MINST Kaggle Digit Recognizer: Contrasting the Random Forest and MLP Neural Network

Andrew Osborne
amo004@uark.edu

Josh Price
jdp024@uark.edu

April Walker
adw027@uark.edu

University of Arkansas,
Fayetteville, AR, 72701, USA

4/23/2019

Presentation Outline

- The MINST Dataset
- GridSearchCV
- Random Forest
  - Implementation
  - Results
- Multi-Layer Perception Classifier
  - Contrasting Gradient Descent Algorithms
  - Implementation
  - Results
- Conclusions

The MINST Dataset

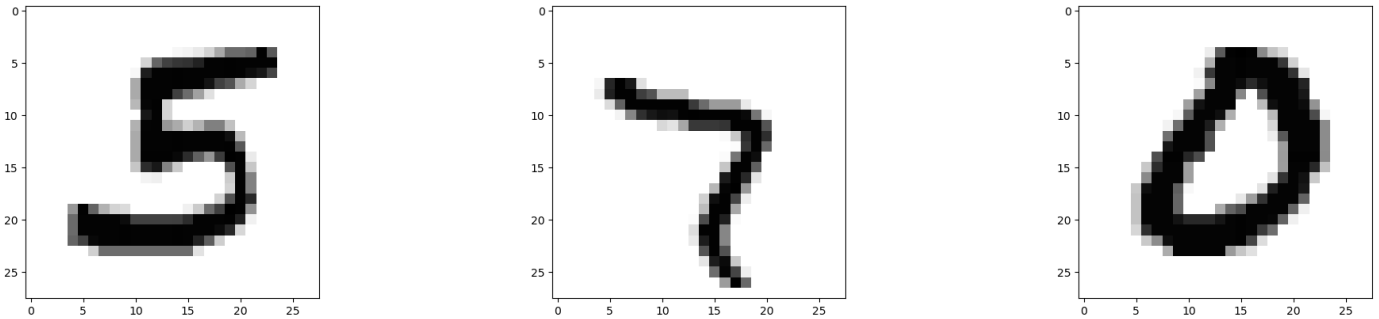


Figure: Image Renderings from the MNIST Dataset

The MINST Dataset

- 70,000 28×28 pixel grey-scale images of handwritten numbers, 0 through 9
- Each image is represented by a vector of length 784, with each element taking a value between 0 and 255 to represent lightness/darkness of the pixel
- Pre-flattened Kaggle dataset
  - 42,000 training examples
  - 28,000 testing examples

GridSearchCV

For both machine learning algorithms, training and cross-validation was done using `sklearn`'s `model_selection.GridSearchCV`

`GridSearchCV` partitions the data into k sections then trains the data on $k-1$ of the sections, leaving the last partition as a pseudo-test dataset

The mean cross-validation score (CVS) is the averaged prediction rate over all k segments of the training data, and the hyper-parameter combination with the best mean CVS is chosen

Random Forest

Our random forest is a "highly random" random forest which splits data with no discretion to promote model volatility

`Sklearn`'s `RandomForestClassifier` combines the probabilistic prediction of each tree, as this has been shown to preform better than the traditional majority vote approach

These models are generally very fast to train, but depending on the complexity, can suffer from slow classification time

High-dimensional data often poses a problem for Random Forests.

Random Forest

Implementation

In order to optimize our model's predictive power, we contrasted the results from the following hyper-parameters:

- `n_estimators` (number of trees)
- `min_samples_split` (minimum leaves to split node)

Once the code was prepared, training took approximately 45 seconds per model.

Random Forest

Implementation

In order to optimize our model's predictive power, we contrasted the results from the following hyper-parameters:

- `n_estimators` {10, 20, 100, 300}
- `min_samples_split` {2,4,8,16}

Once the code was prepared, training took approximately 45 seconds per model.

Random Forest

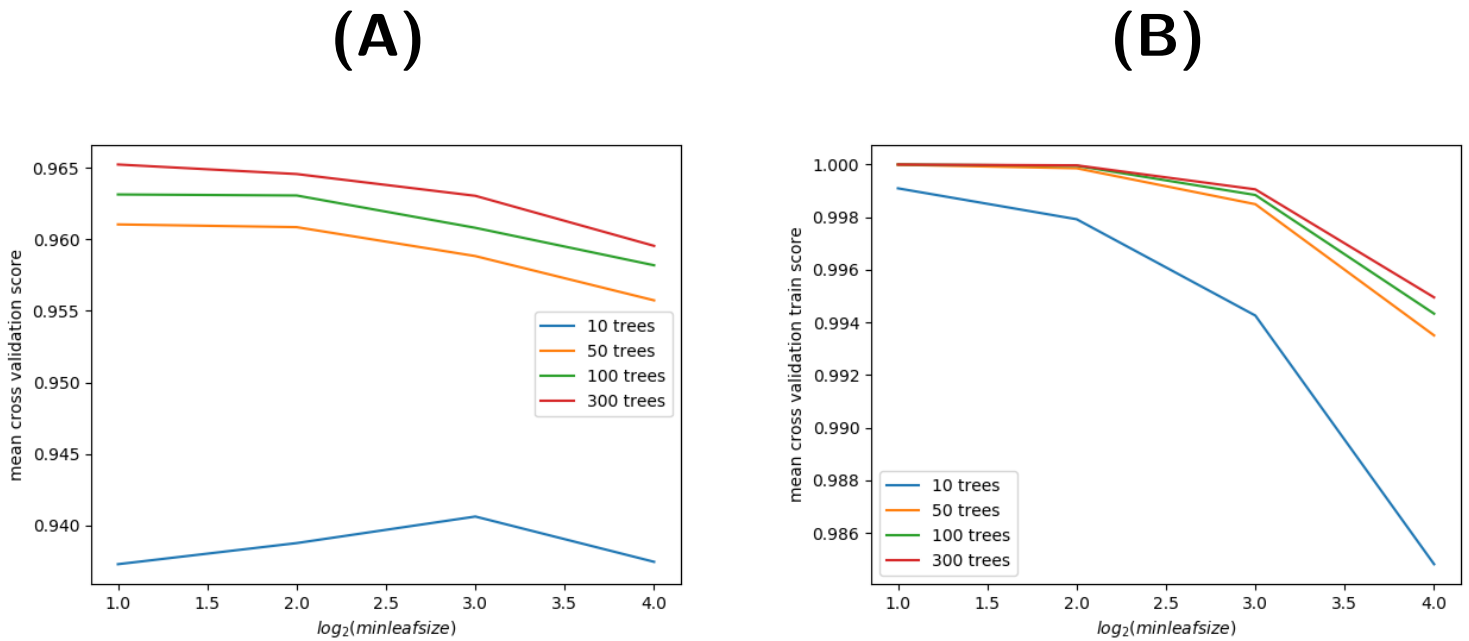


Figure: (A,B) Cross Validation of Hyper-parameters for (A) training data and (B) testing data by contrasting mean cross validation score and $\log_2(\textit{minleafsize})$ giving the minimum leaf threshold as a measure of complexity. The scores for varying numbers of trees are shown.

Random Forest

Cross-Validation and Results

The $\log_2$(minleafsize) which inversely measures the complexity suggests the more complex models performs outstandingly against the training data but has little impact on the testing data, a symptom of overfitting.

Adding trees has a similar positive affect on both datasets, however, also decreases exponentially as more trees are added. These results suggest that our model suffers from extreme levels of overfitting.

Our Kaggle submission of 300 trees and a minimum leaf number of 2 gave a prediction rate of 96.6%

Multi-Layer Perceptron Classifier

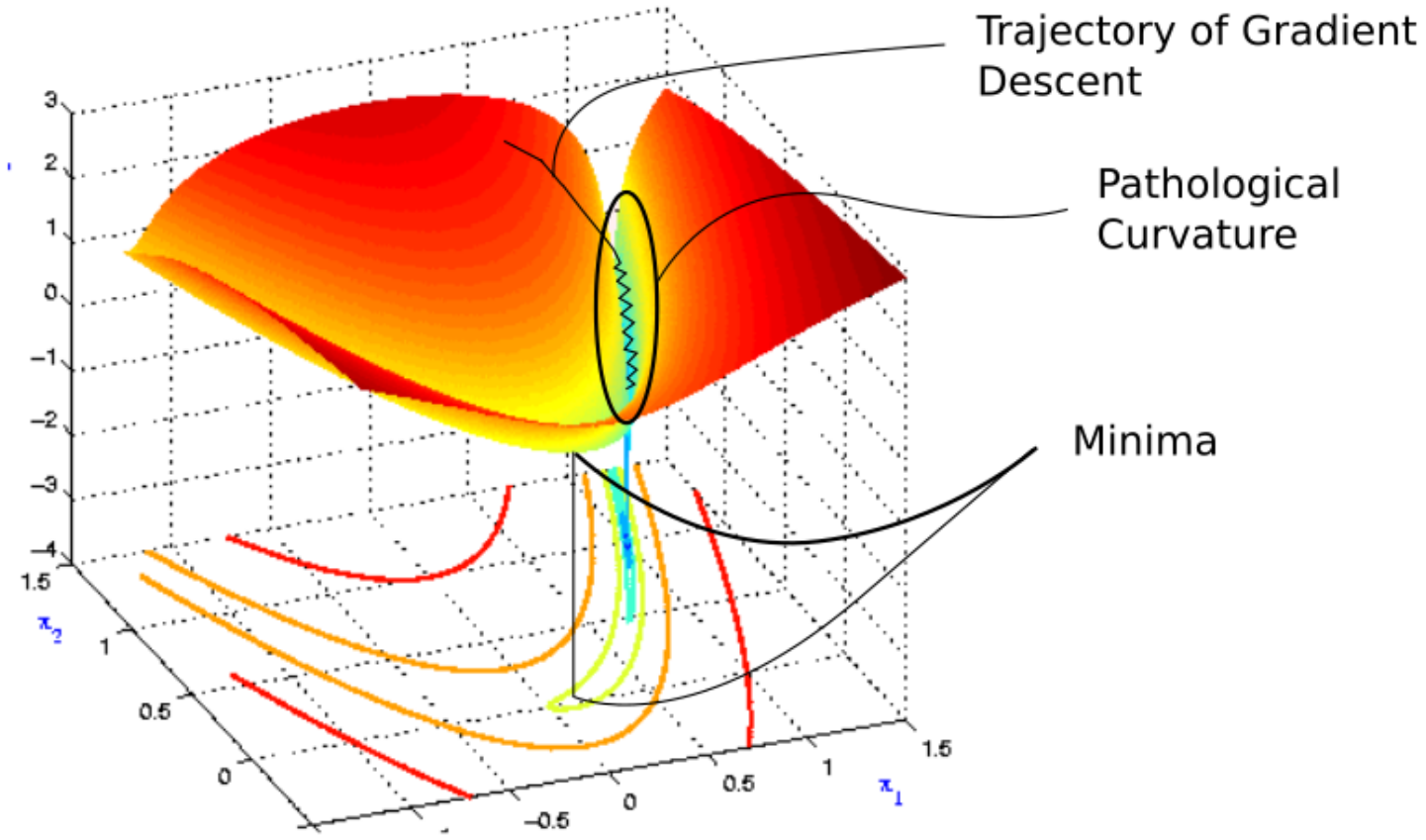
Contrasting Gradient Descent Algorithms

A Multi-layer Perceptron (MLP) is a type of deep neural network that is trained using backpropogation. An MLP consists of at least one hidden layer, an input/features layer, and an output/targets layer

For our MLP, we contrast stochastic gradient descent (SGD) and Adaptive Moment Estimation gradient descent (Adam GD)

Multi-Layer Perceptron Classifier

Optimizing SGD



Multi-Layer Perceptron Classifier

Implementation

In addition to our gradient descent algorithm, the following hyper-parameters were also varied:

- Hidden Nodes: 128, 256, and 512
- Hidden Layers: 1 and 2
- Regularization Parameter (α): 0.5, 0.1, 0.001, and 0.0001

The learning rate was not varied, instead an adaptive rate was chosen

Once the code was prepared, training took approximately 46 minutes per model. Total training time took approximately 20 hours.

Multi-Layer Perceptron Classifier

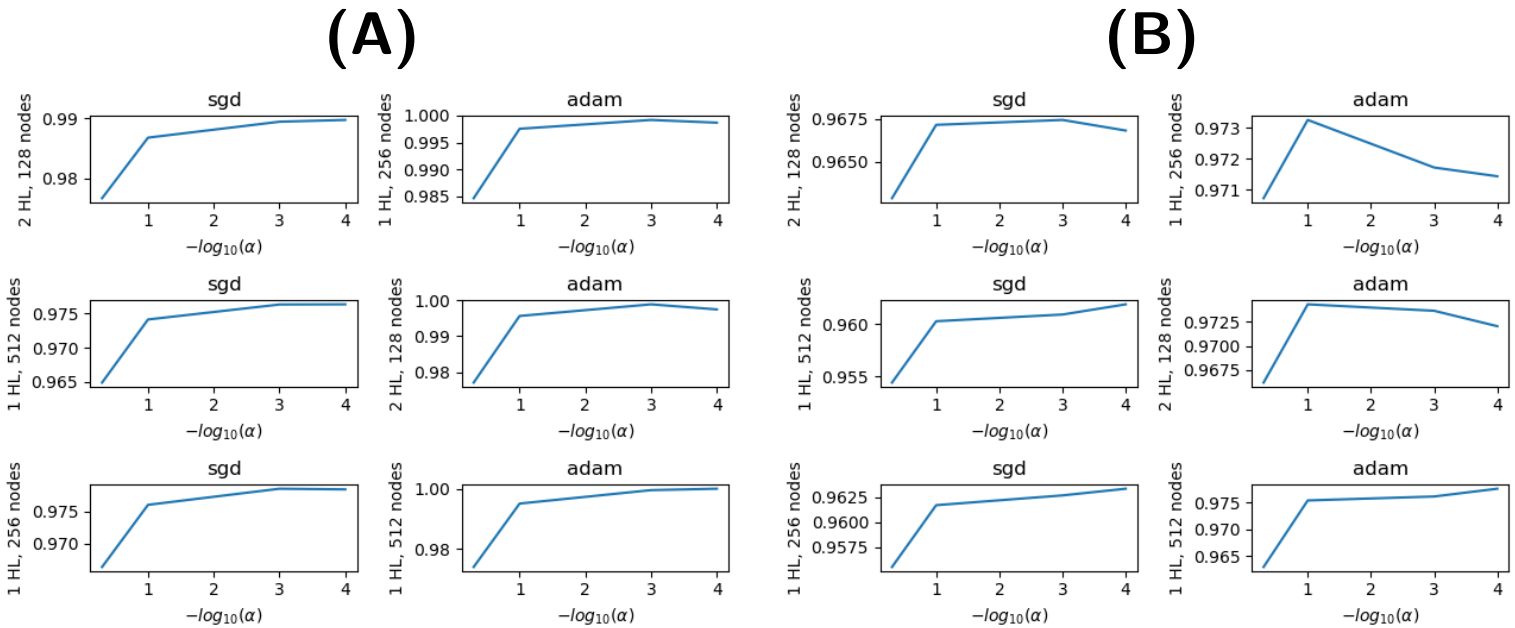


Figure: (A,B) Cross Validation of Hyper-parameters for (A) training data and (B) testing data by contrasting the mean CVS in (A) and the prediction accuracy in (B) with $-\log_{10}(\alpha)$ giving a measure of complexity. For both (A) and (B), the left colomn gives the results for SGD, and the right colmn gives the results for Adam GD.

Multi-Layer Perceptron Classifier

Cross-Validation and Results

- Training and cross-validation was done using `model_selection.GridSearchCV`.
- The measure of complexity is given by $-\log_{10}(\alpha)$, meaning lower α values correspond to higher complexity.

As expected, Adam gradient descent was the clear winner across the board.
Note:

- Smaller α values (that is the larger $-\log_{10}(\alpha)$ values) almost consistently improves the prediction rate in (A), but for (B) in some cases causes severe prediction penalties.
- The most obvious example of this is at 1 HL and 256 nodes in (B).

Our "winning" model with 1 HL and 512 nodes preformed rather consistently between training and testing datasets.

The Kaggle submission of this model gave a prediction rate of 97.90%.

Conclusions

Overall, our MLP preformed better on the MNIST dataset

Data processing to reduce dimensionality and tuning tree depth to add an additional constraint on model complexity could improve the classification accuracy of our RF.

Even with improvements to our RF, neural network architectures are still better suited for computer vision problems.

References

[1] Bernard, S., Adam, S., & Heutte, L. (2007). *Using Random Forests for Handwritten Digit Recognition*. Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) Vol 2. doi:10.1109/icdar.2007.4377074

[2] Hastie, T., Tibshirani, R., & Friedman, J. H. (2017). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. New York, NY: Springer.

[3] Kingma, D. P., & Lei Ba, J. (2015). *Adam: A Method for Stochastic Optimization*. Conference Paper at ICLR. Retrieved from https://arxiv.org/abs/1412.6980.

[4] Pedregosa *et al* (2011) *Scikit-learn: Machine Learning in Python*. JMLR 12, pp. 2825-2830.

[5] Robnik-Šikonja, M. (2004). *Improving Random Forests.* Machine Learning: ECML 2004 Proceedings, Springer, Berlin, 359-370.

[6] Y. LeCun *et al* (1995) *Learning Algorithms For Classification: A Comparison On Handwritten Digit Recognition*, in Oh, J. H. and Kwon, C. and Cho, S. (Eds), Neural Networks: The Statistical Mechanics Perspective, 261-276, World Scientific.

Thank You!