March 29th, 2020

Random Forest from Scratch

Sara van Bezooijen[1], **Arnold Dijk**[1], **Luuk Schrijvers**[1], **Tom de Valk**[1] **and Krishnanand Muthumana Vasudevan**[1]
[1]Premaster Student Business Analytics at VU Amsterdam

ABSTRACT As part of the machine learning course of the VU University in Amsterdam, students are encouraged to obtain a good understanding of machine learning models. To do this, a random forest model has been developed from scratch using only the Pandas and Numpy Python packages. The developed random forest model was tuned and tested on the MNIST data set following the principle of training, validating and testing. An accuracy of 93.99% was achieved on the test data set. The developed random forest algorithm can be enhanced with a number of improvements which have been identified as future research.

KEYWORDS
Random Forest
Decision Tree
MNIST
Machine Learning
VU Amsterdam

1. INTRODUCTION

For the project of the machine learning course of the VU University in Amsterdam, students are encouraged to get a good understanding of machine learning models. One option to get this better understanding, is to develop a machine learning model from scratch. In this case, a random forest model will be developed from scratch. The development of this model will be limited to the use of only the Numpy and Pandas Python packages. The development of the model will also be limited to the available computation power as well as the given time to carry out the research. The goal of this study is to answer the following research question:

To what extent is it possible to use Python to program a random forest classification model from scratch and implement it on a data set to achieve the best possible result?

This research question should be answered thoroughly using the results of the completed research. To ensure that the research question will be answered completely, the following sub-questions have been formulated:

1. *What are decision tree models and how is a decision tree model optimally developed?*
2. *What are random forest models and how is a random forest model optimally developed?*
3. *How can the performance of random forest models be evaluated to find the best suited random forest model?*

These sub-questions will ensure that there is enough knowledge present to develop the random forest model from scratch. The answers to these sub-questions will also be used to formulate a complete answer to the research question.

2. DECISION TREE MODELS

Decision tree models are one of the most crucial elements of random forest models. Therefore, it is imperative that decision tree models are fully understood before the development of a random forest model from scratch. Once decision tree models are fully understood, the corresponding sub-question can be answered. This sub-question goes as follows: *What are decision tree models and how is a decision tree model optimally developed?*

Decision trees are models that are used to support decision-making. These models use tree-like structures to determine the outcome for a certain input. The decision tree model starts with a root node. From there, at least two branches are added in the form of decision nodes. Every decision node leads to a combination of branches and/or leaf nodes. These leaf nodes represent the ends of branches in the decision tree model. Decision tree models have a large scale of possible applications; one of which being machine learning and random forest models.

2.1. Training Decision Tree Models

The training of decision tree models is used to fit a model to a given data set. This training is carried out using a method named induction. According to Perner (2015), there are several known induction algorithms for the training of decision tree models. These algorithms can be broken down into the following steps:

1. Set up a training data set to classify.
2. Determine the feature of the data set that will result in the best possible split of the data set. There are several methods to find the feature that will result in the best possible split. These methods will be discussed in section 2.1.1.
3. Split the data into subsets that contain all of the different possible values for the chosen feature. These subsets represent the nodes in the decision tree model.
4. Recursively generate new nodes by using the previous two steps. This is repeated until the performance of the model has been maximized while the number of splits has been minimized. Other stopping criteria are discussed in section 2.1.2.

This is confirmed by Mantey (2019). The different types of algorithms differ in the way that the most important features are chosen, whether numerical and/or categorical features can be dealt with and how noise is reduced using pruning.

Pruning is a method used to reduce noise and avoid overfitting after a decision tree model has been developed (Bradford *et al.* 1998). According to Bradford *et al.* (1998), the goal of pruning is to reduce the size of a decision tree model in such a way that the error rate is minimal on unseen data, or the validation set. Pruning can be done using one of the following methods (Perner 2015):

- Cost-complexity pruning
- Reduced error pruning
- Pessimistic pruning
- Error-based pruning
- Penalty pruning
- MDL pruning

While these methods are all slightly different, they all rely on the same basic pruning algorithm. This algorithm aims to replace sub-trees with a single leaf when the estimated error of the single leaf is lower than the estimated error of the sub-tree. This process is repeated until no more sub-trees can be replaced by a single leaf without increasing the error. The result of the application of pruning is a lower probability of overfitting of the decision tree model on the training set.

2.1.1. Splitting Criteria

The splitting of data for decision tree models can be done using multiple different criteria. A few of the different splitting criteria are discussed below.

Impurity-Based is a method to determine the impurity measure. This measure is used to show a randomly chosen instance is misclassified when it is randomly classified using the baseline. The goal in machine learning is to have the lowest possible impurity in the model, so that the predictions are the most "pure" (Rokach and Maimon 2005).

Information Gain is an impurity-based criterion that uses the entropy to determine the impurity. The information gain is calculated using the following formula:

$$InformationGain(a_i, S) = Entropy(y, S) - \sum_{v_{i,j} \epsilon dom(a_i)} \frac{|\sigma_{a_i = v_{i,j}} S|}{|S|} * Entropy(y, \sigma_{a_i = v_{i,j}} S)$$

where the entropy is calculated using the following formula:

$$Entropy(y, S) = \sum_{c_j \epsilon dom(y)} -\frac{|\sigma_{y=c_j} S|}{|S|} * log_2 \frac{|\sigma_{y=c_j} S|}{|S|}$$

This means that the information gain criterion can be described as the entropy before the split reduced by the weighted entropy after the split (Rokach and Maimon 2005) .

Likelihood-Ratio Chi-Squared Statistic is a ratio that is used to measure the statistical significance of information gain. With this, a null-hypothesis is formulated where the input and target variables are conditionally independent. If the null-hypothesis is accepted, G^2 is chi-square distributed with $(dom(a_i) - 1) * (dom(y) - 1)$ degrees of freedom. G^2, the likelihood-ratio is defined using the following formula:

$$G^2(a_i, S) = 2 * ln(2) * |S| * InformationGain(a_i, S)$$

(Rokach and Maimon 2005)

DKM Criterion is an impurity-based criterion for binary classes. The function for the DKM Criterion is equal to:

$$DKM(y, S) = 2 * \sqrt{(\frac{\sigma_{y=c1} S}{|S|}) * (\frac{\sigma_{y=c2} S}{|S|})}$$

According to Rokach and Maimon (2005), the DKM criterion requires the use of decision tree models with a smaller depth to obtain an error that is different than the other impurity-based criteria: information gain and gini index.

Normalized Impurity-Based Criterion is, in principle, the same measure as the impurity-based method, but the result is normalized to reduce the influence of variables with big values (Rokach and Maimon 2005).

Gain Ratio is a ratio is a normalized information gain. The gain ratio is calculated using the following formula:

$$GainRatio(a_i, S) = \frac{InformationGain(a_i, S)}{Entropy(a_i, S)}$$

Considering the formula mentioned above, the ratio is calculated using the information gain criterion and the calculated entropy. When the entropy is equal to zero, the gain ratio cannot be calculated (Rokach and Maimon 2005).

Distance Measure normalizes the impurity measure, similar to the gain ratio. The distance measure is calculated using the following formula:

$$DistanceMeasure = \frac{\Delta\Phi(a_i, S)}{-\sum_{v_{i,j} \epsilon dom(a_i)} \sum_{c_k \epsilon dom(y)} \frac{|\sigma_{a_i = v_{i,j} AND y = c_k} S|}{|S|} * log_2 \frac{|\sigma_{a_i = v_{i,j} AND y = c_k} S|}{|S|}}$$

(Rokach and Maimon 2005)

Binary Criteria is used, as the name states, for creating binary decision tree models. This criterion is based on the division of the input attribute into two sub-domains. The binary criterion value can be calculated using the following formula: $\beta(a_i, dom_1(a_i), dom_2(a_i), S)$. Here, a_i is the attribute over sample S and $dom_1(a_i)$ and $dom_2(a_i)$ are sub-domains (Rokach and Maimon 2005).

Twoing Criterion is a criterion that simplifies large classification problems by reviewing the problem as a binary classification problem. When the number of target classes is equal to two, the twoing criterion will be equal to the gini index (Rokach and Maimon 2005). When this is not the case, the twoing criterion will be more accurate than the gini Index. The twoing criterion is defined by the following formula:

$$twoing(a_i, dom_1(a_i), dom_2(a_i), S) = 0.25 * \frac{|\sigma_{a_i \epsilon dom_1(a_i)} S|}{|S|} * \frac{|\sigma_{a_i \epsilon dom_2(a_i)} S|}{|S|} * (\sum_{c_i \epsilon dom(y)} |\frac{|\sigma_{a_i \epsilon dom_1(a_i) AND y = c_i} S|}{|\sigma_{a_i \epsilon dom_1(a_i)} S|} - \frac{|\sigma_{a_i \epsilon dom_2(a_i) AND y = c_i} S|}{|\sigma_{a_i \epsilon dom_2(a_i)} S|}|)^2$$

Orthogonal Criterion (ORT) is used for creating binary decision tree models. According to Rokach and Maimon (2005), the ORT criterion can be calculated using the following formula:

$$ORT(a_i, dom_1(a_i), dom_2(a_i), S) = 1 - cos\theta(P_{y,1}, P_{y,2})$$

In this formula, $\theta(P_{y,1}, P_{y,2})$ is the angle between two vectors. These vectors represent the probability distribution of the target feature in the segments $\sigma_{a_i \epsilon dom_1(a_i)} S$ and $\sigma_{a_i \epsilon dom_2(a_i)} S$. According to Rokach and Maimon (2005), the ORT criterion outperforms the information gain and gini index criteria in some specific cases.

Kolmogorov-Smirnov Criterion is a splitting criteria that solely works for the creation of binary decision tree models. Assuming a binary target attribute $dom(y) = (c_1, c_2)$, the criterion can be defined using the following equation:

$$KS(a_i, dom_1(a_i), dom_2(a_i), S) = |\frac{|\sigma_{a_i \epsilon dom_1(a_i) AND y = c_i} S|}{|\sigma_{y=c_1} S|} - \frac{|\sigma_{a_i \epsilon dom_2(a_i) AND y = c_i} S|}{|\sigma_{y=c_2} S|}|$$

This splitting criterion outperforms the gain ratio criteria (Rokach and Maimon 2005).

AUC-Splitting Criteria is, according to Rokach and Maimon (2005), a splitting criterion that tries to find the maximal area under the convex hall of the ROC-curve (see section 4). The AUC-splitting criteria outperforms other splitting criteria, but it can only be used for binary decision tree models.

Entropy is a measure of uncertainty. The the goal of machine learning is to have as little uncertainty as possible when predicting. The entropy is calculated using the following formula:

$$Entropy(y, S) = \sum_{c_j \epsilon dom(y)} -\frac{|\sigma_{y=c_j} S|}{|S|} * log_2 \frac{|\sigma_{y=c_j} S|}{|S|}$$

(Rokach and Maimon 2005)

Gini Index is a measure that shows how often a randomly chosen element is labeled incorrectly. This is expressed in impurity, where the gini index can vary from pure with a value of 0, to impure with a value of 0.5. The gini index is defined by the following formula:

$$Gini = 1 - \sum_j p_j^2$$

where p is the probability of the instance being in class j (Rokach and Maimon 2005). According to Raileanu and Stoffel (2004), the gini index will differ from the entropy in only two percent of all cases. This means that these two splitting criteria are very similar.

Cross-Entropy is a measure that is used in classification problems. It is a measure of uncertainty that is similar to entropy. This measure, however, takes into account what the baseline performance of the problem is when calculating the entropy.

2.1.2. Stopping Criteria

The training of a decision tree model is continued until one of the stopping criteria has been reached. According to Rokach and Maimon (2005) and Mantey (2019), the following stopping criteria can be used when training a decision tree model:

- All instances in the leaf have the same value for the target feature; no further splitting is necessary.
- The maximum depth of the decision tree model has been reached.
- The number of instances in a node is less than the minimum required number of instances for parent nodes.
- When splitting the parent node, the number of instances in the leaves would be less than the minimum required number of instances for leaves.
- The used splitting criterion is smaller than or equal to a certain threshold.

2.1.3. Parameter Tuning for Decision Tree Models

The tuning of parameters for a decision tree model ensures that the decision tree model fits the data as well as possible. When the decision tree model has achieved an optimal fit to the data, it will present the best possible predictions. Below are some of the parameters that can be adjusted for tuning the decision tree models.

Maximum Depth is the maximum depth of a decision tree model. Generally, the performance of a decision tree model improves as the maximum depth increases. Galarnyk (2019) recommends starting with a small value for the maximum depth and raising this value until the performance no longer improves significantly.

Minimum Samples Split is the minimum number of instances required to split an internal node. When the number of instances in a node is less than this threshold, the node is considered a leaf. The default value for the minimum number of instances required to split an internal node is equal to two. A large value for the minimum number of instances required to split an internal node, may result in overfitting (Mithrakumar 2019).

Minimum Samples Leaf is the minimum number of instances required in a leaf node. According to Probst *et al.* (2019), the default value for the minimum number of instances in a leaf node is equal to one for classification models and two for regressions models.

Minimum Weight Fraction Leaf is the minimum weighted fraction of the total sum of the weights required to be at a leaf node. This is only applicable when the instances have a weight other than one.

Number of Features is the size of the random subset of features which is used to create a decision tree model. The default value for the number of features is equal to $\sqrt{M}$ where M represents the total number of features. According to Bernard (2009), using this value is a reasonable way to induct near-optimal decision tree model.

Maximum Leaf Nodes creates a decision tree model with a maximum number of leaf nodes in best-first fashion. The maximum number of leaves is, by default, equal to $2^{\text{maximum depth}}$. A smaller number of leaves leads to a decision tree model in which not every branch can reach the length of the maximum depth.

Minimum Impurity Decrease is a threshold that determines whether a split will be carried out. The split is carried out when it induces a decrease of impurity greater than or equal to the value of this threshold. The best way to tune this parameter is by plotting the decision tree model and looking at the gini index. Looking at the splits that lead to a gini index of 0, it is possible to determine if the classification is acceptable. If the classification is not acceptable, the minimum impurity decrease can be increased so that the node will not be split further than the given threshold.

CCP Alpha is the complexity parameter that is used for cost-complexity pruning. This is a value that is used to reduce overfitting in a decision tree model. The value chosen is the maximum value of complexity within cost-complexity pruning.

2.2. Predictions Using Decision Tree Models

After a decision tree model is created, it can be used to the predict the labels of given instances. Starting at the root node of the decision tree model, the value for a certain feature of the instance is assessed using the conditional statement for the root node. If the conditional statement returns the value *true*, the instance will move to the next node on the right where the process is repeated for the following nodes. If the conditional statement returns the value *false*, the instance will move to the next node on the left where the process is repeated for the following nodes. The process is repeated until the instance reaches a leaf where it is assigned a certain label.

2.3. Conclusion

Given the information regarding decision tree models mentioned above, it is possible to answer the following sub-question: *What are decision tree models and how is a decision tree model optimally developed?* The training of a decision tree model is carried out using induction. During the training of a decision tree model, stopping and splitting criteria need to be used. After the training of a decision tree model,

certain parameters can be tuned. Some of these parameters have a larger influence on the performance of the decision tree model than others.

3. RANDOM FOREST MODELS

Random forest models consist of numerous, unique decision tree models, making it a forest of decision trees. These decision tree models operate ensemble, which means that a random forest model is an ensemble machine learning method. Due to the fact that the decision tree models are used ensemble, random forest models provide a higher accuracy and carry a smaller risk of overfitting than a single decision tree model.

3.1. Training Random Forest Models

The training of random forest models can be explained in terms of the training of decision tree models. The training of random forest models is simply the training of a given number of unique decision tree models. Every decision tree model in a random forest model is created using a different bootstrapped data set and a random subset of features. This ensures that all of the decision tree models in a random forest model are unique and will, therefore, not always give the same output.

3.1.1. Parameter Tuning for Random Forest Models

The tuning of parameters for a random forest model ensures that the random forest model fits the data as well as possible. When the random forest model has achieved an optimal fit to the data, it will present the best possible predictions. The parameters for random forest models are largely the same as the parameters for decision tree models as a random forest contains numerous decision tree models with equal parameters. The the number of features and the maximum depth, for instance, are equal for all of the decision tree models. These parameters have been described in section 2.1.3. The bootstrap sample size parameter is equal for all of the decision tree models as well, but since it is only applicable to decision tree models in random forest models, it is mentioned in this section. Additional parameters that can be changed for a random forest model include the number of decision tree models that are incorporated in it and the number

Bootstrap Sample Size is the size of the sample used to create a decision tree model. Per definition, the bootstrap sample size is equal to the size of the training data set (Wager and Athey 2018).

Number of Decision Tree Models in a random forest model drastically affects the performance of a random forest model. A random forest model with one decision tree model will give the same result as a decision tree model whereas a random forest model with one thousand decision tree models for example, will combine the results from the various decision tree models to give a more accurate prediction. Based on Probst and Boulesteix (2017), increasing the number of decision tree models will not negatively affect the performance of the model. Empirical results show that for most of the examined data sets, the biggest performance gain is achieved when training the first one hundred decision tree models.

3.2. Predictions Using Random Forest Models

Just as the training of random forest models, predictions using random forest models rely heavily on the predictions made by decision tree models. When making a prediction using the random forest model, every decision tree model in the random forest model is consulted. In doing so, every decision tree model makes a prediction for a given instance. The prediction of the random forest model is equal to the most frequently-occurring prediction among the decision tree models. This sequence of steps is then repeated for all of the instances in the data set. The parallel use of decision tree models leads to a better performance than when a single decision tree is used due to the wisdom of crowds (Yiu 2019).

3.3. Conclusion

Given the information about random forest mentioned above, it is possible to answer the following sub-question: *What are random forest models and how is a random forest model optimally developed?* Random forest models are trained by training the decision tree models that it incorporates. These decision tree models are trained using a unique, bootstrapped set of instances. Random forest models can be tuned by tuning the parameters for the decision tree models, as well as the additional parameters for random forest models. When making a prediction with the trained random forest model, the decision tree models are consulted. The most frequently-occurring prediction among the decision tree models then becomes the prediction for the random forest model.

4. EVALUATION OF CLASSIFICATION MODELS

There are multiple different evaluation metrics that are used to evaluate classification models. A few of the most well-known ones are mentioned below.

4.1. Confusion Matrix

Confusion matrices can be used for the evaluation of either binary classification models or multi-class classification models. Confusion matrices are matrices that contains every combination of predicted versus actual values. The actual values are spread along the rows of the matrix and the predicted values are spread among the columns of the matrix. Every element (x, y) of the matrix contains the number of instances that have actual value x and predicted value y. This means that the diagonal of the matrix contains the number of instances that have been predicted correctly. Confusion matrices help to indicate which instances have been predicted incorrectly and which value these instances have been given.

For binary classification models, the different values are described as positive and negative. The different cells of the confusion matrix can be described as follows:

- Positive and predicted positive: True Positive (TP)
- Negative and predicted positive: False Positive (FP)
- Negative and predicted negative: True Negative (TN)
- Positive and predicted negative: False Negative (FN)

It can be observed that many other evaluation metrics for binary classification models can be described in terms of true and false positives and negatives.

For multi-class classification models, every row and column represents a possible class to be predicted. This means that the confusion matrix for multi-class classification models is significantly larger than the confusion matrix for binary classification models. Furthermore, the labels of true and false positives and negatives cannot be used for multi-class classification models.

4.2. Precision

Precision is an evaluation metric for binary classification models (Hossin and Sulaiman 2015). According to Davis and Goadrich

(2006), the precision of a model can be calculated using the following formula:

$$\text{precision} = \frac{TP}{TP + FP}$$
$$= \frac{\text{number of instances with class X predicted correctly}}{\text{number of instances with predicted class X}}$$

The precision indicates how many of the instances with predicted class X, actually belong to class X. Considering the formula mentioned above, the precision can also be calculated per class for multi-class classification models.

4.3. Recall

Recall is an evaluation metric for binary classification models (Hossin and Sulaiman 2015). According to Davis and Goadrich (2006), the recall of a model can be calculated using the following formula:

$$\text{recall} = \frac{TP}{TP + FN}$$
$$= \frac{\text{number of instances with class X predicted correctly}}{\text{number of instances with class X}}$$

The recall indicates how many of the instances with class X, are predicted as class X. This is equal to the True Positive Rate. Considering the formula mentioned above, the recall can also be calculated per class for multi-class classification models.

4.4. Accuracy

The accuracy can be used for the evaluation of multi-class and binary classification models. According to Hossin and Sulaiman (2015), the accuracy of a model can be determined using the following formula:

$$\text{accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$
$$= \frac{\text{number of instances predicted correctly}}{\text{number of instances}}$$

This means that the accuracy indicates which proportion of the instances was classified correctly.

4.5. Error Rate

T error rate can be used for the evaluation of multi-class and binary classification models. According to Hossin and Sulaiman (2015), the error rate of a model can be determined using the following formula:

$$\text{error rate} = \frac{FP + FN}{TP + FP + TN + FN}$$
$$= \frac{\text{number of instances predicted incorrectly}}{\text{number of instances}}$$

This means that the error rate indicates which proportion of the instances was classified incorrectly.

4.6. Sensitivity

The sensitivity is an evaluation metric for the evaluation of binary classification models (Hossin and Sulaiman 2015). According to Lalkhen and McCluskey (2008), the sensitivity of a classification model can be calculated using the following formula:

$$\text{sensitivity} = \frac{TP}{TP + FN}$$
$$= \frac{\text{number of instances with class X predicted correctly}}{\text{number of instances with class X}}$$

This is equal to the recall and true positive rate evaluation metrics.

4.7. Specificity

The specificity is an evaluation metric for the evaluation of binary classification models (Hossin and Sulaiman 2015). According to Lalkhen and McCluskey (2008), the specificity of a classification model can be calculated using the following formula:

$$\text{specificity} = \frac{TN}{TN + FP}$$
$$= \frac{\text{number of instances with class Y predicted incorrectly}}{\text{number of instances with class Y}}$$

This means that the specificity is equal to the proportion of negative instances that is classified correctly. This is equal to the true negative rate.

4.8. ROC-Curve

The ROC-curve is an evaluation technique for binary classification models. This is a plot where the sensitivity is plotted as function of the aspecificity, or $1 -$ specificity. The ROC-curve can also be made by plotting the true positive rate against the false positive rate (Davis and Goadrich 2006).

4.9. Area Under the Curve

The area under the curve, or AUC, is an evaluation technique for binary classification models (Hossin and Sulaiman 2015). The AUC is equal to the area under the ROC-curve which represents the probability of ranking a positive instance higher than a negative instance (Davis and Goadrich 2006).

4.10. Precision-Recall-Curve

The precision-recall-curve is an evaluation technique for the evaluation of binary classification models. The curve that is mentioned in the name is the plot of the precision as a function of the recall. This curve is a straight line from $(0, 0)$ to $(1, 1)$ when the model does not perform better than a random guess. When the curve is above the previously mentioned line, the model is better than a random guess. (Brownlee 2020)

4.11. Out-of-Bag Sampling

The out-of-bag sampling error is an evaluation metric that is used for the evaluation of multi-class classification models that use ensemble learning. This evaluation metric evaluates the predictions of instance x while only using the base models that are not trained on instance x (Bhatia 2019).

4.12. Conclusion

Using the information about evaluation techniques mentioned above, it is possible to answer the following sub research question: *How can the performance of random forest models be evaluated to find the best suited random forest model?* Some of the evaluation metrics mentioned above can be used to evaluate binary classification models, while others can also be used for the evaluation of multi-class classification models. Furthermore, some of the evaluation metrics focus on the instances that have been classified correctly, while some others focus on the instances that have been classified incorrectly. Which evaluation metric can be used when tuning a random forest model for a certain data set, depends on the data set that is being modelled.

5. METHOD

To ensure that the research is reproducible, the approach of the research has been elaborated below.

5.1. Literature Study

Initially, a literature study was conducted to obtain the necessary information on random forest models, decision tree models and the evaluation of classification models. This ensures that all of the options for the different models are understood and can be explored during the development of the final random forest model. Decision tree models are the main element of random forest models and must, therefore, be fully understood in order to be able to develop a random forest model. To be able to determine which developed random forest model is the best, the evaluation of classification models must be researched. Once the literature study is complete, a random forest model can be created and the optimal random forest model can be selected.

5.2. Development of Algorithm

Once the literature study is complete, decision tree and random forest models are fully understood. This enables the conductors of the research to start the development of the random forest model. The development of the random forest model can be broken up into the following four segments: the programming of the training of decision tree models, the programming of the predictions using decision tree models, the programming of the training of random forest models and the programming of the predictions using random forest models.

5.2.1. Programming of Training of Decision Tree Models

The programming of the training of decision tree models is the first element of the complete algorithm that can be developed. This is due to the fact that the training of decision tree models is the most important element of the development of the random forest model. Without the algorithm for the training of decision tree models, no further progress can be made.

There are some variable elements of the training of decision tree models. One of these elements is which stopping criteria is used. For the decision tree models in the developed random forest model, the chosen stopping criteria are as follows:

- The maximum tree depth has been reached
- The remaining data contains less than two instances, this is the same as being pure
- The remaining data is pure; the instances in the data set have the same label

Furthermore, the splitting criterion is also variable for decision tree models. The chosen splitting criterion is the entropy. With the determined stopping and splitting criteria, the programming of the training of decision tree models can be completed.

5.2.2. Programming of Predictions Using Decision Tree Models

To ensure that the training of decision tree models works, the element of the algorithm that makes predictions using decision tree models, needs to be developed. Once this has been done, there is more confidence that the decision tree models are trained correctly and can then be used for random forest models.

5.2.3. Programming of Training of Random Forest Models

Once the algorithm for a decision tree model has been programmed, the training of random forest models can be programmed. The training of a random forest model is the process of creating a given number of decision tree models. To ensure that all of the decision tree models in the random forest model are unique, it has been decided that all of the decision tree models will be created using a unique, bootstrapped data set. This data set is a bootstrap sample of instances from the original data set. Furthermore, it has been decided that this bootstrapped data set will contain a certain number of random features rather than all of the available features. This ensures extra randomness during the creation of decision tree models. This also reduces the computation time of the random forest model while the accuracy will barely be affected.

5.2.4. Programming of Predictions Using Random Forest Models

Once the training of the random forest model works, the element of the algorithm that makes predictions using random forest models, can be developed. Making predictions using random forest models relies heavily on the making of predictions using the decision tree models in the random forest model. Once the programming of this element of the algorithm, the development of the algorithm itself will be complete.

5.3. Preparation of Data Set

In order to be able to validate the performance of the developed random forest model, a data set needed to be selected. One of the most well-known data sets in the field of machine learning is the MNIST Handwritten Digit data set. This data set contains thousands of handwritten digits from zero to nine. A further description of the data set is given in Appendix A. Considering the percentages in Table 3 in Appendix A, it is assumed that class imbalance will not cause any issues for this research as all of the different labels occur in approximately the same proportions.

There are multiple reasons why this data set has been chosen to be used for the testing of the developed random forest model. The data set has been pre-processed which means that there are no missing values or labels. Furthermore, the data set contains only numeric features which means that the data set is slightly easier to interpret than data sets that contain a mixture of categorical and numeric features. Another reason for choosing the MNIST data set is the fact that it requires a multi-class classification model, such as a random forest model, in order to classify the handwritten digits. Lastly, another reason for the MNIST data set is that a significant amount of research has been done using the MNIST data set. This ensures that there is a lot of literature available on the topic and that there are many classification models that the developed model can be compared to.

When using a data set for machine learning, it needs to be split into a training set, a validation set and a testing set. By default, the MNIST data set is split into a training set and a testing set using the distribution of 6:1 respectively. To create a validation set, the default training set was split into a training set and a validation set using the distribution 9:1. Table 1 shows the number of instances per subset.

Table 1 Distribution of Data Among Subsets

Subset	Number of Instances
Training Set	54,000
Validation Set	6,000
Testing Set	10,000

5.4. Tuning of Random Forest Model

After the preparation of the data set, the developed random forest model can be tuned so that it fits the given data set best. This

tuning can be done by slightly changing the parameters for every iteration. The parameters that can be adjusted during the training of the random forest model are:

- Maximum depth of the decision tree models
- Bootstrap sample size for every decision tree model
- Size of the random subset of features for every decision tree model
- Number of trees in the random forest model

Parameters such as the class weight for random forest models and the minimum weight fraction for a leaf node for decision tree models are not applicable to the fitting of a model to the MNIST data set as these parameters are used when there is class imbalance present in the data set. As mentioned before, that is not the case for this research.

Next to the parameters mentioned above, there are other parameters that can be tuned according to the literature research. Due to time limitations for this study, parameters such as minimum samples per leaf and minimum samples per split will not be tuned and will be left for future research. These parameters will be used with a default value that was found in the literature study.

To determine which parameters are used to create the best possible random forest model, evaluation metrics have to be used. According to the literature study, the accuracy and error rate metrics are best for the evaluation of multi-class classification models such as this one. Confusion matrices will also be used to evaluate the developed random forest models. There are other evaluation metrics which can be used for multi-class classification models, but due to time limitations, these will not be used to evaluate models in this study.

Relying on the results of numerous tests, the maximum depth of the decision tree models does not greatly affect the accuracy of the random forest model beyond a maximum depth of ten. This is shown in Figure 1. Therefore, a maximum depth of ten will be used in the final random forest model. It can be seen that the Python scikit-learn package produces a higher accuracy for all of the different bootstrap sample sizes and maximum depths.

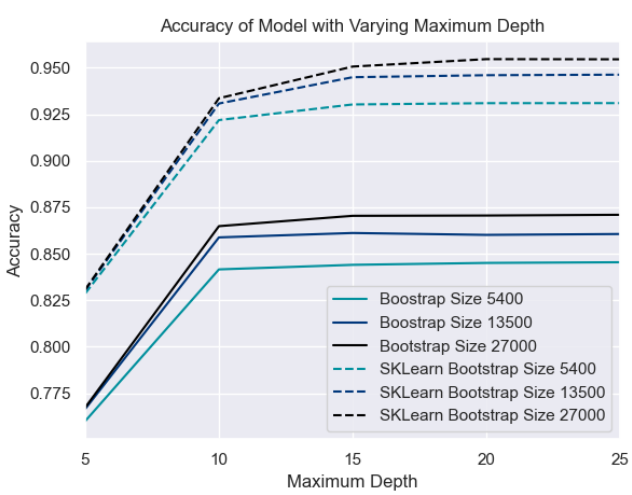


Figure 1 Plot of the accuracy of the random forest model with a varying maximum depth for the decision tree models.

For the rest of the parameters that can be tuned, the accuracy is increased when the value for the parameter is increased. At some point, however, the parameter value exceeds a certain value where the increase in accuracy is no longer significant. Figure 2 shows that the accuracy no longer improves greatly when the number of trees exceeds fifty. Therefore, the final random forest model will contain fifty decision tree models. It can be observed that the developed random forest model is more accurate than the Python scikit-learn model for all of the different bootstrap sample sizes and numbers of trees.

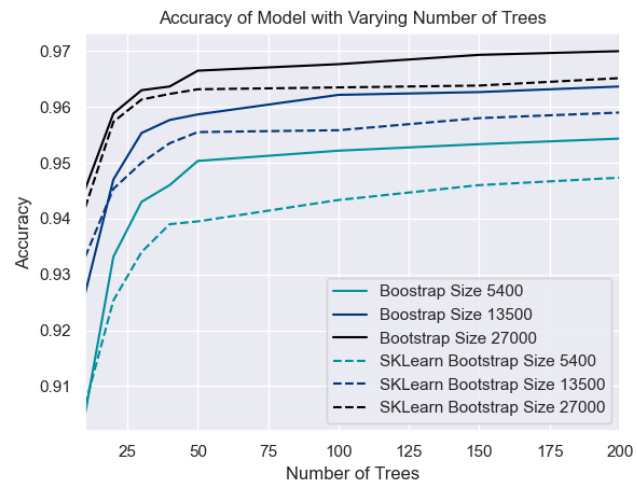


Figure 2 Plot of the accuracy of the random forest model with a varying number of decision tree models.

For the size of the random subset of features, Figure 3 shows that the accuracy significantly improves until fifty features. Therefore, the final random forest will be created using fifty random features per decision tree model. This plot does not provide much clarity regarding the difference between the developed random forest model and the random forest model from the Python scikit-learn package. Therefore, it can be concluded that there is no significant difference between the two random forest models.

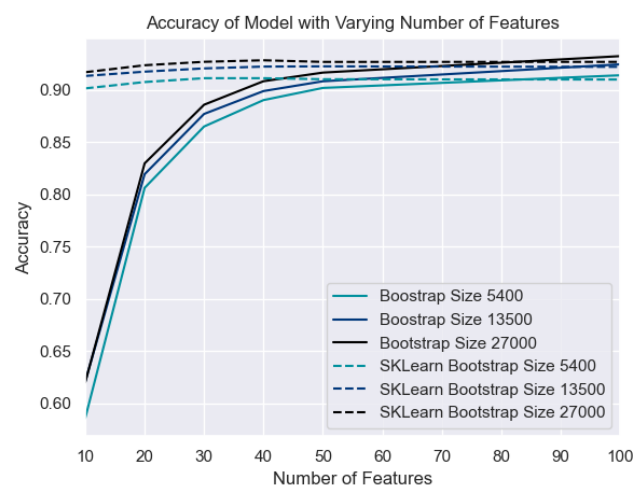


Figure 3 Plot of the accuracy of the random forest model with a varying number of features.

Figures 1, 2 and 3 show the influence of parameters with varying values for the bootstrap sample size. The figures show that the

accuracy is always the highest for the largest bootstrap sample size that was tested, 27,000. Due to computational limitations, random forest models with a larger bootstrap sample size up to 60,000, could not be tested. Therefore, a bootstrap size of 27,000 instances will be used in the final random forest model

The best combination of parameters will give a random forest model with an optimal value for accuracy.

5.5. Testing of Random Forest model

Once the random forest model has been completed and the parameters have been tuned, the optimal random forest model for the data set has been found. This optimal random forest model can then be applied to the testing set. The application of the random forest model to the testing set will give the final results of the random forest model. These results will show how good the random forest model is at classifying unseen data. The final results will be expressed in the accuracy and the error rate as these are the best evaluation metrics for a multi-class classification. Furthermore, the final results will be shown in a confusion matrix to determine which of the handwritten digits are most likely to be classified incorrectly by the random forest model. To further evaluate the performance of the random forest model, it will be compared to a common random forest model. This random forest model is provided by the Python scikit-learn package. Training this model on the same training data will show how the developed random forest model compares to another, well-known random forest model.

6. RESULTS

The results of this research can broken up into two sections. The first section describes the developed random forest model and the second section describes the performance of the developed random forest model on the MNIST data set.

6.1. Model Description

The description of the developed random forest model can be split into a training algorithm and a prediction algorithm. Both of the elements of the model have been described below.

6.1.1. Training

The training of a random forest model consists of creating a given number of decision tree models. For every decision tree model, a unique bootstrapped data set is assembled. The size of this bootstrapped data set is entered as a parameter. This bootstrapped data set is also limited to a certain number of random features of the original data set. The bootstrapped data set is used to create a new, unique decision tree model. This decision tree model is then added to the random forest model before another decision tree model is created. The flowchart in Figure 4 illustrates how the described steps lead to the development of the final random forest model. This flowchart is shown in greater detail in Appendix B.

The most essential aspect of the training of a random forest model is the training of the individual decision tree models. Using the previously mentioned bootstrapped data set, the following steps are carried out using recursion to create an individual decision tree model.

1. If the remaining data contains less than five instances or the maximum depth of the decision tree model has been reached, a leaf will be created. The leaf will classify the instances in the leaf with the label that occurs most frequently in the data. If the remaining data consists of at least five instances and the maximum depth of the decision tree model has not been reached, a new node will be created using the following steps.

2. Every node in the decision tree model has a conditional statement which returns true or false for a given instance. In this step, all of the possible conditional statements that can be used to split the remaining data, are defined. This is done determining every unique value for the all of the features in the data set.

3. After determining all of the possible splits, the entropy is calculated for every split to determine which split will lead to the best result. The split with the lowest value for entropy will be used to create the next node in the decision tree model.

4. The remaining data will be split according to the best possible split from the previous step.

5. For each of the two remaining sets of data after the split, a leaf will be created or a sub-decision tree model will be added recursively. This recursive addition method of sub-decision tree models is repeated until all of the data is split into leaves.

The flowchart in Figure 5 illustrates how the described steps lead to the creation of a decision tree model in a given random forest model. This flowchart is shown in greater detail in Appendix C.

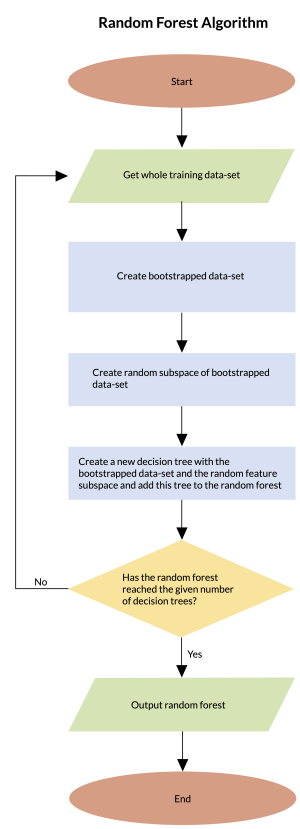


Figure 4 Flowchart of the training algorithm of random forest models

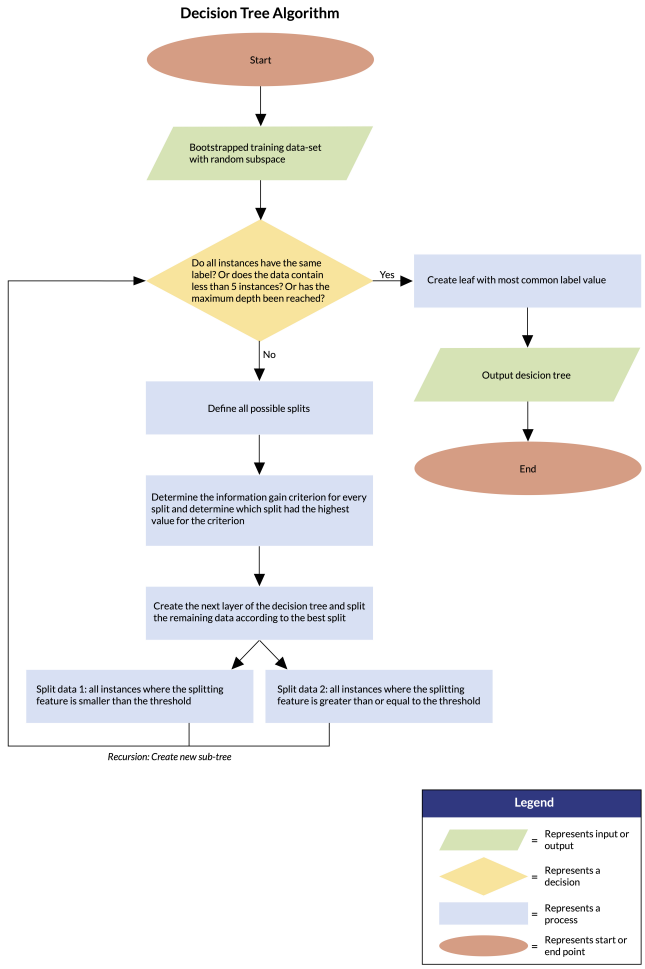


Figure 5 Flowchart of the training algorithm of decision tree models

6.1.2. Prediction

When an unlabeled data set is fed into the random forest model, the algorithm can be used to predict the label given a number of input values. For every instance in the given data set, a prediction is made using every decision tree model in the random forest model. The most frequently occurring label among the decision tree models is considered to be the final prediction for the label of the given instance.

For every decision tree model that is used to predict the label for a given instance, the given instance starts at the root node. At the root node, the value for a certain feature of the instance is assessed using the conditional statement for the root node. If the conditional statement returns the value *true*, the instance will move to the next node on the right where the process is repeated for the following nodes. If the conditional statement returns the value *false*, the instance will move to the next node on the left where the process is repeated for the following nodes. The process is repeated until the instance reaches a leaf where it is assigned a certain label. This predicted label is returned to the random forest model.

6.2. Model Performance

As the developed random forest model can only be tested using a data set, a sample data set has been chosen. The MNIST data set will be used to test the developed random forest model. A description of this data set can be found in Appendix A.

The baseline for the significance of the model is calculated using the following formula:

$$\text{baseline} = \frac{1}{\text{number of possible classes}} = \frac{1}{10} = 0.1 = 10\%$$

This means that the model performs better than a random guess when an accuracy of at least 10% is reached.

The final random forest model uses the parameters that have been listed in Table 2.

Table 2 Distribution of Data Among Subsets

Parameter	Parameter Value
Bootstrap Sample Size	27,000
Number of Trees	50
Number of Features	50
Maximum Tree Depth	10

The random forest model that is developed using the parameters in Table 2, achieves an accuracy of 93.99% on the testing set. This means that the model is relevant as this accuracy is higher than the baseline of 10%. An accuracy of 93.99% on the testing set means than an error rate of 6.01% has been achieved. In comparison to the random forest model with the same parameters from the Python scikit-learn package, the accuracy is 1.06% higher. This result is rather insignificant when the difference in computation time is considered. The time to compute a model of the same size is significantly longer when using the developed model than when the model from the Python scikit-learn package is used.

Figure 6 shows the confusion matrix of the results. The actual labels of instances are spread over the rows, while the predicted labels of instances are spread over the columns of the matrix. In this confusion matrix, it can be observed that a large majority of the instances are on the diagonal of the matrix. This means that these instances have been classified correctly. Outside of the diagonal, there are no clear indications of frequently-occurring misclassifications. However, it can be observed that handwritten fours are sometimes classified as nines and that fives are sometimes classified as threes. These frequencies are rather insignificant considering the number of instances that was classified.

7. CONCLUSION

Using the research results, it is possible to answer the following research question: *To what extent is it possible to use Python to program a random forest classification model from scratch and implement it on a data set to achieve the best possible result?* This research answers the research question as it has been proven possible to develop a random forest classification model from scratch in Python. The developed algorithm was tested on the MNIST data set. Even though the MNIST data set only contains numerical attributes, the developed algorithm has been programmed so that it can handle data sets with numerical and/or categorical features. the can be used on data sets with numerical, as well as categorical features. The developed algorithm can be made to fit a data set by tuning the following parameters:

- The bootstrap sample size for every decision tree model
- The size of the random subset of features for every decision tree model

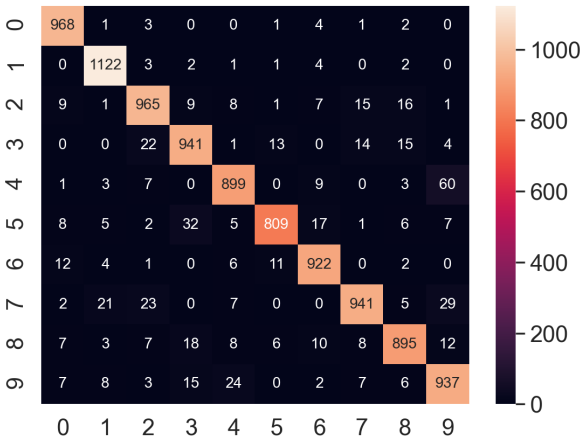


Figure 6 Confusion matrix of the predictions of the test set using the chosen random forest model.

- The maximum depth of every decision tree model
- The number of decision tree models that the random forest contains

By tuning the parameters mentioned above, it was possible to fit a random forest model to the MNIST data set. The random forest model was used to achieve an accuracy of 93.99% on the withheld test set. This accuracy is 1.06% higher than the random forest model from the Python scikit-learn package with the same parameters. However, no conclusion can be made regarding the comparison with the random forest models from the Python scikit-learn package. Figures 1, 2 and 3 show that the accuracy is lower for some parameters, and higher for others.

Disregarding the previously mentioned result, the random forest model that was used to fit the MNIST data set was not optimal. Due to the fact that this research was limited by computing power and time, the optimal result could not be reached. The possible adjustments that could optimize the tuning of the random forest model are elaborated in the discussion section.

8. DISCUSSION

The results of this research can be improved in a number of ways. These improvements could not be made due to a lack of resources and time. A few possible improvements have been elaborated in the following sections.

8.1. Reduction of Computation Time

In this research, we came across some obstacles when we were trying to run several different random forest models. The computation of these models took an extremely long time, especially compared to the computation of the same model using the Python scikit-learn package. Due to the time restrictions for this project, we did not find out why the difference in computation time was so significant. An option for future research is, therefore, to reduce the computation time.

Due to the fact that the computation of a random forest model took a very long time, we were unable to use very large parameter values while fitting the model to the MNIST data set. When future research leads to a reduction in computation time, it may be possible to achieve a better performance on the MNIST data set when fitting a random forest model with larger parameter values. Fitting a random forest model using larger parameter values is, therefore, another topic which can be investigated in future research.

8.2. Tuning with Other Parameters

Another potential topic for future research is the influence of other parameters on the performance of random forest models. The literature study for this research uncovered that there are far more parameters for random forest and decision tree models than that are tuned in this research. This is also a result of the computing power and time limitations. Therefore, one of the potential topics for future research is the investigation of parameters such as the minimum number of instances per split, the minimum number of instances per leaf and the minimum impurity decrease.

8.3. Tuning with Other Splitting Criteria

In the developed random forest model, the decision tree models use the entropy as a splitting criterion. The literature study for this research, however, showed that there are many more splitting criteria. Some of these splitting criteria may result in better performance for the random forest model. Due to time limitations for this research, these options have not been explored. Therefore, this is also a potential topic for future research.

8.4. Use of Other Evaluation Metrics

In this research, accuracy, error rate and confusion matrices were used to evaluate the performance of the random forest model. Based on the values of these metrics, decisions were made regarding the further tuning of the random forest model. In some cases, it may have been the case that different evaluation metrics would have resulted in different decisions. For that reason, another potential topic for future research is the use of other evaluation metrics for the tuning of the random forest model.

Of the evaluation metrics that have been investigated in the literature study of this research, only one metric is applicable to the tuning of the random forest model to fit the MNIST data set. This evaluation metric is the out-of-bag sample score. Using the out-of-bag sample score to tune the random forest model may result in better decisions for the tuning random forest model and is, therefore, worth investigating.

8.5. Application to Other Data Sets

In this research, a random forest model was created from scratch. This model was then tested on the MNIST data set. This data set includes only numerical features. To ensure that the developed random forest model works on every type of data set, a potential topic for future research is to fit the developed random forest model to other data sets. These other data sets can include data sets with numerical as well as categorical attributes.

8.6. Pruning of Decision Tree Models

The pruning of decision tree models is a process where sub-trees are replaced by leaves when the estimated error of a single leaf is lower than the estimated error of the sub-tree. This ensures that the decision tree models do not overfit, which can lead to a better performance on unseen data. This process, however, adds to the computation time of the training of decision tree models. Due to the fact that computation times were already an issue in this study without pruning, it was decided not to apply pruning to the developed random forest. However, pruning may lead to better performance on unseen data which is why it is another potential topic for future research.

LITERATURE CITED

Bernard, A., Heutte, 2009 Influence of hyperparameters on random forest accuracy **5519**: 171–180.

Bhatia, N., 2019 What is out of bag (oob) score in random forest?

Bradford, J. P., C. Kunz, R. Kohavi, C. Brunk, and C. E. Brodley, 1998 Pruning decision trees with misclassification costs. In *European Conference on Machine Learning*, pp. 131–136, Springer.

Brownlee, J., 2020 Roc curves and precision-recall curves for imbalanced classification.

Davis, J. and M. Goadrich, 2006 The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pp. 233–240.

Famishedrover, 2017 Getting familiar with mnist.

Galarnyk, M., 2019 Understanding decision trees for classification (python).

Hossin, M. and M. Sulaiman, 2015 A review on evaluation metrics for data classification evaluations. International Journal of Data Mining & Knowledge Management Process **5**: 1.

Lalkhen, A. G. and A. McCluskey, 2008 Clinical tests: sensitivity and specificity. Continuing Education in Anaesthesia Critical Care & Pain **8**: 221–223.

Mantey, S., 2019 Coding a decision tree classifier from scratch in python.

Mithrakumar, M., 2019 How to tune a decision tree?

Perner, P., 2015 pp. 57–88 in *Decision Tree Induction Methods and Their Application to Big Data*.

Probst, P. and A.-L. Boulesteix, 2017 To tune or not to tune the number of trees in random forest. The Journal of Machine Learning Research **18**: 6673–6690.

Probst, P., M. N. Wright, and A.-L. Boulesteix, 2019 Hyperparameters and tuning strategies for random forest. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **9**: e1301.

Raileanu and Stoffel, 2004 Theoretical comparison between the gini index and information gain criteria. Annals of Mathematics and Artificial Intelligence **41**: 77–93.

Rokach, L. and O. Maimon, 2005 Decision trees. In *Data mining and knowledge discovery handbook*, pp. 165–192, Springer.

Wager, S. and S. Athey, 2018 Estimation and inference of heterogeneous treatment effects using random forests. Journal of the American Statistical Association **113**: 1228–1242.

Yiu, T., 2019 Understanding random forest.

APPENDIX

A. DESCRIPTION OF THE DATA SET

The data set that has been chosen for the testing of the model is the MNIST data set. The data consists of seventy thousand images containing a handwritten digit. A few of these handwritten digits are shown in Figure 7. The MNIST data set represents seventy thousand images of twenty-eight pixels by twenty-eight pixels. This means that the MNIST data set has seventy thousand rows of 784 columns where every column represents a certain pixel in the images. The last column of the data set contains the label of the image, or in this case, which handwritten number is depicted in the image. The label column of an instance can contain a value between zero and nine. Every column that corresponds to a pixel can contain a value between zero and 255.

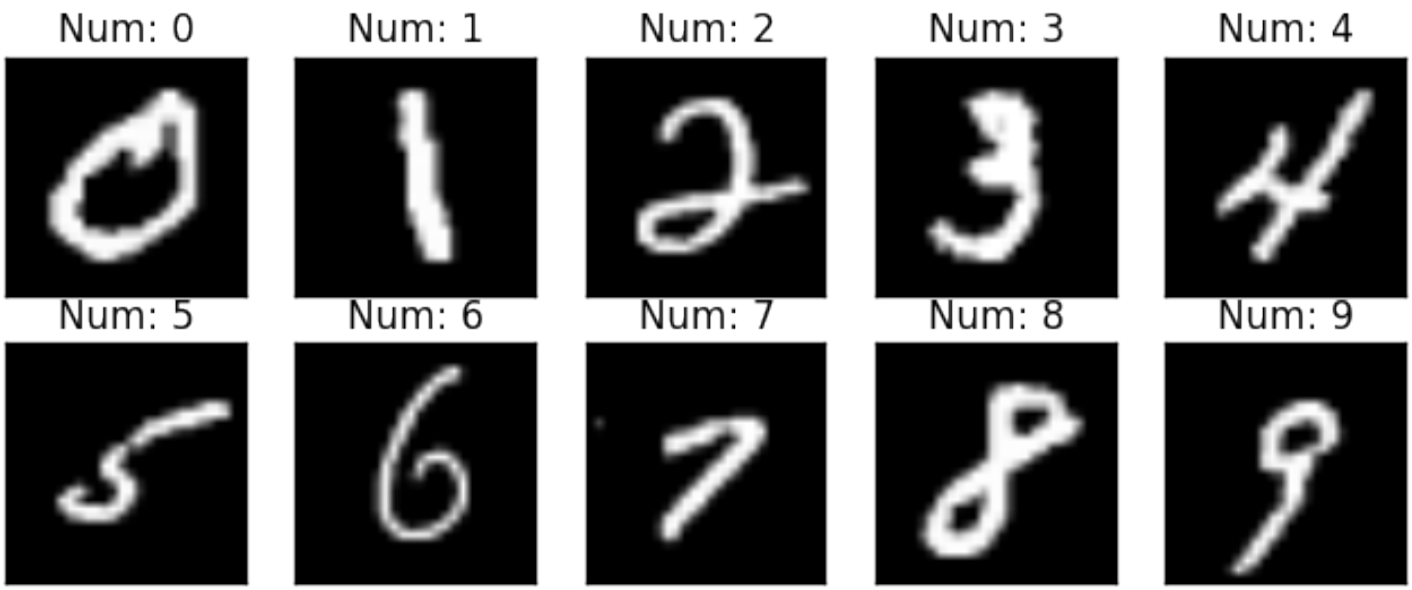


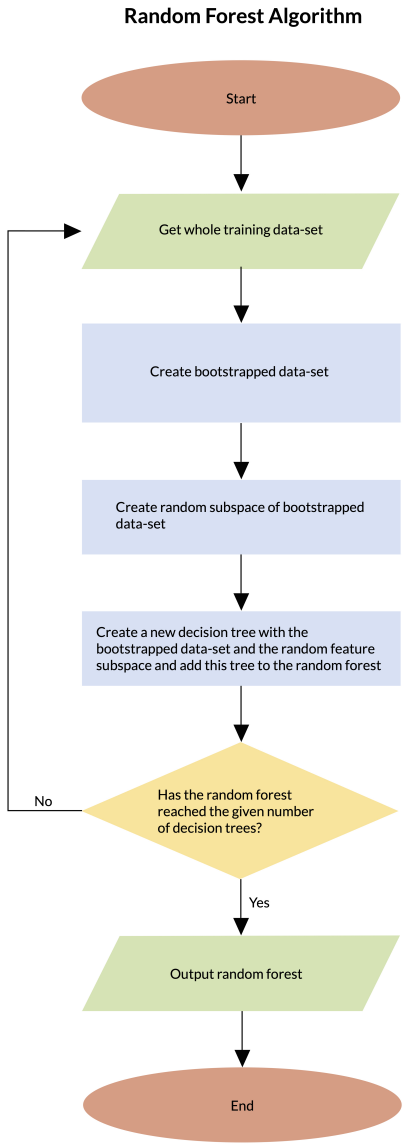
Figure 7 A small sample from the handwritten digits in the MNIST data set (Famishedrover 2017)

By default, the MNIST data set is split into a training set and a testing set using the distribution of 6:1 respectively. The distribution of the different digits in the default training set is described in Table 3.

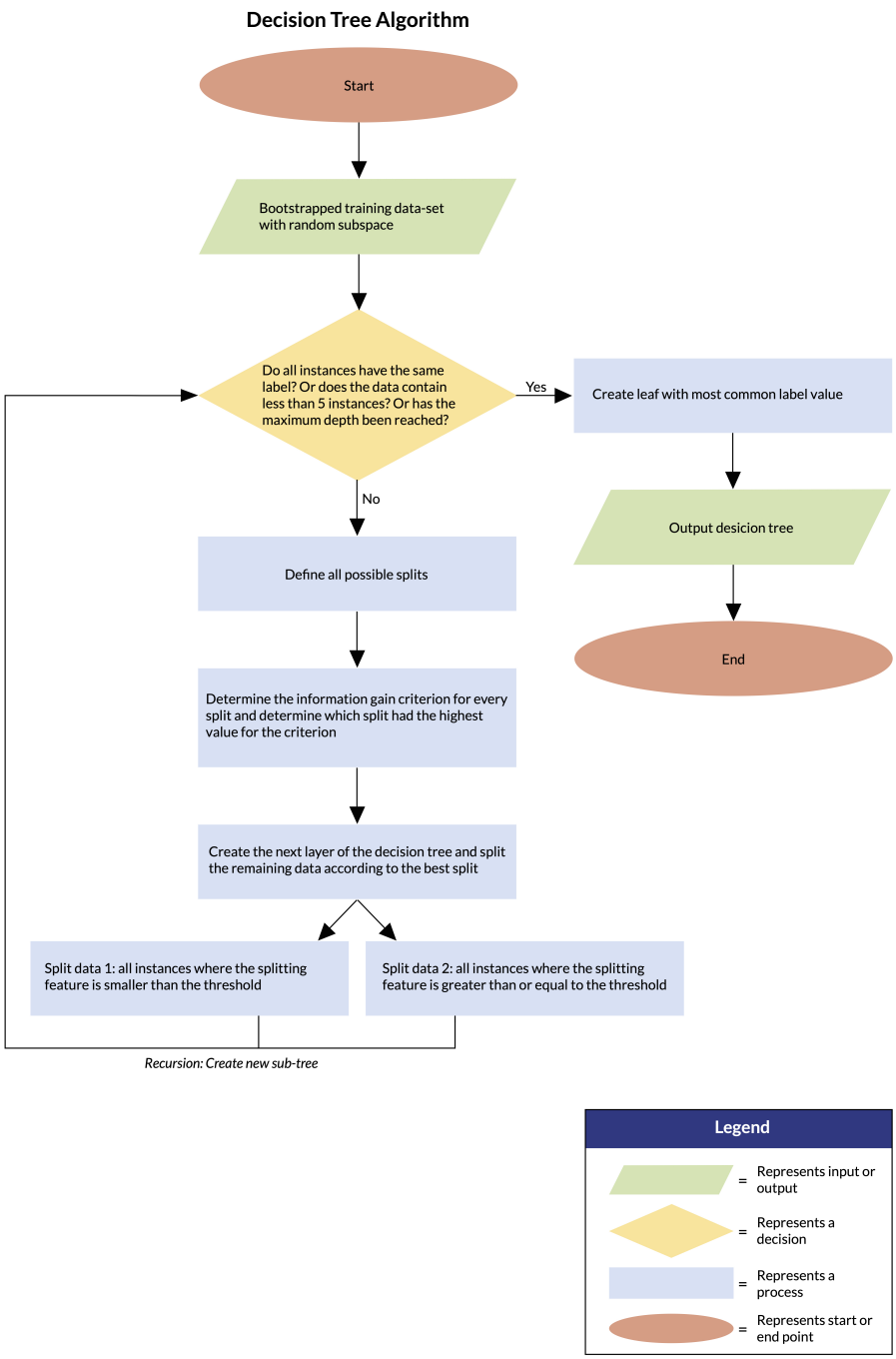
Table 3 Distribution of Data Among Labels

Target Digit	Number	Percentage
0	5923	9.87 %
1	6742	11.24 %
2	5958	9.93 %
3	6131	10.22 %
4	5842	9.73 %
5	5421	9.03 %
6	5918	9.86 %
7	6265	10.44 %
8	5851	9.75 %
9	5949	9.91 %

B. RANDOM FOREST MODEL TRAINING ALGORITHM



C. DECISION TREE MODEL TRAINING ALGORITHM



D. PYTHON CODE AND RESULTS

The full Python code can be found in the following GitHub repository: https://github.com/SaravB/VU-MachineLearning-Project
This repository also contains tables consisting of the results of all of the random forest models that have been run as well as the random forest models from the Python scikit-learn library as comparison.