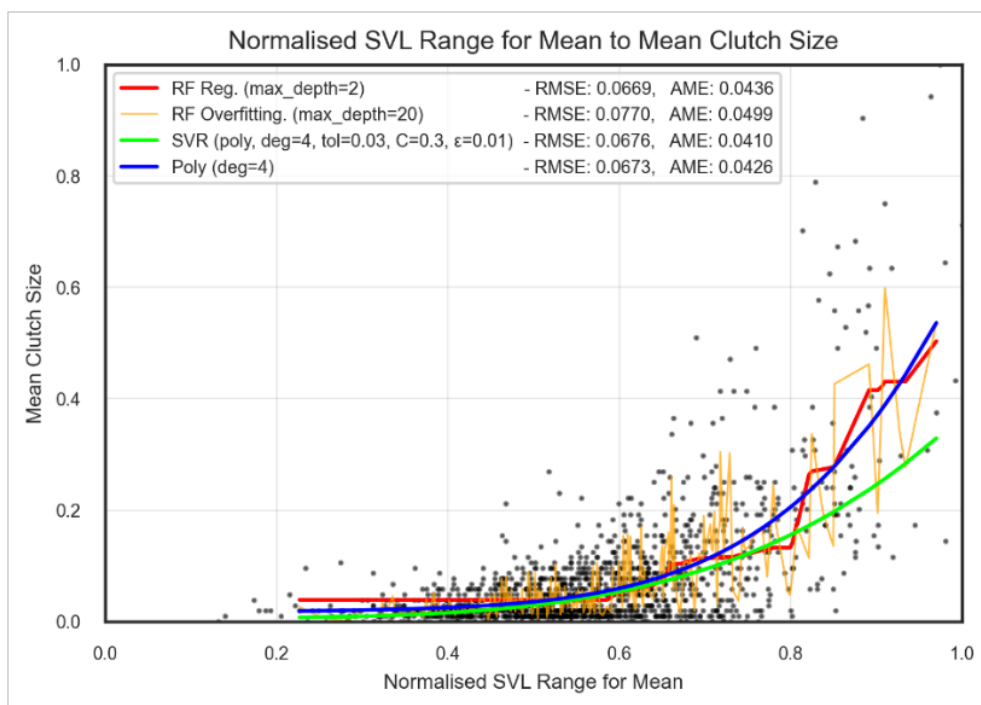


Figure 14: Normalised SVL range to mean clutch size, regressor comparison

Average SVL to mean clutch size has a mostly linear, gradually uplifting trend. Linear and polynomial regressors were fitted, along with RF regression, shown in Figure 15. Linear performed the worst, with order 4 polynomial regression performing the best, tied with tuned RF regression. There was negligible improvement to RMSE and AME with higher orders than 3.

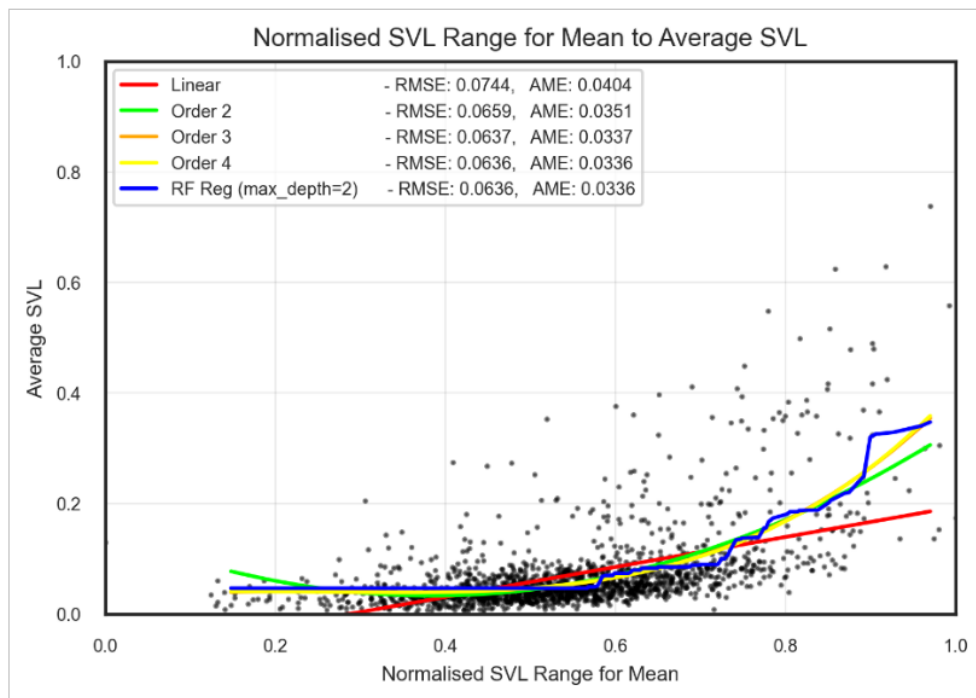


Figure 15: Normalised SVL range to average SVL regression

3.2.2. Predicting Largest Clutch Size from the Mean

Mean and maximum clutch sizes correlated strongly, as anticipated. This relationship is insightful in predicting the theoretical maximum given a mean. Standard linear regression and a linear SVR were applied and compared. Both produced excellent RMSE and AME results as shown in Figure 16. Once tuned, they performed almost identically, as seen in the overlapping lines and performance metrics. The SVR regressor did not benefit meaningfully from modifying hyperparameters.

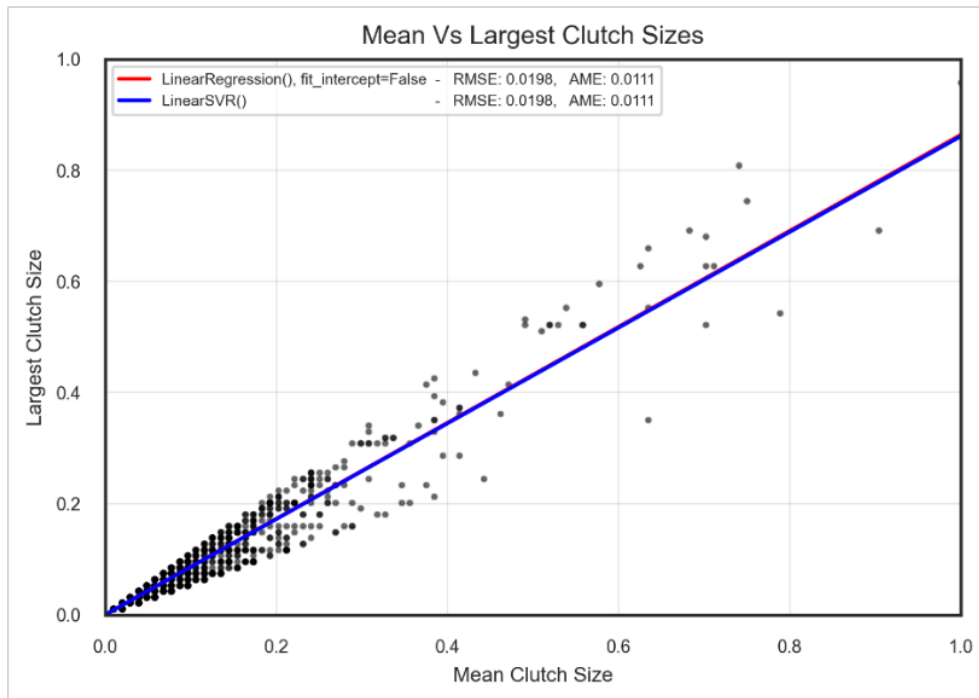


Figure 16: Mean clutch size to largest clutch size regression

3.2.3. Average SVL to Clutch Size

Lizard size correlates with the number of eggs laid, though with a wide range. Figure 17 shows various regressors, appearing to not fit the test data. Figure 18 shows the same results with lower alpha values for data points, highlighting a dense group which aligns with regression trendlines. Polynomial order 2 had the best RMSE, though the linear SVR had the best AME, despite having the worst RMSE. Had this dataset contained more data, this dense trend is expected to be more concentrated, as such the SVR trend line is probably the most accurate.

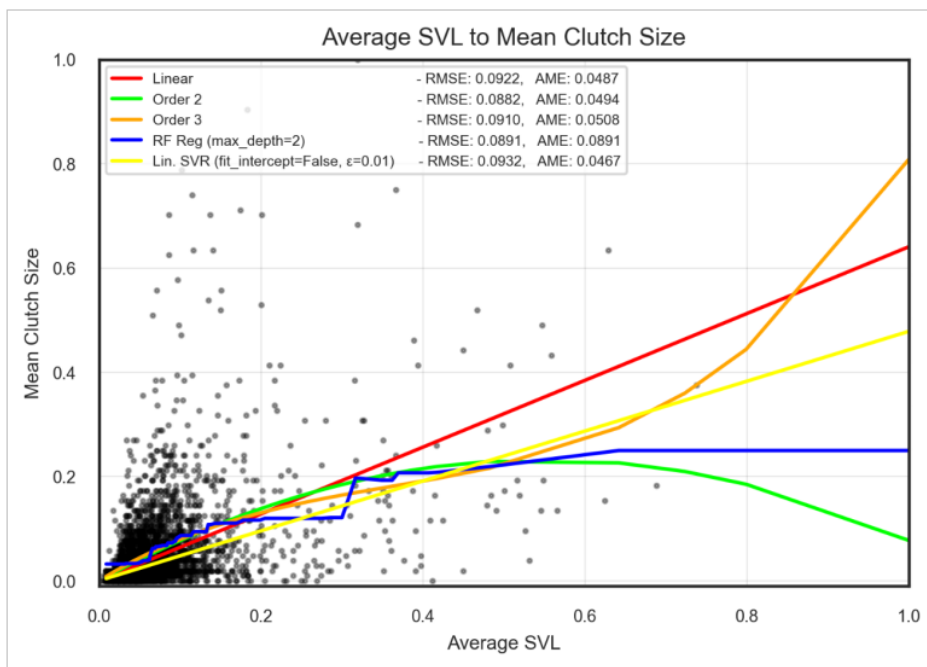


Figure 17: Clutch size to SVL - high alpha

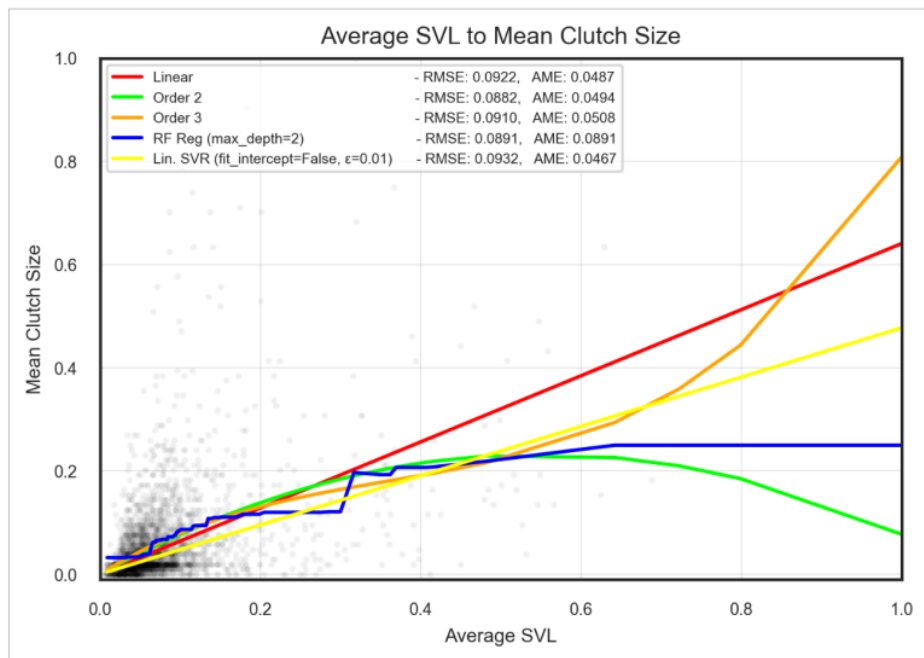


Figure 18: Clutch size to SVL - low alpha

3.2.4. Mean Body Temperature to Predict Range and Maximum

Mean to range of body temperature was insightful. As mean body temperature decreased, range increased. Except for RF regression, hyperparameter tuning had little impact. RF regression performed best by a small margin, with the standard LinearRegression() and SVR performing similarly, as seen in Figure 19. Despite RF regression performing best, it is likely that linear is most indicative of the trend.

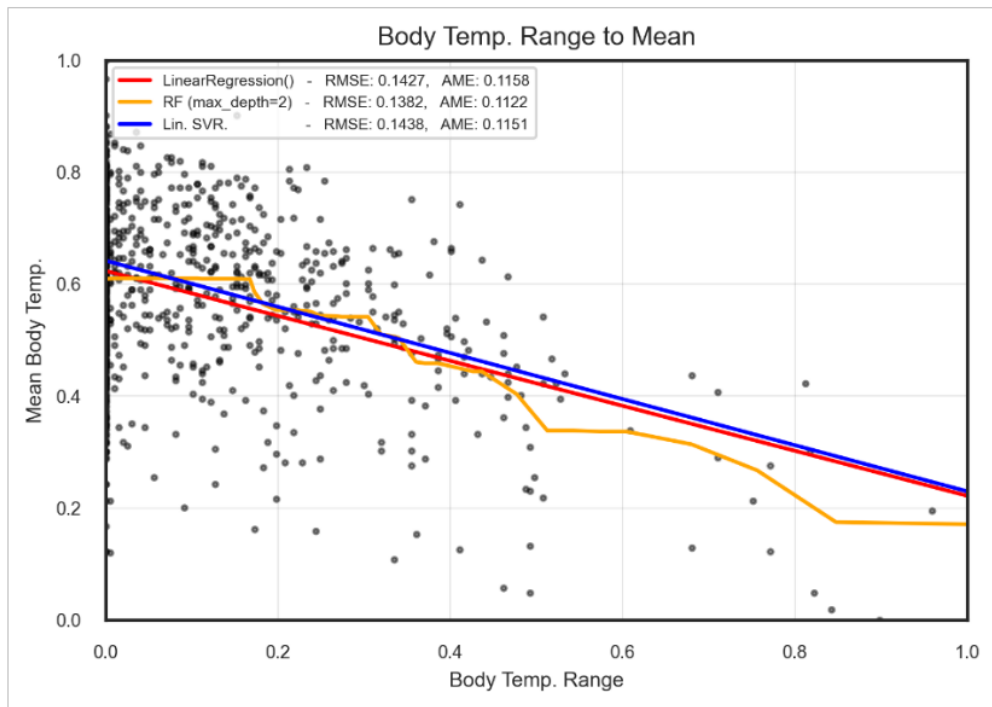


Figure 19: Range to mean body temperature regression

Regression on mean to maximum body temperature provided interesting results. Highly correlated, as anticipated, but with less noise than expected, the trend becomes less noisy as mean temperature increases, as shown in the regression lines' slopes. Tuned RF regression performed the best, reflective of the noise in the data – followed closely by both linear regression algorithms, seen in Figure 20.

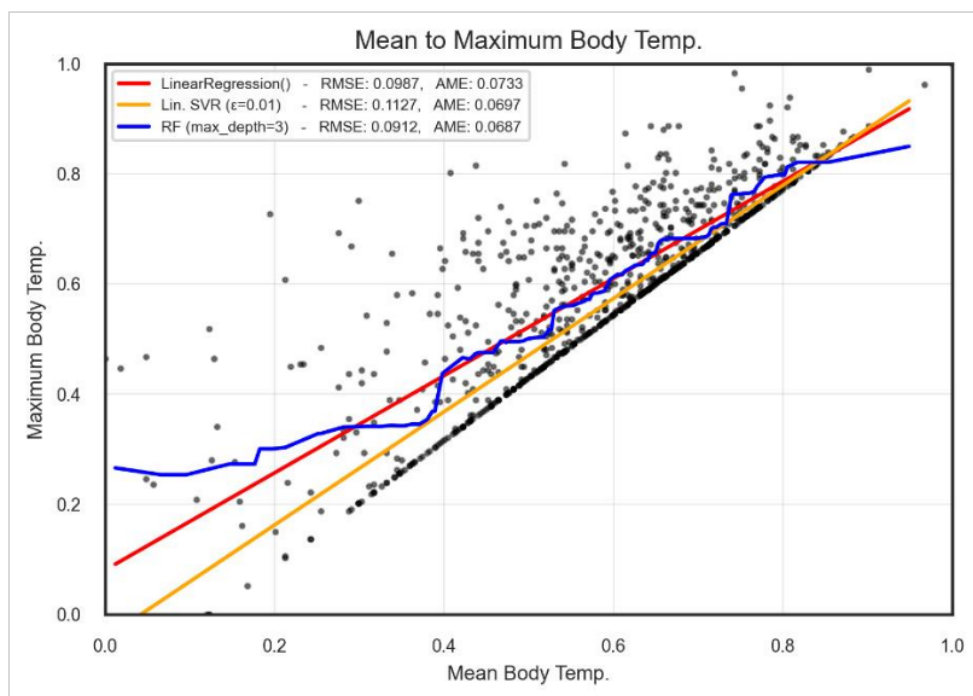


Figure 20: Mean to maximum body temperature regression

4. Conclusion and Future Work

This assignment has explored machine learning techniques including data analysis and clean-up, classification, and regression algorithms. Classification performance was strong, while regression was insightful. Findings aligned with initial discussion by S. Meiri, especially the relationship between mean and maximum clutch size, and body size to clutch size.

Future work could encompass classification of further features such as future extinction, and multiple regression using several features. Background knowledge of reptiles did not align with some findings, such as the relationship between lizard size and clutch size being inverse to what was expected. This could be further explored.

References

- [1] P. Bhavsar, I. Safro, N. Bouaynaya, R. Polikar, and D. Dera, "Chapter 12 - Machine Learning in Transportation Data Analytics," in *Data Analytics for Intelligent Transportation Systems*, M. Chowdhury, A. Apon, and K. Dey, Eds. Elsevier, 2017, pp. 283–307.
- [2] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM J. Res. Dev.*, vol. 3, no. 3, pp. 210–229, 1959.
- [3] pandas.pydata.org, "pandas - Python Data Analysis Library," 2020. <https://pandas.pydata.org/> (accessed Nov. 10, 2020).
- [4] scikit-learn, "scikit-learn: machine learning in Python — scikit-learn 0.23.2 documentation." <https://scikit-learn.org/stable/> (accessed Nov. 10, 2020).
- [5] S. Meiri, "Traits of lizards of the world: Variation around a successful evolutionary design," *Glob. Ecol. Biogeogr.*, vol. 27, no. 10, pp. 1–5, 2018, doi: 10.1111/geb.12773.
- [6] D. B. Miles, "Can Morphology Predict the Conservation Status of Iguanian Lizards?," *Integr. Comp. Biol.*, vol. 60, no. 2, pp. 535–548, 2020.
- [7] C. J. Lortie *et al.*, "Shrub and vegetation cover predict resource selection use by an endangered species of desert lizard," *Sci. Rep.*, vol. 10, no. 1, pp. 1–7, 2020, doi: 10.1038/s41598-020-61880-9.
- [8] M. Fang, X. Zhu, Y. Du, L. Zhang, and L. Lin, "Macroecological patterns of climatic niche breadth variation in lacertid lizards," *Asian Herpetol. Res.*, vol. 10, no. 1, pp. 41–47, 2019, doi: 10.16373/j.cnki.ahr.180034.
- [9] E. R. Pianka, L. J. Vitt, N. Pelegrin, D. B. Fitzgerald, and K. O. Winemiller, "Toward a Periodic Table of Niches, or Exploring the Lizard Niche Hypervolume," *Am. Nat.*, vol. 190, no. 5, pp. 601–616, 2017.
- [10] R. Kekes, T. Kalloo, and K. Opaleychuk, "Habitat Suitability Index for Five-Lined Skink: A species at risk Lizard in Thousand Islands National Park," 2019. [Online]. Available: <https://luna.flemingcollege.ca/Collab/Reports/1918.pdf>.
- [11] M. Ebrahimi, E. Ebrahimie, and C. M. Bull, "Minimizing the cost of translocation failure with decision-tree models that predict species' behavioral response in translocation sites," *Conserv. Biol.*, vol. 29, no. 4, pp. 1208–1216, 2015.
- [12] H. L. Bateman and D. M. Merritt, "Complex riparian habitats predict reptile and amphibian diversity," *Glob. Ecol. Conserv.*, vol. 22, p. e00957, 2020, doi: <https://doi.org/10.1016/j.gecco.2020.e00957>.

- [13] X. A. Jenkins, A. C. Pritchard, A. D. Marsh, B. T. Kligman, C. A. Sidor, and K. E. Reed, "Using manual ungual morphology to predict substrate use in the Drepanosauromorpha and the description of a new species," *J. Vertebr. Paleontol.*, 2020, doi: 10.1080/02724634.2020.1810058.
- [14] S. Tomlinson, "The mathematics of thermal sub-optimality: Nonlinear regression characterization of thermal performance of reptile metabolic rates," *J. Therm. Biol.*, vol. 81, pp. 49–58, 2019, doi: <https://doi.org/10.1016/j.jtherbio.2019.02.008>.
- [15] J. L. M. Judson, C. R. Knapp, and M. E. Welch, "Age-dependent, negative heterozygosity–fitness correlations and local effects in an endangered Caribbean reptile, Iguana delicatissima," *Ecol. Evol.*, vol. 8, no. 4, pp. 2088–2096, 2018, doi: <https://doi.org/10.1002/ece3.3826>.
- [16] A. C. H. Choong and N. K. Lee, "Evaluation of convolutionary neural networks modeling of DNA sequences using ordinal versus one-hot encoding method," in *2017 International Conference on Computer and Drone Applications (ICONDA)*, Nov. 2017, pp. 60–65, doi: 10.1109/ICONDA.2017.8270400.
- [17] M. G. Arnold, I. Kouretas, V. Paliouras, and A. Morgan, "One-Hot Residue Logarithmic Number Systems," in *2019 29th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Jul. 2019, pp. 97–102, doi: 10.1109/PATMOS.2019.8862159.
- [18] M. Zabidi, M. Mokji, I. Kamisian, and N. Ramli, "Implementing One Hot State Machines Using Schematics," *Appl. Note AN6, Digit. Electron. Lab.*, vol. 2019, pp. 1–4, 2019.
- [19] H. Kang, "The prevention and handling of the missing data," *Korean J. Anesthesiol.*, vol. 64, no. 5, pp. 402–406, May 2013, doi: 10.4097/kjae.2013.64.5.402.
- [20] B. L. Browning and S. R. Browning, "Genotype Imputation with Millions of Reference Samples," *Am. J. Hum. Genet.*, vol. 98, no. 1, pp. 116–126, 2016, doi: <https://doi.org/10.1016/j.ajhg.2015.11.020>.
- [21] L. Beretta and A. Santaniello, "Nearest neighbor imputation algorithms: a critical evaluation," *BMC Med. Inform. Decis. Mak.*, vol. 16–3, no. 74, pp. 198–208, 2016, doi: 10.1186/s12911-016-0318-z.
- [22] M. Martinez-Luengo, M. Shafiee, and A. Kolios, "Data management for structural integrity assessment of offshore wind turbine support structures: data cleansing and missing data imputation," *Ocean Eng.*, vol. 173, pp. 867–883, 2019, doi: <https://doi.org/10.1016/j.oceaneng.2019.01.003>.

- [23] C.-F. Tsai, M.-L. Li, and W.-C. Lin, "A class center based approach for missing value imputation," *Knowledge-based Syst.*, vol. 151, pp. 124–135, 2018.
- [24] T. K. Ho, "Random decision forests," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, Aug. 1995, vol. 1, pp. 278–282, doi: 10.1109/ICDAR.1995.598994.
- [25] T. Hillel, M. Bierlaire, M. Elshafie, and Y. Jin, "A systematic review of machine learning classification methodologies for modelling passenger mode choice," *J. Choice Model.*, no. August, 2020, doi: 10.1016/j.jocm.2020.100221.
- [26] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "A review on machine learning-based approaches for Internet traffic classification," *Ann. Telecommun.*, 2020, [Online]. Available: <https://doi.org/10.1007/s12243-020-00770-7>.
- [27] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA: O'Reilly, 2019.
- [28] Q. Li, L. Chen, X. Li, X. Lv, S. Xia, and Y. Kang, "PRF-RW: a progressive random forest-based random walk approach for interactive semi-automated pulmonary lobes segmentation," *Int. J. Mach. Learn. Cybern.*, vol. 11, no. 10, pp. 2221–2235, 2020, doi: 10.1007/s13042-020-01111-9.
- [29] D. Moses, T. Deepa, B. Patri, and M. Sowmya, "A Review of Decision Tree Algorithms for Predictive Analysis in Data Mining," *i-Manager's J. Softw. Eng.*, vol. 12, no. 1, pp. 38–46, 2018.
- [30] A. Singh and C. S. Tucker, "A machine learning approach to product review disambiguation based on function, form and behavior classification," *Decis. Support Syst.*, vol. 97, no. March, pp. 81–91, 2017, doi: 10.1016/j.dss.2017.03.007.
- [31] Encyclopaedia Britannica, "Animal Taxonomy," 2015. <https://www.britannica.com/science/genus-taxon#/media/1/229422/148663%0A> (accessed Oct. 27, 2020).
- [32] scikit-learn, "sklearn.multiclass.OneVsRestClassifier — scikit-learn 0.23.2 documentation." <https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html> (accessed Nov. 04, 2020).
- [33] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, 2006, doi: 10.1016/j.patrec.2005.10.010.

- [34] L. Yang, S. Liu, S. Tsoka, and L. G. Papageorgiou, "Mathematical programming for piecewise linear regression analysis," *Expert Syst. Appl.*, vol. 44, no. February 2016, pp. 156–167, 2016, doi: 10.1016/j.eswa.2015.08.034.
- [35] B. Sen and M. Meyer, "Testing against a linear regression model using ideas from shape-restricted estimation," *J. R. Stat. Soc. Ser. B Stat. Methodol.*, vol. 79, no. 2, pp. 423–448, 2017, doi: 10.1111/rssb.12178.
- [36] C. D. Nye, J. Prasad, J. Bradburn, and F. Elizondo, "Improving the operationalization of interest congruence using polynomial regression," *J. Vocat. Behav.*, vol. 104, pp. 154–169, 2018, doi: 10.1016/j.jvb.2017.10.012.
- [37] M. Su, Y. Guo, C. Men, and W. Wang, "A robust self-weighted SELO regression model," *Int. J. Mach. Learn. Cybern.*, vol. 10, no. 11, pp. 3189–3199, 2019, doi: 10.1007/s13042-019-01009-1.
- [38] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to Linear Regression Analysis*, 5th ed. New Jersey: Wiley, 2012.
- [39] M. H. Sibley, M. Campeze, and J. S. Raiker, "Reexamining ADHD-Related Self-Reporting Problems Using Polynomial Regression," *Assessment*, vol. 26, no. 2, pp. 305–314, 2019, doi: 10.1177/1073191117693349.
- [40] G. Dişken, Z. Tüfekci, and U. Çevik, "A robust polynomial regression-based voice activity detector for speaker verification," *Eurasip J. Audio, Speech, Music Process.*, vol. 2017, no. 1, 2017, doi: 10.1186/s13636-017-0120-6.

Background Reading

A. B and S. A. A. Balamurugan, "Review On Feature Selection Techniques And Its Impact For Effective Data Classification Using UCI Machine Learning Repository Dataset," *J. Eng. Sci. Technol.*, vol. 11, no. 11, pp. 1639–1646, 2016.

D. Phiri and J. Morgenroth, "Developments in Landsat Land Cover Classification Methods: A Review," *Remote. Sens.*, vol. 9, p. 967, 2017.

J. Li et al., "Feature Selection: A Data Perspective," *ACM Comput. Surv.*, vol. 50, no. 6, pp. 1–45, 2018.

M. Raza, F. K. Hussain, O. K. Hussain, M. Zhao, and Z. ur Rehman, "A comparative analysis of machine learning models for quality pillar assessment of SaaS services by multi-class text classification of users' reviews," *Futur. Gener. Comput. Syst.*, vol. 101, pp. 341–371, 2019.

Q. Tao, P. van der Tol, F. F. Berendsen, E. H. M. Paiman, H. J. Lamb, and R. J. van der Geest, "Robust motion correction for myocardial T 1 and extracellular volume mapping by principle component analysis-based groupwise image registration," *J. Magn. Reson. imaging*, vol. 47, no. 5, p. 1397, 2018.

S. Dimitriadis, D. Liparas, and ADNI, "How random is the random forest? Random forest algorithm on the service of structural imaging biomarkers for Alzheimer's disease: from Alzheimer's disease neuroimaging initiative (ADNI) database," *Neural Regen. Res.*, vol. 13, no. 6, pp. 962–970, 2018.

S. Gu, R. Cheng, and Y. Jin, "Feature selection for high-dimensional classification using a competitive swarm optimizer," *Soft Comput. (Berlin, Ger.)*, vol. 22, no. 3, pp. 811–822, 2016.

S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Essex: Pearson Education, 2016.

S. Rahman Fahim, S. K. Sarker, S. M. Mueen, M. R. I. Sheikh, and S. K. Das, "Microgrid Fault Detection and Classification: Machine Learning Based Approach, Comparison, and Reviews," *Energies (Basel)*, vol. 13, no. 13, p. 3460, 2020.

W. Wang and Y. Sun, "Penalized local polynomial regression for spatial data," *Biometrics*, vol. 75, no. 4, pp. 1179–1190, 2019.

W. Yao and L. Li, "A New Regression Model: Modal Linear Regression," *Scand. J. Stat.*, vol. 41, no. 3, pp. 656–671, 2014.

Y. G. Akhlaghi, X. Ma, X. Zhao, S. Shittu, and J. Li, "A statistical model for dew point air cooler based on the multiple polynomial regression approach," *Energy (Oxford)*, vol. 181, pp. 868–881, 2019.

Appendices

Appendix A

Library dependencies:

- matplotlib
- cartopy – not essential
- numpy
- pandas
- sklearn
- os
- sys
- six
- io
- IPython
- Pydotplus – not essential
- seaborn
- warnings – not essential

Code included on the following pages.

Initial Setup and Fetching the Data

Import libraries, set up any general parameters, and fetch the dataset from a remote source

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import numpy as np
import pandas as pd
import sklearn
import os
import sys
from six.moves import urllib
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import tree
from sklearn import preprocessing
from sklearn.impute import SimpleImputer
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn import svm
from sklearn.metrics import precision_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from io import StringIO
from IPython.display import Image
import pydotplus
from sklearn.tree import export_graphviz
from sklearn.svm import LinearSVR
from sklearn.svm import SVR
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import cross_val_predict
import seaborn as sn
import warnings

warnings.filterwarnings('ignore')
mpl.rcParams['figure.dpi'] = 300

# In production this could be an absolute file path but for this
# assignment, I assume it's more convenient to mark it with file paths specified relatively
CURRENT_DIR = os.path.dirname(os.path.realpath('__file__'))
NEW_DATA_DIR = "datasets"

DATASETS_DIR = CURRENT_DIR + "/" + NEW_DATA_DIR

# Fetch the dataset from a URL
def FetchData():
```

```

os.makedirs(DATASETS_DIR, exist_ok=True)

# Download the dataset file and place it in the specified location
data_url = "https://datadryad.org/stash/downloads/file_stream/78828"
dataset_path = os.path.join(DATASETS_DIR, "reptile_traits.csv")
urllib.request.urlretrieve(data_url, dataset_path)

FetchData()

%%html
<style>

/* Stop the cells from scrolling unless they are really tall */

.output_wrapper, .output {
  height:auto !important;
  max-height:1500px
}
.output_scroll {
  box-shadow:none !important;
  webkit-box-shadow:none !important;
}
</style>

```

Read the Dataset into a Pandas DataFrame Object, and Select Features

Read the dataset and do some basic tidying up, mainly:

- Convert feature names to lower case for consistency
- Remove features which are not useful, such as those with remark or references
- Rename some features to be shorter and/or clearer

```
# Load the data into a pandas dataframe object and return it
def LoadData(file_name, data_path=DATASETS_DIR):
    csv_path = os.path.join(data_path, file_name)
    return pd.read_csv(csv_path, encoding = 'unicode_escape')

rep_traits = LoadData("reptile_traits.csv")

# Drop the columns that won't be useful, such as those including references or discussions
# / comments
rep_traits = rep_traits.drop(columns=[
    "valid (reptile database; February 2018)",
    "known only from the only type?",
    "mass equation (Feldman et al. 2016 unless stated)",
    "intercept",
    "slope",
    "Activity time: comments",
    "substrate: comments",
    "diet: comments",
    "foraging mode (comments)",
    "References: SVL of males",
    "References: SVL of females",
    "References: SVL of unsexed individuals, neonates and hatchlings",
    "Remarks",
    "phylogenetic data",
    "Phylogeny",
    "mean body temperature of active animals in the wild",
    "clutch size",
    "breeding age (months)",
    "largest mean clutch size",
    "smallest mean clutch size",
    "References (Biology: all columns except M, N and O).",
], axis=1)

# Rename some columns
rep_traits.rename(columns={'Longitude centroid (from Roll et al. 2017)': 'longitude centroid'}, inplace=True)
rep_traits.rename(columns={'Latitude centroid (from Roll et al. 2017)': 'latitude centroid'}, inplace=True)

# Remove upper case so naming is consistent
rep_traits = rep_traits.rename(columns=str.lower)

# Have a look at what we're working with
rep_traits.head()
```

Data Cleaning and Augmentation

First, let's work with numerical features

Create new, more useful features, from existing ones

Calculate new features, for example some means, ranges, subtract two columns etc.

```
# Mean breeding age
rep_traits['mean breeding age (months)'] = rep_traits[
    ['oldest age at first breeding (months)', 'youngest age at first breeding (months)']].mean(
    axis=1)
rep_traits.drop(['oldest age at first breeding (months)', 'youngest age at first breeding (months)'], axis=1, inplace=True)

# Mean clutch size
rep_traits['mean clutch size'] = rep_traits[
    ['smallest clutch', 'largest clutch']].mean(axis=1)

rep_traits['clutch size range'] = rep_traits['largest clutch'] - rep_traits['smallest clutch']

# Mean and range of body temperature
rep_traits['mean tb'] = rep_traits[
    ['minimum mean tb', 'maximum svl']].mean(axis=1)

rep_traits['tb range'] = rep_traits['maximum mean tb'].astype(float) - abs(rep_traits['minimum mean tb'].astype(float))

# Lizard size average, range from hatchling to adult, and range from hatchling to adult normalised for the size of the lizard
rep_traits["hatchling/neonate svl"] = pd.to_numeric(rep_traits["hatchling/neonate svl"], errors='coerce')
rep_traits["maximum svl"] = pd.to_numeric(rep_traits["maximum svl"], errors='coerce')

rep_traits['neonate to adult svl range'] = rep_traits['maximum svl'].astype(float) - rep_traits['hatchling/neonate svl'].astype(float)
rep_traits["average svl"] = rep_traits[['hatchling/neonate svl', 'maximum svl']].mean(axis=1)
rep_traits["normalised svl range for mean"] = rep_traits['neonate to adult svl range'] / rep_traits["average svl"]

# Create a new column for distance from the equator
rep_traits["distance from equator"] = abs(rep_traits["latitude centroid"])

# Find the number of countries each is present in by getting the number of commas and adding 1
rep_traits["# countries present in"] = rep_traits["geographic range"].str.count(",") + 1

rep_traits.drop(["geographic range"], axis=1, inplace=True)
```

Data Exploration

Show correlations for all remaining columns

```
# Select features which are appropriate for correlation
corr_df = pd.DataFrame(rep_traits, columns=[
    "year of description",
    "longitude centroid",
    "distance from equator",
    "maximum svl",
    "female svl",
    "hatchling/neonate svl",
    "neonate to adult svl range",
    "average svl",
    "normalised svl range for mean",
    "# countries present in",
    "smallest clutch",
    "largest clutch",
    "mean clutch size",
    "clutch size range",
    "minimum mean tb",
    "maximum mean tb",
    "mean tb",
    "tb range",
    "mean breeding age (months)"
])

# Rename some fields from the copy dataframe to include line breaks for better readability
# in the heatmap
corr_df.rename(columns={"normalised svl range for mean": "normalised svl\nrange for mean"},
errors="raise", inplace=True)
corr_df.rename(columns={"# countries present in": "# countries\npresent in"}, errors="raise",
inplace=True)
corr_df.rename(columns={"year of description": "year of desc."}, errors="raise", inplace=True)
corr_df.rename(columns={"mean breeding age (months)": "mean breeding\nage (months)"}, errors="raise",
inplace=True)
corr_df.rename(columns={"neonate to adult svl range": "neonate to\nadult svl range"}, errors="raise",
inplace=True)
corr_df.rename(columns={"maximum mean tb": "max mean tb"}, errors="raise", inplace=True)
corr_df.rename(columns={"minimum mean tb": "min mean tb"}, errors="raise", inplace=True)
corr_df.rename(columns={"hatchling/neonate svl": "hatchling/\nneonate svl"}, errors="raise",
inplace=True)
corr_df.rename(columns={"distance from equator": "distance\nfrom equator"}, errors="raise",
inplace=True)

# Create a correlation matrix
corr_matrix = corr_df.corr()

sn.set(font_scale = 0.4) # Label size
plt.title('Correlations', fontsize=16);

# Present the correlation matrix as a heatmap
sn.heatmap(corr_matrix, annot=False, vmin=-1.0, vmax=1.0, linewidths=0.25, linecolor="black")
plt.show()
```

Narrow down correlation features

Pick some of the best and then look in more detail at those. Also check their correlations

```
# Specify a list of attributes to use
attrs=["female svl",
       "average svl",
       "normalised svl\nrange for mean",
       "mean clutch size",
       "mean tb",
       "tb range",
       "smallest clutch",
       "largest clutch",
       "min mean tb",
       "max mean tb"]

# Get the correlation for each relationship and annotate it
def CorrelationFunction(*args, **kwargs):
    corr_r = args[0].corr(args[1], 'pearson')
    corr_text = f"{corr_r:2.2f}"
    ax = plt.gca()
    ax.set_axis_off()
    marker_size = 70000
    ax.scatter([.5], [.5], marker_size, [corr_r], alpha=0.6, cmap="coolwarm",
              vmin=-1, vmax=1, transform=ax.transAxes)
    font_size = 90
    ax.annotate(corr_text, [.5, .5], xycoords="axes fraction",
               ha='center', va='center', fontsize=font_size)

sn.set(style='white', font_scale=5)
g = sn.PairGrid(corr_df[attrs], aspect=1.0, diag_sharey=False, height=7)
g.map_diag(sn.distplot, kde_kws={'color': 'black'}) # Diagonal bar/distribution plots
g.map_upper(CorrelationFunction)                  # Top-right side correlation vlaues
g.map_lower(sn.scatterplot, edgecolor="none")      # Bottom-left side scatter plots

kseaborn.axisgrid.PairGrid at 0x1e3ccc75188>
```

Context: explore location of each lizard record

```
ax = plt.axes(projection=ccrs.PlateCarree())
ax.stock_img()

# Plot equator line
plt.plot([-180, 180], [0, 0],
         color='red', linewidth=0.75, marker='', linestyle="--",
         transform=ccrs.PlateCarree(),
         )
# Plot each latitude and longitude
plt.scatter(rep_traits["longitude centroid"], rep_traits["latitude centroid"],
           color="blue", s=0.1, alpha=0.5, marker="o")

plt.show()
```

Imputation of missing values

If a feature has at least a specified proportion of values intact, then impute the rest.
If it has few non-empty values, then do not impute, but place into a separate dataframe to work with independently.

```
# Returns the number of numeric values in a dataframe's column/feature
def GetCountNumericalVals(dataframe, feature):
    return (pd.to_numeric(dataframe[feature], errors='coerce')).notnull().sum()

# Returns the number of non-null categorical values
def GetCountNotNullVals(dataframe, feature):
    return (rep_traits[feature]).notnull().sum()

# Return the result of imputing a feature by median
def ImputeMedian(dataframe, feature_name):
    # Convert everything to numeric. If the value can't be converted (i.e. text), set it to
    NaN
    dataframe[feature_name] = pd.to_numeric(dataframe[feature_name], errors='coerce')

    median_imputer = SimpleImputer(missing_values=np.nan, strategy='median')

    # Now impute to replace the NaN values with the median of the feature
    return median_imputer.fit_transform(dataframe[feature_name].to_numpy().reshape(-1, 1))

# Return the result of imputing a feature by mode (for categorical data mostly)
def ImputeMode(dataframe, feature_name):
    mode_imputer = SimpleImputer(missing_values=np.nan, strategy='most_frequent')

    # Now impute to replace the NaN values with the median of the feature
    return mode_imputer.fit_transform(rep_traits[feature_name].to_numpy().reshape(-1, 1))

# Features to impute medians of
numerical_features = ["year of description",
                     "latitude centroid",
                     "longitude centroid",
                     "distance from equator",
```

```

        "maximum svl",
        "female svl",
        "hatchling/neonate svl",
        "smallest clutch",
        "largest clutch",
        "mean clutch size",
        "mean tb",
    ]

# Features to impute modes of
categorical_bool_features = ["binomial",
                             "genus",
                             "epithet",
                             "country described from",
                             "main biogeographic realm",
                             "insular endemic",
                             "leg development",
                             "activity time",
                             "substrate",
                             "diet",
                             "foraging mode",
                             "reproductive mode",
                             "family",
                             "iucn redlist assessment",
                             "iucn population trend",
                             "extant/extinct",
                             ]

# A list of the features which were kept after imputing, i.e. have enough non-null values
# This is used for normalisation later
numerical_features_kept = []

# Empty dataframe to add to
imputed_rep_traits = pd.DataFrame()

# Columns less than this proportion of values intact will be ignored
NON_MISSING_THRESHOLD = 0.70

# For every numerical feature
for feature_name in numerical_features:
    print(f"{GetCountNumericalVals(rep_traits, feature_name)}\t numerical values in '{feature_name}'")

    # If more than a specified amount of the data is not null, impute and then add it to a new dataset
    if GetCountNumericalVals(rep_traits, feature_name) > (len(rep_traits.index) * NON_MISSING_THRESHOLD):
        imputed_rep_traits[feature_name] = ImputeMedian(rep_traits, feature_name).reshape(-1, -1)
        numerical_features_kept.append(feature_name)

# For every categorical / boolean feature
for feature_name in categorical_bool_features:
    print(f"{GetCountNotNullVals(rep_traits, feature_name)}\t non-null categorical values in '{feature_name}'")

    # If more than a specified amount of the data is not null, impute and then add it to a new dataset
    if GetCountNotNullVals(rep_traits, feature_name) > (len(rep_traits.index) * NON_MISSING_THRESHOLD):
        imputed_rep_traits[feature_name] = ImputeMode(rep_traits, feature_name).reshape(-1, -1)
print("\n")
print(f"Number of rows in imputed dataframe: {len(imputed_rep_traits.index)}")

```

Normalise the numeric columns

```
# Normalise all of the values that are numerical in our imputed dataframe
min_max_scaler = preprocessing.MinMaxScaler()

# For each feature, normalise it and replace the original with the normalised
for feature_name in numerical_features_kept:
    column_values = imputed_rep_traits[feature_name].values.reshape(-1, 1)
    column_values_scaled = min_max_scaler.fit_transform(column_values)
    imputed_rep_traits[feature_name] = pd.DataFrame(column_values_scaled)

# Check the output since being normalised
imputed_rep_traits.head()
```

Working with boolean values

Because the SimpleImputer doesn't support booleans, these will be converted to bools after the imputation

```
# Map to boolean, then convert column type
imputed_rep_traits['insular endemic'] = imputed_rep_traits['insular endemic'].map({'yes': True, 'no': False})
imputed_rep_traits["insular endemic"] = imputed_rep_traits["insular endemic"].astype(bool)

imputed_rep_traits["extinct?"] = imputed_rep_traits["extant/extinct"].map({'extant': False, 'extinct': True})
imputed_rep_traits.drop(["extant/extinct"], axis=1, inplace=True)
imputed_rep_traits["extinct?"] = imputed_rep_traits["extinct?"].astype(bool)
```

Decision Tree Classification: Predicting Genus and Family

First let's classify for reptiles' genus

A genus is a taxonomic term for classifying living things. It is more general than species, but more specific than family

The first step is to make a new dataframe with only what is used for this classification

```
genus_class_df = pd.concat([imputed_rep_traits["genus"],
                           imputed_rep_traits["leg development"],
                           imputed_rep_traits["activity time"],
                           imputed_rep_traits["substrate"],
                           imputed_rep_traits["maximum svl"],
                           imputed_rep_traits["latitude centroid"],
                           imputed_rep_traits["longitude centroid"],
                           ], axis=1)

# One hot encoding using pandas get_dummies. Despite the name, this does
# indeed get one hot encoded values, and not dummies
# Add the one hot encoded columns for a given original column to the dataframe
# ... and then drop the original

# List to one hot encode
one_hot_encode_features = ["leg development",
                           "activity time",
                           "substrate"]

# One hot encode every relevant column
for feature_name in one_hot_encode_features:
    genus_class_df = pd.concat([genus_class_df, pd.get_dummies(genus_class_df[feature_name],
                                                                prefix=feature_name)], axis = 1)
    genus_class_df.drop([feature_name],axis=1, inplace=True)

# Classify for the most common X genres.
# We don't do all of them because many only occur once or twice, so it would be hard to classify on
num_most_common = 15
most_common_genuses = genus_class_df['genus'].value_counts()[ :num_most_common].index.tolist()

# Only keep rows whose genus value is one of the top n specified
# i.e only keep rows which contain the most frequent n genus values
# NOTE: this is done so we only classify against classes which we have a decent amount of instances of
genus_class_df = genus_class_df[genus_class_df["genus"].isin(most_common_genuses)]

# Number of rows left after filtering data
print(f"Number of rows for selected genres: {len(genus_class_df.index)}")
Number of rows for selected genres: 2145
```

Training, fitting, predicting, and evaluating for reptile genus

```
# Get the attributes (all but genus), and the label (genus)
X = genus_class_df.drop("genus", axis = 1)
y = genus_class_df["genus"]

# Split into train and test, 80:20
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

# Fit to our training data, then predict

#classifier = tree.DecisionTreeClassifier(random_state = 0, max_leaf_nodes = 150)
#classifier = SGDClassifier(random_state = 0, penalty="l1", max_iter = 50)
#classifier = svm.SVC(random_state = 0, C = 1000)
classifier = RandomForestClassifier(random_state = 0, n_estimators = 100, max_features = 6,
max_depth = 13)

classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

actual_vs_pred = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

# Display a graphical confusion matrix, followed by a classification report
confusionMatrix = pd.crosstab(actual_vs_pred['Actual'], actual_vs_pred['Predicted'],
rownames=['Actual'], colnames=['Predicted'])

sn.set(font_scale = 0.8) # Label size
sn.heatmap(confusionMatrix, annot=True, annot_kws={"size": 6}, fmt="d") # Font size for numbers
plt.show()

print(classification_report(y_test, y_pred, zero_division="warn"))

print("\nOVERALL:")
print("Accuracy score: ", round(accuracy_score(y_test, y_pred), 2))
print("Precision score: ", round(precision_score(y_test, y_pred, average = "macro"), 2))
print("Recall score: ", round(recall_score(y_test, y_pred, average = "macro"), 2))
print("F1 score: ", round(f1_score(y_test, y_pred, average = "macro"), 2))
```

Show one of the Random Forest's Decision Trees

```
# The ampersand character is a special character in HTML, which is used by some libraries later on - so replace it
genus_class_df.columns = genus_class_df.columns.str.replace("&", " and ")

# Extract single tree from the random forest
estimator = classifier.estimators_[0]

dot_data = StringIO()

# Export using the feature names and the class values
export_graphviz(estimator, out_file=dot_data,
                 filled=True, rounded=True,
                 special_characters=True, feature_names = genus_class_df.drop("genus", axis =
1).columns,
                 class_names=genus_class_df['genus'].value_counts()[ :num_most_common].index.
tolist())

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('genus.png')
Image(graph.create_png())
```

ROC Curve and AUC Scores

Use a `OneVsRestClassifier`, passing it our `RandomForestClassifier`. Here we only use `predict()` to get class predictions, and not `predict_proba()`, as this produced unrealistically optimistic ROC curves, as discussed in the report.

```
X = genus_class_df.drop("genus", axis = 1)
y = genus_class_df["genus"]

# Binarize the output
y = label_binarize(y, classes=most_common_genuses)
n_classes = y.shape[1]

# Split training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=0)

#classifier = OneVsRestClassifier(tree.DecisionTreeClassifier(random_state = 0, max_leaf_no
des = 150))
#classifier = OneVsRestClassifier(SGDClassifier(random_state = 0, penalty="l1", max_iter =
50))
#classifier = OneVsRestClassifier(svm.SVC(random_state = 0, C = 1000))
classifier = OneVsRestClassifier(RandomForestClassifier(random_state = 0, n_estimators = 10
0, max_features = 6, max_depth = 13))

classifier.fit(X_train, y_train)

y_score = classifier.predict(X_test)
#y_score = cross_val_predict(classifier,X_test,y_test)

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot ROC curve
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC (AUC = {0:0.2f})'
         ''.format(roc_auc["micro"]))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label='ROC curve of class {0} (AUC = {1:0.2f})'
             ''.format(most_common_genuses[i], roc_auc[i]), linewidth=0.75)

plt.plot([0, 1], [0, 1], 'k--') # The dotted diagonal line

# Set limits of axes = make the height go slightly past 1 to account for the line thickness
plt.xlim([-0.0025, 1.0])
plt.ylim([0.0, 1.0075])

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Genus Multi-class ROC curve with AUC scores (OneVsRestClassivier)')
plt.legend(loc="lower right", prop={'size': 6})
plt.show()
```

Now let's classify for reptiles' family

Like a genus, a family is another way of classifying and ordering living things. A family is more general than a genus, but more specific than an order. Therefore, this will be predicting a more general classification than previously

```
family_class_df = pd.concat([imputed_rep_traits["family"],
                             imputed_rep_traits["leg development"],
                             imputed_rep_traits["activity time"],
                             imputed_rep_traits["substrate"],
                             imputed_rep_traits["maximum svl"],
                             imputed_rep_traits["latitude centroid"],
                             imputed_rep_traits["longitude centroid"],
                             ], axis=1)

# One hot encoding using pandas get_dummies. Despite the name, this does
# indeed get one hot encoded values, and not dummies
# Add the one hot encoded columns for a given original column to the dataframe
# ... and then drop the original

# List to one hot encode
one_hot_encode_features = ["leg development",
                           "activity time",
                           "substrate"]

# One hot encode every relevant column
for feature_name in one_hot_encode_features:
    family_class_df = pd.concat([family_class_df, pd.get_dummies(family_class_df[feature_name], prefix=feature_name)], axis = 1)
    family_class_df.drop([feature_name],axis=1, inplace=True)

num_most_common = 15
most_common_families = family_class_df['family'].value_counts()[:num_most_common].index.tolist()

# Only keep rows whose genus value is one of the top n specified
# i.e only keep rows which contain the most frequent n genus values
# NOTE: this is done so we only classify against classes which we have a decent amount of instances of
family_class_df = family_class_df[family_class_df["family"].isin(most_common_families)]

# Number of rows left after filtering data
print(f"Number of rows for selected genres: {len(family_class_df.index)}")
```

Training, fitting, predicting, and evaluating for reptile family

```
# Get the attributes (all but genus), and the label (genus)
X = family_class_df.drop("family", axis = 1)
y = family_class_df["family"]

# Split into train and test, 80:20
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

# Fit to our training data, then predict
#classifier = tree.DecisionTreeClassifier(random_state = 0, max_depth = 11)
#classifier = SGDClassifier(random_state = 0, penalty="l1", max_iter = 50)
#classifier = svm.SVC(random_state = 0, C = 1000)
classifier = RandomForestClassifier(random_state = 0, n_estimators = 100, max_depth = 16, max_features = 7)

classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

actual_vs_pred = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

# Display a graphical confusion matrix, followed by a classification report
confusionMatrix = pd.crosstab(actual_vs_pred['Actual'], actual_vs_pred['Predicted'],
                               rownames=['Actual'], colnames=['Predicted'])

sn.set(font_scale = 0.5) # Label size
sn.heatmap(confusionMatrix, annot=True, annot_kws={"size": 6}, fmt="d") # Font size for numbers
plt.show()

print(classification_report(y_test, y_pred, zero_division="warn"))

print("\nOVERALL:")
print("Accuracy score: ", round(accuracy_score(y_test, y_pred), 2))
print("Precision score: ", round(precision_score(y_test, y_pred, average = "macro"), 2))
print("Recall score: ", round(recall_score(y_test, y_pred, average = "macro"), 2))
print("F1 score: ", round(f1_score(y_test, y_pred, average = "macro"), 2))
```

Show one of the Random Forest's Decision Trees

```
# The ampersand character is a special character in HTML, which is used by some libraries later on - so replace it
family_class_df.columns = family_class_df.columns.str.replace("&", " and ")

# Extract single tree from the random forest
estimator = classifier.estimators_[0]

dot_data = StringIO()

# Export using the feature names and the class values
export_graphviz(estimator, out_file=dot_data,
                 filled=True, rounded=True,
                 special_characters=True, feature_names = family_class_df.drop("family", axis
= 1).columns,
                 class_names=family_class_df['family'].value_counts()[ :num_most_common].index.tolist())

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('family.png')
Image(graph.create_png())
```

```

X = family_class_df.drop("family", axis = 1)
y = family_class_df["family"]

# Binarize the output
y = label_binarize(y, classes=most_common_families)
n_classes = y.shape[1]

# Split training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=0)
#classifier = OneVsRestClassifier(svm.SVC(random_state = 0, C = 1000))
#classifier = OneVsRestClassifier(tree.DecisionTreeClassifier(random_state = 0, ma
x_depth = 11))
#classifier = OneVsRestClassifier(SGDClassifier(random_state = 0, penalty="l1", ma
x_iter = 50))
classifier = OneVsRestClassifier(RandomForestClassifier(random_state = 0, n_estima
tors = 100, max_features = 7, max_depth = 16))

classifier.fit(X_train, y_train)

y_score = classifier.predict(X_test)
#y_score = cross_val_predict(classifier,X_test,y_test)

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot ROC curve
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC (AUC = {0:0.2f})'
         ''.format(roc_auc["micro"]))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label='ROC curve of class {0} (AUC = {1:0.2f})'
             ''.format(most_common_families[i], roc_auc[i]),
            linewidth=0.75)

plt.plot([0, 1], [0, 1], 'k--') # The dotted diagonal line

# Set limits of axes = make the height go slightly past 1 to account for the line
thickness
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0075])

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Family Multi-class ROC curve with AUC scores')
plt.legend(loc="lower right", prop={'size': 6})
plt.show()

```

Regression

Perform linear and polynomial regression, as well as Random Forest regression, on a range of attributes with promising trends and correlations

```
regression_df = pd.concat([rep_traits["average svl"],
                             rep_traits["normalised svl range for mean"],
                             rep_traits["mean clutch size"],
                             rep_traits["mean tb"],
                             rep_traits["tb range"],
                             rep_traits["minimum mean tb"],
                             rep_traits["maximum mean tb"],
                             rep_traits["smallest clutch"],
                             rep_traits["largest clutch"]
                             ], axis=1)

# For each feature, normalise it and replace the original with the normalised
for feature_name in regression_df:
    column_values = regression_df[feature_name].values.reshape(-1, 1)
    column_values_scaled = min_max_scaler.fit_transform(column_values)
    regression_df[feature_name] = pd.DataFrame(column_values_scaled)

def PrepareRegressionData(dataframe_src, X_feature, y_feature):
    # Dependent and independent variables, get rid of null values
    filtered_df = dataframe_src[dataframe_src[X_feature].notnull()]
    filtered_df = filtered_df[filtered_df[y_feature].notnull()]

    X = filtered_df[X_feature].values.reshape(-1, 1)
    y = filtered_df[y_feature].values.reshape(-1, 1)

    # Split the data 80:20. train:test.
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

    # Sort the data - sort X, but then sort y based on X, so the x and y values remain intact.
    # This is mostly done for graphing later on
    testOrders = np.argsort(X_test.ravel())
    y_test = y_test[testOrders]
    X_test = X_test[testOrders]

    trainOrders = np.argsort(X_train.ravel())
    y_train = y_train[trainOrders]
    X_train = X_train[trainOrders]

    packed_test_train_data = {'X_train': X_train,
                              'y_train': y_train,
                              'X_test': X_test,
                              'y_test': y_test}

    return packed_test_train_data
```

Normalised SVL Range to Mean Clutch Size

Polynomial features, so compare different polynomial orders

```
# Data setup
test_train_data = PrepareRegressionData(regression_df, "normalised svl range for mean", "mean clutch size")

# Select the model
lin_reg = LinearRegression()

chartColours = ["red", "lime", "orange", "yellow", "pink", "blue"]

# Set up graph and plot the original data points (scatter)
plt.title("Normalised SVL Range for Mean to Mean Clutch Size", fontsize=10)
plt.xlabel("Normalised SVL Range for Mean", fontsize=8)
plt.ylabel('Mean Clutch Size', fontsize=8)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
sn.set(style='white', font_scale=1)
plt.scatter(test_train_data["X_train"], test_train_data["y_train"], color='black', s = 1.5,
alpha=0.45)
numberOfOrders = 6

# Try a few different orders
for orderValue in range(1, numberOfOrders + 1):

    poly_features = PolynomialFeatures(degree = orderValue, include_bias = False)
    X_poly = poly_features.fit_transform(test_train_data["X_train"])

    lin_reg.fit(X_poly, test_train_data["y_train"])
    #print(lin_reg.intercept_, lin_reg.coef_)

    y_pred = lin_reg.predict(poly_features.fit_transform(test_train_data["X_test"]))

    # Calculate RMSE and AME
    rmse = mean_squared_error(test_train_data["y_test"], y_pred, squared = False)
    ame = mean_absolute_error(test_train_data["y_test"], y_pred)

    plt.plot(test_train_data["X_test"], y_pred, color = chartColours[orderValue - 1],
             linewidth = 1.5, label = f"Linear - RMSE: {round(rmse, 4):.4f}, AME: {round(ame, 4):.4f}" if orderValue == 1
             else f"Order {orderValue} - RMSE: {round(rmse, 4):.4f}, AME: {round(ame, 4):.4f}")

plt.legend(loc="upper left", prop={'size': 7})
plt.rcParams.update({'font.size': 10})
plt.xticks(fontsize=7)
plt.yticks(fontsize=7)
plt.grid(color='black', alpha=0.1, linestyle='solid', linewidth = 0.5)

plt.show()
```

Normalised SVL Range to Mean Clutch Size

Warning: takes an eternity to run!

```
# Data setup
test_train_data = PrepareRegressionData(regression_df, "normalised svl range for mean", "mean clutch size")

# Select the models
lin_reg = LinearRegression()
rf_reg = RandomForestRegressor(random_state=0, max_depth = 2)
rf_reg_overfit = RandomForestRegressor(random_state=0, max_depth = 20)
svm_poly_reg = SVR(kernel="poly", degree = 4, tol = 0.03, C = 0.3, epsilon = 0.01)

# Set up graph and plot the original data points (scatter)
plt.title("Normalised SVL Range for Mean to Mean Clutch Size", fontsize=10)
plt.xlabel("Normalised SVL Range for Mean", fontsize=8)
plt.ylabel('Mean Clutch Size', fontsize=8)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
sns.set(style='white', font_scale=1)
plt.scatter(test_train_data["X_train"], test_train_data["y_train"], color='black', s = 1.5, alpha=0.45)
numberOfOrders = 4

# Now fit and predict for each classifier
# RF
rf_reg.fit(test_train_data["X_train"], test_train_data["y_train"])
y_pred_rf = rf_reg.predict(test_train_data["X_test"])

rf_rmse = mean_squared_error(test_train_data["y_test"], y_pred_rf, squared = False)
rf_ame = mean_absolute_error(test_train_data["y_test"], y_pred_rf)

# RF overfitting example
rf_reg_overfit.fit(test_train_data["X_train"], test_train_data["y_train"])
y_pred_rf_overfit = rf_reg_overfit.predict(test_train_data["X_test"])

rf_overfit_rmse = mean_squared_error(test_train_data["y_test"], y_pred_rf_overfit, squared = False)
rf_overfit_ame = mean_absolute_error(test_train_data["y_test"], y_pred_rf_overfit)

# Polynomial Regression using LinearRegression()
# Set up for polynomial regression
poly_features = PolynomialFeatures(degree = 4, include_bias = False)
X_poly = poly_features.fit_transform(test_train_data["X_train"])

lin_reg.fit(X_poly, test_train_data["y_train"])
y_pred_poly = lin_reg.predict(poly_features.fit_transform(test_train_data["X_test"]))

# Calculate RMSE and AME
poly_rmse = mean_squared_error(test_train_data["y_test"], y_pred, squared = False)
poly_ame = mean_absolute_error(test_train_data["y_test"], y_pred)

# SVM
svm_poly_reg.fit(test_train_data["X_train"], test_train_data["y_train"])
```

```

y_pred_svm = svm_poly_reg.predict(test_train_data["X_test"])

svm_rmse = mean_squared_error(test_train_data["y_test"], y_pred_svm, squared = False)
svm_ame = mean_absolute_error(test_train_data["y_test"], y_pred_svm)


# Now plot the results
plt.plot(test_train_data["X_test"], y_pred_rf, color = "red",
         linewidth = 1.5, label = f"RF Reg. (max_depth=2)
- RMSE: {round(rf_rmse, 4):.4f},   AME: {round(rf_ame, 4):.4f}")

plt.plot(test_train_data["X_test"], y_pred_rf_overfit, color = "orange",
         linewidth = 0.75, alpha=0.75, label = f"RF Overfitting. (max_depth=20)
- RMSE: {round(rf_overfit_rmse, 4):.4f},   AME: {round(rf_overfit_ame, 4):.4f}")

plt.plot(test_train_data["X_test"], y_pred_svm, color = "lime",
         linewidth = 1.5, label = f"SVR (poly, deg=4, tol=0.03, C=0.3, ε=0.01) - RMSE:
{round(svm_rmse, 4):.4f},   AME: {round(svm_ame, 4):.4f}")

plt.plot(test_train_data["X_test"], y_pred_poly, color = "blue",
         linewidth = 1.5, label = f"Poly (deg=4)
- RMSE: {round(poly_rmse, 4):.4f},   AME: {round(poly_ame, 4):.4f}")

plt.legend(loc="upper left", prop={'size': 7})
plt.rcParams.update({'font.size': 10})
plt.xticks(fontsize=7)
plt.yticks(fontsize=7)
plt.grid(color='black', alpha=0.1, linestyle='solid', linewidth = 0.5)

plt.show()

```

Normalised SVL Range to Average SVL

```
# Data setup
test_train_data = PrepareRegressionData(regression_df, "normalised svl range for mean", "average svl")

# Select the models
lin_reg = LinearRegression()
rf_reg = RandomForestRegressor(random_state=0, max_depth = 2)

# Set up graph and plot the original data points (scatter)
plt.title("Normalised SVL Range for Mean to Average SVL", fontsize=10)
plt.xlabel("Normalised SVL Range for Mean", fontsize=8)
plt.ylabel('Average SVL', fontsize=8)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
sn.set(style='white', font_scale=1)
plt.scatter(test_train_data["X_train"], test_train_data["y_train"], color='black', s = 1.5,
alpha=0.45)

chartColours = ["red", "lime", "orange", "yellow", "pink", "blue"]

numberOfOrders = 4

# Try a few different orders
for orderValue in range(1, numberOfOrders + 1):

    poly_features = PolynomialFeatures(degree = orderValue, include_bias = False)
    X_poly = poly_features.fit_transform(test_train_data["X_train"])

    lin_reg.fit(X_poly, test_train_data["y_train"])
    #print(lin_reg.intercept_, lin_reg.coef_)

    y_pred = lin_reg.predict(poly_features.fit_transform(test_train_data["X_test"]))

    # Calculate RMSE and AME
    rmse = mean_squared_error(test_train_data["y_test"], y_pred, squared = False)
    ame = mean_absolute_error(test_train_data["y_test"], y_pred)

    plt.plot(test_train_data["X_test"], y_pred, color = chartColours[orderValue - 1],
             linewidth = 1.5, label = f"Linear - RMSE: {round(rmse, 4):.4f}, AME: {round(ame, 4):.4f}" if orderValue == 1
             else f"Order {orderValue} - RMSE: {round(rmse, 4):.4f}, AME: {round(ame, 4):.4f}")

    rf_reg.fit(test_train_data["X_train"], test_train_data["y_train"])
    y_pred_rf = rf_reg.predict(test_train_data["X_test"])

    rf_rmse = mean_squared_error(test_train_data["y_test"], y_pred_rf, squared = False)
    rf_ame = mean_absolute_error(test_train_data["y_test"], y_pred_rf)

    plt.plot(test_train_data["X_test"], y_pred_rf, color = "blue",
             linewidth = 1.5, label = f"RF Reg (max_depth=2) - RMSE: {round(rmse, 4):.4f}, AME: {round(ame, 4):.4f}")

plt.legend(loc="upper left", prop={'size': 7})
plt.rcParams.update({'font.size': 10})
plt.xticks(fontsize=7)
plt.yticks(fontsize=7)
plt.grid(color='black', alpha=0.1, linestyle='solid', linewidth = 0.5)

plt.show()
```

Mean and Largest Clutch Size

These correlate very highly and linearly, so use only linear regression here

```
# Data setup
test_train_data = PrepareRegressionData(regression_df, "mean clutch size", "largest clutch"
)

# Select the model
lin_reg_svm = LinearSVR()
lin_reg = LinearRegression(fit_intercept = False)

lin_reg_svm.fit(test_train_data["X_train"], test_train_data["y_train"])
lin_reg.fit(test_train_data["X_train"], test_train_data["y_train"])

y_pred_svm = lin_reg_svm.predict(test_train_data["X_test"])
y_pred_lin = lin_reg.predict(test_train_data["X_test"])

# Calculate RMSE and AME
rmse_svm = mean_squared_error(test_train_data["y_test"], y_pred_svm, squared = False)
ame_svm = mean_absolute_error(test_train_data["y_test"], y_pred_svm)

rmse_lin = mean_squared_error(test_train_data["y_test"], y_pred_lin, squared = False)
ame_lin = mean_absolute_error(test_train_data["y_test"], y_pred_lin)
plt.title('Mean Vs Largest Clutch Sizes', fontsize=10)
plt.ylabel('Largest Clutch Size', fontsize=8)
plt.xlabel('Mean Clutch Size', fontsize=8)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
sn.set(style='white', font_scale=1)
plt.scatter(test_train_data["X_train"], test_train_data["y_train"], color='black', s = 3, alpha = 0.45)

plt.plot(test_train_data["X_test"], y_pred_lin, color = "red",
         linewidth = 1.5, label = f"LinearRegression(), fit_intercept=False - RMSE: {
round(rmse_lin, 4):.4f}, AME: {round(ame_lin, 4):.4f}")

plt.plot(test_train_data["X_test"], y_pred_svm, color = "blue",
         linewidth = 1.5, label = f"LinearSVR()
- RMSE: {round(rmse_svm, 4):.4f}, AME: {round(ame_svm, 4):.4f}")

plt.legend(loc="upper left", prop={'size': 6})

plt.rcParams.update({'font.size': 10})
plt.xticks(fontsize=7)
plt.yticks(fontsize=7)
plt.grid(color='black', alpha=0.1, linestyle='solid', linewidth = 0.5)
plt.show()
```

Average SVL to Mean Clutch Size

```
# Data setup
test_train_data = PrepareRegressionData(regression_df, "average svl", "mean clutch size")

# Select the model
lin_reg = LinearRegression(fit_intercept = False)
rf_reg = RandomForestRegressor(random_state=0, max_depth = 2)
lin_reg_svm = LinearSVR(epsilon=0.01, fit_intercept = False)

plt.title('Average SVL to Mean Clutch Size', fontsize=10)
plt.ylabel('Mean Clutch Size', fontsize=8)
plt.xlabel('Average SVL', fontsize=8)
plt.xlim([0.0, 1.0])
plt.ylim([-0.01, 1.0])
sns.set(style='white', font_scale=1)
plt.scatter(test_train_data["X_train"], test_train_data["y_train"], color='black', s = 3, alpha = 0.035)

# Linear Regression
numberOfOrders = 3
# Try a few different orders
for orderValue in range(1, numberOfOrders + 1):

    poly_features = PolynomialFeatures(degree = orderValue, include_bias = False)
    X_poly = poly_features.fit_transform(test_train_data["X_train"])

    lin_reg.fit(X_poly, test_train_data["y_train"])
    #print(lin_reg.intercept_, lin_reg.coef_)

    y_pred = lin_reg.predict(poly_features.fit_transform(test_train_data["X_test"]))

    # Calculate RMSE and AME
    rmse = mean_squared_error(test_train_data["y_test"], y_pred, squared = False)
    ame = mean_absolute_error(test_train_data["y_test"], y_pred)

    plt.plot(test_train_data["X_test"], y_pred, color = chartColours[orderValue - 1],
             linewidth = 1.5, label = f"Linear
- RMSE: {round(rmse, 4):.4f},    AME: {round(ame, 4):.4f}" if orderValue == 1
             else f"Order {orderValue}"
- RMSE: {round(rmse, 4):.4f},    AME: {round(ame, 4):.4f}")

# Random Forest
rf_reg.fit(test_train_data["X_train"], test_train_data["y_train"])
y_pred_rf = rf_reg.predict(test_train_data["X_test"])

rf_rmse = mean_squared_error(test_train_data["y_test"], y_pred_rf, squared = False)
rf_ame = mean_absolute_error(test_train_data["y_test"], y_pred_rf)

# SVM
lin_reg_svm.fit(test_train_data["X_train"], test_train_data["y_train"])
y_pred_svr = lin_reg_svm.predict(test_train_data["X_test"])

svr_rmse = mean_squared_error(test_train_data["y_test"], y_pred_svr, squared = False)
svr_ame = mean_absolute_error(test_train_data["y_test"], y_pred_svr)
```

```

plt.plot(test_train_data["X_test"], y_pred_rf, color = "blue",
         linewidth = 1.5, label = f"RF Reg (max_depth=2) - RMSE: {round(rf_rmse, 4):.4f}, AME: {round(rf_ame, 4):.4f}")

plt.plot(test_train_data["X_test"], y_pred_svr, color = "yellow",
         linewidth = 1.5, label = f"Lin. SVR (fit_intercept=False, ε=0.01) - RMSE: {round(svr_rmse, 4):.4f}, AME: {round(svr_ame, 4):.4f}")

plt.legend(loc="upper left", prop={'size': 6})

plt.rcParams.update({'font.size': 10})
plt.xticks(fontsize=7)
plt.yticks(fontsize=7)
plt.grid(color='black', alpha=0.1, linestyle='solid', linewidth = 0.5)
plt.show()

```

Mean Body Temperature to Body Temperature Range

```
# Data setup
test_train_data = PrepareRegressionData(regression_df, "tb_range", "mean tb")

# Select the model
lin_reg = LinearRegression()
rf_reg = RandomForestRegressor(random_state=0, max_depth = 2)
lin_reg_svm = LinearSVR(random_state=0)

lin_reg.fit(test_train_data["X_train"], test_train_data["y_train"])
rf_reg.fit(test_train_data["X_train"], test_train_data["y_train"])
lin_reg_svm.fit(test_train_data["X_train"], test_train_data["y_train"])

y_pred_lin = lin_reg.predict(test_train_data["X_test"])
y_pred_rf = rf_reg.predict(test_train_data["X_test"])
y_pred_svm = lin_reg_svm.predict(test_train_data["X_test"])

rmse_lin = mean_squared_error(test_train_data["y_test"], y_pred_lin, squared = False)
ame_lin = mean_absolute_error(test_train_data["y_test"], y_pred_lin)

rf_rmse = mean_squared_error(test_train_data["y_test"], y_pred_rf, squared = False)
rf_ame = mean_absolute_error(test_train_data["y_test"], y_pred_rf)

svm_rmse = mean_squared_error(test_train_data["y_test"], y_pred_svm, squared = False)
svm_ame = mean_absolute_error(test_train_data["y_test"], y_pred_svm)

plt.title('Body Temp. Range to Mean', fontsize=10)
plt.ylabel('Mean Body Temp.', fontsize=8)
plt.xlabel('Body Temp. Range', fontsize=8)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
sns.set(style='white', font_scale=1)
plt.scatter(test_train_data["X_train"], test_train_data["y_train"], color='black', s = 4, alpha = 0.45)

plt.plot(test_train_data["X_test"], y_pred_lin, color = "red",
         linewidth = 1.5, label = f"LinearRegression() - RMSE: {round(rmse_lin, 4):.4f}, AME: {round(ame_lin, 4):.4f}")

plt.plot(test_train_data["X_test"], y_pred_rf, color = "orange",
         linewidth = 1.5, label = f"RF (max_depth=2) - RMSE: {round(rf_rmse, 4):.4f}, AME: {round(rf_ame, 4):.4f}")

plt.plot(test_train_data["X_test"], y_pred_svm, color = "blue",
         linewidth = 1.5, label = f"Lin. SVR. - RMSE: {round(svm_rmse, 4):.4f}, AME: {round(svm_ame, 4):.4f}")

plt.legend(loc="upper left", prop={'size': 6})

plt.rcParams.update({'font.size': 10})
plt.xticks(fontsize=7)
plt.yticks(fontsize=7)
plt.grid(color='black', alpha=0.1, linestyle='solid', linewidth = 0.5)
plt.show()

# Data setup
test_train_data = PrepareRegressionData(regression_df, "mean tb", "maximum mean tb")

# Select the model
lin_reg = LinearRegression()
```

```

rf_reg = RandomForestRegressor(random_state=0, max_depth = 3)
lin_reg_svm = LinearSVR(random_state=0, epsilon = 0.01)

lin_reg.fit(test_train_data["X_train"], test_train_data["y_train"])
rf_reg.fit(test_train_data["X_train"], test_train_data["y_train"])
lin_reg_svm.fit(test_train_data["X_train"], test_train_data["y_train"])

y_pred_lin = lin_reg.predict(test_train_data["X_test"])
y_pred_rf = rf_reg.predict(test_train_data["X_test"])
y_pred_svm = lin_reg_svm.predict(test_train_data["X_test"])

rmse_lin = mean_squared_error(test_train_data["y_test"], y_pred_lin, squared = False)
ame_lin = mean_absolute_error(test_train_data["y_test"], y_pred_lin)

rf_rmse = mean_squared_error(test_train_data["y_test"], y_pred_rf, squared = False)
rf_ame = mean_absolute_error(test_train_data["y_test"], y_pred_rf)

svm_rmse = mean_squared_error(test_train_data["y_test"], y_pred_svm, squared = False)
svm_ame = mean_absolute_error(test_train_data["y_test"], y_pred_svm)

plt.title('Mean to Maximum Body Temp.', fontsize=10)
plt.ylabel('Maximum Body Temp.', fontsize=8)
plt.xlabel('Mean Body Temp.', fontsize=8)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
sns.set(style='white', font_scale=1)
plt.scatter(test_train_data["X_train"], test_train_data["y_train"], color='black', s = 3, alpha = 0.45)

plt.plot(test_train_data["X_test"], y_pred_lin, color = "red",
         linewidth = 1.5, label = f"LinearRegression() - RMSE: {round(rmse_lin, 4):.4f}, AME: {round(ame_lin, 4):.4f}")

plt.plot(test_train_data["X_test"], y_pred_svm, color = "orange",
         linewidth = 1.5, label = f"Lin. SVR (ε=0.01) - RMSE: {round(svm_rmse, 4):.4f}, AME: {round(svm_ame, 4):.4f}")

plt.plot(test_train_data["X_test"], y_pred_rf, color = "blue",
         linewidth = 1.5, label = f"RF (max_depth=3) - RMSE: {round(rf_rmse, 4):.4f}, AME: {round(rf_ame, 4):.4f}")

plt.legend(loc="upper left", prop={'size': 6})

plt.rcParams.update({'font.size': 10})
plt.xticks(fontsize=7)
plt.yticks(fontsize=7)
plt.grid(color='black', alpha=0.1, linestyle='solid', linewidth = 0.5)
plt.show()

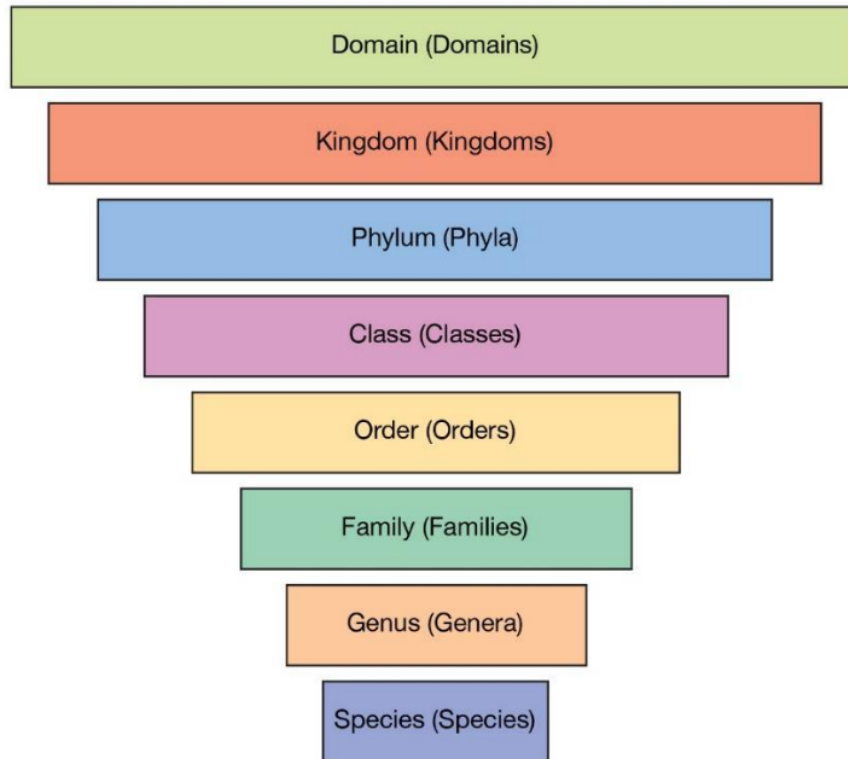
```

Appendix B

Feature Name	Data Type
Binomial	Categorical
Genus	Categorical
Epithet	Categorical
Valid (reptile database, February 2018)	Boolean (yes / no)
Year of description	Integer
Country described from	Categorical
Main biogeographic Realm	Categorical
Geographic Range	Categorical, list
Known only from the only type?	Boolean (yes / no)
Latitude centroid (from Roll et al. 2017)	Decimal
Longitude centroid (from Roll et al. 2017)	Decimal
Insular endemic	Boolean (yes / no)
Maximum SVL	Decimal
Female SVL	Decimal
Hatchling/neonate SVL	Decimal
Leg development	Categorical
Mass equation (Feldman et al. 2016 unless stated)	Categorical
Intercept	Decimal
Slope	Decimal
Activity time	Categorical
Activity time: comments	Text (references)
Substrate	Categorical
Substrate: comments	Text (references)
Diet	Categorical
Diet: comments	Text (references)
Foraging mode	Categorical
Foraging mode (comments)	Text (references)
Reproductive mode	Categorical
Clutch size	Text (references)
Smallest clutch	Integer
Largest clutch	Integer
Smallest mean clutch size	Decimal
Largest mean clutch size	Decimal
Breeding age (months)	Text (references)
Youngest age at first breeding (months)	Integer
Oldest age at first breeding (months)	Integer
Mean body temperature of active animals in the wild	Text (references)
Minimum mean Tb	Decimal
Maximum mean Tb	Decimal
Family	Categorical
Phylogeny	Text (references)
Mhylogenetic data	Text (references)
IUCN redlist assessment	Categorical
IUCN population trend	Categorical
Extant/Extinct	Categorical
Remarks	Text (references)
References (Biology: all columns except M, N and O).	Text (references)
References: SVL of unsexed individuals, neonates and hatchlings	Text (references)
References: SVL of females	Text (references)
References: SVL of males	Text (references)

Appendix C

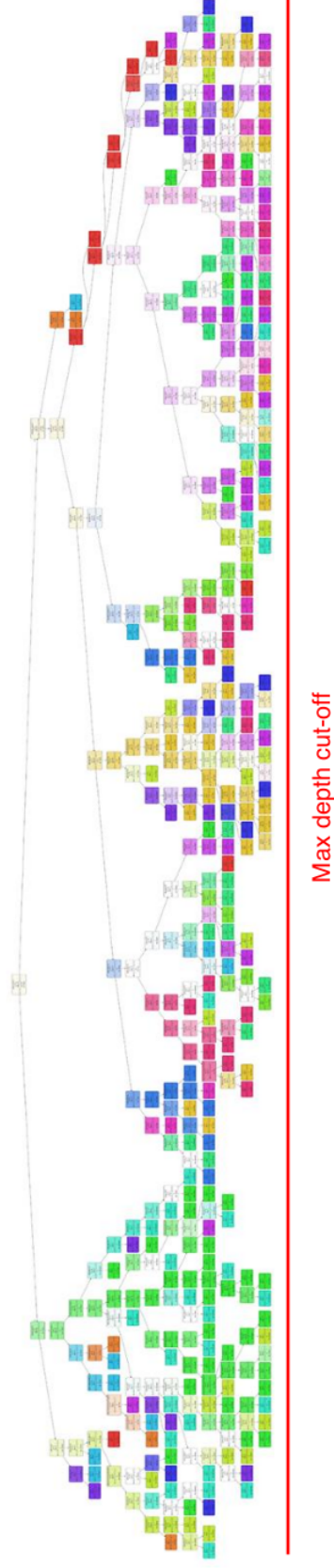
How animals are classified



© 2015 Encyclopædia Britannica, Inc.

Appendix D

Unfortunately, the complexity of the tree is too high for leaf text to be legible, and so the tree was included for illustration purpose only.



AHooper ML Assignment 1 FINAL

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

90/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47

PAGE 48

PAGE 49

PAGE 50

PAGE 51

PAGE 52

PAGE 53

PAGE 54

PAGE 55

PAGE 56

PAGE 57

PAGE 58

PAGE 59

PAGE 60

PAGE 61

PAGE 62

PAGE 63

GRADING FORM: SMU GENERIC_GJ

AUSTIN HOOPER

90

RESEARCH-INFORMED LITERATURE



Some excellent work here informed by a wide range of good references.

8.5

KNOWLEDGE AND UNDERSTANDING



A thorough understanding of the principles involved in cleaning data and applying classification and regression algorithms.

18

ANALYSIS



A wide range of classifiers evaluated. Perhaps you could have reduced the number of classifiers and concentrated on just a few, however the results here are excellent. You may think about one or two tables towards the end that bring together the results for easy comparison. PCA would have been helpful but nevertheless some reasonable choices of features identified using alternative methods.

32

PRACTICAL APPLICATION AND DEP.



Excellent, thorough, informed with appropriate results. A lot of work has gone into this - Well done. Comment in general good.

27.5

SKILLS FOR PROFESSIONAL PRAC.



Good use of references. References appropriate of a high quality generally.

4