Project 2: The Metal Part SORT-R

Foundations of Intelligent Systems
Prof. Zanibbi
Due Date: Friday Dec. 8th, 2017 (11:59pm)

Submission Instructions

Through the dropbox in MyCourses, submit:

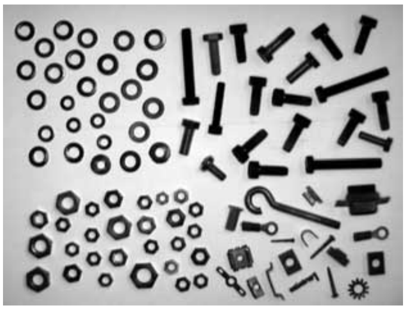
1. Your project write-up as a .pdf file, and
2. A .zip archive containing:
   - Source code (.py) files (with main program files `trainMLP.py`, `executeMLP.py`, `trainDT.py` and `executeDT.py`.)
   - Data files needed to run the trained classifiers described in your report
   - A README file explaining how to run your programs and interpret their output

---

Problem Description

You own a start-up company that sorts and then sells used metal parts. As one of your first contracts, RIT has asked you to do this for parts taken from an old printing press. You need to sort the parts into **four categories: 1. bolts, 2. nuts, 3. rings, and 4. scrap.** After price negotiations with the school and some research of your own, you produce the table below defining the profit or cost in cents ($0.01) obtained after sorting each part, based on the actual type of the part, and the class assigned by your SORT-R system (patent pending).

Profit (in cents)				
	Actual			
Assigned	bolt	nut	ring	scrap
bolt	20	-7	-7	-7
nut	-7	15	-7	-7
ring	-7	-7	5	-7
scrap	-3	-3	-3	-3



Your system consists of a conveyer belt on which parts are carried, a camera and computer vision system that measure visual features for each part, and a sorter that places each part in a given bin. The vision system provides two measurements for each metal part:

1. **Six-fold rotational symmetry:** a measure of the average symmetry in the shape of the object, obtained by folding a black-and-white image in half in six different directions, each time counting the percentage of disagreeing pixels between 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 'uncircular' an object is). This helps discriminate bolts from nuts and rings.

Your remaining task is to use machine learning algorithms to create a classifier that accurately identifies the type of each part as it arrives on the conveyor belt.

Data: CSV format training and testing datasets are provided. Both files contain one sample per line, with rotational symmetry in the first column, eccentricity in the second column, and the correct class **(given as 1 for bolt, 2 for nut, 3 ring and 4 scrap)** in the third column.

Classifier 1: Multi-Layer Perceptron (MLP)

You will create programs to train and execute MLPs with one hidden layer.
Parameters: Network weights should initialized randomly using values in the interval [-1,1]. Use the sigmoid (logistic) function as the node activation function, and a learning rate (α) of 0.01.

1. `trainMLP.py` takes a file containing training data as input and produces as output:

   (a) **Five (5) files** containing the trained neural network weights after 0 (for initial weights), 10, 100, 1000, and 10,000 epochs.[1] **Use batch training, repeatedly going over the training samples in-order, updating weights after each training sample is run.** You will use these files to later run different network configurations on test samples using `executeMLP.py`, described below.

   (b) An image containing a plot of the *learning curve*. The learning curve represents the total sum of squared error (SSE) over all training samples after each *epoch* (i.e. one complete pass over all training samples). Use the python matplotlib library (see http://matplotlib.org/users/index.html) to produce the plots.

   On the CS computer systems, include '`from pylab import *`' at the top of your program to import matplotlib.

2. `executeMLP.py` takes a file defining a trained network and a data file, and runs the network on the data file. The program should produce:

   (a) **(standard output)** On the command line:
   i. Recognition rate (% correct)
   ii. Profit obtained
   iii. A *confusion matrix*, which is a histogram counting the number of occurrences for each combination of assigned class (rows) and actual class (columns). The main diagonal contains counts for correctly assigned samples, all remaining cells correspond to counts for different types of error.

   (b) **(class region image)** An image file containing a plot of the test samples (using shapes/color to represent classes) along with the *classification regions*. The program runs the current network over a grid of equally spaced points in the region (0,0) to (1,1) (the limits of the feature space), using a different color to represent the assigned class (i.e. classification decision by the MLP) at each point. Use matplotlib to produce this image.

Classifier 2: Decision Tree (DT)

We will use a modified form of decision tree that allows *continuous* features to be partitioned using binary splits (similar to what is used in the C4.5 algorithm). To find the best split for a continuous feature, we: 1) sort samples in increasing order by their values for one attribute, and then 2) compute the information gain at every possible split point between adjacent samples, using their midpoint as the split value s. Samples with an attribute value $\geq s$ are then put in

[1] Use a simple text file (e.g. CSV file) to store the MLP network architecture and weights, and then have a constructor in your code that can instantiate a network from a provided CSV parameter file.

the ‘right’ child of the current node, and otherwise the left child of the current node. **Note that unlike with discrete attributes, we will need to re-use features**.

We will also apply Chi-Squared pruning to the learned tree, to avoid overfitting. For the Chi-Squared test, try using a significance level of 5% or 1%.

You need to produce two programs for your decision trees.

1. `trainDT.py` takes a training data file and produces:

   (a) **Two files** representing the trained decision tree 1) after automatic induction, and 2) after subsequent pruning using the Chi-Squared test. You will later run these trees using `executeDT.py`, described below.[2]

   (b) **Two images illustrating splits in the two decision trees.** For each tree, use matplotlib to create a plot visualizing how classes are assigned to regions by the decision tree, by drawing a box around both regions created at every internal (‘split’) node in the tree. **You should also show the training samples in the plots.**

   (c) On the terminal, print a summary of the contents for each decision tree:
      i. Number of nodes, and number of leaf (decision) nodes
      ii. Maximum, minimum and average depth of root-to-leaf paths in the decision tree

2. `executeDT.py` takes a file defining a decision tree and a data set, and produces the same output as `executeMLP` for decision tree in the file.

Experiment

Use `trainNN.py` and the train_data.csv data file to **train a multi-layer perceptron with two input nodes, five hidden nodes, and four output nodes (for four classes), plus bias nodes**. Then, use `executeNN.py` to run each network obtained after different numbers of training epochs on test_data.csv (5 networks in total). Collect the performance metrics and classification region images for each classifier. Repeat this a number of times (**remembering to save your previous results!**), and then report the results for what you consider to be the best run.

Then run `trainDT.py` to create your decision trees from train_data.csv, and then use `executeDT.py` to run your decision tree before and after pruning over test_data.csv.

[2] For storing decision trees in files, use a CSV file to represent parent-child node relationships (i.e. the tree structure), and define the selected feature and split threshold value at each node.

Write-Up

1. **Algorithms:** Briefly discuss the similarities and differences between the two learning algorithms. Which did you expect to perform better in the experiment, and why?
2. **Data:** Provide a plot of the training data set. Show sample classes using colors and/or shapes. Comment on the distribution of the classes.
3. **Results:** Provide the following for the **test data** results.
   (a) MLP:
      i. Plots showing the test samples and classification regions produced by the different numbers of training epochs (0, 10, 100, 1000, and 10,000).
      ii. Learning curve (SSD vs. Epoch) for the trained MLP from 0 to 10,000 epochs.
      iii. Table showing recognition rate and profit for each number of saved epochs (0, 10, 100, 1000, and 10,000).
   (b) Decision Trees:
      i. Plots showing the test samples and classification regions produced by each of your decision tree before and after pruning.
      ii. Plots showing how feature space is split your decision tree before and after pruning.
      iii. A table providing the recognition rate and profit obtained before and after pruning, along with the tree metrics produced by `trainDT.py`.

4. **Discussion:** Comment on the results (in prose, not bullet points!), making sure to answer the following questions:
   - Which of the different classifiers performed best in terms of 1) accuracy and 2) profit? Did this meet your expectations?
   - How do the hypotheses (i.e. class boundaries) and performance metrics differ between the different version of the MLP and decision trees, and why?

   Use your result figures and tables, along with the confusion matrices support your arguments. Also provide any additional comments on the problem, classifiers, results or process that you feel are informative or interesting.

Bonus (max +5 points)

Train and test another neural network with fewer than five hidden nodes, or more than five nodes in the hidden layer, and report the results and classification region plots in your write-up. Make sure to include this in your discussion of results.

or

Use a different Chi-Squared pruning threshold to obtain a different decision tree, and include the new results and classification region plots in your write-up. Make sure to include this in your discussion of results.

Grading

- 40% Correctness of implementation
- 10% Test set accuracy for the best classifiers
- 10% Coding style (use a research programming style)
- 40% Write-up

Acknowledgement

This dataset and the part image on page 1 are taken from the textbook, “Classification, Parameter Estimation and State Estimation" by van der Heijden, Duin, Ridder and Tax (Wiley, 2004).