CSE4/574: Introduction to Machine Learning (Fall 2018)
Project 3: Classification

Instructor: Sargur N. Srihari

Due Date: November 21, 2018

1 Overview

This project is to implement machine learning methods for the task of classification. You will first implement an ensemble of four classifiers for a given task. Then the results of the individual classifiers are combined to make a final decision.

The classification task will be that of recognizing a 28×28 grayscale handwritten digit image and identify it as a digit among 0, 1, 2, ... , 9. You are required to train the following four classifiers using MNIST digit images.

1. Logistic regression, which you implement yourself using backpropagtion and tune hyperparameters.
2. A publicly available multilayer perceptron neural network, train it on the MNIST digit images and tune hyperparameters.
3. A publicly available Random Forest package, train it on the MNIST digit images and tune hyperparameters.
4. A publicly available SVM package, train it on the MNIST digit images and tune hyperparameters.

2 Questions to be answered

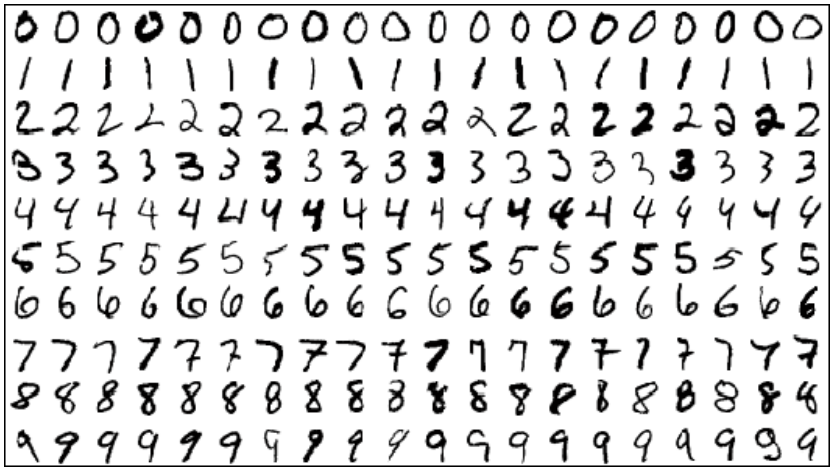
Based on the above implementations we wish to answer the following questions:

1. We test the MNIST trained models on two different test sets: the test set from MNIST and a test set from the USPS data set. Do your results support the "No Free Lunch" theorem?
2. Observe the confusion matrix of each classifier and describe the relative strengths/weaknesses of each classifier. Which classifier has the overall best performance?
3. Combine the results of the individual classifiers using a classifier combination method such as majority voting. Is the overall combined performance better than that of any individual classifier?

3 Datasets

3.1 MNIST Data

For both training and testing of our classifiers, we will use the MNIST dataset. The MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems.The database is also widely used for training and testing in the field of machine learning.



The database contains 60,000 training images and 10,000 testing images. The dataset could be downloaded from here:

http://yann.lecun.com/exdb/mnist/

The original black and white (bilevel) images from MNIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

3.2 USPS Data

We use USPS handwritten digit as another testing data for this project to test whether your models could be generalize to a new population of data. Examples of each of the digits are given below. The dataset will be available on UBLearns.

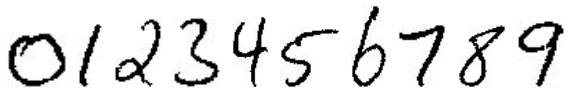


Figure 1: Examples of each of the digits

Each digit has 2000 samples available for testing. These are segmented images scanned at a resolution of 100ppi and cropped. Resize or fill the images to 28x28 like MNIST digits and feed this into your trained model and compare the result on USPS data and MNIST test data.

4 Plan of Work

1. **Extract feature values and labels from the data**: Download the MNIST dataset from the Internet and process the original data file into a Numpy array that contains the feature vectors and a Numpy array that contains the labels.
2. **Data Partition**: The MNIST dataset is originally partitioned into a training set and a testing set. You will use this partition and train your model on the training set.
3. **Train model parameter**: For a given group of hyper-parameters such as the number of layers and the number of nodes in each layer, train the model parameters on the training set.
4. **Tune hyper-parameters**: Validate the classfication performance of your model on the validation set. Change your hyper-parameters and repeat step 3. Try to find what values those hyper-parameters should take so as to give better performance on the testing set.
5. **Evaluate on testing sets**: Test the trained models on both MNIST test set and USPS data. Discuss your findings.

5 Evaluation

1. Evaluate each solution on the test set using classification accuracy,

$$Acc = \frac{N_{correct}}{N}, \tag{1}$$

where $N_{correct}$ is the number of corrected classified data samples, and N is the total number of samples of the validation set. Under the 1-of-K coding scheme, each data sample will be assigned a class label as

$$C = \text{argmax } y_i, \tag{2}$$

where $y \in \mathcal{R}^K$ is the output probability distribution over classes.

2. Construct a confusion matrix for each classifier and observe the relative strengths and weaknesses.
3. Evaluate the performance of the ensemble classifier

6 Deliverables

There are two parts in your submission:

6.1 Report

The report describes your implementations and results using graphs, tables, etc. You report should include: concise descriptions of your implementations for each of the classification models, the validation accuracy regarding different hyperparameter settings, and the best testing accuracy you can obtain for each model on two testing sets.

Your report should be edited in PDF format. Additional grading considerations will include the performance of the model, creativity in paramter tuning and the clarity and flow of your report. Highlight the innovative parts and do not include what is already in the project description. You should also include the printed out results from your code in your report.
Submission:
Submit the PDF on a CSE student server with the following script:
submit_cse474 proj3.pdf for undergraduates
submit_cse574 proj3.pdf for graduates

6.2 Code

The code for your implementations. Code in Python is the only accepted one for this project. You can submit multiple files, but the name of the entrance file should be main.py. All Python code files should be packed in a ZIP file named proj3code.zip. After extracting the ZIP file and executing command python main.py in the first level directory, it should be able to generate all the results and plots you used in your report and print them out in a clear manner.
Submission:
Submit the Python code on a CSE student server with the following script:
submit_cse474 proj3code.zip for undergraduates
submit_cse574 proj3code.zip for graduates

Appendix 1 Logistic Regression

Suppose we use 1-of-K coding scheme $\mathbf{t} = [t_1, ..., t_K]$ for our multiclass classification task. Our multiclass logistic regression model could be represented in the form,

$$p(\mathcal{C}_k|\mathbf{x}) = y_k(\mathbf{x}) = \frac{exp(a_k)}{\sum_j exp(a_j)}, \tag{3}$$

where the activation a_k are given by $a_k = \mathbf{w}\top\mathbf{x} + b_k$. The cross-entropy error function for multiclass classification problem in terms of a training sample $\mathbf{x}$ is,

$$E(\mathbf{x}) = -\sum_{k=1}^{K} t_k \ln y_k, \tag{4}$$

where $y_k = y_k(\mathbf{x})$. The gradient of the error function would be,

$$\bigtriangledown_{w_j} E(\mathbf{x}) = (y_j - t_j)\mathbf{x}. \tag{5}$$

You can use stochastic gradient descent which uses first order derivatives to update

$$\mathbf{w}_j^{t+1} = \mathbf{w}_j^t - \eta \bigtriangledown_{w_j} E(\mathbf{x}). \tag{6}$$

to find the optimum of the error function and find the solution for $\mathbf{w}_j$.

For logistic regression, do not use any Python libraries/toolboxes, built-in functions, or external tools/libraries that directly perform the learning or prediction. Using any external code will result in 0 points for that problem.

Appendix 2 Neural Networks

For multilayer deep neural network (DNN) and convolutional neural network (CNN), you can use packages from websites, such as tensorflow and keras. In this part, we will not go into detail of the implementation. In the project report, you need to describe how you choose the structure of the neural networks and how well the model performs in detail. For DNN, you can try to vary the number of hidden layers, hidden nodes, learning rate, etc. For CNN, you can try to change filter size, number layers, learning rate, etc.

Appendix 3 Support Vector Machine

In this part of assignment, you are asked to use the Support Vector Machine tool in sklearn.svm.SVM to perform classification on MNIST data set. The details about the tool are provided here: http://scikit-learn. org/stable/modules/generated/sklearn.svm.SVC.html. Try the following settings:

1. Using linear kernel (all other parameters are kept default).
2. Using radial basis function with value of gamma setting to 1 (all other parameters are kept default).
3. Using radial basis function with value of gamma setting to default (all other parameters are kept default).

Appendix 4 Random Forest

In this part of assignment, you are asked to use the Random Forest tool in sklearn.ensemble to perform classification on MNIST data set. The details about the tool are provided here: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. Try different number of trees in the forest.

Appendix 5 Mini-batch stochastic gradient descent

Mini-batch stochastic gradient descent is something between batch gradient descent and stochastic gradient descent. In each iteration of the mini-batch SGD, it samples a small chunk of samples $\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_m$ from the training data and uses this chunk to update the parameters $\mathbf{w}$:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \sum_{i=1}^{m} \nabla_{\mathbf{w}} E\left(\mathbf{z}_i\right)$$

The strength of mini-batch SGD compared to SGD is that the computation of $\sum_{i=1}^{m} \nabla_{\mathbf{w}} E\left(\mathbf{z}_i\right)$ can usually be performed using matrix operation and thus largely out-performs the speed of computing $\nabla_{\mathbf{w}} E\left(\mathbf{z}_i\right)$ individually and updating $\mathbf{w}$ sequentially. However, within same computing time, mini-batch SGD updates the weights much more often than batch gradient descent, which gives mini-batch SGD faster converging speed. The choice of mini-batch size m is the tradeoff of the two effects.

Instead of randomly sampling $\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_m$ from the training data each time, the normal practice is we randomly shuffle the training set $\mathbf{x}_1, ..., \mathbf{x}_N$, partition it into mini-batches of size m and feed the chunks sequentially to the mini-batch SGD. We loop over all training mini-batches until the training converges.