Project 2: Automated Part Sorting

Introduction to Artificial Intelligence, Fall 2012
Prof. Zanibbi

Due Date: Thursday Oct. 25th, 2012 11:59pm

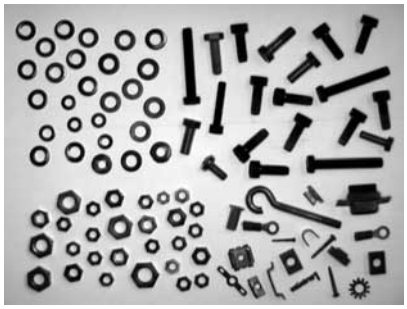
Submission Instructions

Through MyCourses, submit a .zip archive containing 1) your program files as .py files, 2) additional data files needed to run your (trained) neural networks, 3) a brief README file explaining how to run your program and interpret the output, and 4) a .pdf file containing the project write-up.

Problem Description

You have been assigned the task of sorting used metal in a machine shop for the purposes of recycling the metal (for profit!). You have been asked to sort recycled metal parts into four categories: nuts, bolts, rings, and scrap. The company that purchases the metal after it has been sorted provides the following table, identifying the profit or cost for each correctly/incorrectly identified part once they have received them. **Profits are positive-valued, and costs are negative valued.**

Profit function (in cents)				
	Correct Class			
Assigned Class	bolt	nut	ring	scrap
bolt	0.20	-0.07	-0.07	-0.07
nut	-0.07	0.15	-0.07	-0.07
ring	-0.07	-0.07	0.05	-0.07
scrap	-0.03	-0.03	-0.03	-0.03



After a week of doing this task by hand, your employer takes pity on you, and purchases a conveyor belt with a computer vision system, a sorting machine, and a computer on which to implement a program identifying the type of each piece of metal. Given the type, the sorter can automatically place the assigned class (e.g. bolt) in the corresponding bin. **You have been asked to maximize profit using your system.**

The vision system provides two *features* (attributes) from which your system needs to classify the type of each piece of metal to be recycled. They are:

1. **Six-fold rotational symmetry:** a measure of the average symmetry in the shape of the object, obtained by folding the image in half six times, and averaging differences in the two sides of the fold. This helps to discriminate rings and nuts.
2. **Eccentricity:** a measure of variation in the distance to the center of the object as we trace the object's outer contour (roughly, a measure of how 'un-circular' an object is). This helps discriminate bolts from nuts and rings.

Your employer has provided some training data from which you are to train different neural networks for this task (see the file `nuts_bolts.csv` provided through MyCourses). In that file, 94 samples are provided one per row, with the first column providing rotational symmetry, the second eccentricity, and the third column contains the correct class (1 - bolt, 2 - nut, 3 - ring, 4 - scrap).

Code

You need to implement **two programs**, `train.py` and `execute.py`:

1. `train.py` takes a file containing training data as input (specifically, `nuts_bolts.csv`), and produces as output:
   (a) A file containing the trained neural network (for use with `execute.py`).[1]
   (b) A file containing the *learning curve*. The learning curve represents the total sum of squared error (SSE) over all training samples after each *epoch* (i.e. after each complete pass over the training data). This can be in any simple format (e.g. a text file with the total SSE for each epoch on a separate line) - note that you will need to be able to plot this curve.
2. `execute.py` takes one or more files defining trained networks, that you can then run feed-forward for classification. When run using a neural net defined in a file and the `nuts_bolts.csv` data, the `execute.py` program should return:
   (a) On standard output, a *confusion matrix* that shows how many objects of each type are correctly classified, and counts for classification errors made (the confusion matrix is structured similarly to the profit function above)
   (b) On standard output, the total profit (or loss) produced by the classifier for the training data.
   (c) A file containing the output layer values produced (i.e. the class 'confidence' values) and chosen class for each input.

Neural Networks

Use these programs to train and execute three different multi-layer perceptrons using the `nuts_bolts.csv` data file, as listed below. **Note that all 'neurons' should be logistic regressors, using the logistic function (sigmoid) as the activation function:**

1. A multi-layer perception with one input layer (2 features + a bias), two hidden nodes, and four output nodes (with each output node representing the 'confidence' for each class).
2. The same architecture as in the previous network, but with five hidden nodes.
3. The same three layer architecture, but with ten hidden nodes.

Note: Recall that network weights should initially be set at random. It is suggested that you use initial values in either the interval [0,1] or [-1,1].

Notes:

- During training, update weights after each training sample is examined.
- You will need to use a stopping criterion for the training algorithm; training stops when either: 1) the weights converge (i.e. don't change after a complete pass over the data, known as an 'epoch'), or 2) after a fixed number of passes over the training data set (i.e. a fixed number of epochs).
- **Plotting:** you will need to plot learning curves and the class regions for the write-up. You may use any tool to do this, such as gnu plot, matlab or matplotlib, a Python library: http://matplotlib.org/users/index.html - available on the CS system using:
  ```
  from pylab import *
  ```
  – **For plotting the class regions**, given a trained classifier, show the regions in the feature space (i.e. 'attribute space') where each class will be chosen. To do this, you can sample locations in the feature space at equally spaced points using a grid (the

1. It is strongly recommended that you use the Python 'pickle' module, which allows you to save and recover object states from files easily.

feature space is bounded within (0,0) through (1,1)), and then indicate which class is selected at each point, using colors or symbols. You can use the output file from `execute.py` to produce the data needed for the class region plots.

Write-Up

1. **Network Design:** Provide a diagram (one diagram!) illustrating the architecture of your neural networks. Comment on the sets of functions that the different architectures are capable of representing. Which architecture did you expect to be most successful, and why?
2. **Results:** Provide the following.
   (a) A plot of the learning curves for each network showing the total sum of squared error over the training samples vs. training epoch. Do this using one plot, for comparison.
   (b) For each network, the confusion matrices produced for the training data, *after* training is complete, and the **profit** obtained by each network.
   (c) A plot of the training data set in feature space, such that samples from each class can be identified.
   (d) A plot of the class regions produced by **each classifier.**
3. **Discussion:** Comment on the results. Which network performed best, and did this match your expectation? How do the classification regions and boundaries differ between the classifiers? What factors contributed to the results that you observed? **Also provide** any additional comments that you have about the design, results or process you went through to construct the networks that is informative or interesting.

Grading

- 40% Correctness of implementation
- 10% Accuracy of best network
- 10% Coding style (use a research programming style as described under the "Resources" link for the course web page)
- 40% Write-up

Acknowledgement

This problem and the images are borrowed from the introductory pattern recognition textbook, "Classification, Parameter Estimation and State Estimation" by van der Heijden, Duin, Ridder and Tax (Wiley, 2004).