Comp7703 Report

Tsung-Tse Tu 47801873

Abstract

This report addresses the challenge of predicting species classification using a dataset derived from biological samples. The approach combined PCA for feature reduction and Random Forest for classification, emphasizing the ability to handle high-dimensional data and interpret complex models. The findings suggest that while initial model performance was moderate, iterative tuning and refinement of the model parameters led to significant improvements in predictive accuracy.

Table of Contents

Data Preprocessing

1. **Initial Dataset Information:**

   - The dataset contains 1731 entries with 20 columns and no null values.

```
RangeIndex: 1731 entries, 0 to 1730
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Species        1731 non-null   object
 1   Population     1731 non-null   object
 2   Latitude       1731 non-null   float64
 3   Longitude      1731 non-null   float64
 4   Year_start     1731 non-null   int64
 5   Year_end       1731 non-null   int64
 6   Temperature    1731 non-null   int64
 7   Vial           1731 non-null   int64
 8   Replicate      1731 non-null   int64
 9   Sex            1731 non-null   object
 10  Thorax_length  1731 non-null   object
 11  l2             1731 non-null   float64
 12  l3p            1731 non-null   float64
 13  l3d            1731 non-null   float64
 14  lpd            1731 non-null   float64
 15  l3             1731 non-null   float64
 16  w1             1731 non-null   float64
 17  w2             1731 non-null   float64
 18  w3             1731 non-null   float64
 19  wing_loading   1731 non-null   object
dtypes: float64(10), int64(5), object(5)
```

   - The columns **Thorax_length** and **wing_loading** are of type '*object*', which is faulty considering their first few rows are numeric values.

	Thorax_length	wing_loading
0	1.238	1.914
1	1.113	1.928
2	1.215	1.908
3	1.123	1.860
4	1.218	1.886

2. **Inspection of Unique Values:**

   - Some values in **Thorax_length** and **wing_loading** are not numeric, causing them to be treated as *objects*.

3. **Conversion to Numeric:**

   - Both **Thorax_length** and **wing_loading** are converted to numeric types, with non-numeric values coerced to '*NAN*', and have type '*float64*'**.**

4. **Drop unnecessary features:**

   - All values of **Year_start** and **Year_end** share the same value '*1994*', which doesn't give any insights when using a single dataset.

	Latitude	Longitude	Year_start	Year_end
count	1731.000000	1731.000000	1731.0	1731.0
mean	-24.794910	150.821693	1994.0	1994.0
std	1.958099	1.220711	0.0	0.0
min	-27.680000	148.850000	1994.0	1994.0
25%	-25.520000	150.170000	1994.0	1994.0
50%	-25.200000	151.170000	1994.0	1994.0
75%	-23.770000	151.450000	1994.0	1994.0
max	-21.770000	152.450000	1994.0	1994.0

5. **Check for duplicates:**

```
# Check for duplicate rows
duplicate_rows = data.duplicated()
print(f"Number of duplicate rows: {duplicate_rows.sum()}")

# No duplicates
```

```
Number of duplicate rows: 0
```

6. **Inspect all numeric type columns:**

   - **Latitude** and **Longitude** have limited unique values, consistent with a few distinct sampling locations. After confirming, there are 5 unique pairs of these two features, so I merged them into a new column called **Location.**

Latitude	Longitude
-25.52	151.45
-23.77	150.17
-27.68	152.45
-21.77	148.85
-25.20	151.17

```
['Location (-25.52, 151.45)' 'Location (-23.77, 150.17)'
 'Location (-27.68, 152.45)' 'Location (-21.77, 148.85)'
 'Location (-25.2, 151.17)']
```

   - **Temperature** was used as a treatment in the experimental design, where flies were reared at three distinct temperature settings.
   - **Vial** as noted in the related work, the vial number represents different experimental units or groups in which different traits were measured.
   - **Replicate** refers to the repetition of experiments to ensure reliability and statistical validity according to related work.

- I changed the above numeric features' type to 'category' because they contain limited unique value and to reduce memory usage and speed up operations.

```
Unique values in 'Latitude':
[-25.52 -23.77 -27.68 -21.77 -25.2 ]

Unique values in 'Longitude':
[151.45 150.17 152.45 148.85 151.17]

Unique values in 'Temperature':
[20 25 30]

Unique values in 'Vial':
[ 1  2  3  4  5  6  7  8  9 10]

Unique values in 'Replicate':
[1 2 3]
```

- Other numeric values:

  Some rows consisted of zero values, which from the pattern observed are unlikely from the nature of the features. We changed them to NAN to deal with them altogether.

```
     Replicate     Sex  Thorax_length   l2  l3p  l3d  lpd   l3   w1     w2  \
61           1  female          1.106  0.0  0.6  0.0  0.0  0.0  0.0  1.252
698          3  female          1.151  0.0  0.0  0.0  0.0  0.0  0.0  0.000

      w3  wing_loading
61   0.0           0.0
698  0.0           0.0
```

7. **Remove rows with NAN**

   Considering in 1731 entries only 3 of them (rows 61, 253, 698) have NAN, we simply remove them to minimizes the risk of introducing bias through imputation.

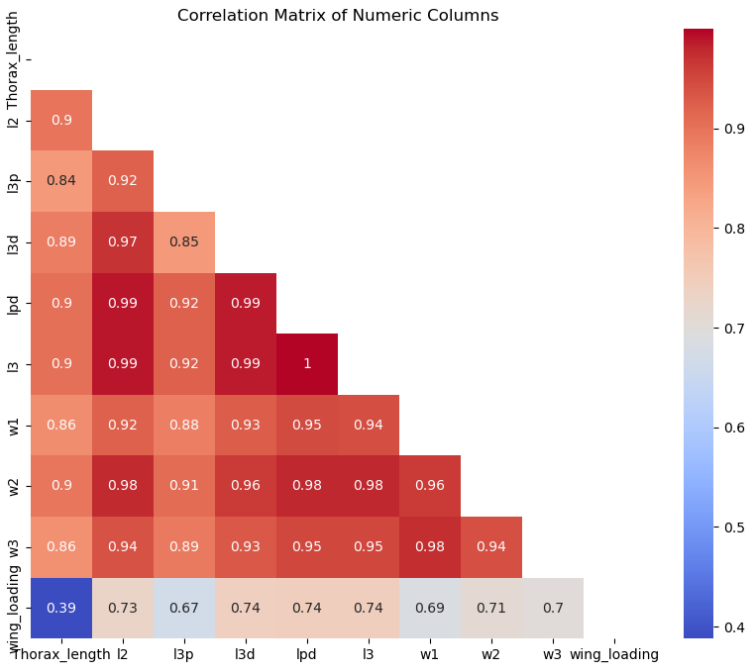
8. Check Categorical data (Dtype= category)

```
Unique values in Species:
Species
D._buzzatii     891
D._aldrichi     837
Name: count, dtype: int64


Unique values in Population:
Population
Gogango_Creek     353
Grandchester      349
Wahruna           345
Oxford_Downs      341
Binjour           340
Name: count, dtype: int64


Unique values in Temperature:
Temperature
25     579
20     577
30     572
Name: count, dtype: int64
```

```
Unique values in Vial:
Vial
5     177
8     177
6     175
7     173
2     172
4     172
9     172
10    172
1     170
3     168
Name: count, dtype: int64


Unique values in Replicate:
Replicate
1    599
2    586
3    543
Name: count, dtype: int64


Unique values in Sex:
Sex
male      872
female    856
Name: count, dtype: int64


Unique values in Location:
Location
Location (-23.77, 150.17)    353
Location (-27.68, 152.45)    349
Location (-25.2, 151.17)     345
Location (-21.77, 148.85)    341
Location (-25.52, 151.45)    340
Name: count, dtype: int64
```

Everything seems to be fine.

9. Correlation map (numeric features)



Such high degrees of correlation suggest that these variables contain overlapping information, which can lead to multicollinearity when these variables are used in predictive models.

Given these reasons, I think performing PCA is a great idea.

10. Outlier evaluation before PCA

Before I implement PCA, we must consider if outliers would be a problem. I used IQR to identify any potential outliers first. I compared the original dataset and the outlier-deleted dataset to examine the impact. We can see that **l3p** and **wing_loading** have a relatively minimal impact on the standard deviations and mean after deletion. Since the data shows small changes even after outliers are removed, using **StandardScaler** would be appropriate for most of the features. This should not lead to misrepresentations in the scaled data.

	Feature	Original Mean	Cleaned Mean	Original Std	Cleaned Std
0	Latitude	-24.794850	-24.794850	1.959542	1.959542
1	Longitude	150.821505	150.821505	1.221548	1.221548
2	Temperature	24.985532	24.985532	4.078318	4.078318
3	Vial	5.518519	5.518519	2.863363	2.863363
4	Replicate	1.967593	1.967593	0.812534	0.812534
5	Thorax_length	1.126501	1.126501	0.065632	0.065632
6	l2	1.725973	1.725973	0.154921	0.154921
7	l3p	0.586193	0.586102	0.051768	0.051647
8	l3d	1.457535	1.457535	0.118176	0.118176
9	lpd	2.043562	2.043562	0.164270	0.164270
10	l3	2.042683	2.042683	0.164430	0.164430
11	w1	0.915105	0.915105	0.067386	0.067386
12	w2	1.252947	1.252947	0.102529	0.102529
13	w3	1.039501	1.039501	0.082481	0.082481
14	wing_loading	1.811679	1.811835	0.068179	0.067536

11. Implementation of PCA

(Step 1): Standardize the Data

Transforms each numeric feature to have zero mean and unit variance. This ensures that each feature contributes equally to the analysis and prevents features with larger scales from dominating.

```
# Step 1: Standardize the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data[remain_numeric_columns])
```

(Step 2): Apply PCA

Setting **n_components** to 0.95 tells PCA to choose the minimum number of principal components such that 95% of the variance is retained, which is enough content. By doing so, PCA reduces dimensionality by finding new axes (principal components) that maximize variance and are orthogonal (independent) from each other. The **fit_transform** of the PCA class computes the principal components and uses them to transform the data into a new coordinate system.

```
# Step 2: Apply PCA
pca = PCA(n_components=0.95)  # Keep 95% of the variance
principal_components = pca.fit_transform(data_scaled)
```

(Step 3): Showcase the new data

The first array shows the percentage of the variance captured in the data. We can see that the high percentage **PC1** captured has a significant part of the information from the original dataset. By using 2 principal components, we retained nearly 96% (0.9619) of the information presented in the original dataset, confirming that most of the significant variability in the dataset has been preserved.

```
Variance explained by each component: [0.89884037 0.06313771]
Total variance explained: 0.9619780712732408
        PC1       PC2
0 -5.719656  0.107209
1 -1.685873  1.487611
2 -4.605792  0.284268
3 -0.881411  0.575929
4 -4.461361 -0.024122
  Component  Explained Variance  Cumulative
0       PC1            0.898840    0.898840
1       PC2            0.063138    0.961978
```

12. Final dataset

After the PCA process, combine **PC1** and **PC2** with the categorical columns, and use one-hot encoding since most of the categories don't have meaningful orders. Next, we will work on the analysis part.

Location_Location (-27.68, 152.45)	Species_D._aldrichi	Species_D._buzzatii	Vial_1	Vial_2	Vial_3	Vial_4	Vial_5
0	1	0	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	0	0	1	0	0	0
...	...	...	...	...	...	...	...
0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0

Data analysis

1. **Why random forest for classification?**

- **Robustness to Overfitting**

Given the reduced dimensions from PCA and the expanded feature set from one-hot encoded categorical variables, there is the risk of overfitting. Random Forest mitigates this risk by constructing multiple decision trees and averaging their outputs. This approach not only helps in reducing overfitting but also enhances the generalizability of the model.

- **Feature Importance**

With the inclusion of principal components and various categorical variables, identifying which features significantly influence your model's predictions is crucial. Random Forest offers an in-built mechanism to evaluate feature importance, which is particularly useful for detecting the relative impact of PCA components versus categorical variables.

- **Capability to Model Non-linear Relationships**

The interactions between the different types of features in your dataset, especially between PCA components and categorical variables, are likely complex and non-linear. Random Forest can capture intricate relationships, thanks to its ability to construct multiple decision boundaries.

- **No Need for Feature Scaling**

While PCA components result from scaled data, combining them with binary categorical data causes a mixture of scales. Random Forest operates effectively regardless of the scale of input features, excluding the need for additional scaling steps.

2. First experiment (Baseline)

I first created a single **species** column from one-hot encoded species data simplified the dataset and prepared it for classification tasks. The **np.select** function assigns a species name based on which species column has a value of 1, it manages scenarios where multiple one-hot encoded columns represent different species. The use of default='Other' ensures that any data point not fitting the predefined conditions is classified as 'Other,' to deal with any unexpected or missing data.

```
# Create a single 'species' column from one-hot encoded species data
conditions = [
    (final_df['Species_D._aldrichi'] == 1),
    (final_df['Species_D._buzzatii'] == 1)
]
choices = ['D._aldrichi', 'D._buzzatii']
final_df['species'] = np.select(conditions, choices, default='Other')
```

Once the unified species column is created, the original one-hot encoded species columns are removed from the dataset. The features (X) and the target (y) are then defined, with 'species' set as the target and the remaining dataset as the features. The dataset is then split into training and testing subsets, with 30% reserved for testing to evaluate the model's performance on unseen data. The **RandomForestClassifier** is then initialized with 100 trees and trained on the training data. After training, the model is used to make predictions on the test set, and the outcomes are assessed using accuracy scores and detailed classification reports. These reports break down the model's precision, recall, and F1-score by class, providing insights into how effectively the model is classifying each species.

```python
# Set 'species' as the target
y = final_df['species']
X = final_df.drop('species', axis=1)  # Exclude the target variable from the features

# Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=66)

# Initialize and train the RandomForest classifier
clf = RandomForestClassifier(n_estimators=100, random_state=66)
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Evaluate the model's performance
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

The model achieves an accuracy of approximately 51.63%, which suggests moderate performance. This level indicates that, on average, the model correctly predicts the species slightly more than half the time. Considering the complexity and variability in biological data, this performance serves as a baseline, indicating that <u>there is significant room for improvement</u>. The classification report reveals that **D._buzzatii** is somewhat easier for the model to predict than **D._aldrichi**, as evidenced by its slightly higher F1 score.

```
Accuracy: 0.5163776493256262
Classification Report:
                precision    recall  f1-score   support

  D._aldrichi       0.51      0.46      0.48       255
  D._buzzatii       0.52      0.57      0.54       264

     accuracy                           0.52       519
    macro avg       0.52      0.52      0.51       519
 weighted avg       0.52      0.52      0.52       519
```

3. Feature Importance

After the baseline has been established, I retrieve the feature importances from the **clf** model, which is previouly trained by **RandomForestClassifier**. In Random Forest, feature importance is calculated based on how much each feature decreases the weighted impurity in a tree. In other words, it is a measure of how important each feature is for making accurate predictions.

```python
# Feature Importances
importances = clf.feature_importances_
feature_names = final_df.drop(['species'], axis=1).columns
feature_importance_dict = dict(zip(feature_names, importances))
```

From the output, we see the ranked list of features based on their importance derived from the Random Forest model. The two principal components (**PC1** and **PC2**) hold the highest importance scores, indicating that they capture significant variance and crucial information in the dataset. For the others, they all show lesser importance. To find the best hyperparameter combination, I will use **Grid Search** to find the optimal settings for the Random Forest.

```
Feature Importances:
                                Feature  Importance
0                                   PC1    0.308715
1                                   PC2    0.267089
2                           Replicate_1    0.023454
3                           Replicate_2    0.023364
4                           Replicate_3    0.022491
5                        Temperature_30    0.021023
6                        Temperature_25    0.020144
7                            Sex_female    0.019221
8                              Sex_male    0.018684
9                                Vial_8    0.017864
10                       Temperature_20    0.017370
11                              Vial_10    0.016895
12                               Vial_7    0.016749
13                               Vial_4    0.016604
14                               Vial_5    0.016486
15                               Vial_9    0.015906
16                               Vial_2    0.015481
17                               Vial_1    0.015476
18                               Vial_3    0.015357
19                               Vial_6    0.015182
20                   Population_Binjour    0.011868
21  Location_Location (-25.52, 151.45)    0.011103
22   Location_Location (-25.2, 151.17)    0.010152
23  Location_Location (-27.68, 152.45)    0.009952
24             Population_Oxford_Downs    0.009696
25             Population_Grandchester    0.009369
26            Population_Gogango_Creek    0.008704
27  Location_Location (-23.77, 150.17)    0.008635
28  Location_Location (-21.77, 148.85)    0.008614
29                   Population_Wahruna    0.008355
```

4. Hyperparameter Tuning with Grid Search

In the given code**, param_grid** defines a series of settings for a Random Forest classifier. The parameters explored include **n_estimators**, representing the number of trees in the forest with values set to [100, 200, 300]; **max_depth**, which controls the maximum depth of each tree with options including None (trees grow until all leaves are pure) and fixed depths [10, 20, 30]; **min_samples_split**, specifying the minimum number of

samples required to split an internal node; and **min_samples_leaf**, the minimum number needed at a leaf node.

The RandomForest classifier (**clf**) is configured with the **param_grid** in a **GridSearchCV** object. This object uses 3-fold cross-validation to ensure that each parameter combination is fairly evaluated by partitioning the training data into three parts. This setup uses all available CPU cores (**n_jobs=-1**) to expedite the computation and is verbose (**verbose=2**), providing detailed progress logs.

The **fit** method of **GridSearchCV** trains the Random Forest model using each parameter combination specified in **param_grid**. This training is then conducted across all folds specified in the cross-validation setup.

Once **GridSearchCV** completes its search, **best_params_** reveals the parameter combination that yielded the best performance across the evaluated folds. This optimal parameter set is then used to initialize the best version of the model, **grid_search.best_estimator_**, which is retrained on the entire training dataset to fully adapt to the data characteristics. This optimized model is at last utilized to predict the test dataset (**X_test**), and its predictions (**y_pred**) are assessed against the actual labels (**y_test**).

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, cv=3, n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)
print("Best Parameters:", grid_search.best_params_)

# Re-train with the best parameters
best_clf = grid_search.best_estimator_
best_clf.fit(X_train, y_train)
y_pred = best_clf.predict(X_test)
print("Improved Accuracy:", accuracy_score(y_test, y_pred))
print("Improved Classification Report:\n", classification_report(y_test, y_pred))
```

From the picture below, we can see the process involved fitting 3 folds for each of the 108 candidate models, total of 324 fits. The optimal parameters identified are:

- **max_depth**: None, allowing trees to grow until all leaves are pure or until they contain less than **min_samples_split** samples.

- **min_samples_leaf**: 4, requiring at least four samples to be at a leaf node.
- **min_samples_split**: 10, requiring at least ten samples to split an internal node.
- **n_estimators**: 300, using 300 trees in the forest for a robust ensemble model.

The model now achieves an accuracy of approximately 58.57%, a substantial increase from initial results.

```
Fitting 3 folds for each of 108 candidates, totalling 324 fits
Best Parameters: {'max_depth': None, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 300}
Improved Accuracy: 0.5857418111753372
Improved Classification Report:
              precision    recall  f1-score   support

 D._aldrichi       0.59      0.54      0.56       255
 D._buzzatii       0.59      0.63      0.61       264

    accuracy                           0.59       519
   macro avg       0.59      0.58      0.58       519
weighted avg       0.59      0.59      0.58       519
```

5. **Extend GridSerach one more time**

I tried more hyperparameters to provide a broader exploration of the model space, which can uncover a more optimal model configuration that was not assessed in previous searches. A more extensive grid search allows the model to be better tailored to the specific characteristics of your data.

```
param_grid = {
    'n_estimators': [100, 200, 300, 400],  # More options
    'max_depth': [None, 10, 20, 30, 40],   # More depth options
    'min_samples_split': [2, 5, 10, 15],   # More granularity
    'min_samples_leaf': [1, 2, 4, 6],      # More granularity
    'max_features': [None, 'sqrt', 'log2'],  # Different features to consider
    'bootstrap': [True, False],            # Whether to use bootstrap samples
    'criterion': ['gini', 'entropy']       # Different measures for quality of a split
}
```

From the picture below, we can see the process involved fitting 3 folds for each of the 3840 candidate models, total of 11520 fits. The optimal parameters identified are:

- **Bootstrap**: True, indicating that bootstrap samples were used.
- **Criterion**: 'entropy', suggesting that using the information gain as a criterion led to better model performance.
- **Max Depth**: None, which means trees were allowed to grow until all leaves were pure or until other constraints were met.
- **Max Features**: None, allowing the model to consider all features when making a split.

- **Min Samples Leaf**: 2, providing a good balance to prevent the model from being too fit to the noise of the training data.
- **Min Samples Split**: 2, the minimum required to decide a node split.
- **N Estimators**: 100, indicating a forest of 100 trees was sufficient to achieve the best performance.

The optimized model, configured with the best parameters from the GridSearch, achieved an accuracy of approximately 66.67%, a noticeable improvement over previous iterations.

```
Fitting 3 folds for each of 3840 candidates, totalling 11520 fits
Best Parameters: {'bootstrap': True, 'criterion': 'entropy', 'max_depth': None, 'max_features': None, 'min_samples_leaf': 2, 'm
in_samples_split': 2, 'n_estimators': 100}
Improved Accuracy: 0.6666666666666666
Improved Classification Report:
               precision    recall  f1-score   support

 D._aldrichi       0.66      0.66      0.66       255
 D._buzzatii       0.67      0.67      0.67       264

    accuracy                           0.67       519
   macro avg       0.67      0.67      0.67       519
weighted avg       0.67      0.67      0.67       519
```

Conclusion

In this report, we used dimensionality reduction and ensemble machine learning methods to classify species from complex biological data accurately. Initially employing Principal Component Analysis (PCA), the study effectively reduced the dimensionality of the data without significant information loss. Next, a Random Forest classifier was utilized to predict species classification, managing both numerical and categorical data effectively.

Through iterative model tuning and the strategic adjustment of hyperparameters via **GridSearchCV**, there was a noticeable enhancement in the model's performance. The integration of additional hyperparameters expanded the model's complexity and adaptiveness, leading to an accuracy improvement from 51.63% in the baseline model to 66.67% in the final iteration. This improvement demonstrates the potential of comprehensive machine learning approaches in biological classification tasks, particularly when combined with comprehensive data preprocessing and model optimization strategies.