**Candidate Number: 262860**

## 1. Introduction

This report addresses the issues of implementing a classifier for a binary classification problem. To successfully determine an optimal solution the code uses a Random Forest Classifier. A strong and adaptable model that works especially well with high-dimensional. A Random Forest Classifier is a "meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve predictive accuracy and control over-fitting." (Scikit-learn, 2018) Several decision trees are trained on various data subsets to create the ensemble method's predictions, which are then combined by majority vote or average. This method makes use of the random feature selection for each tree and the independence of mistakes, both of which contribute to a decrease in overfitting.

Furthermore, a Bayesian optimization technique was utilized to effectively adjust the model parameters. By finding a balance between exploring the parameter space and taking use of well-established excellent regions, Bayesian Optimization repeatedly chooses the most promising hyperparameters based on previous assessments. (Wang, 2022). This technique is also helpful for making sure the Random Forest Classifier performs at its best for our classification problem.

## 2. Methods

### 2.1 Training and Testing the Classifier

To train and assess the classifier, the dataset was divided into training, validation, and test sets. There were two sets of training data: one with all the values and the other with some missing. These datasets were merged in order to train the model more thoroughly. Utilizing the train_test_split function from sklearn, we divided the whole training set into training and validation sets using a 80–20 split to assess the performance of the models. The model was trainged using the combined training data with sample weights from the confidence column. The test data was used exclusively for final evaluation to avoid overfitting.

### 2.2. Handling Data and Data Rescaling

To handle missing values in the dataset, a SimpleImputer with the mean method for continuous features was utilized. By using the mean of the corresponding feature in place of missing values, this method ensured that imputation did not add bias based on data dispersion.

StandardScaler from scikit-learn was used to rescale the data, eliminating the mean and scaling to the unit variance in order to standardize the features. This was a crucial step since Random Forest classifiers can still benefit from normalized feature values to make sure that each feature contributes equally to the model's performance, even though they are less likely to feature scaling than certain other models.

### 2.3 Feature Selection and Confidence Labels

Feature selection was used to lower computational complexity and enhance model performance due to the large dimensionality of the data. There were 384 gist features and 3072 CNN features in the dataset. The Random Forest model was adopted to determine the feature significance scores, aiding in the identification of the most important features. In addition, the top 250 featrures were automatically chosen using SelectFromModel in accordance with their relevance values. (Scikit-learn.org, 2019)

Both CNN and gist characteristics made a substantial contribution to the model's performance, as a result, a combination of both feature types was used in the final model.

Further details on the reliability of the class label for each training sample were supplied by the confidence labels. During the training stage, these confidence labels were used as sample weights. The algorithm was able to select more trustworthy data points by assigning samples with higher confidence (1.0) greater weight than samples with lower confidence (0.66).

## 3. Results and Discussion

### 3.1 Impact of Hyperparameters

As the max_depth increased from 3 to about 10, the model's performance improved, suggesting that deeper trees are better able to capture complex patterns. But this increases in performance decreased beyond a depth of 10, indicating that very deep trees would not add much to performance and might even cause overfitting.

When max_features was set to about 7, the model worked as well as it could. The model's capacity to identify appropriate trends was hindered by the use of too little features (e.g., 3 or 4), while the use of too many features (e.g., 8) raised the possibility of overfitting and produced no further performance gains.

Lower values of min_samples_split often resulted in better model performance, indicating that the model was able to learn from the data more efficiently by permitting more splits. Nevertheless, if the model is very sensitive to even little changes in the training set of data, exceedingly low values (such as 2) may cause overfitting.

Generally speaking, the model performed better with more trees (n_estimators) up to about 200 trees, after which

the performance improvements began to decline. This suggests that while improved performance may be achieved with more trees, there is a threshold at which adding more trees just increases computational cost and does not yield appreciable benefits.

## 3.2 Performance across different training sees

The model was trained and evaluated using different subsets of the data to assess the impact of incompleete data and the importance of confidence labels

The results indiciate that the complete dataset provides the highest accuracy, demonstrating that complete data is more informative and beneficial for training robust models.

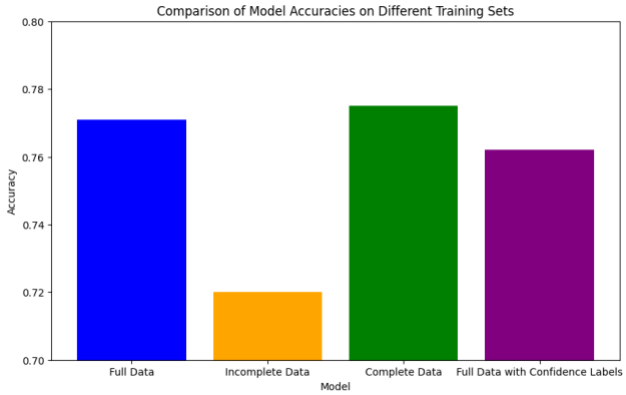


Figure 1: Visualization of different classifiers on different training sets

Incomplete training data was found to be less useful, as evidenced by the lower accuracy 72%. This suggests that missing values and incomplete features significantly hinder the model's learning process.

The inclusion of confidence labels enhanced the model's performance. The model scored 76.2% with confidence labels, compred to initialy scoring 74.3%. This indicates that the model can benefit from additional information about the reliability of each training instance. Confidence labels act as weights, giving more importance to more reliable data points, which helps the model focus on higher quality information during training.

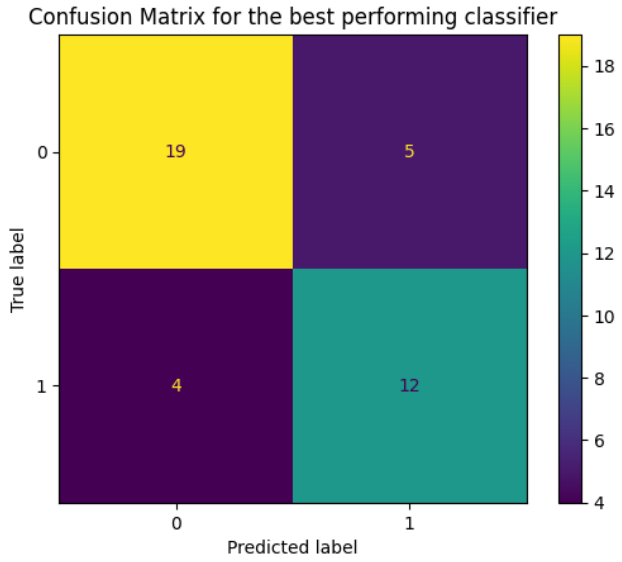


Figure 1: Confusion matrix for the best performing classifier

The figure above includes a confusion matrix for the best performing classifier. In 77.5% of cases, the model predicts the class labels accurately. Seventy-five percent of all the occurrences that were anticipated to be good came true. This indicates a small number of false positives. Of all true positive events, 75% are accurately identified by the model. This indicates a few false negatives. While the model is effective at recognizing true negatives, it also contains some false positives, properly identifying 79.2% of all actual negative cases. The F1 score is 72.6%; it strikes a balance between recall and accuracy. This is a useful measure of the model's overall effectiveness, particularly in the case of uneven class distributions. Overall, the model performs reasonably well with an F1 score of 72.6%, but there's room for improvement in reducing misclassifications.

## 4. Conclusion

In conclusion, the system was able to successfully implement and optimize a Random Forest classifier for a binary classification. In order to improve model performance, a variety of training subsets were utilized, along with confidence labels. The results of thorough investigation and methodical testing demonstrated how important high-quality data is for developing reliable machine learning models. The findings show that models with confidence labels and those trained on whole datasets can achieve greater accuracy and improved generalization.

To further increase the model's accuracy and resilience, future research should examine more complex approaches to handle missing data, such as multiple imputation or data augmentation strategies.

## References


[1] Scikit-learn (2018). *sklearn.ensemble.RandomForestClassifier — scikit-learn 0.20.3 documentation*. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[2] Wang, W. (2022). *Bayesian Optimization Concept Explained in Layman Terms*. [online] Medium. Available at: https://towardsdatascience.com/bayesian-optimization-concept-explained-in-layman-terms-1d2bcdeaf12f#:~:text=So%20what%20is%20Bayesian%20Optimization [Accessed 23 May 2024].

[3] Scikit-learn.org. (2019). *1.13. Feature selection — scikit-learn 0.21.3 documentation*. [online] Available at: https://scikit-learn.org/stable/modules/feature_selection.html.

[4] https://towardsdatascience.com/bayesian-optimization-concept-explained-in-layman-terms-1d2bcdeaf12f#:~:text=So%20what%20is%20Bayesian%20Optimization,actually%20delivers%20a%20simple%20message.