Team: SG-90

Assignment-4

Report

1. We chose to build the RandomForest classifier using Decision Trees as the decision functions for voting. The RandomForest classifier we built was made from scratch, as required by the assignment regulations. We kept the ensemble classifier as simple as possible, given the small size of the data. The classifier can be run by running the 'SG-90_classifier.py' command in the terminal opened from the same folder where the program and training data file lies. The user will be prompted to input the 'test.csv file location' after running the program, and executing this will produce a 'predictions.csv' file containing the predicted labels. Although, the folder also contains the predicted labels after running the classifier.

   We obtained an accuracy of 96.36% from our classifier on a test split of 20% of the given training data.

   Note:- The 'penguins_train.csv' and 'test.csv' file must be kept in the same folder as the 'SG-90_classifier.py' for the classifier to run correctly.

2. We experimented with multiple classifiers, including ours. The classifiers are KNN, Gaussian Naive Bayes, Decision Tree, SVM, and RandomForest. The summary of the performance of these classifiers is as follows:

S. no:	**Classifier**	**Accuracy**
1	Gaussian Naive Bayes	0.9818
2	RandomForest	0.9636
3	Decision Tree (criterion = 'entropy')	0.9454
4	KNN (algorithm = 'brute')	0.7636
5	SVM (kernel = 'rbf')	0.7272

   As from the above summary table we can see that the Gaussian naive Bayes classifier performed best among all other classifiers. The RandomForest, when built directly using the 'sklearn' module, performed with an accuracy of 0.9818, the same as the Naive Bayes one. But our self-built classifier performed with 0.9636 accuracy, still better than the other classifiers built directly from the 'sklearn' module.

3. The difference between using the one-vs-all and all-vs-all classifiers on this dataset is as follows:

   One-vs-all: In this type of classifier, we train binary classifiers taking one class at a time. This is done for each class, and the learned labels are appropriately assigned by taking the predictions from each binary classifier.
   Example: Suppose we have three classes, namely {0, 1, 2}, to be classified. The one-vs-all classifier will make three copies of the data assigning one class as ‘1’ and the remaining as ‘0’. Hence we will have three binary classification problems to solve for each class. We will perform the three models in the new input and get three predictions. At last, we will pick the non-zero predictions from the three predictions and assign them to the corresponding class. For ‘n’ classes, we need ‘n’ classifiers.

   All-vs-all: In this classifier, we take two classes at a time and then train a binary classifier for all the combinations and store the predictions. If there are ‘n’ classes, there will be ‘n(n-1)/2’ models. The predicted values will then be voted against the number of predictions for a particular class by all the models. The class that bags the maximum number of votes is then assigned as the predicted class.

   Note:- The major difference between one-vs-all (ova) and all-vs-all (ava) is that the latter trains on a small subset of the dataset while the previous one trains on the whole dataset for every class. The ‘ava’ is superior to the ‘ova’ as the binary classifiers are sensitive to errors, and ‘ava’ can be robust in such cases as the vote share gives the most likeable class prediction. The class prediction probability compensates for the errors. Hence, ‘ava’ is the best suited type for our small dataset.

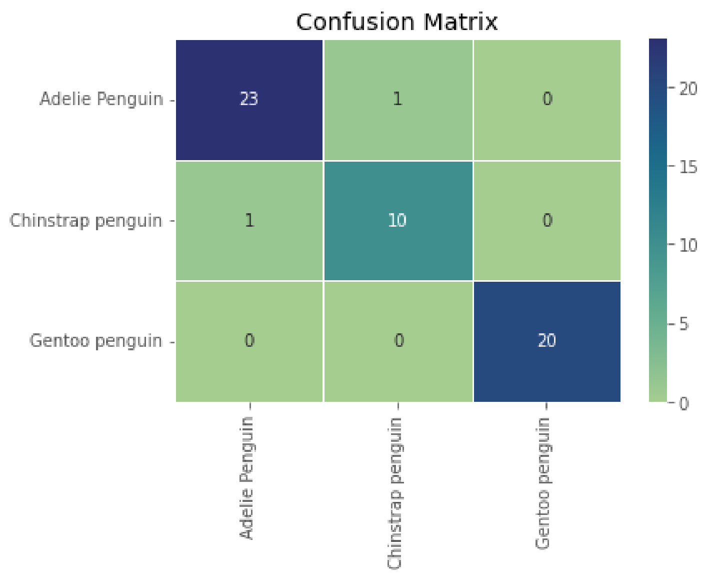
4. The given dataset is very small, and hence it is prone to overfitting. To avoid this, we:
   - Use a model with low complexity or high bias.
   - Use a simpler classifier model, like a short decision tree, rather than a deeper one, which is less susceptible to overfitting.
   - Use ensemble methods in which the voting compensates for the individual overlearning by the models.
   - Use bootstrapping to compensate for the low amount of data.

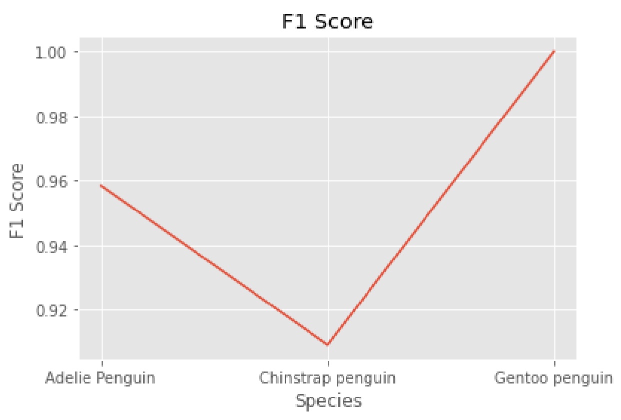
5. In our ensemble-type RandomForest classifier:

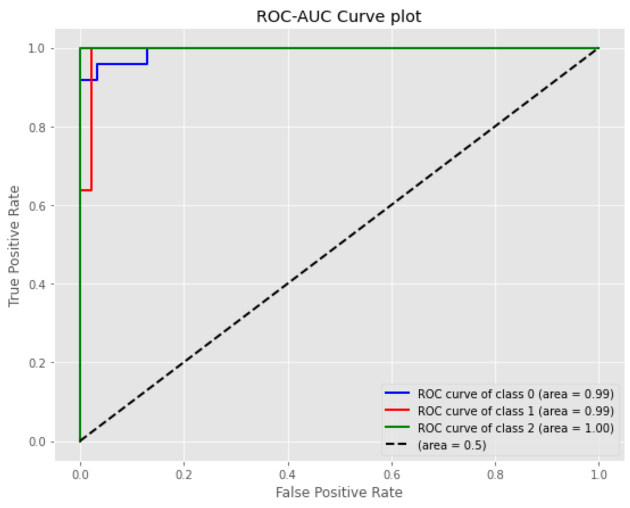
   For feature selection, we considered coding a random selection of the subset of features and performing the predictions. The class prediction probability captures the performance of the feature selection in the classifier. After using this method, we found that the probabilities of the class predictions came out to be very strong, and it was weak at only two or three positions.

   For feature engineering, we tried considering a new feature as ‘approx culmen area’ by combining ‘culmen length’ and ‘culmen depth’ but didn’t get improved accuracy as the metrics can give many similar areas but the culmen dimensions can vary for different classes. We didn’t see any significant feature generation that can improve the model further although, normalisation of ‘body mass’ helped till some extent but the difference was not statistically significant.

6. The plots of the error metrics are as follows:







```
Class 0 = Adelie Penguin
Class 1 = Chinstrap penguin
Class 2 = Gentoo penguin
```

Since, this dataset is an imbalance dataset, the best suited metrics for evaluation are F1_Score, ROC-AUC curve and confusion matrix. The confusion matrix is often hard to interpret in multiclass classification. Hence, the remaining good options are F1_score and ROC-AUC Curve. Since ROC-AUC Curve often leads to misleading results for an imbalanced datasets and F1_score can still manage to appropriately evaluate the performance when the data is skewed, **F1_score** is the best suited error metric for this dataset.