LETTER RECOGNITION PROJECT​

​

Brandon Tobar

Justin Le

Summer Lake

Edin Sehic

Eric Son

Introduction

The UCI Letter Recognition Data Set has been selected to explore the capabilities of machine learning algorithms in accurately classifying capital letters based on their pixel features. The data set, sourced from the UCI Machine Learning Repository, contains 20,000 observations and 17 variables that describe various attributes of capital letters, such as their position, size, and pixel density.

The primary question we aim to answer is: Can a machine learning model accurately classify capital letters based on their pixel features? Additionally, we are interested in determining which features are most important in distinguishing between different letters and comparing the performance of different machine learning algorithms, specifically Decision Trees, Random Forests, and Neural Networks.

The response variable, "Lettr," represents the capital letter (A-Z) that we want to predict. Potential predictor variables include x-ege (mean edge count left to right), y-ege (mean edge count bottom to top), x-box (horizontal position of box), and y-box (vertical position of box).

Our primary focus will be on prediction, with the goal of developing a model that can accurately classify capital letters based on the provided features. To achieve this, we will employ Random Forests and Neural Networks, two distinct models covered in this course. Random Forests are selected for their ability to handle complex interactions and potential for high accuracy, while Neural Networks allow the ability to detect all possible interactions between predictors.

To compare the performance of these models, we will implement K-fold cross-validation. Each model will be trained and evaluated multiple times using different subsets of the data for validation and training. Performance metrics, such as accuracy, will be collected for each fold, and the average performance for each model will be calculated. By comparing these averages, we will determine which model performs best in recognizing letters based on the given features.

Through this project, we hope to gain insights into the effectiveness of machine learning algorithms in letter recognition tasks and identify the most important features for distinguishing between different capital letters.

Random Forest

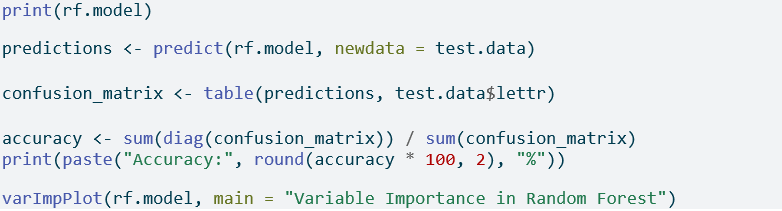
Choosing a Random Forest for this dataset is a strategic decision driven by several key factors. Random Forests excel in classification tasks, particularly when dealing with complex and high-dimensional data like this one. The primary advantages of using Random Forest here include its ability to handle large numbers of features, manage both numerical and categorical data effectively, and mitigate overfitting through its inherent bagging and feature selection processes. In this case, the dataset involves classifying letters, which suggests a high-dimensional feature space and potentially noisy data. Random Forest’s robustness in handling noisy data and its capability to model intricate interactions between features make it a suitable choice. Additionally, Random Forest provides reliable estimates of feature importance, which can help in understanding which features are most influential in distinguishing between different classes. This is particularly useful in a dataset with numerous predictors, as it helps in identifying and focusing on the most significant features. We used the randomForest package as well as the function to create our model in R:

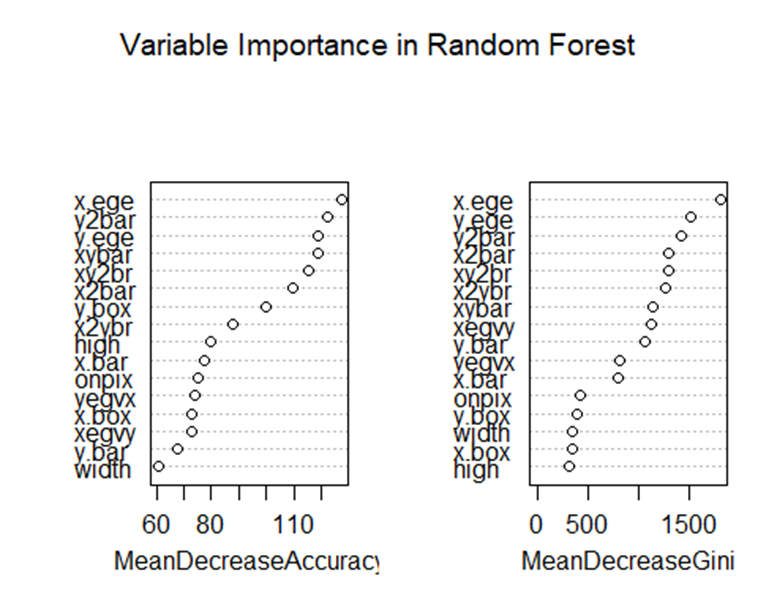


First we read the dataset and organize the data by response variables



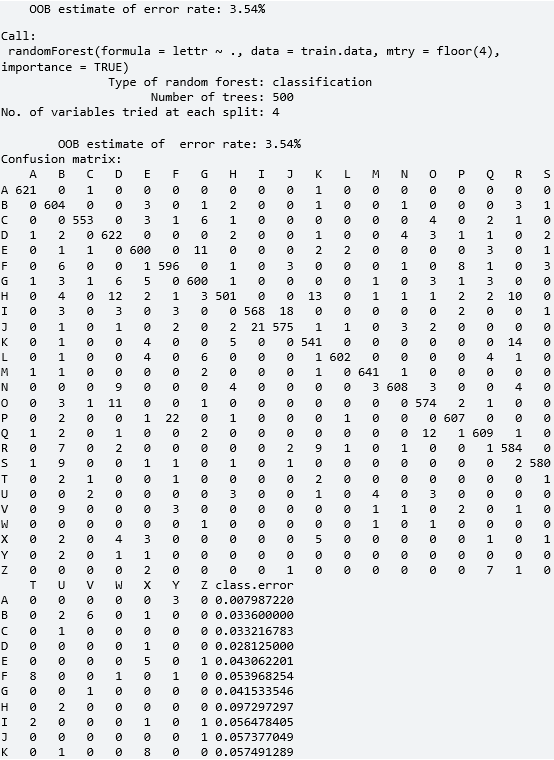
This is our random forest equation with an 80% training and 20% testing split. There are 16 variables total so our mtry will be the square root of that, which is 4.







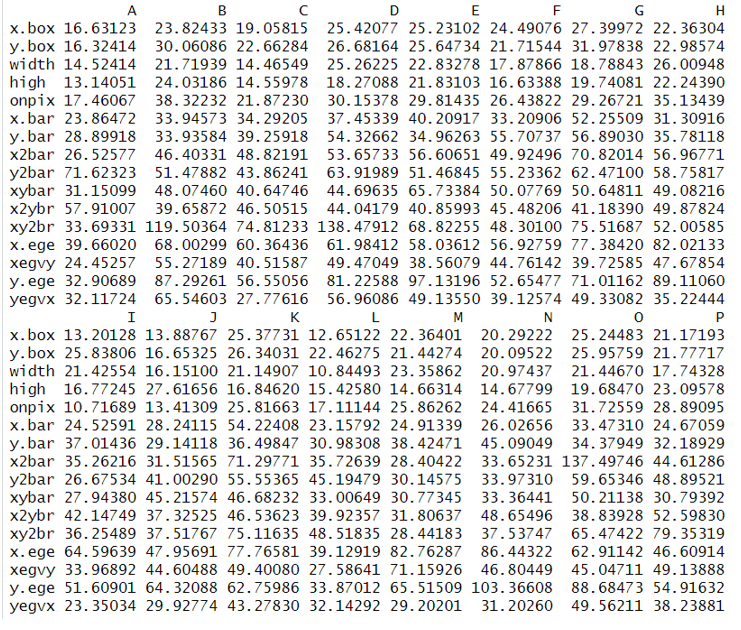
With the importance plot results, we can conclude that the 3 most important predictors in this dataset are “x.ege” , “y. ege” , and “y2bar”.

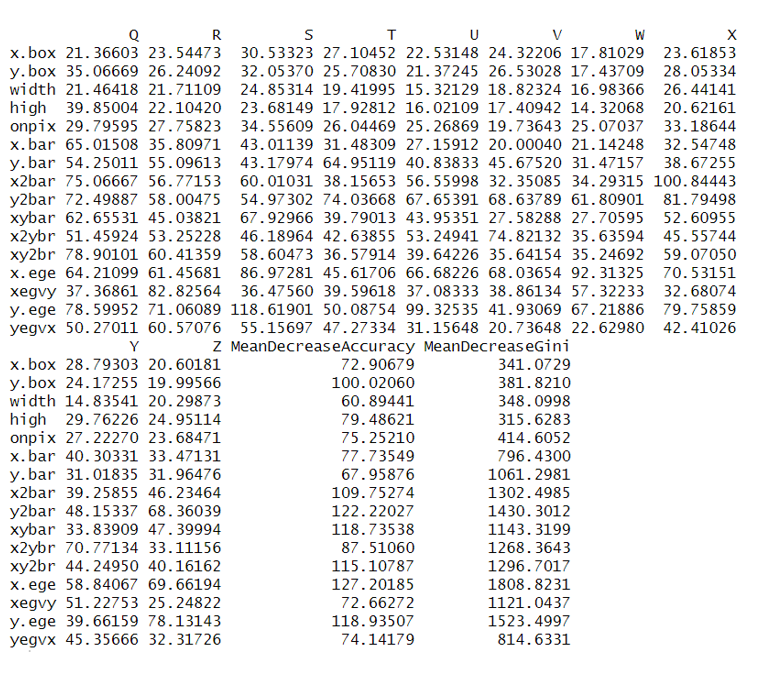


The confusion matrix above represents the performance of a Random Forest classifier on a dataset where the goal is to recognize letters from A to Z. Each row in the matrix corresponds to the true letter class, while each column represents the predicted letter class. The diagonal elements show the number of correctly classified instances for each letter. For instance, the letter “A” was correctly classified 621 times, with only minor misclassifications for letters “C” and “K.” Similarly, the letter “B” was correctly identified 604 times, with some confusion with letters “E,” “G,” and “R.” Overall, the classifier shows high accuracy for most letters, particularly “A,” “B,” “C,” “D,” “M,” and “P,” with few misclassifications. However, there are some notable confusions, such as “H” being misclassified as “D,” “J” being confused with “I,” and “R” having higher confusion rates with other letters like “T” and “K.” The matrix indicates a strong overall performance with areas of improvement in distinguishing letters with similar visual features, as evidenced by the higher off-diagonal values for certain letter pairs.



The Random Forest classifier achieved an impressive accuracy of 96.47% on the letter recognition dataset. This high accuracy indicates that the model correctly classified approximately 96.5% of the letters in the test set, demonstrating its strong performance in recognizing various characters from the dataset. Such a level of accuracy is particularly notable given the complexity and potential similarities between certain letters, which can often lead to misclassification in less robust models. This method reduces overfitting and increases generalization, allowing the model to handle noise and subtle variations in the data effectively. Additionally, the use of a significant portion of the dataset for training (80%) allows the model to capture the underlying patterns of the letters comprehensively. The achieved accuracy suggests that the model can reliably distinguish between different letters, making it suitable for real-world applications such as optical character recognition (OCR) systems, where precise and efficient letter recognition is crucial.





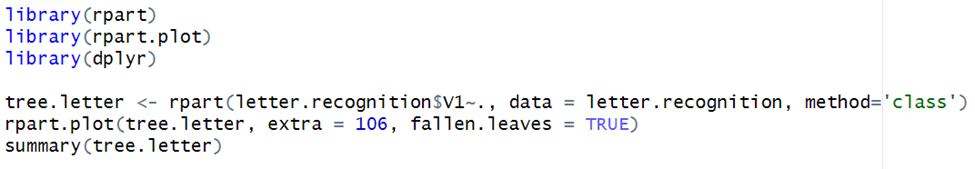
The given plots illustrate the average values of several features used in the Random Forest model for each letter in the dataset. These features include attributes such as x.box, y.box, width, high, onpix, x.bar, y.bar, x2bar, y2bar, xybar, x2ybr, xy2br, x.ege, xegvy, y.ege, and yegvx. Each feature captures a specific aspect of a letter’s shape, orientation, or pixel distribution. For instance, x.box and y.box measure the horizontal and vertical dimensions of the smallest enclosing box for a letter, while width and high represent the letter’s width and height within that box. The onpix feature reflects the number of pixels used to draw the letter, which often varies significantly across different letters due to their inherent complexity. Observing the feature means across different letters reveals distinctive patterns; for instance, letters like ‘G’ and ‘Q’ tend to have larger x.box values, suggesting they occupy wider horizontal spaces. Similarly, onpix is notably higher for letters like ‘E’ and ‘H’, indicating they require more pixels to be depicted clearly. Features like x.bar and y.bar, which represent the mean x and y coordinates of the pixels in a letter, highlight differences in pixel distribution, crucial for differentiating between similar-looking letters. The variability in values across these features underscores their importance in distinguishing letters effectively, as the Random Forest algorithm leverages these distinctions to achieve high classification accuracy.

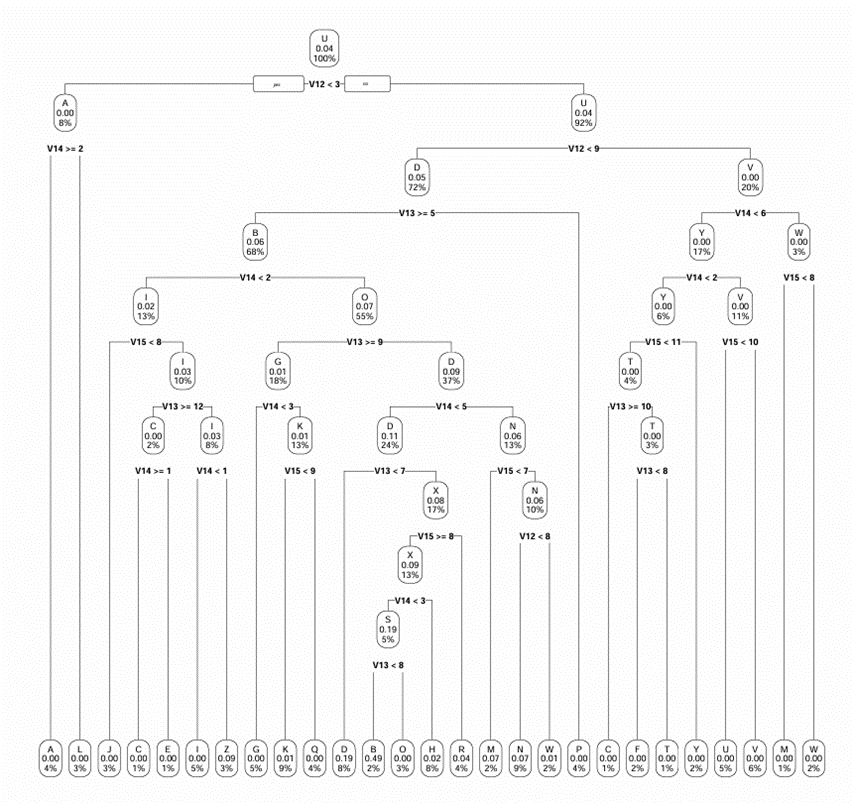
Decision Tree

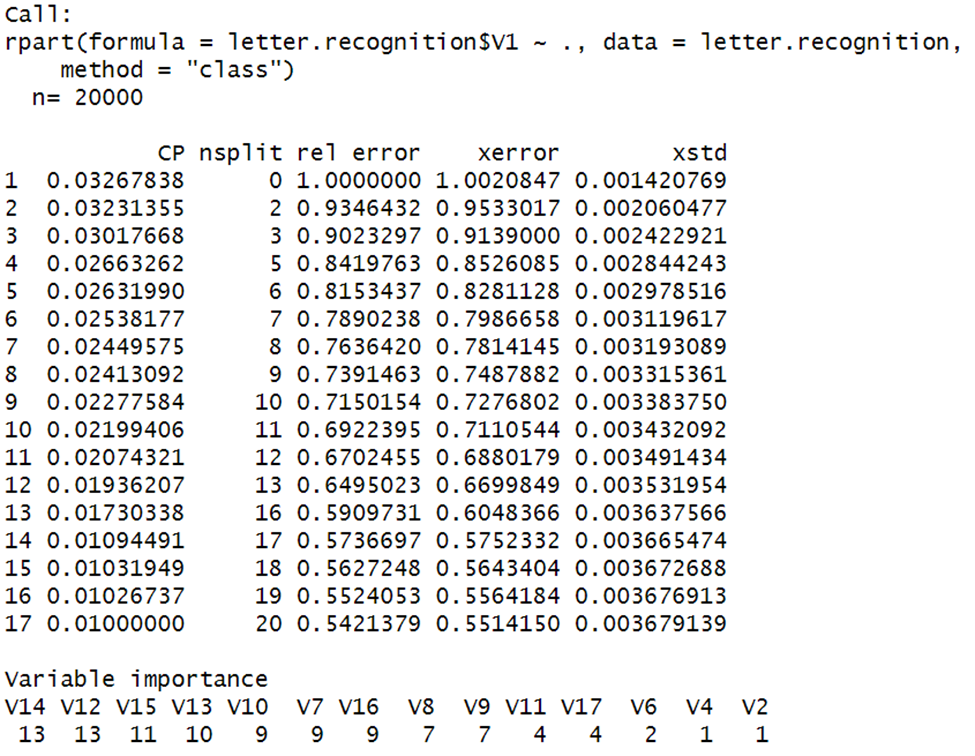
We use decision tree models because they are straightforward to explain, visualize, and interpret, allowing us to see the data's visualization clearly. Decision trees are organized in a hierarchical, tree-like structure that is easy to comprehend. However, a drawback is that they typically do not achieve the same level of predictive accuracy as some other regression and classification methods.

To make the decision tree we used rpart() since our response variable, lettr, is a categorical variable with 26 values. This made it difficult to use tree(), so we used rpart() instead.

When we run this we get:

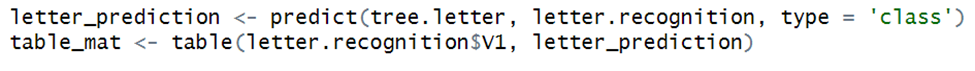


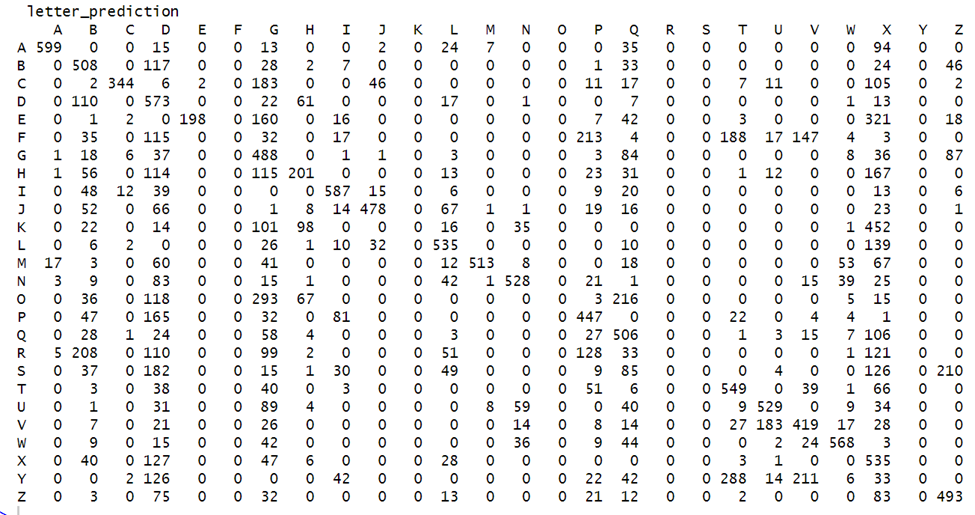
When we run this we get:



Looking at the variables of importance, we decided to use variables above the importance of 7 in order to get rid of insignificant variables.

Now we want to predict which letter is going to be read. To do this we use the predict() command and we make a table.





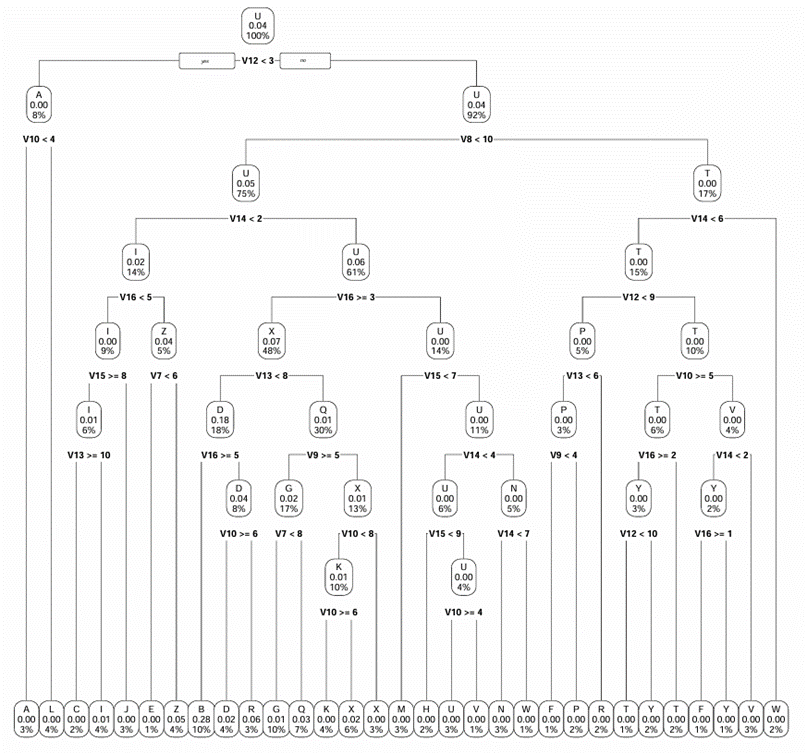
Using this table, we can find the error rate and accuracy. We compute the accuracy by using this formula:

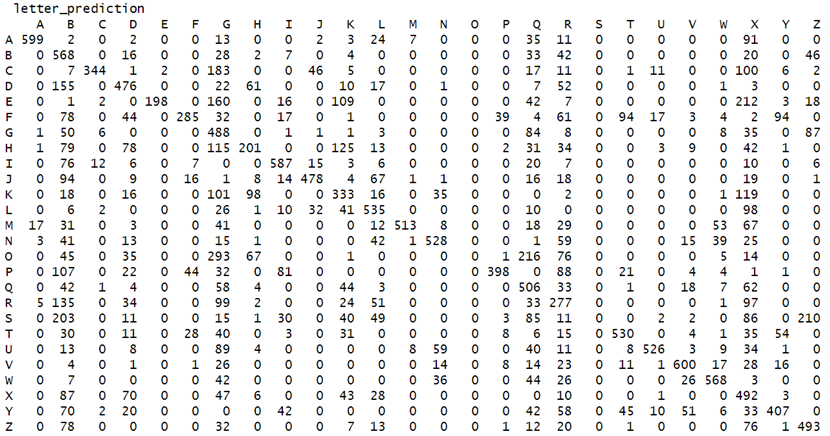


So the accuracy of this set is 48.065% and the error rate is 51.935%. We can increase the accuracy by modifying the minsplit, minbucket, and cp value in the rpart() command. We set the these values as follows:



The new tree and table we get from this is:

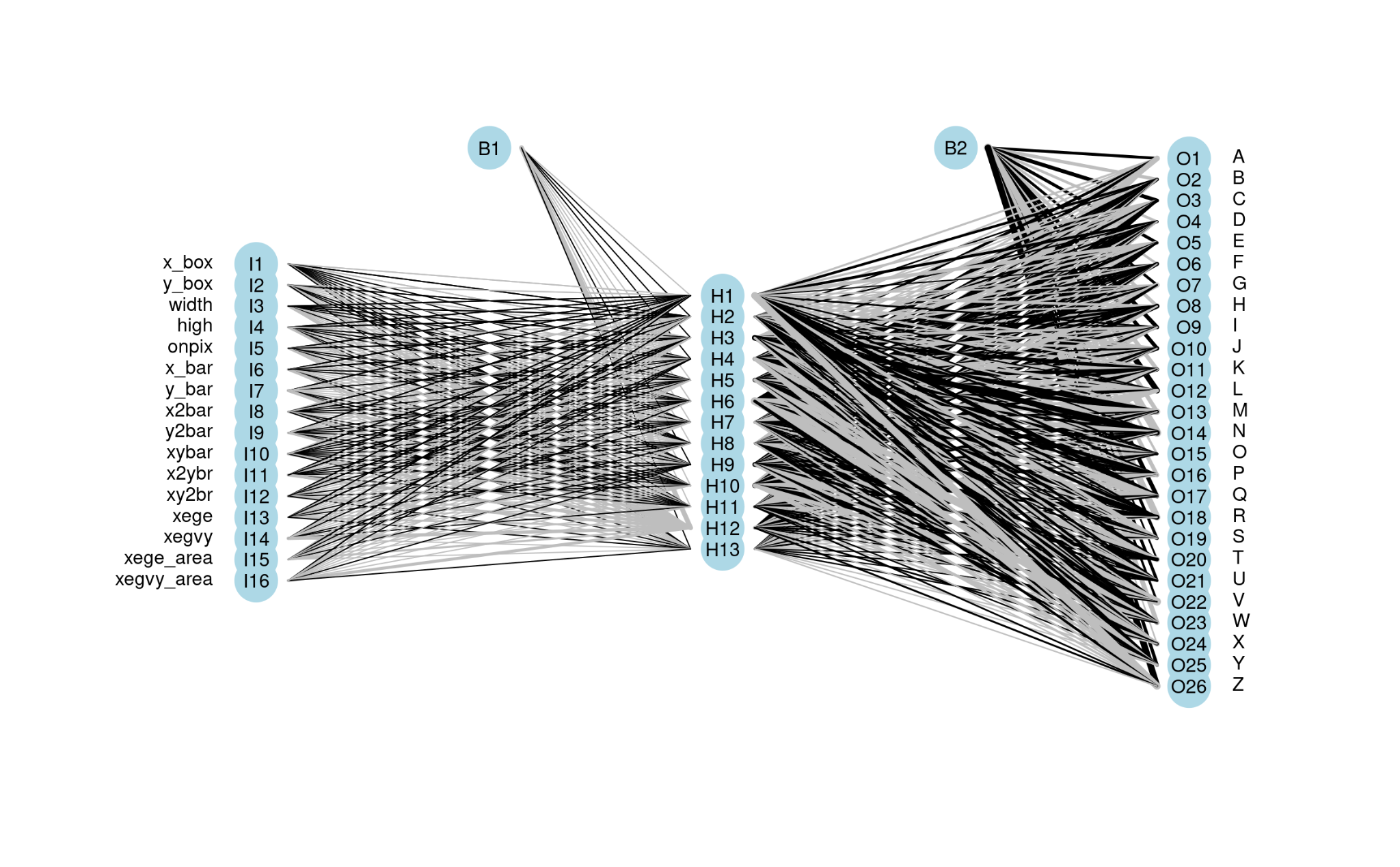




The new accuracy we get is 54.65% and the new error rate is 45.35%. With this, we conclude that our model is fairly accurate when it comes to predicting the letter.

Neural Networks

Model:



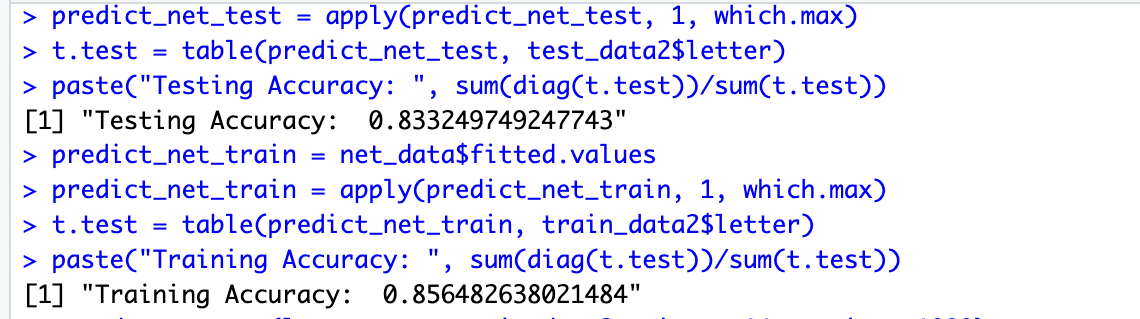
· For this Neural Network, I first loaded in the data set, via URL, and immediately scaled the values. Afterward I retrieved the column names for the data set and manually set them

· Next, I split the data into training and test and began to pre-process the data. For the training and test data I used a 80/20 split in favor of training. Data was pre-processed using the range method to scale the data to the interval between zero and one for both the training and test sets.

· With all the data set I formed the neural network with 13 hidden layers. Originally, I tested with 8 and 10 hidden layers and received an accuracy percentage of just below 79 percent and just above 80 percent respectively. The decision to go with the amount of 13 was a matter of accuracy, time efficiency and ratio relative to the number of predictors. For my last test I decided to see if 14 hidden layers would yield better results. Not only did the run time increase, but the model accuracy decreased by around 2%. From this I determined that I should stick with the amount of 13 hidden layers.

· After concluding the test to determine the number of hidden layers to use, I then checked if increasing the iterations of fitting the model would improve accuracy, but it seems that with 5 iterations, the accuracy was about 80%. From this analysis in combination with the amount of time it took for the loop to finish, I settled with only one iteration.

· The mean test prediction accuracy for the model using 13 hidden layers and only one iteration is 0.8565; the mean prediction test error was 0.1435.



Source Code:



Conclusion & Results

Results:

Random Forest:

* The random forest model used an 80% training and 20% testing split, with mtry=4 (square root of 16 variables).
* The 3 most important predictor variables were identified as "x.ege", "y.ege", and "y2bar" based on the variable importance plot.
* The confusion matrix showed the random forest classifier achieved high accuracy for most letters, with some notable confusions between visually similar letters like "H" and "D", "J" and "I", etc.
* The overall accuracy of the random forest model was an impressive 96.47% on the test set.

Decision Tree:

* A decision tree was created using rpart() since the response variable "lettr" is categorical with 26 values.
* Variables with importance above 7 were used to eliminate insignificant variables.
* The initial decision tree model had an accuracy of 48.065% and error rate of 51.935%.
* By modifying the minsplit, minbucket, and cp values in rpart(), the accuracy was increased to 54.65% with an error rate of 45.35%.

Neural Network:

* A neural network model was built using 13 hidden layers, which was determined to be optimal based on accuracy and efficiency compared to 8, 10 and 14 hidden layers.
* The data was split into 80% training and 20% test sets, and preprocessed using range scaling between 0 and 1.
* Increasing the number of fitting iterations did not significantly improve accuracy beyond 1 iteration.
* The final neural network model with 13 hidden layers and 1 iteration achieved a mean test prediction accuracy of 85.65% and mean test error of 14.35%.

Conclusions: Among the three models, the Random Forest performed the best with an accuracy of 96.47% on the letter recognition task. This proves the effectiveness of ensemble methods like random forests in handling complex, high-dimensional data, and achieving high classification accuracy.

The decision tree model, while providing an easily interpretable visualization of the classification rules, had a lower accuracy of 54.65% even after tuning. This highlights the potential limitations of single tree models in capturing complex patterns.

Lastly, the neural network with 13 hidden layers achieved a reasonably high accuracy of 85.65%, showing the power of deep learning approaches, but it did not outperform the random forest.

In conclusion, for the given letter recognition dataset, the random forest model proved to be the most effective in accurately classifying letters based on their pixel features. The results underscore the importance of comparing multiple modeling approaches and tuning them appropriately to achieve the best performance on a given task.

Bibliography

Slate, David. “Letter Recognition.” *UCI Machine Learning Repository*, UCI Machine Learning Repository, archive.ics.uci.edu/dataset/59/letter+recognition. Accessed 30 July 2024.