IST707 Applied Machine Learning

School of Information Studies

Syracuse University

Hendi Kushta

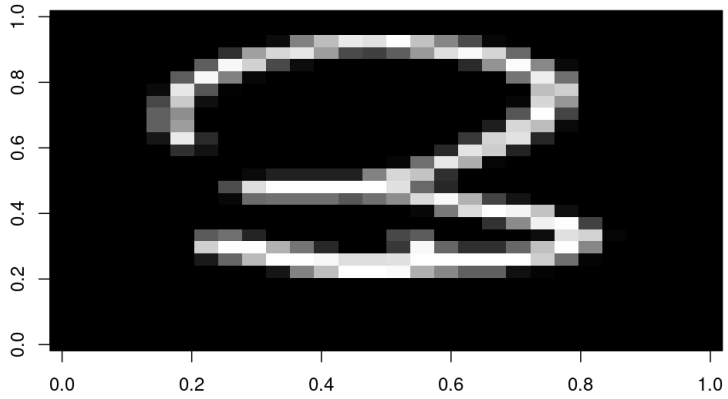
HW6: Naive Bayes and Decision Tree for Digit Recognition

Task description: The data set comes from the **Kaggle Digit Recognizer** competition. The goal is to recognize digits 0 to 9 in handwriting images. Because the original data set is too large, I have systematically sampled 5% each of the original training and testing data, stored in files called *digit-train.csv* and *digit-test.csv,* respectively. You are going to use the training data to construct prediction models using both naïve Bayes and decision tree algorithms. Tune their parameters to get the best model (measured by cross validation) and compare which algorithms provide better performance on the testing data.

Section 1: Introduction

Briefly describe the classification problem and general data preprocessing. Note that some data preprocessing steps maybe specific to a particular algorithm. Report those steps under each algorithm section.

The MNIST dataset comprises images of handwritten digits, where each image is a 28x28 pixel box, and each pixel contains a value representing the digit. The training dataset has 4198 rows and 785 columns, while the testing dataset has the same number of rows and columns.



To build accurate prediction models, it's important to preprocess the data. Preprocessing steps may include cleaning the data, selecting relevant features, scaling the pixel values, reducing dimensionality, and augmenting the data through techniques like image rotation or scaling.

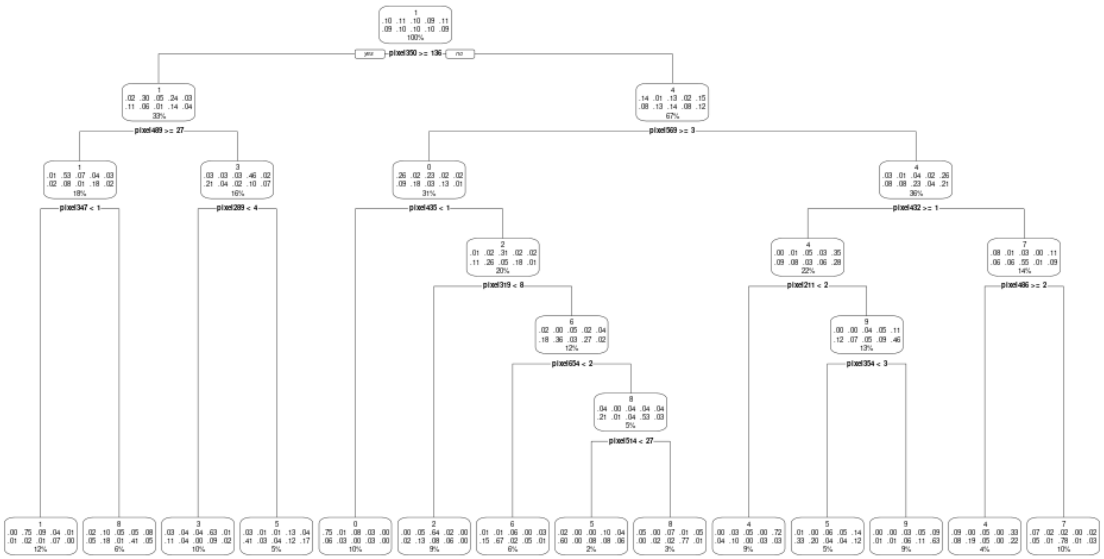
For example, to clean the data, we may check for missing values or errors. There are no missing values or duplicated values in our data. For feature selection, we may identify which pixels are most relevant for classification. To scale the pixel values, we may ensure they are within a similar range.

Before building decision tree or naive bayes models, I have split the data in training and testing. 60% training and 40% testing, since it takes lots of time to process.

Section 2: Decision Tree

Build a Decision Tree model. Tune the parameters (describe what you did), then report the 5-fold CV accuracy.

The second step is to build a simple decision tree. First, the rpart package is loaded to access the decision tree model building functions.



In order to improve the performance of a decision tree algorithm, it is possible to tune several parameters, such as the minimum number of leaf nodes, the maximum depth of the tree, and the type of cross-validation algorithm used (such as k-fold, stratified, or leave-one-out).

To find the optimal values for the maximum depth and minimum samples per leaf, we performed several iterations, testing different values for both parameters. The range of values we tested for minimum samples per leaf was 2, 3, 4, 5, 6, 7, 8, 9, and 10, while the range for maximum depth was 2, 3, 4, 5, 6, 7, 8, and 9.

After running multiple iterations, we achieved the best accuracy score of 78% using k-fold cross-validation with a maximum depth of 9 and a minimum samples per leaf of 4.

The stratified k-fold cross-validation is a method that samples the data based on the characteristics of the dataset, ensuring that each sample is selected in the same proportion as they appear in the original population. In our case, the dataset is divided into 10 strata based on the digits from 0-9.
Using this technique with the Decision Tree model, we achieved an average accuracy of 80%.

The cross-validation scores for each fold were: [0.78452381 0.77261905 0.81547619 0.7845238, 0.80357143 0.79880952 0.8045292 0.8045292 0.80095352 0.79976162], and the execution time was 7.13 ms.

On the other hand, shuffle split cross-validation randomly splits the dataset into training and testing sets, with the size of each set specified by the user. The algorithm then repeats this process for a specified number of times, and the results are averaged. For our dataset, we achieved an average cross-validation score of 0.77 using this method.
The cross-validation scores for each split were: [0.76060029 0.76703192 0.78156265 0.77084326, 0.76417342 0.76584088 0.75059552 0.76512625 0.77536922 0.75488328], and the execution time was 4.18 ms.

Section 3: Naïve Bayes

Build a Naïve Bayes model. Tune the parameters, such as the smoothing alpha (describe what you did), then report the 5-fold CV.

A Naïve Bayes model was built and parameters were tuned, specifically the smoothing alpha, to compare the results. The Naïve Bayes algorithm has three types: Gaussian, Bernoulli, and Multinomial. The Gaussian model is suitable for classification with features that follow a normal distribution, while the Bernoulli model works for binary feature vectors. The Multinomial model is preferred for data that follows a multinomial distribution. For the given dataset, the selected sub-models were Gaussian and Multinomial. Both models assume that the attributes are independent and normally distributed, and in the case of Multinomial, all values are categorical.

To execute the Gaussian algorithm, the implementation from the sklearn.naive_bayes package was used. The results from the execution without smoothing showed:

Classification Report

	precision	recall	f1-score	support
0	0.54	0.93	0.68	414
1	0.68	0.93	0.78	478
2	0.78	0.35	0.49	420
3	0.67	0.26	0.37	446
4	0.74	0.10	0.17	404
5	0.46	0.11	0.18	388
6	0.66	0.93	0.77	404
7	0.89	0.28	0.43	465
8	0.30	0.44	0.35	393
9	0.35	0.92	0.50	386
accuracy			0.53	4198
macro avg	0.61	0.53	0.47	4198
weighted avg	0.62	0.53	0.48	4198

Accuracy 53%

A Naïve Bayes model was built and parameters were tuned, specifically the smoothing alpha, to compare the results. The Naïve Bayes algorithm has three types: Gaussian, Bernoulli, and

Multinomial. The Gaussian model is suitable for classification with features that follow a normal distribution, while the Bernoulli model works for binary feature vectors. The Multinomial model is preferred for data that follows a multinomial distribution. For the given dataset, the selected sub-models were Gaussian and Multinomial. Both models assume that the attributes are independent and normally distributed, and in the case of Multinomial, all values are categorical.

To execute the Gaussian algorithm, the implementation from the sklearn.naive_bayes package was used. The results from the execution without smoothing showed:
accuracy 82%

The performance of Multinomial algorithm with different smoothing values is evaluated. The best smoothing value is selected by testing a range of values using cross-validation. The results show that there is no significant improvement in accuracy compared to the Multinomial model without smoothing. The best smoothing value obtained is 0.0035, which results in an average cross-validation score of 0.83. The test set score with the best parameters is 0.82, and the execution time is 0.35 ms.

Section 4: Algorithm performance comparison

Compare the results from the two algorithms. Which one reached higher accuracy? Which one runs faster? Provide an explanation for why one runs faster than the other.

To answer the questions regarding the performance of different machine learning algorithms, we have collected the results of accuracy and execution time for each algorithm. One notable observation is that the DecisionTree algorithm has a much higher execution time compared to the Naïve Bayes models. This is because DecisionTree algorithm recursively partitions the inputted data based on the 785 attributes, which requires a significant amount of computation. On the other hand, Naïve Bayes only needs to calculate the probabilities for the given classes, which requires less computation.

In terms of accuracy, we have tested different variations of the Naïve Bayes algorithm. The Gaussian Naïve Bayes algorithm assumes that features follow a normal distribution, while the Bernoulli Naïve Bayes algorithm is useful if feature vectors are binary. The Multinomial Naïve Bayes algorithm is preferred for data that is multinomially distributed. Based on the characteristics of the given dataset, we have selected Gaussian and Multinomial sub-models, as they make assumptions that the attributes are normally distributed and independent.

For the Gaussian Naïve Bayes algorithm, we have used the implementation from the sklearn.naive_bayes package. By tuning the 'var_smoothing' parameter, we have found that a value of 0.06 gives the highest accuracy of 80%. The execution time for this algorithm was 0.44 ms. On the other hand, the Multinomial Naïve Bayes algorithm was also implemented using the same package. The results show that the accuracy of the Multinomial Naïve Bayes algorithm is significantly higher compared to the Gaussian model. The execution time for this algorithm was 0.18ms.

To further improve the accuracy of the Multinomial Naïve Bayes algorithm, we have also tested the smoothing values. However, no significant improvements were observed for the accuracy. The best cross-validation score achieved for this algorithm was 0.83, with a test set score of 0.82. The execution time for this model was 0.35 ms.