Digits Classification Using Random Forest Classifier

Ngan Junn Fai
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
tp059397@mail.apu.edu.my

Keong Yan Qi
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
tp060661@mail.apu.edu.my

Raymond Jee Meng Chun
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
tp060797@mail.apu.edu.my

Wong Kai Wey
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
tp060469@mail.apu.edu.my

Gan Jun Xian
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
tp060912@mail.apu.edu.my

Zailan Arabee bin Abdul Salam
School of Computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
zailan@apu.edu.my

***Abstract*— The objective of this paper is to investigate the performance of a random forest classifier for the task of digit classification using a standard dataset of handwritten digits. This paper focuses on hyperparameter tuning to evaluate the individual and combined influence of different hyperparameter settings on the accuracy of the random forest classifier, using stratified-k fold cross-validation as the performance criteria. The result of this study shows that the random forest classifier achieves an accuracy of 0.9416. The impact of different hyperparameter settings on the classifier's performance is also analyzed and it is found that certain settings either improve or diminish the accuracy of the model while some trade-off each other. The findings demonstrate the effectiveness of the random forest classifier for digit classification tasks and suggest that it could be useful in other applications where accurate classification is important.**

Keywords—Random Forest, digits classification, parameter tuning, Stratified K-Folds, multiclass classification

I. INTRODUCTION

Digits classification and recognition are done seamlessly by humans through the frontoparietal cortex of the human brain. The neurons in the human brain fire in a chain of events which results in the identification of digits. In machine learning, handwritten digits classification involves using algorithms to recognize and classify handwritten digits based on their visual appearance. This falls under the multiclass classification as it classifies instances into one of three or more classes.

To represent the handwritten digits in a way that can be processed by a machine learning algorithm, it is common to use features that capture various characteristics of the digits through feature extraction, such as breaking pixels into features with the pixel intensities being the variable of the feature (Abdulrazzaq and Saeed, 2019; Karakaya and Kazan, 2021). The algorithm learns to recognize patterns in the data that are associated with each digit and the labelled class, and it can then be used to classify new, unseen digits based on these learned patterns.

Digit recognition can help to ease daily life problems by automating tasks that would otherwise be time-consuming for humans to perform. Digit classification can be used to automate data entry to automatically extract and transcribe data from scanned documents or images. This can save time and avoid overstraining caused by human transcription as the machine learning algorithms can be trained to accurately transcribe the data with a high degree of accuracy and making it easier and more efficient for individuals or organizations to manage and process large amounts of data. During the unforeseen global pandemic, COVID-19, machine learning technology experienced a groundbreaking upsurge in demand worldwide (Fortune Business Insight, 2022). A higher growth of 36.1% in 2020 compared to 2019 has been disclosed by the global machine learning market.

There are many different approaches to handwritten digit classification, and the choice of algorithm and features used can have a significant impact on the performance of the system. Thus, this paper aims to study the usability and performance of using a Random Forest Classifier (RFC) to classify handwritten digits.

II. LITERATURE REVIEW

The Random Forest Algorithm (RFA), an ensemble classifier that combines many decision trees to provide predictions, was reviewed by Parmar, Katariya, and Patel in 2018. The background, key principles, and useful applications of the random forest approach are all covered in detail in this study. The random forest classifier creates several decision trees and then combines the decision trees' predictions to provide a final prediction. Each decision tree in the random forest is trained to provide predictions using a random subset of the features and a random subset of the training data. All of the decision trees in the random forest's predictions are then combined to produce the final prediction, which is usually decided by a majority vote. According to Parmar et al. (2018), the random forest classifier has strong classification accuracy with large datasets and is resistant to overfitting by handling noises present in data sets, making it a good option for many classification tasks.

Research has been done on machine learning which utilizes various algorithms such as SVM, KNN, Convolutional Neural Networks (CNN), etc. Akhtar & Ali (2020) implemented RFC as a character recognizer in their work of proposing an automatic number plate recognition.

They have compared four classifiers namely K-Nearest Neighbor (KNN), Neural Network (NN), Support Vector Machine (SVM) and RFC. The results depict that RFC is prevalent with an accuracy of 90.9% whereby the others are within 83 to 90 percent. Two parameters, the number of features in each split (Fs) and the number of decision trees (Nt) were set to the square root of the number of features for the model and 100 respectively. This accuracy was due to RFC thoroughly removing the ambiguity of certain similar characters such as "G" and "6". Therefore, the result was justified due to the nature of RFC being a multiclass classifier whereas SVM is fundamentally a binary classifier. However, it has also been noted that RFC takes a slightly longer time as compared to the other 3 classifiers (Akhtar & Ali, 2020).

Similar research has been done by Shamim et al. (2018) describes that the RF algorithm, whether regression or classification, contains a weakness when learning from an extremely unbalanced training dataset. This is because RF is designed to diminish the overall error rate. RF tends to focus more on the prediction efficiency of the majority class, resulting in consecutively poor accuracy for the minority class. In their paper, the performances of different machine learning algorithms including Multilayer Perceptron (MLP), SVM, RF, Bayes Net, Naïve Bayes, J48 and Random Tree using the MNIST dataset. Among all the machine learning algorithms used, the highest accuracy with a score of 90.37% was achieved by MLP. RF came in third with an accuracy of 85.75% (Shamim et al., 2018). To prevent this issue, identical prior probability for both classes (majority and minority) should be done which results in the minority class being overrepresented. This raises the class's posterior probability, which shifts the class's classification boundary and causes more observations to be classified to the class. The weightage of the minority class is increased by defining the benefit of selecting the best decision for a case from a minority class should be more than the majority class (Zhou et al., 2020).

Literature by Moo et al. (2021) attempted to employ Neural Network and Genetic Algorithm (GA) together to play the snake game, where the NN controls the moving direction while GA controls the snake's evolution. Moo et al. attempted to tune the mutation rate, population size, number of hidden layers and number of hidden neurons, separately and one at a time. Moo et al. subsequently identified the sole optimum value for parameters that yields the most favourable results but they have also suggested that a better result could be possibly attained if these hyperparameters are cross-tuned simultaneously. Although this piece of work studies hyperparameter tuning, the cross-implementation of two machine learning algorithms on one problem may have influenced the learning process as it introduces a level of uniqueness to the hybrid algorithm. Thus, it lacks a baseline for comparison and has limited generalisability. The level of significance for each parameter might have changed in the cross-implementation.

Moving on, research done by Bernard et al. (2007) focuses on imparting rules on parameter settings for RFC practitioners. In their paper, it is stated that Breiman founded the Bootstrap Aggregation (Bagging) technique which entails creating a group of base classifiers, each of which has been trained using a bootstrap replica of the training data. The subsequent combination of outputs with a plurality or majority vote results in predictions. Breiman then proposed Random Forest a few years later which consists of a general strategy for developing a Multiple Classifier System (MCS) that uses Decision Trees as the fundamental classifiers. This ensemble is unique in that each member must be constructed using a set of random parameters. The essential point is that the ensemble of the base classifiers becomes more diverse because of this randomization (Bernard et al., 2007). Through this understanding, Bernard et al. experimented with two parameter values, the Number of Trees (L), and the Number of Features preselected for the splitting process (K), and found that the increase of L and K shows improvement in recognition rates within a certain range ($100 \leqslant L \leqslant 300$; $5 \leqslant K \leqslant 20$).

III. Materials and Methods

A. Selection of Materials

1) Source Code: The Python programming language (version 3.10.1) was used to implement the Random Forest Classifier in this study. It is a widely used, high-level programming language known for its simplicity and readability. The source code of the Random Forest Classifier was obtained through the ensemble module of the Python scikit-learn package (version 1.1.2).

2) Machine: A device running on Windows 11 Pro (version 22H2, build 22621.963) operating system equipped with a 12th Gen Intel® Core™ i5-12600 central processing unit and 16 GB of RAM installed was deployed to conduct this study.

3) Dataset: The digits dataset chosen is from scikit-learn which is a collection of handwritten digits images that is commonly used as a benchmark for machine learning algorithms. It contains 1,797 images of handwritten digits which ranges from 0 to 9 and are almost evenly distributed in each class. Each image represents an 8 x 8 pixels grayscaled digit, where each pixel's intensity is indicated by an integer value between 0 and 16. The dataset comprises 64 features in total, which matches the number of pixels in the image. The dataset is ready to use and no preprocessing is needed.

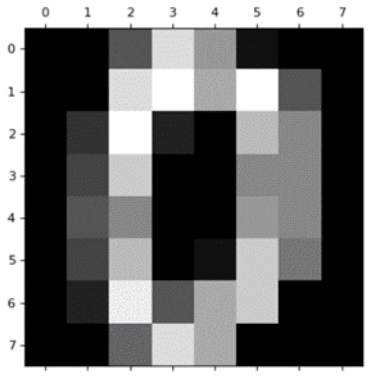

Fig. 1. Sample image in the scikit-learn digits dataset.

B. Algorithm Implementation

1) Decision Tree: A decision tree is a graphical representation for making predictions or judgements depending on certain conditions. The root node is at the top, and the leaf nodes also known as terminal nodes are at the bottom, making up the structure's nodes. The leaf nodes reflect the potential outcomes of each decision, whereas each

node represents a decision or query. Based on the circumstances at each node, the algorithm analyses the information at hand to identify the most likely path through the tree (Safavian et.al, 1991). The default division criterion, also known as the split rule used by decision trees in the scikit-learn package is the Gini index (Géron, 2017) and the criterion will remain unchanged throughout the entire study. Fig. 2 shows the illustration of how a decision tree works.

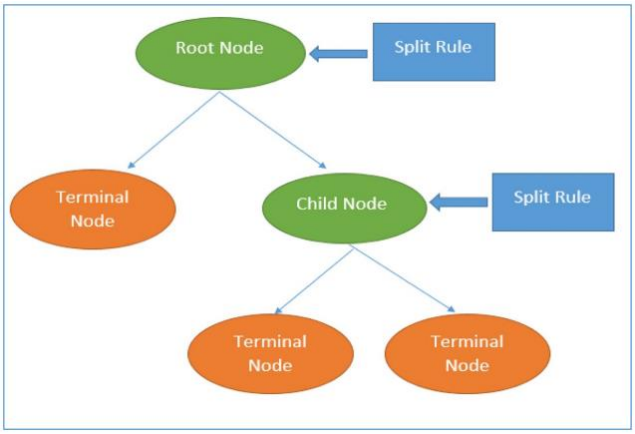

Fig. 2. Decision tree illustration (Zhou et al., 2020).

2) Random Forest: An RFC is utilized for this digit classification study. RFC is a method that constructs a forest which consists of a large number of decision trees at training time and outputting the class that is the mode of the classes (classification) yielded from the individual trees. Each decision tree inside the forest was built and trained using a random feature subset of fixed size. The final prediction made by the random forest is the mode of the class predictions made by the individual decision trees, also known as the majority voting. In other words, the class label that is predicted by most of the decision trees will be the final prediction of the algorithm (Breiman, 2001). This concept of random forest is to address the overfitting that can occur when using only a single decision tree. Fig. 3 demonstrates the algorithm's logic in a schematic representation.

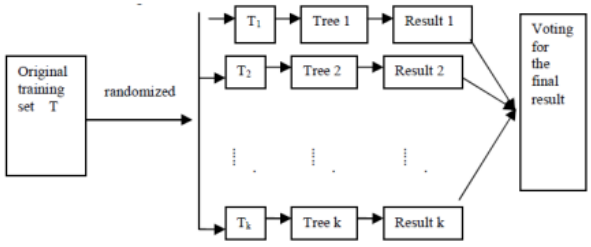

Fig. 3. Random forest schematic (Liu et al., 2012).

C. Parameters

In this study, one constant parameter was involved and three hyperparameters will be modified or tuned to investigate the influence of each studied hyperparameter on the model's performance. The default value will be compared together with another four custom values, for each of the hyperparameters to be tuned.

1) `random_state`: To enable a fair evaluation of the effects of each studied hyperparameter, the randomness will be set to a static number, 42 so that the model behaves consistently across different trials. In this source code, the default value for this parameter is `None`, which means that the pseudorandom algorithm used to initialize the model will not be initialized with a fixed seed value, and the results of the model will not be reproducible.

2) `n_estimators`: This parameter controls the number of decision trees in the ensemble. The default value of this parameter is 100, which means 100 decision trees will be built in the forest. It is widely understood that increasing the value of this parameter may allow for more diverse predictions to be made. Another four custom values to be studied are 50, 75, 125 and 150 respectively.

3) `max_depth`: This parameter controls the maximum depth of each decision tree in the ensemble. The depth of a decision tree refers to the number of splits that are made from the root node to the leaf nodes. The default value of this parameter is `None`, which means that the trees in the forest can grow indefinitely, hence the maximum model complexity. Another four custom values to be studied are 2, 5, 8 and 10 respectively.

4) `max_features`: This parameter controls the number of features that are randomly selected for the construction of decision trees. The default value of this parameter is `sqrt` which uses the square root of the total number of features from the dataset. It is a well-established relationship that increases the value of `max_features` will make the model access to more features when making split decisions, which can increase the complexity of the model and potentially improve its performance but also the risk of overfitting. Another four custom values to be studied are 2, 4, 16 and 32 respectively.

D. Performance Criteria

The performance criteria that will be used in this study is the stratified k-folds cross-validation. It involves dividing a dataset into `k` folds where the proportion of samples for each class is approximately the same as the entire dataset in each fold. Each fold represents a subset of the data. The model is then trained and evaluated `k` times, with a different fold being used as the test set and the rest will be the training data in each iteration. This is important when the class distribution is imbalanced, as it stratifies the dataset into `k` chunks accordingly to the original proportion of classes. It helps to ensure that the model is trained and evaluated on a balanced subset of the data, reducing the risk of bias in the evaluation of the model's performance (Berrar, 2019).

IV. Results and Discussion

Generalisation is the ability of a machine learning algorithm to perform well on unobserved data. Generalisation error is a measurement of the accuracy of the model when acting on unseen data (Guissous, 2019). In the scenario where the model performs poorly on unobserved data, there could be two reasons. The first reason is that the model has insufficient capacity and underfit the underlying function. The second reason is the model has excessive capacity and overfits the

underlying function. In order to obtain a model with low generalisation error, the training error and test error must be low. In the instance where there is high training and testing error, the model is said to be underfitted. The solution for this is to increase the capacity by tuning the related hyperparameters. On the other hand, low training error but high testing error is a result of an overfitted model. The model should decrease its capacity by shrinking techniques. This reduces the variance of the model without causing a significant increase in its bias. Fig. 4 delivers the above concepts in a graphical view.

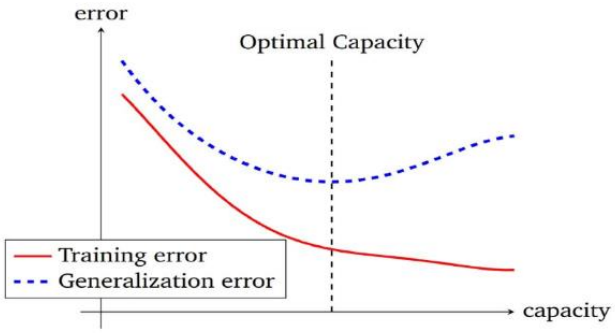

Fig. 4. Generalisation error (Guissous, 2019).

The use of the precise capacity of a model is pivotal in addressing the generalisation error complication. When a model is underfitting, the performance of the model is said to be critically inaccurate whereby the model fails to classify the training dataset which leads to highly inaccurate predictions of the unseen dataset. Meanwhile, an overfitted model is when the model achieves high accuracy on the testing dataset but fails to accurately predict the unseen dataset. This is due to the model being overly sensitive to certain features which results in the misclassification of unseen data. Fig. 5 illustrates the model learning performance of each condition.

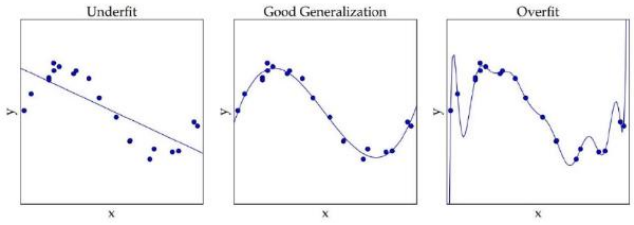

Fig. 5. Underfit, good generalisation and overfit (Guissous, 2019).

A. `n_estimators`

Table I shows the results of stratified k-folds cross-validation accuracy for different values of the `n_estimators` hyperparameter in the Random Forest classifier. The default value of 100 did not produce the most accurate result for digits classification as the default value returns an acceptable accuracy of 0.9394. However, when the `n_estimators` is set to 125, the model returns a slightly better accuracy which is also better than the result when `n_estimators` = 150. This result suggests that setting 125 for the `n_estimators` hyperparameter truly approximates the optimum model capacity compared to 100 and 150, among the 5 different values studied in this test.

TABLE I. ACCURACY FOR TUNING THE `N_ESTIMATORS ` HYPERPARAMETER

n_estimators	*50*	*75*	*100*	*125*	*150*
Accuracy	0.9399	0.9371	0.9394	0.9416	0.9410

A higher value for `n_estimators` results in a more diverse and robust model but also increases the computational complexity, and time required to train and predict with the model. These results suggest that increasing the `n_estimators` can increase the stability of the Random Forest classifier, but the accuracy begins to diminish after a certain point as seen in Table I. The diminish occurs because the learning model starts to overfit. In this case, it is better to reduce the number instead of increasing it. Therefore, it is important to consider the trade-off between model performance and computational efficiency when tuning the value for `n_estimators`.

B. `max_depth`

Table II shows the results of k-folds cross-validation accuracy for different values of the `max_depth` hyperparameter in the Random Forest classifier. The default value of `max_depth` (`None`) produced the most accurate result for digits classification which is 0.9394. By setting specific numbers for the `max_depth` hyperparameter, it showed lower accuracies compared with the default value. Thus, setting to `None` for the `max_depth` hyperparameter allows the model to approach its optimum capacity, among the 5 different values studied in this test.

TABLE II. ACCURACY FOR TUNING THE `MAX_DEPTH` HYPERPARAMETER

max_depth	*2*	*5*	*8*	*10*	*`None`*
Accuracy	0.7863	0.8904	0.9243	0.9371	0.9394

A higher value of `max_depth` results in a deeper tree thus a more complex model is made possible. These results suggest that decreasing the `max_depth` value reduces the complexity of the Random Forest classifier. A lower value indicates that the number of splits each tree can make is limited, thus when it reaches this limit, although the current nodes are still impure, no further split can be made. This consequentially made the impurities in each node stay and got classified into the wrong class. In short, the optimal value for `max_depth` still depends on the specific dataset, the classification task being performed and the desired outcomes.

C. `max_features`

The default value for `max_features` in this RFC model does not produce the most accurate result for digits classification. The default value of `max_features` is the square root of the total features in a class. Hence, the square root of 64, 8, returns an acceptable accuracy margin (0.9394). However, when the `max_features` value is set to 4, the model returns a consistently better result (0.9416) and it's the optimal figure. This result suggests that setting 4 for the `max_features` hyperparameter truly approximates the optimum model capacity compared to 8, among the 5 different values studied in this experiment as tabulated in Table III.

TABLE III. ACCURACY FOR TUNING THE `MAX_FEATURES` HYPERPARAMETER

max_features	*2*	*4*	*sqrt/8*	*16*	*32*
Accuracy	0.9349	0.9416	0.9394	0.9399	0.9266

Increasing the number of features to consider for each tree does not always increase the accuracy of the result as the model tends to overfit while reducing the value of `max_features` reduces the model complexity thus making the model less sensitive to noise or anomalies, but may also underfit the training data. Therefore, it can be inferred that the default value of `max_features` does not always result in the best possible accuracy in an RFC model.

D. Cross-Tuning of Three Hyperparameters

After examining the influence of the sole hyperparameters on the model's performance, Table IV tabulates the prediction accuracy when the three parameters aforementioned are cross-tuned.

Perusing the prediction accuracy of different `max_depth` values, the overall accuracy is increasing in relatively larger steps as the `max_depth` increases. Yet, it can be seen that no matter how the `max_features` and `n_estimators` parameters are being tuned, the accuracy doesn't change much, unless the `max_depth` value was stepped up. Therefore, the `max_depth` hyperparameter is deemed to be the most influential among the three studied subjects. This hyperparameter is at the maximum capacity by default as it doesn't have a finite limit, thus it is usually used to reduce the model compacity. This is warranted by the findings in Table IV, where the lowest accuracy among all 125 combinations of different hyperparameter tuning was when `max_depth` = 2, while the highest was found when `max_depth` = `None`.

The `max_features` is deemed to be the second most influential. This is the only hyperparameter of the RFC model that can be used to both increases or decrease the model capacity. Despite the `max_depth` being tuned, a higher `max_features` results in the model overfitting the training data. This is because the model will consider more features in the construction of each tree, thus each feature is being learned by the model repeatedly, more frequently. Yet, although the `max_features` was set to 2, the decline in accuracy is not as drastic as `max_depth`, this is due to the `n_estimators` saturated the model's access to each feature in the dataset, thus the underfitting was minimized. In particular, the `n_estimators` does not interfere with the construction of individual decision trees, but it affects the stability of the RFC model.

When tuning each of the three hyperparameters separately, the optimum values identified were `n_estimators` = 125, `max_depth` = `None` and `max_features` = 4. When these three parameters were conjunctively tuned to their respective solo optimum value, its model's accuracy is very high, yet it was not the highest instance of accuracy found in these 125 hyperparameter combinations. Vice versa, the solo inferior values of each parameter do not make up the lowest accuracy in this experiment. A simple inference can be made whereas different parameters would trade-off each other's influences, even though each parameter is tuned to its solo peak value.

TABLE IV. ACCURACY FOR CROSS-TUNING THREE HYPERPARAMETERS

max_ depth	**max_ features**	**n_estimators**				
		50	***75***	***100***	***125***	***150***
2	*2*	0.7986	0.8063	0.8047	0.8119	0.8197
	4	0.8186	0.8225	0.8191	0.8219	0.8275
	sqrt/8	0.7780	0.7869	0.7863	0.7841	0.7924
	16	0.7290	0.7301	0.7318	0.7334	0.8848
	32	0.6522	0.6611	0.6739	0.6767	0.6789
5	*2*	0.8871	0.8932	0.8921	0.8982	0.8948
	4	0.8993	0.9015	0.9021	0.9021	0.9049
	sqrt/8	0.8909	0.8871	0.8904	0.8954	0.8932
	16	0.8848	0.8876	0.8948	0.8954	0.8948
	32	0.8698	0.8732	0.8732	0.8715	0.8743
8	*2*	0.9260	0.9266	0.9255	0.9316	0.9321
	4	0.9277	0.9305	0.9316	0.9338	0.9344
	sqrt/8	0.9271	0.9249	0.9243	0.9266	0.9321
	16	0.9210	0.9204	0.9221	0.9238	0.9249
	32	0.9110	0.9149	0.9121	0.9138	0.9160
10	*2*	0.9255	0.9305	0.9310	0.9349	0.9355
	4	0.9371	0.9394	0.9399	0.9394	0.9405
	sqrt/8	0.9377	0.9371	0.9371	0.9383	0.9366
	16	0.9321	0.9310	0.9349	0.9382	0.9377
	32	0.9199	0.9210	0.9238	0.9249	0.9277
`None`	*2*	0.9332	0.9338	0.9349	0.9371	0.9366
	4	0.9355	0.9394	0.9416	0.9410	0.9394
	sqrt/8	0.9399	0.9371	0.9394	0.9416	0.9410
	16	0.9382	0.9377	0.9399	0.9366	0.9366
	32	0.9243	0.9260	0.9266	0.9255	0.9266

V. CONCLUSION

Random Forest Classifier (RFC) algorithm has been used to classify digits by their respective class. Initial results using default parameter values were fairly decent but unsatisfactory. Hence, through the implementation of hyperparameter tuning, each substantial hyperparameter is delicately tuned to achieve the lowest possible generalisation error which corresponds to the model approaching the approximate optimal model capacity. This returns the best possible accuracy. However, for

the model to truly reach true optimal capacity, it would be time-consuming and costly. It has been noted throughout the research that hyperparameter tuning in an RFC is comparatively simpler and visualizable by graphical or tabular representation. Besides that, RFC by its nature, is a generalisation algorithm because of the involvement of multiple decision trees in its ensemble. This results in RFC being less prone to overfitting. Hence, RFC is objectively suitable for new machine learning practitioners as resources are abundantly available and arguably easy to comprehend and implement.

ACKNOWLEDGMENT

The authors would like to thank to all School of Computing members who involved in this study. This study was conducted for the purpose of digits classification using random forest classifier project.

REFERENCES

Abdulrazzaq, M. B., & Saeed, J. N. (2019, April). A comparison of three classification algorithms for handwritten digit recognition. In *2019 International Conference on Advanced Science and Engineering (ICOASE)* (pp. 58-63). IEEE.

Akhtar, Z., & Ali, R. (2020). Automatic number plate recognition using random forest classifier. *SN Computer Science*, *1*(3), 1-9.

Bernard, S., Adam, S., & Heutte, L. (2007, September). Using random forests for handwritten digit recognition. In *Ninth international conference on document analysis and recognition (ICDAR 2007)* (Vol. 2, pp. 1043-1047). IEEE.

Berrar, D. (2019). Cross-Validation. *Encyclopedia of Bioinformatics and Computational Biology*, 542–545.

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5-32.

Fortune Business Insight. (2022). Market Research Report. In *Fortune Business Insight* (No. FBI102226). Retrieved December 21, 2022, from https://www.fortunebusinessinsights.com/machine-learning-market-102226.

Géron, A. (2017). Hands-on machine learning with scikit-learn and tensorflow: Concepts. *Tools, and Techniques to build intelligent systems*.

Guissous, A. E. (2019). Skin lesion classification using deep neural network. arXiv preprint arXiv:1911.07817.

Karakaya, R., & Kazan, S. (2021). Handwritten digit recognition using machine learning. *Sakarya University Journal of Science*, 25(1), 65-71.

Liu, Y., Wang, Y., & Zhang, J. (2012, September). New machine learning algorithm: Random forest. In International Conference on Information Computing and Applications (pp. 246-252). Springer, Berlin, Heidelberg.

Moo, C. Y., Lee, W. Y., Chen, E. Y. K., Syed, S. Q. & Salam, Z. A. A. (2021). Investigating parameters of genetic algorithm and neural network on classic snake game. *Journal of Applied Technology and Innovation*, *5*(2).

Parmar, A., Katariya, R., & Patel, V. (2018, August). A review on random forest: An ensemble classifier. In *International Conference on Intelligent Data Communication Technologies and Internet of Things* (pp. 758-763). Springer, Cham.

Safavian, S. R., & Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3), 660-674.

Shamim, S. M., Miah, M. B. A., Angona Sarker, M. R., & Al Jobair, A. (2018). Handwritten digit recognition using machine learning algorithms. *Global Journal Of Computer Science And Technology*.

Zhou, X., Lu, P., Zheng, Z., Tolliver, D., & Keramati, A. (2020). Accident prediction accuracy assessment for highway-rail grade crossings using random forest algorithm compared with decision tree. *Reliability Engineering & System Safety*, *200*, 106931.