Predictive Modeling of Handwritten Digits Using Decision Tree Classification Techniques: A Comparative Analysis

Emre Guvenilir & Rett Kinsey

Introduction & Data

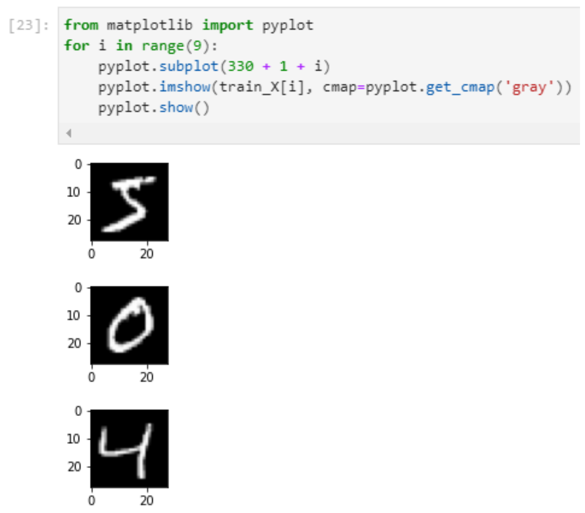
This paper utilizes the MNIST dataset is a database of handwritten digits used to train machine learning and image processing models. It contains 60,000 training set data points, and the test set contains 10,000 examples. Each image is of a digit 0-9. Each image has 28x28 grayscale pixels. It is known what digit each image represents, meaning the accuracy score can be calculated accordingly. The data within the project was imported from keras.datasets, a module that provides datasets vectorized and in Numpy format to be used.

Objectives

Given the dataset, we are attempting to be able to identify new, unseen handwritten digits in a similar format. By doing so, we can apply this concept to broader handwriting recognition, such as letters, meaning we can identify words and sentences. To do this, we built multiple decision tree classification models to test how our handwritten approach would do compared to the sci-kit-learn models with different hyperparameters. We will look to evaluate the model through accuracy and other metrics such as recall, precision, and F1-score to determine the overall effectiveness of the models and our approaches.

Methods

In this section, we define our approach in detail. For the data exploration and preprocessing, we did research on the functionality of the MNIST dataset and found that it utilizes grayscale images and each image is in a 28x28 pixel format (Khodabakhsh, 2019). We also printed out a sample of 9 images from the dataset to visualize the image and decide on an approach. We decided we would not use K-nearest-neighbors as we felt the cost of compilation time to search a 28x28 pixel image would outweigh the result that would be yielded. We felt there would be too many features, as each pixel would be its own feature, and the curse of dimensionality would negatively affect the performance of the KnN algorithm.



```
[23]: from matplotlib import pyplot
      for i in range(9):
          pyplot.subplot(330 + 1 + i)
          pyplot.imshow(train_X[i], cmap=pyplot.get_cmap('gray'))
          pyplot.show()
```

Figure 1: The code we utilized to print out digit images and three out of nine printed images from the dataset

We decided to use decision tree classifications for our models. These types of models can be built upon more easily than KnN algorithms as it can handle large datasets efficiently. We also decided that one approach we would use would be to reduce the scale of the image from 28x28 to a 7x7 pixel image. We achieved this by creating an empty 7x7 array, iterating over each 4x4 block and calculating the average pixel value to store in each corresponding pixel in the resized image. We then created a new array to store the resized dataset in while maintaining the original structure of the MNIST dataset. The results of that approach will be discussed later. Another essential part of the handwritten model was to convert the parts of the image that were not exclusively black or white to black or white using the bisecting value of 128. We created our own node class for the tree that would have a left and right child, a feature attribute to find the best feature, and a value attribute for when the tree reaches its end point and decides on a number. We recursively ran our Find_Features() function until both the left and right child had null values, at which point we reached the end of the node. We decided to reduce the image size to reduce the number of features to 49, meaning that the tree would be significantly more shallow than a 28x28 pixel image if ran to completion without a max_depth, reducing the complexity of our model. We encountered issues when dealing with series in parsing and accessing and writing information, which our value, or digit result, was stored in, so we decided to create a new list to store our answer, and used a simple counter to determine the accuracy of our results.

Our other models were created with the help of sci-kit-learn. We created four additional models, two which ran the decision tree classification on the 28x28 pixel image, and two which ran it on the images scaled down to a 7x7 level. We had to use the reshape method so that the classifier would be able to process the data without error. We ran tests with different hyperparameters such as adjusting the splitter to “random” from “best”, max_features to a range of numbers from 3-49 where default is “none”, but found that the default parameters ran with the highest accuracy scores. We were also able to calculate the precision, recall, and F1-scores of these models to compare how each model did in addition to its accuracy.

We addressed overfitting by varying the level of the trees. We did not have a set pattern for tuning, but started with a smaller number of levels and increased the number of levels until accuracy scores plateaued. Typically, this plateau would occur around 12-15 levels. Our handwritten model plateaued significantly earlier, as 9 levels performed marginally worse than 14 levels.

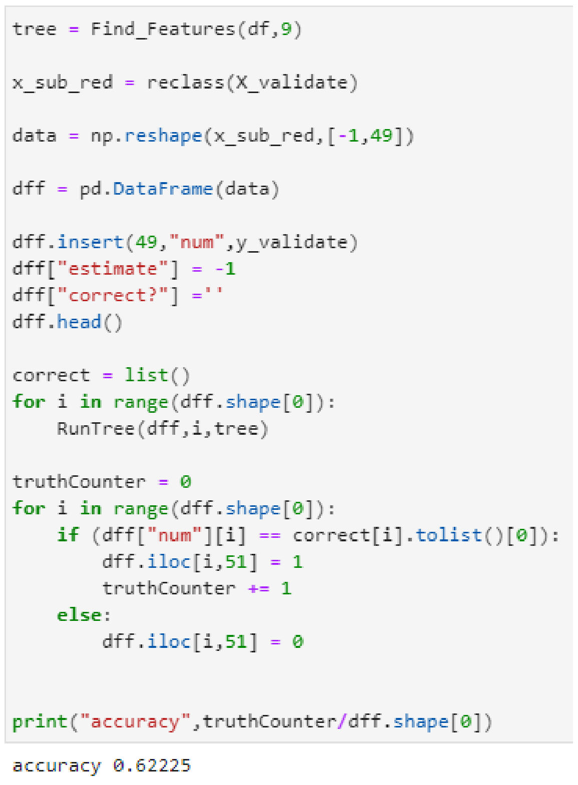
Another adjustment we made was to the Gain function. While not a significant adjustment, we separated the grayscale values if they were over 128, then calculated the gain for a given feature. We did not have to handle any missing values in the dataset, and decision trees are helpful in handling outliers with noisy data points that act as outliers. This is because decision trees make decisions based on features and the pixels, rather than relying on the entire dataset.

Results

Predictive Modeling of Handwritten Digits Using Decision Tree Classification Techniques: A Comparative Analysis

Emre Guvenilir & Rett Kinsey

For the handwritten model, we only calculated the accuracy score on a 14 level tree, which was 0.6706666666666666 or 67.07% accurate. This accuracy score was significantly less than all four of the sci-kit-learn models. We also ran a second model using a tree depth of 9, which yielded an accuracy score of 0.62225, or 62.22%.



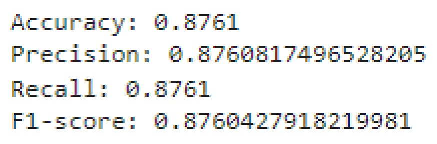
```
tree = Find_Features(df,9)

x_sub_red = reclass(X_validate)

data = np.reshape(x_sub_red,[-1,49])

dff = pd.DataFrame(data)

dff.insert(49,"num",y_validate)
dff["estimate"] = -1
dff["correct?"] =''
dff.head()

correct = list()
for i in range(dff.shape[0]):
    RunTree(dff,i,tree)

truthCounter = 0
for i in range(dff.shape[0]):
    if (dff["num"][i] == correct[i].tolist()[0]):
        dff.iloc[i,51] = 1
        truthCounter += 1
    else:
        dff.iloc[i,51] = 0


print("accuracy",truthCounter/dff.shape[0])
```

```
accuracy 0.62225
```

Figure 2: Results for handwritten model #2, with a tree level of 9

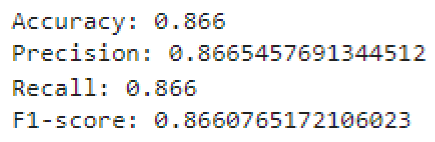
We ran a third model to make sure the hyperparameter tuning was nearly at capacity, so we ran a model with 18 levels, which yielded a marginal increase of an accuracy score of 67.56%.

We created four additional models utilizing sci-kit-learn, in which we will compare the results of two 28x28 pixel models and two 7x7 pixel models, and within those two one had all default hyperparameters, and one had a maximum tree depth level of 10. The results are as follows:



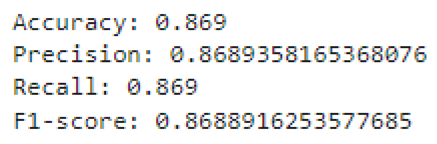
```
Accuracy: 0.8761
Precision: 0.8760817496528205
Recall: 0.8761
F1-score: 0.8760427918219981
```

Figure 3: Results for sci-kit-learn model #1, 28x28 pixels and no max_depth



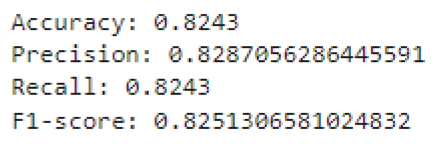
```
Accuracy: 0.866
Precision: 0.8665457691344512
Recall: 0.866
F1-score: 0.8660765172106023
```

Figure 4: Results for sci-kit-learn model #2, 28x28 pixels and max_depth = 10



```
Accuracy: 0.869
Precision: 0.8689358165368076
Recall: 0.869
F1-score: 0.8688916253577685
```

Figure 5: Results for sci-kit-learn model #3, 7x7 reduced image, no max_depth



```
Accuracy: 0.8243
Precision: 0.8287056286445591
Recall: 0.8243
F1-score: 0.8251306581024832
```

Figure 6: Results for sci-kit-learn model #4, 7x7 reduced image, max_depth = 10

Although not documented, we found that the plateau where the default no max levels typically equals the accuracy and other metrics around level 12-15. This was found as part of our hyperparameter tuning process.

Analysis

The biggest drawback to our handwritten approach is that we currently do not have a way to calculate the precision, F1-score, and recall of those results. Given more time, we would have developed methods to calculate these metrics, giving us additional data to analyze and to compare with the sci-kit-learn models. Our results find that our models all plateaued at a certain number of levels, although it differed slightly between the handwritten and sci-kit-learn models. We prevented overfitting by not exceeding the plateau in the models we chose max levels for. What we found was when underfitting the model, our accuracy and other metrics were noticeably lower. It was also evident that underfitting with max_levels = 10 on the 7x7 pixel models undercut the performance by approximately 4% more compared to the 28x28 pixel model. This means that our approach to reducing the size of the image likely undercut performance on the handwritten model when underfitting. However, when using the default hyperparameters, we found that performance was largely the same between the downsized image and the regularly sized image. One downside to our handwritten model is that the compilation time is extremely high. Even though we picked decision tree classifications to have faster compilation time, it still lacked the speed to allow us to develop more advanced metrics to thoroughly compare these models to the sci-kit-learn models.

Online we were able to locate a paper from a group of researchers in Nigeria at Ladoke Akintola University of Technology who took a similar approach of downsizing the image. They utilized more data, as they used 7x7, 14x14, 21x21, and 28x28 images, and also created results for another dataset. These researchers also utilized multiple model types, including a decision tree, though they do not discern if it is a classification or regression. On the MNIST digits with a decision tree. However, they do not denote the size of the image used, as they only do so for the other dataset they used. They provided two values for the decision tree results: 28.3% and 71.24% (Omidiora EO et. al). We are assuming that those images were at full scale 28x28 pixels since they do not denote any image size. This study was the closest I could find to our approach, as other studies reduce the image size but do not use our model type on the MNIST dataset. Comparing their results only to our 28x28 pixel images, our models perform better comparatively, although much of their documentation is not available in the paper so their complete methods are unknown, so meaningful and verifiable conclusions cannot be drawn.

When comparing our handwritten model to the sci-kit-learn models, our model underperformed significantly in nearly every category. Our process was likely much more data heavy in processing the data through the decision tree, as our compilation time was significantly higher, although we have not provided empirical evidence to support those claims.

Ethics

Using the NeurIPS Code of Ethics, it is clear that the contribution of this paper falls in line with the standards set. The data used does not include any human subjects or participants, does not have any identifiable information on who wrote any of the handwritten digits, and the dataset is publicly available and not deprecated. This project is not vulnerable to misuse as the handwriting is not identifiable to a singular individual or group of individuals. When looking at the generalizability of this dataset, there are some concerns in how it could be used. With a possible expansion to identify letters, such a model could be used to identify patterns in handwritten, intercepted letters. Whether this means identifying writing styles to individuals or identifying writing could be used for fraudulent purposes if this model is used without the consent of individuals. This model, and other similar models, should be used for research and education applications where security concerns are minimal such as automating data entry tasks or sorting mail. Such a model should not be used to assist in forging documents, or where sensitive data is used such as in medical, legal, and personal educational records where there are laws that must be abided by in their handling. This model should also not be used for individual handwriting analysis, especially without consent, as that may lead to privacy breaches.

These results were produced in a Jupyter environment using Python and libraries such as numpy, matplotlib, and sklearn for the models and classification implementations. All code is available in the 'csc371-machinelearning/project-2-rett-and-emre' repository on github.

References

Khodabakhsh H (2019) Mnist dataset. In: Kaggle. https://www.kaggle.com/datasets/hojjatk/mnist-dataset. Accessed 26 Mar 2024

Omidiora EO, Adeyanju IA, Fenwa ODIn: Comparison of machine learning classifiers for recognition ... https://core.ac.uk/download/pdf/234644708.pdf. Accessed 27 Mar 2024