



Faculty of Engineering and Applied Science  
SOFE - 4620U Machine Learning  
Thursday, April 4, 2024

## **Digit Recognizer**

### **Supervised Learning Classification**

## **Project Description and Background**

Digit recognition plays a vital role in automation tasks that involve processing handwritten information. However, recognizing digits from images of handwritten characters can be challenging due to the difference in handwriting styles. Various machine-learning methods can be used to build a model capable of accurately predicting digits based on these images.

Therefore, the goal of this project is to develop a model that can identify single digits from images of handwritten characters. The main task of this project requires building a model capable of classifying digits from a given dataset of handwritten digits ranging from 0 to 9, using different machine-learning methods, and comparing their accuracies.

## **Major Machine Learning Methods**

In this project, the major machine learning methods used in this project takes images of handwritten single digits as inputs and predicts the digit in the image. The focus of our model used the Random Forest and Multilayer Perceptron methods. The use of these 2 models will involve data processing, training the model using different techniques, and evaluating the model. It also gives the accuracy for each method used for the model, therefore we can compare the results of the 2 models and determine the model best suited for digit recognition.

## **Random Forest**

Random forest is a supervised learning method which can be used for both classification and regression. The random forest algorithm works by creating multiple decision trees which work together to produce a single more accurate prediction. This follows the principle of ensemble learning, where many weak learners combined together produce a smarter learner [1].

Random forests most often use bagging which is the process of splitting the training data into subsets of the same size with replacement [2]. This introduces diversity and produces better results. Once each tree is made, they will each produce a prediction for each input. The output with the most votes is selected as the final prediction. This is the general idea behind random forest and will be explained in more detail in the following sections.

## Data Preprocessing

Before building a model the data needs to be prepared. The data is first normalized to a range of zero to one, this is commonly done to ensure data with larger values do not skew the training process helping enhance model performance. The train.csv data is split 80:20 into training and validation data. Features which always have the same value are dropped since they are useless and do not contribute to the randomness/diversity of the random forest model. Additionally, since the process of building a random forest model is time consuming, only the best “n” features may be selected by using the ExtraTreesClassifier library in Python, reducing the dimensionality of data and speeding up the training process.

## Decision Tree Methods

The process of generating decision trees begins by finding the best feature to split on which is done using either Gini impurity or information gain calculated by entropy. The data is then split based on this feature, creating two subsets, one where the feature is above 0.8 and one where it is below 0.8. These subsets are used to initialize the left and right child of the current node. This process is repeated recursively until the tree reaches the specified depth.

Method	Description
Split Data	The data is split based on a feature index. Two subsets will be created, one where the feature is above 0.8 and one where it is below 0.8.
Best Split (Gini Impurity)	Creates a subset of random features indices. For each feature in the subset, the data is split into subsets, one where the value of the current feature is above 0.8 and one where it is below 0.8. The Gini impurity, which is a measure of misclassification [3], is then calculated for each subset, and then for the feature. The feature with the lowest Gini Impurity is considered the best feature to split on.
Best Split (Information Gain)	Creates a random subset of feature indices. The information gain (measure of reduction in entropy) [4] of each feature in the subset is calculated. To calculate information gain of a feature, it is split into two subsets, one where the feature is above 0.8 and one where it is below 0.8. The entropy is then calculated for each subset which is then used to find the information gain of the current feature. This is repeated for all features, while keeping track of the feature resulting in the highest information gain, which is considered the best feature to split on.
Build Tree	This function works recursively. It finds the best feature to split on to create two child nodes which call this function. This function stops when the current node has no labels, has 1 label, or has reached max depth. If max depth is reached, and multiple labels are present, the most common will be used.
Evaluate	This function does not need to be used. If the training data is split into a smaller training and validation set, after the tree is built, the validation set can be used to calculate the accuracy of the tree. Used for weighted voting in a random forest.
Predict	This function works recursively. Starting at the parent node, it is checked if any of

	the features of the parent are non zero in the current data point. If there is, the true branch of the current node is used, else the false branch. This process is repeated until a leaf node is reached, where its label is returned as the prediction.
--	---

## Random Forest Methods

The random forest is made by generating multiple decision trees each with a different subset of the training data which is done through bootstrapping. Additionally each tree has a depth within the specified range. The quality of a random forest model depends on its randomness and the diversity of its trees. Each tree in the random forest generates predictions for the given input, the prediction with the most votes is selected.

Method	Description
Bootstrap	Samples from the original data with replacement to create random subsets of the same size for each tree.
Build Forest	Builds decision trees, each with different training data. Each tree has a depth within the specified range providing further diversity/randomness. If building a weighted random forest, the data is split into a training and validation set. The validation set is used to determine the tree's accuracy.
Predict	Gets each tree in the forest to make a prediction for the current input. In a majority random forest, the most common prediction is selected. If it is a weighted random forest, trees with better accuracies are given stronger votes, the label which has the most votes is chosen as the final prediction.
Calculate Class Probabilities	Calculates the probability of predicting each label. This is used for the log loss and ROC AUC score which are evaluation metrics.

## Hyper Parameter Tuning

Each function used to construct a decision tree and a forest was manually created, because all calculations are done entirely in Python the process of building a random forest model is quite slow. Due to this, typical hyper parameter tuning cannot be done in a reasonable amount of time. Instead, inputs will be manually tested, with there being a significant difference between each attempted value.

Hyper Parameter	Result
Number of Trees	5 Trees = 0.43 accuracy 20 Trees = 0.60 accuracy 50 Trees = 0.63 accuracy 100 Trees = 0.65 accuracy 200 Trees = 0.70 accuracy  As the number of trees increases so does accuracy and the time to build.

Tree Depth	3 to 5 = 0.61 accuracy 5 to 10 = 0.69 accuracy 7 to 15 = 0.76 accuracy 10 to 20 = 0.78 accuracy  A range of tree depths is chosen to increase diversity and randomness of the forest. As depth increases so does accuracy and the time to build.
Gini Impurity vs Information Gain	Gini Impurity = 0.59 accuracy Information Gain = 0.57 accuracy  Small difference in accuracy. Since each random forest built will produce different accuracies, it will be assumed gini and information gain are similar in terms of performance.
Number of Features to Use	100 Best = 0.57 accuracy 200 Best = 0.59 accuracy 300 Best = 0.57 accuracy All = 0.60 accuracy  Using more features does seem to raise accuracy by a small amount, however, it is not worth the time it takes. Therefore, only the best 100 features will be considered.
Majority vs Weighted Predictions	Majority = 0.51 accuracy Weighted = 0.59 accuracy  Weighted random forest will be used as it seems to perform better.

## Evaluation

Various evaluation metrics were used such as precision, sensitivity, F1 score, ROC AUC score, log loss, and accuracy to validate the built random forest model. Not all of these metrics were implemented at the time an accuracy of 90% was achieved which is why a different model will be built, one with a lower performance but quicker build time. This model was made using 50 trees, a depth between 7-10, uses weighted predictions, and took 12 minutes to build. Other than accuracy, all evaluation metrics were calculated by passing the predictions and labels to the sklearn library.

- Precision is the number of true positive predictions compared to all positive predictions. This model got 86% of the positive predictions made are correct
- Sensitivity is the percentage of actual positives that the model has correctly predicted. This model has correctly identified 84% of positive cases.
- F1 score is the measure of the balance between precision and sensitivity, 0.84 indicates a good balance.
- ROC AUC score is the true positive rate against the false positive rate. 0.99 indicates the model does a good job distinguishing between classes

- Log loss is a loss function for classification models. 0.68 is quite high indicating the model could be further improved.

### Figure 1: Evaluation Metrics of Random Forest

```
Accuracy: 0.8385714285714285
Total Precision: 0.86
Total Sensitivity: 0.84
Total F1 Score: 0.84
ROC AUC: 0.99
Log Loss: 0.68
```

\* Total precision, sensitivity, and F1 score are calculated by taking the weighted average of each class

Figure 2: Confusion Matrix of Random Forest

True Values \ Predicted Values	0	1	2	3	4	5	6	7	8	9
0	813	0	7	11	3	29	13	2	3	9
1	1	914	3	7	5	8	7	19	12	4
2	5	7	693	31	9	7	11	48	4	23
3	3	3	3	696	1	84	0	5	5	15
4	0	3	4	2	608	12	4	9	3	6
5	5	1	5	28	1	451	0	1	0	4
6	5	1	15	5	30	38	774	0	10	10
7	0	1	2	5	0	0	0	661	0	7
8	19	19	65	85	52	94	22	52	761	89
9	1	1	4	12	102	24	0	87	4	673

### Figure 3: Kaggle Submissions for Random Forest




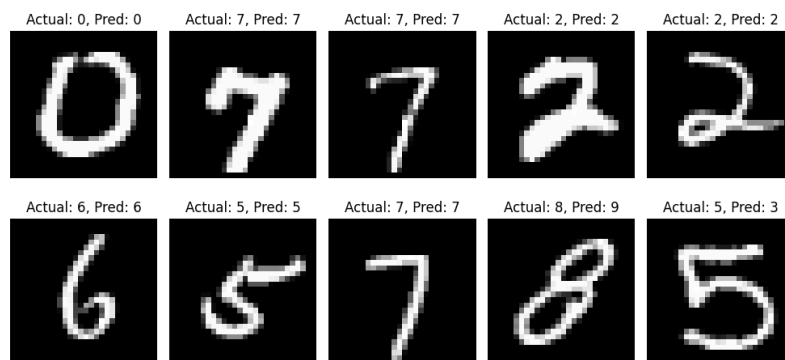
	<b>submission.csv</b> Complete · 11d ago · 200 Trees, Depth between 5 and 20, Using all Features, Weighted Random Forest	<b>0.90525</b>
	<b>submission.csv</b> Complete · 12d ago · 100 Trees, Max Depth between 5 and 15, Best 250 Features, Weighted Random Forest	<b>0.88478</b>
	<b>submission.csv</b> Complete · 12d ago · 5 Trees, Max Depth between 5 and 10, Uses Best 100 Features, Weighted Random Forest	<b>0.72189</b>

Figure 4: Actual vs Predicted Labels Comparison for Random Forest



# Multi-Layer Perceptron (MLP)

The Multilayer Perceptron (MLP) method uses multiple layers of neurons(nodes), and the input travels from one layer to another. The model takes the images as inputs and then uses forward and backward propagation for training and evaluation.

## Forward Propagation

Forward propagation is used to predict the output of the model. The forward propagation takes the images as a 1-D vector, and then these images are passed as input to the input layer of the network. The neurons in the input layer represent each feature of the image. Then, these inputs are sent to hidden layers. Each neuron in the hidden layers represents the learned features, which are the weighted sum of the input from the previous layer, and then this weighted sum is passed to an activation function, which helps to understand the complex patterns and the output produced by the activation method is the output of that layer. If there are multiple hidden layers, then the output of previous layers is used as input by the next hidden layer. After that, from the last hidden layer, the output of that hidden layer gets passed to the output layer, and the output layer also calculates the weighted sum. Each neuron corresponds to each class, and each neuron's weighted sum (logits) is passed through only one softmax activation function. This gives the final output, which is a probability distribution containing probabilities associated with each neuron, so basically, it denotes the probability of each class. The class with the highest probability is the final output. This way, the model generates the predicted output.

## Backward Propagation

Then, in the training process, the loss function is used to calculate the difference between the actual output and predicted output(error). Then, this error is used to train the model by propagating it from the output layer to the input layer and updating the parameters using optimization. By updating the parameters, the model learns to predict the correct output.





## Hyperparameter Tuning

Hyperparameter	Result
Epochs	With Hidden size = 600, learning rate = 0.01 50 = 0.9752(0.97) accuracy, build time = 221.64 seconds 100= 0.9739(0.97) accuracy, build time = 347.77 seconds 150 = 0.9742(0.97) accuracy, build time = 621.44 seconds As we see the increasing the number of epochs did not affect the accuracy much , but it did increase the epoch time because as the number of epochs increase the for

	loop will run also run that much time
Learning Rate	<p>With hidden size = 600, epoch = 50</p> <p>0.001 = 0.9487(0.94) accuracy, build time = 179.18 seconds</p> <p>0.01 = 0.9752(0.97) accuracy, build time = 221.64 seconds</p> <p>0.1 = 0.0935(0.09) accuracy, build time = 194.61</p> <p>Increasing the learning rate to 0.1 resulted in a lower accuracy. We can deduce this is due to the fact that it caused the gradients to skip over the minimum solution and which results in poor convergence.</p> <p>However if we decrease the Learning Rate to 0.001, it leads to reduced accuracy and slower convergence. Therefore 0.01 is the ideal learning rate</p>
Hidden size	<p>With epoch = 50, learning rate = 0.01</p> <p>100 = 0.9595(0.96) or around 0.95 accuracy, build time = 63.53 seconds</p> <p>300 = 0.9702(0.97) accuracy, build time = 113.72 seconds</p> <p>600 = 0.9726(0.97) accuracy, build time = 190.90 seconds</p> <p>As we see, increasing the hidden size results in a slight improvement in accuracy, but not significantly. However, the increased hidden size leads to an increase in build time.</p>

## Evaluation

### Results on the Test Set

Submission and Description	Public Score 
 <b>Test_labels.csv</b> Complete · now	<b>0.97139</b>
 <b>Test_labels.csv</b> Complete · 2d ago	<b>0.95967</b>
 <b>predicted_labels.csv</b> Complete · 3d ago	<b>0.97017</b>

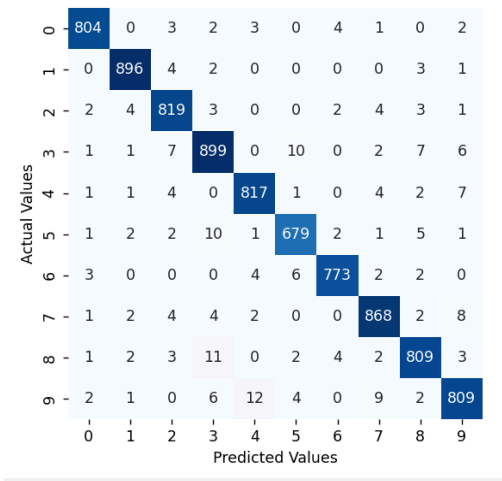
### Results on the Validation Set      Confusion Matrix



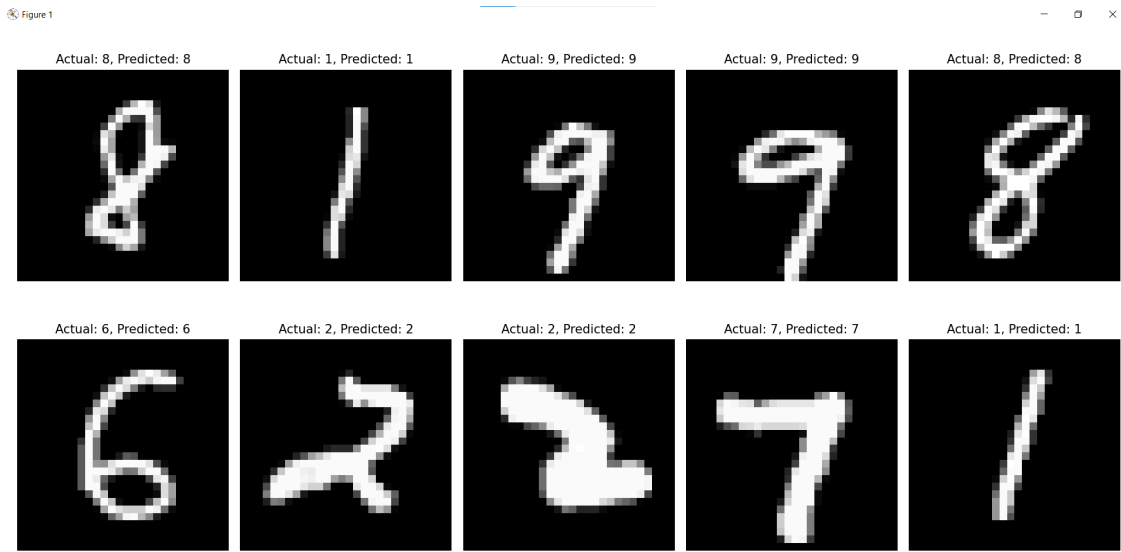
The accuracy of 97% shows that the model functioned with high accuracy of correct predictions.

Validation Set:  
Accuracy: 0.9730

The higher numbers on the diagonal show that the model is generating correct predictions for most instances



Images with Actual and Predicted Results



Comparison

Our exploration of machine learning models for digit recognition yielded distinct advantages for both the Multi-Layered Perceptron (MLP) and Random Forest approaches. While both achieved high accuracy, the MLP emerged as the clear winner in terms of training efficiency. The MLP achieved a stellar accuracy of 97% within a manageable timeframe of 221 seconds (approximately 3 minutes 41 seconds) using a learning rate of 0.01, 50 epochs, and 600 nodes and four layers. This rapid training time is particularly noteworthy when compared to the Random Forest, which, despite achieving an accuracy of 90% with weighted inputs, all features, a depth of 5-20, and 200 trees, required a significantly longer training period

of around 250 minutes. By changing the parameters we got the Random forest implementation to run in under 3 minutes but it returned an accuracy of about 60%.

## Conclusion

Our investigation into Random Forests and Multi-Layered Perceptrons (MLPs) for digit recognition revealed a clear advantage for MLPs in terms of accuracy and training efficiency. The MLP achieved a remarkable 97% accuracy within a reasonable timeframe (3 minutes 41 seconds), significantly outperforming the Random Forest's training time (255 minutes) for a comparable accuracy.

However, it's important to acknowledge the strengths of both approaches. While MLPs excel in accuracy, Random Forests offer a distinct advantage for smaller datasets. Their bagging technique with bootstrapping inherently reduces bias and variance, making them well-suited for scenarios where data availability might be limited.

## Participation

Akshat Kapoor	MLP Model
Nathanael Selvaraj	MLP Model
Vishan Patel	Random Forest
Angad Singh	Random Forest

## References

- [1] Singh, A. (2023, November 22). A comprehensive guide to ensemble learning (with python codes). Analytics Vidhya.  
<https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/>
- [2] Google. (n.d.). Machine learning | google for developers. Google.  
[https://developers.google.com/machine-learning/decision-forests/random-forests#:~:text=Bagging%20\(bootstrap%20aggregating\)%20means%20training,a%20different%20subset%20of%20examples](https://developers.google.com/machine-learning/decision-forests/random-forests#:~:text=Bagging%20(bootstrap%20aggregating)%20means%20training,a%20different%20subset%20of%20examples)
- [3] Victor Zhou. (2019, March 29). A simple explanation of Gini impurity.  
[https://victorzhou.com/blog/gini-impurity/#:~:text=Gini%20Impurity%20is%20the%20probability,E2%88%92p\(i\)\)](https://victorzhou.com/blog/gini-impurity/#:~:text=Gini%20Impurity%20is%20the%20probability,E2%88%92p(i)))
- [4] Khandelwal, R. (2018, November 14). Decision tree and Random Forest. Medium.  
<https://medium.datadriveninvestor.com/decision-tree-and-random-forest-e174686dd9eb?gi=326ee509039e#:~:text=Information%20Gain%20is%20the%20difference,with%20the%20purest%20child%20nodes>