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

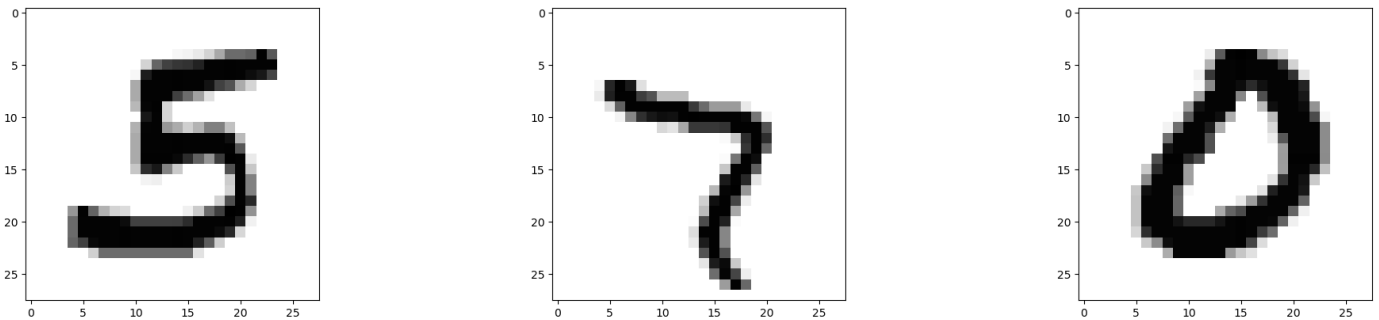

Figure: Image Renderings from the MNIST Dataset

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`.

The grid search takes in a grid of the hyper-parameters the user wishes to contrast and exhaustively considers all possible parameter combinations.

`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 [4].

Random Forest

Each tree is formed from a bootstrap sample (random sampling with replacement) then grown very similarly to a decision tree.

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 approach of picking a classification based on a majority vote [4,5].

Random Forest

These models are generally very fast to train, but depending on the complexity, can suffer from slow run-time performance.

Generally, overfitting can be negated by adding more trees to your model [2].

This approach eventually results in diminishing returns, and for high-dimensional problems such as digit recognition, is especially impractical [5].

Random Forest

Implementation
Data normalized to values in [0,1] using `sklearn`'s `preprocessing.minmax_scale`.

In order to improve our model's predictive power, we contrasted the results from our hyper-parameters `n_estimators` (number of trees) and `min_samples_split` (minimum number of leafs required to split a node).

Specifically we allowed 10, 50, 100, and 300 trees to be developed and 2, 4, 8, and 16 as the minimum number of leaves. The later can be thought of as a measure of complexity, with lower minimum resulting in higher complexity.

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

Random Forest

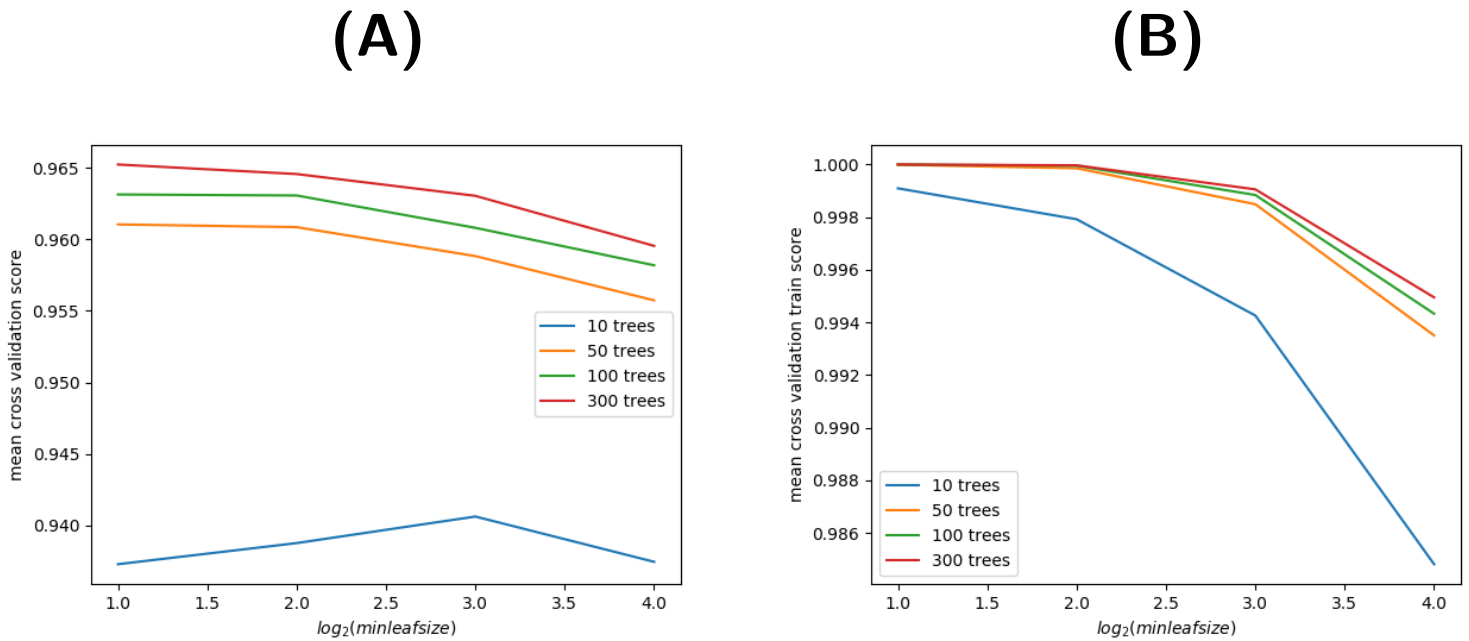

Figure: (A,B) Cross Validation of Hyper-parameters for (A) training data and (B) testing data by contrasting mean cross validation score and $\log_2(\textit{minleafsize})$ giving the minimum leaf threshold as a measure of complexity. The scores for varying numbers of trees are shown.

Random Forest

Cross-Validation and Results

The $\log_2$(minleafsize) which inversely measures the complexity suggests the more complex models preform outstandingly against the training data but have 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
For our MLP, we contrast stochastic gradient descent (SGD) and Adaptive Moment Estimation gradient descent (Adam GD), developed by Kingma and Lei Ba [3].

- SGD approximates the gradient by considering a single training example at a time.
- Adam GD is an optimized SGD which computes adaptive learning rates for each parameter.

Multi-Layer Perceptron Classifier

Adam Parameter Estimation

- Exponentially decaying averages of the past gradient $\nabla_\theta f_t(\theta_{t-1})$ and square of the gradient $\nabla_\theta f_t(\theta_{t-1})^2$ are stored as estimates of the first and second moment of the gradient (mean m_t and variance v_t respectively).
- The exponential decay rates β_1 and β_2 are initialized (generally near 1)[1] such that the moments can be more specifically calculated using the following:

$$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \nabla_\theta f_t(\theta_{t-1})$$
$$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot \nabla_\theta f_t(\theta_{t-1})^2$$

- With the initial m_0 and v_0 set to 0. In order to correct for bias, the following bias-corrected moments are then computed:

$$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$$
$$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$$

- With β_1^t and β_2^t indicating β_1 and β_2 to the power of t. Given some α and ϵ as regularization parameters, the parameters θ_t are then updated using the following:

$$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t + \epsilon})$$

- This process is repeated until θ converges [3].

[1] The developers recommended default is $\beta_1 = 0.9$ and $\beta_2 = 0.999$. [3]

Multi-Layer Perceptron Classifier

Implementation
In addition to contrasting GD algorithms we also varied:

- the number of hidden nodes between 128, 256, and 512
- the number of hidden layers either 1 or 2
- α at 0.5, 0.1, 0.001, and 0.0001

Adam GD regularization parameters were initialized at the developer's (Kingma and Lei Ba) recommended defaults:

- $\epsilon = 10^{-8}$
- $\beta_1 = 0.9$
- $\beta_2 = 0.999$ [3]

The learning rate was not varied, instead an adaptive rate was chosen which divides the current learning rate by 5 with a starting value of 0.001 [4].

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

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%.

Multi-Layer Perceptron Classifier

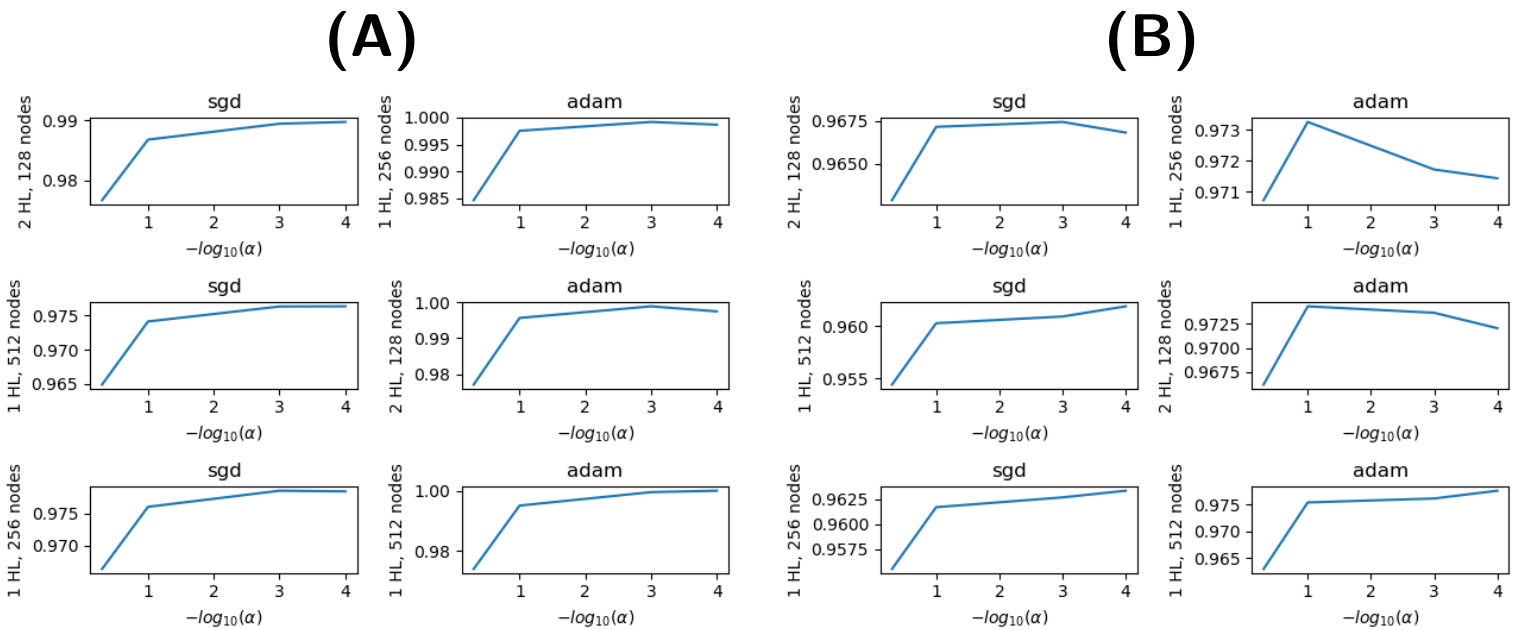

Figure: (A,B) Cross Validation of Hyper-parameters for (A) training data and (B) testing data by contrasting the mean CVS in (A) and the prediction accuracy in (B) with $-\log_{10}(\alpha)$ giving a measure of complexity. For both (A) and (B), the left colomn gives the results for SGD, and the right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

Conclusions

As expected, our MLP was better suited for handwritten digit recognition.

It is possible that data processing to reduce dimensionality would help the RF preform better, however computer vision problems are still better left in the hands of NN's and similar models.

One advantage of the RF is it's training time was approximately $1/60^{th}$ of our MLP's, however adding additional trees would begin to bridge this gap and further improve it's classification accuracy.

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!