CS 4641 Final Project: Handwritten Digits Recognition

Project Introduction

The focus of this project is the classification of handwritten digits. The scope of this report will cover three types of classification methods:

- Random forests with bagging
- Support vector machines with a non-linear kernel
- Neural networks with 2 hidden layers

The objective is to compare the performances of each algorithm to determine which is best suited for correctly identifying handwritten digits.

Dataset Description

For this project, the MNIST database of handwritten digits will be used. The database is split into two subsets: 60,000 labeled training images and 10,000 labeled testing images. Each sample is a 28x28 gray-scale image containing a total of 784 features. Initially, each feature (pixel) was represented by an integer on the interval [0, 255]. For this project, the data was normalized so that each feature is represented by a floating point on the interval [0, 1].

As briefly stated in the introduction, the overall aim of this project is to teach a computer to recognize and classify handwritten digits using several different methods. I chose this specific dataset and supervised learning problem for two reasons. First, I am familiar with the MNIST database from experimenting with neural networks in the past. Second, I have always found image recognition to be one of the more interesting classification problems in artificial intelligence. Thus, by using the MNIST database, I can experience and learn more of this field in machine learning.

The measures of performance used to compare these supervised learning algorithms are simple. The first is the *classification accuracy*. Once a model has been trained, its ability to correctly label a provided image is a good indication of its capacity for learning the MNIST dataset. The second is the *training time* of the model. The ability to learn the MNIST dataset in a reasonable amount of time is an important indicator of applicability to this problem.

Description of Supervised Learning Algorithms

As mentioned in the introduction, this project will compare the performance of three supervised learning algorithms: random forests, support vector machines, and neural networks. The following contains a description of the important hyperparameters that will be tuned in this project to find a good model.

Random Forest Hyperparameters

- Number of trees in the random forest (*n_estimators*)
  - Decreasing it causes a decrease in potential accuracy and computation time.
  - Increasing it causes an increase in complexity and tendency for overfitting.

- Number of features to be considered by a specific tree in the random forest (*max_features*)
  - If it is too small, important features required for learning a good model could be excluded.
  - If it is too large, the diversity of each individual tree can be diminished causing a decrease in classification performance.
  - Increasing it causes an increase in complexity and computation time.

Support Vector Machine Hyperparameters

- The regularization parameter (*C*)
  - When small, the margin of the decision function is permitted to be large, despite the potential incorrect classification of some training points.
  - When large, the margin of the decision function is restricted to be small, enforcing the correct classification of all training points.
  - Increasing it causes an increase in complexity and tendency for overfitting.

- The indicator of how much influence training points far from the hyperplane have in determining the decision boundary (*gamma*)
  - When small, training points far away from the hyperplane can influence the decision boundary.
  - When large, only training points close to the hyperplane influence the decision boundary.
  - Increasing it causes an increase in complexity and tendency for overfitting.

Neural Network Hyperparameters

- The number of hidden layers and the number of neurons within each layer (*hidden_layer_sizes*)
  - An increase in hidden layers / neurons causes an increase in complexity and computation time. It is also likely, but not guaranteed, to cause an increase in accuracy.

- The L2 penalty regularization term (*alpha*)
  - Decreasing it increases the tendency for underfitting.
  - Increasing it causes an increase in complexity and tendency for overfitting.

Hyperparameter Tuning with Training Data

In order to compare the three supervised learning algorithms, we must first find a good combination of hyperparameters for each.

To understand the procedure I used to find good hyperparameter values, one must first understand two useful training related methods: *grid search* and *cross validation*.

Grid search takes a list of values for each hyperparameter and computes a model for each possible combination of hyperparameters. From this, the best hyperparameter combination can be determined.

Cross validation takes a training set and breaks it up into *k* equal and random subsets. Then a model is trained from *k-1* subsets and tested on the remaining subset. This is done *k* times, each time choosing a different subset to test on. For this project, I used *k=5* subsets.

The experiments I have performed to find good hyperparameter values for each of the supervised learning algorithms involved a combination of *grid search* and *5-fold cross validation*.

Note: Since SVM training time scales poorly with the number of samples, I chose to use 50% of the MNIST database in order to avoid extremely *expensive computation times. Due to 30,000 samples still being a solid training set, this halving in no way diminished the resulting models' performance by any significant amount.*

The following figures display the model accuracy and training time of each combination of tested hyperparameters for all three algorithms.

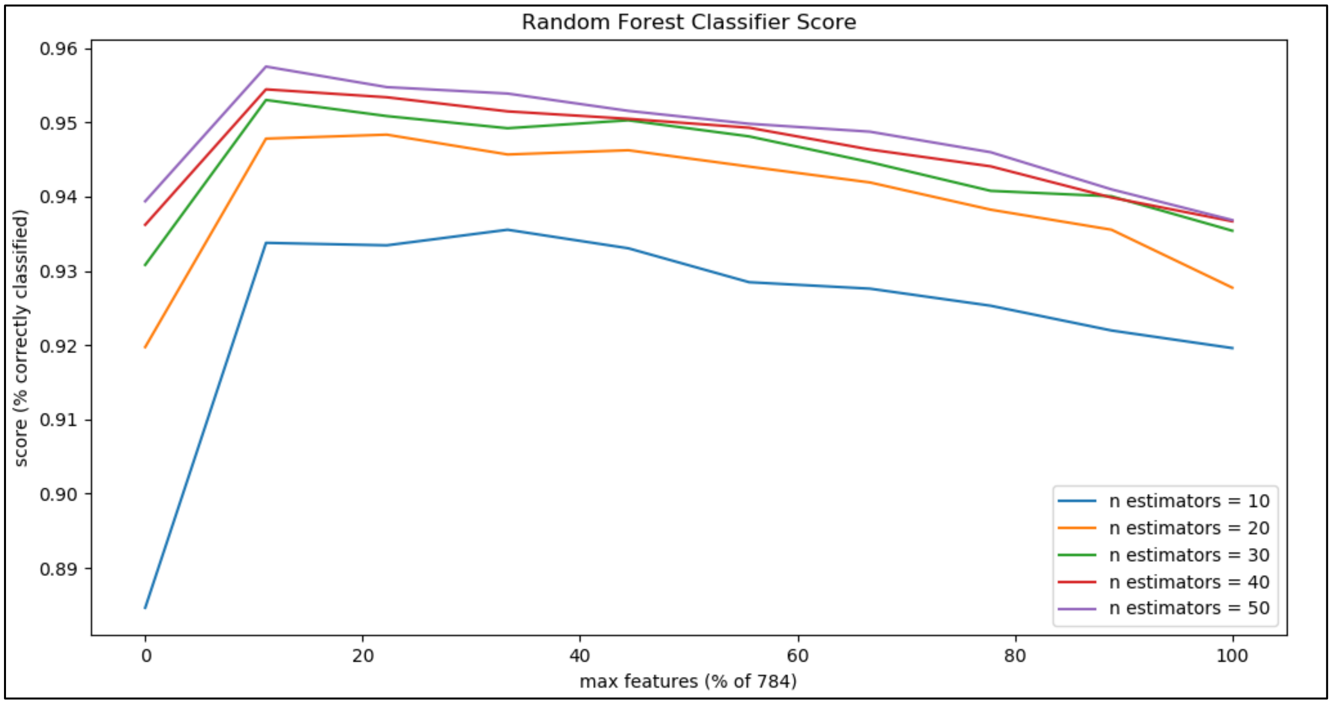

Figure 1: Random Forests - Classification Accuracy vs. Hyperparameters

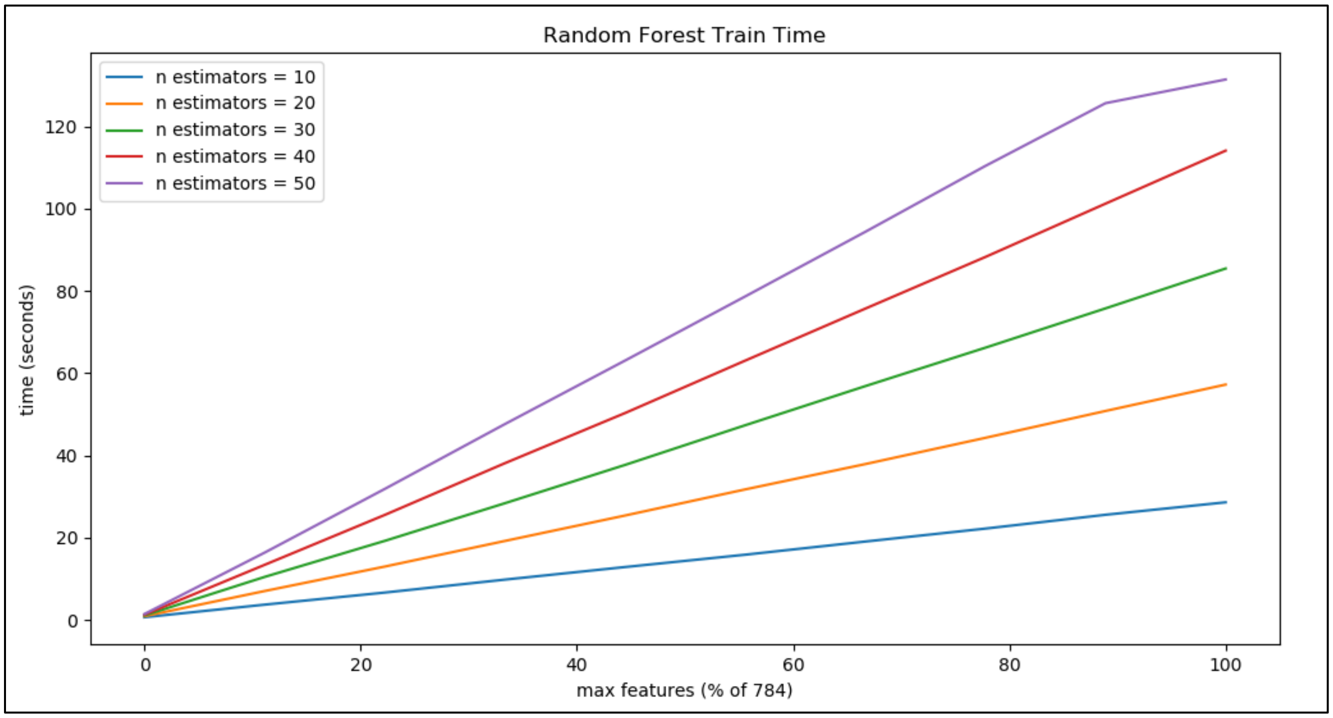

Figure 2: Random Forests - Training Time vs. Hyperparameters

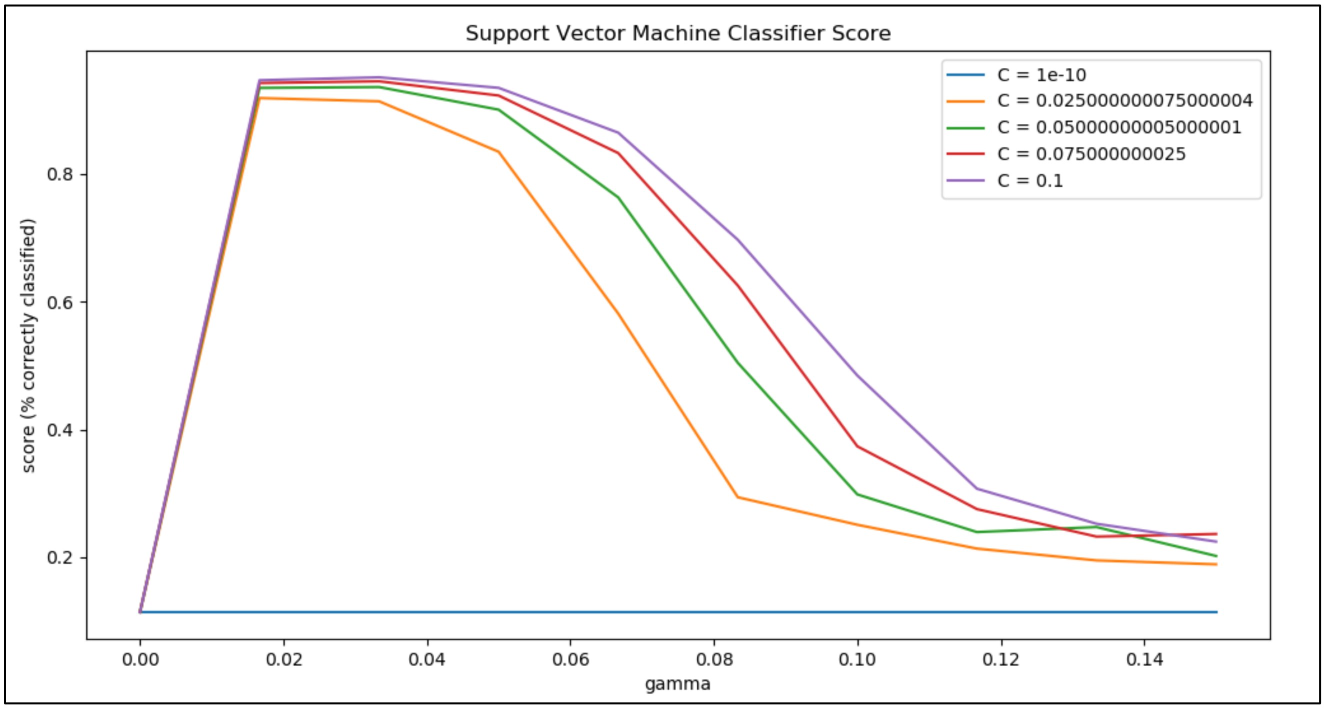

Figure 3: SVM - Classification Accuracy vs. Hyperparameters

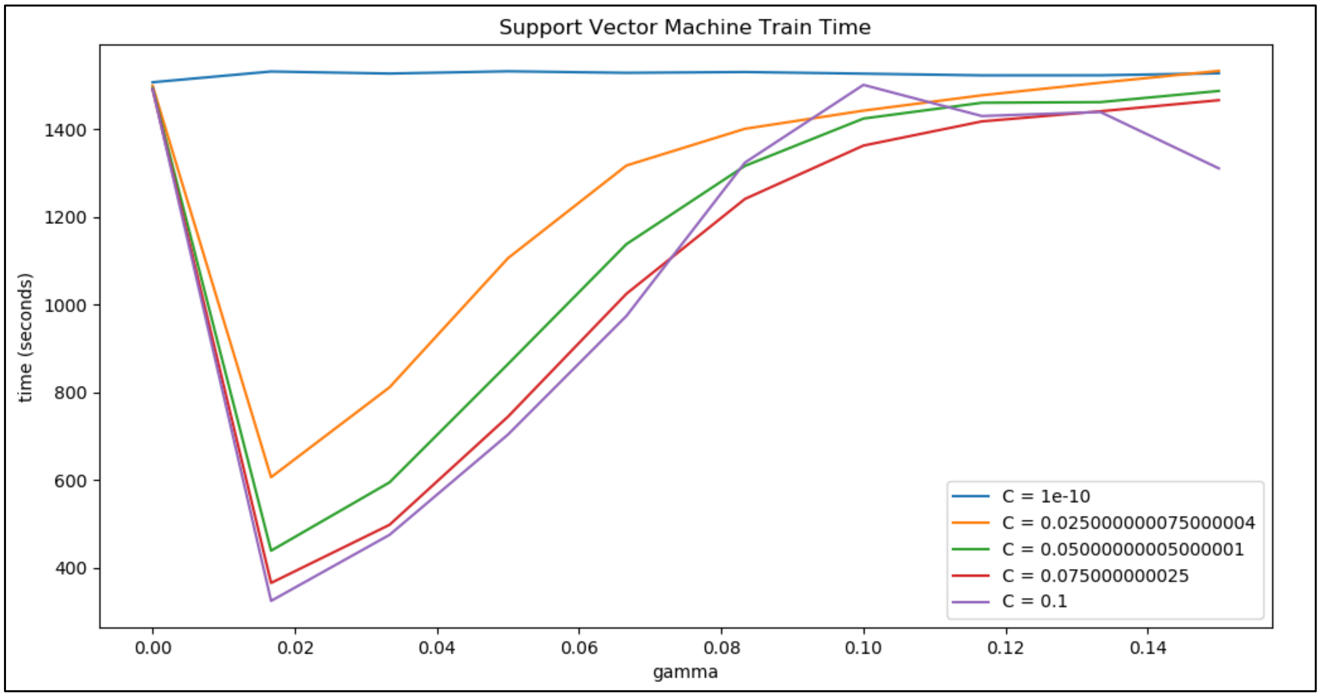

Figure 4: SVM - Training Time vs. Hyperparameters

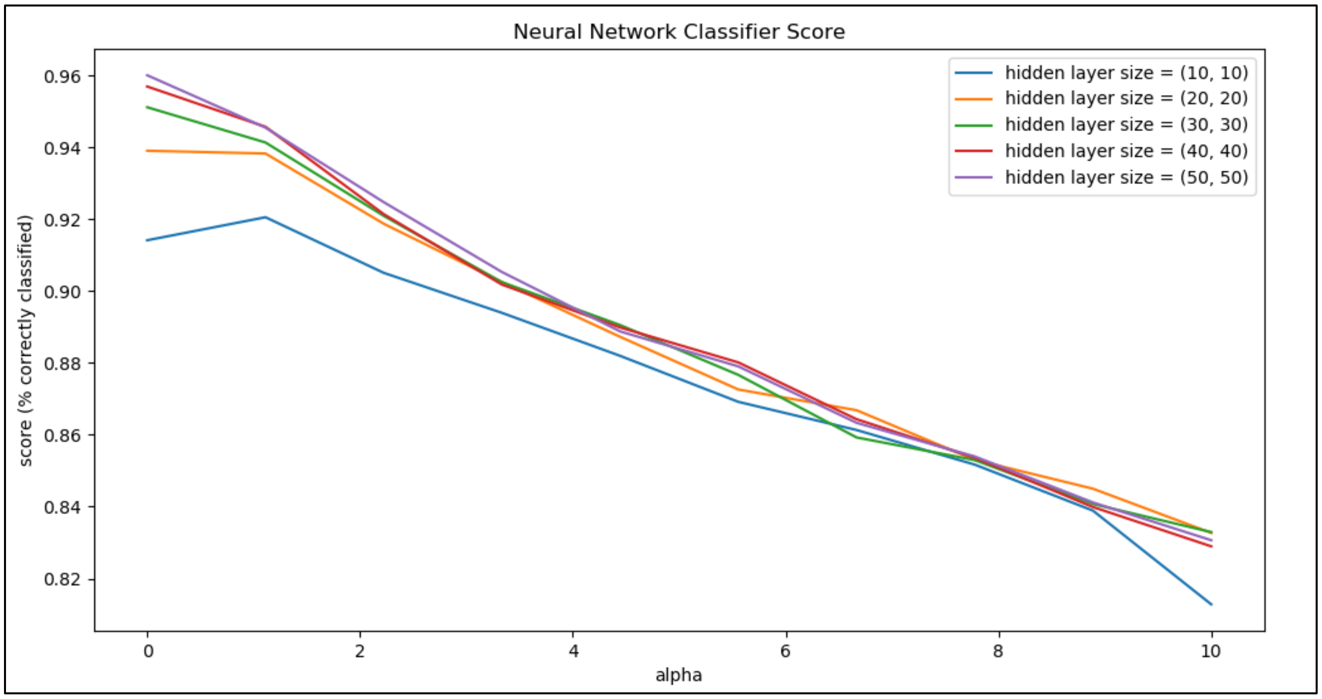

Figure 5: Neural Network - Classification Accuracy vs. Hyperparameters

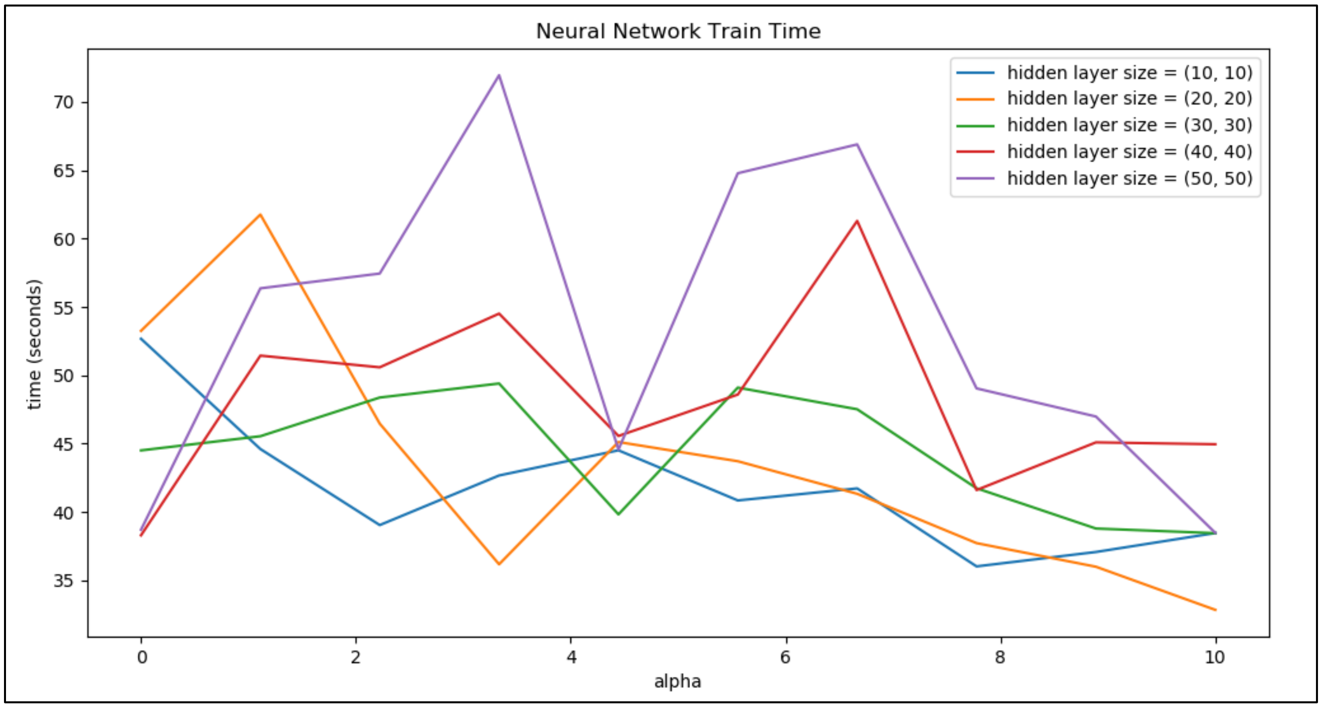

Figure 6: Neural Network - Training Time vs. Hyperparameters

Random Forests Analysis

Looking at Figure 1, there is clearly a “sweet spot” for the number of features hyperparameter. This occurs at ~10% of the 784 features in each sample. This graph also reveals that as the number of trees increases, the accuracy of the model monotonically increases. Due to the nature of the random forests, this observation makes sense and is in line with what we would expect from this algorithm. Afterall, more trees can only improve the estimate.

Figure 2 also reveals an expected result. The computation time consistently increases as both the number of trees and the number of features increases. This observation agrees with expectation due to the greater number of computations required for a higher hyperparameter.

Table 1: Random Forest - Best Hyperparameters

n_estimators	50
max_features	11.1% (87 features)
Classification Accuracy	95.75%
Training Time	16.4 seconds

The total time required to run this experiment was 1846 seconds (~30 minutes)

Support Vector Machine Analysis

Figure 3 also presents a “sweet spot” for the *gamma* hyperparameter. The SVM reaches a maximum in performance with *gamma* values of 0.2-0.4. After this, the graph indicates a significant classification performance drop off for larger values of *gamma*.

Additionally, Figure 3 reveals that increasing the *C* hyperparameter also increases the classification accuracy of the model. However, the performance gain is insignificant for values higher than ~0.05.

On that same note, in Figure 4 we see that increasing the *C* hyperparameter more importantly tends to *decrease* the training time. Thus, higher values of *C* are more advantageous for their time benefit than for their increased accuracy performance.

Table 2: SVM - Best Hyperparameters

gamma	0.03333
C	0.1
Classification Accuracy	95.14%
Training Time	474 seconds (~8 minutes)

The total time required to run this experiment was 59,548 seconds (~16.5 hours)

Neural Network Analysis

Looking at Figure 5, neural network classification accuracy favors *alpha* hyperparameter values close to 0. Most of the graph shows only a decrease in performance as *alpha* increases.

Figure 6 indicates that the training time is sporadic and tends to fluctuate with differing hyperparameter values. There is no discernable pattern as we change the number of neurons / *alpha* in each case.

Table 3: Neural Network - Best Hyperparameters

alpha	$1 * 10^{-10}$
hidden_layer_size	(50, 50) (2 hidden layers, 50 neurons)
Classification Accuracy	96%
Training Time	38.7 seconds

The total time required to run this experiment was 1951 seconds (~32 minutes)

Non-Triviality of MNIST Dataset

By taking on this project using the MNIST dataset, I claim that it is has a non-trivial distribution. In other words, I argue that the data is not linearly separable.

I justify this claim by conducting one more experiment. If the MNIST dataset is, in fact, linearly separable, a neural network with 0 hidden layers should be able to classify with ~100% accuracy.

I used 100% of the MNIST dataset to train a neural network with 0 hidden layers to discover if it could achieve near perfect accuracy.

Table 4: Neural Network with 0 Hidden Layers

alpha	$1 * 10^{-7}$
hidden_layer_size	0 hidden layers
Classification Accuracy	0.9172
Training Time	142.9 seconds

Clearly, while the classification accuracy is decent, it is far from 100%. Thus, I state the MNIST dataset is non-trivial.

Performance Comparisons with Test Data

Now that the previous tuning has presented a set of good hyperparameters for each of the supervised learning techniques, a comparison between them can be drawn using the withheld test data.

For this comparison, each algorithm (using the previously found hyperparameters) was trained using 100% (60,000 samples) of the MNIST training set. Then each model was tested on the 10,000 samples of the MNIST test set. The following are the results of these experiments stated in classification accuracy, confidence intervals, and training time.

Table 5: Performance Comparisons on Test Data

	Classification Accuracy	95% Confidence Interval	Training Time
Random Forest *n_estimators* = 50 *max_features* = 11.1%	96.86%	0.3418%	39.6 seconds
Support Vector Machine *gamma* = 0.03333 *C* = 0.1	96.61%	0.3547%	1090 seconds (~18 minutes)
Neural Network *alpha* = $1 * 10^{-10}$ *hidden_layer_size* = (50, 50)	97.51%	0.3054%	36 seconds

Looking at the classification accuracy's, all three techniques performed with near 97% accuracy. These results can be trusted to be precise as all three 95% confidence intervals are less than a half percent.

Moving to the training time, both the random forest and neural network models required similar times at around 35-40 seconds. However, the support vector machine took 18 minutes to learn the dataset. This is around 20 times longer than the other two algorithms.

All in all, the neural network performed best under the two metrics and had the smallest confidence interval. Thus, under the preliminary research that this project undergoes, the neural network is the best MNIST database classifier of the three considered. I propose this performance is do to the neural network's adaptable structure. By being allowed to define the structure of the layers and neurons within the network, the classifier can be very flexible in learning many problems.

Conclusions

The MNIST database of handwritten digits is one of the simplest instances of image recognition in machine learning. This project, however, can serve as a steppingstone to larger projects in this field. By analyzing the results of the three supervised learning algorithms on a simple problem, one can get a better idea of what will scale effectively to a harder problem such as reading license plates or facial recognition.

Based on the findings of this project, if I were to move from a dataset containing trivial 28x28 gray-scale images to more ambitious dataset containing images with thousands of colored pixels, I would focus on some form of neural network. There are several reasons that lead me to choose this algorithm over random forests and support vector machines.

First, I believe random forests are too simple to handle more complex image classification tasks. Its performance in accuracy and time on the MNIST dataset is respectable. However, only having two major hyperparameters restricts its ability to be appropriately tuned to specific problems. In many cases, this simplicity allows an easy path to *good* performance but misses out on *great* performance. For this reason, a random forest is always a good place to start, yet not the best algorithm to conduct extensive research on for difficult image classification tasks.

As for the support vector machine, its training time is its downfall. Despite its capacity for good classification performance on the MNIST dataset, the results were costly. The parameter tuning using 50% of the dataset took more than 16 hours on a ~4.0 GHz processor using all 6 cores. Had I used 100% of the dataset, I estimate a training time of around 3 days. This computation time will only increase drastically with the number of features. Consequently, unless I had access to an arsenal of servers, I would not consider an SVM to be a realistic option for more complex datasets.

Finally, the neural network exhibited the best results for the MNIST dataset in both classification accuracy and training time. I attribute this success to its flexibility of structure design. The ability to define a neural network’s hidden layers allows a high level of adaptability to a complex problem. Additionally, there are numerous options available for the design of each hidden layer (e.g. convolutional layer). This flexibility combined with realistic training times convinces me that the neural network has the most potential to tackle complex image classification problems.

Acknowledgements

This project's dataset was based on the MNIST database of handwritten digits. This can be found and downloaded at yann.lecun.com/exdb/mnist/

This project made use of the following python libraries:

- NumPy - numpy.org
- Scikit-Learn - scikit-learn.org
- MatPlotLib - matplotlib.org