Project #3 (due 12/17/2018 @ 11:55pm)

Objective

The purpose of this project is to implement a decision tree-based classifiers for hand-written digit recognition. The
project consists of multiple components that involve using different representations of the dataset, implementing two
classification models, assessing the performance of each classifier with the different representations and different
stopping criteria, and analyzing some of the models that were estimated.
You can use C, C++, Java, Python or Matlab to implement the algorithms. Make sure your code can run on a
cselabs Linux machine. If your code is written in Python or Matlab, instead of a Makefile, please attach a
README file that specifies how to run your code and generate the required files. Your code should be
efficient.

The dataset

The dataset is derived from the MNIST data that you used in the first written assignment. We generated two different
feature representations for the data set, "rep1" and "rep2". "rep1" is the original MNIST dataset, "rep2" is generated by
projecting the data points to the first 20 principal components of the original training dataset. For each representation,
there is a training set and a test set; you need to train the classifiers on the training set and evaluate your model on the
corresponding test set. The first column of each .csv file is the label and the remaining columns are the features. You
can download the data from the attachment below.

Classification approaches

You need to develop an algorithm to train decision tree classifiers and use accuracy and confusion matrix to
evaluate the performance of your classifier.
For training the decision tree, you should use Gini Index to select the splitting feature and best split at each node.
You should treat each feature as a continuous feature and induce binary trees. Splitting stops when one of the
following happens:
the node is pure (i.e. the labels of the data points in that node belong to the same class),
the number of the data points in the node is less than a threshold <minfreq>,

all the features are used.
In the last two cases, you should set the label of the leaf node as the one that comes from the majority of the labels in
the node.
You need to develop three programs for training, testing, and evaluation, respectively. The command line for each
program should be:
Training
```
dtinduce <trainfile> <minfreq> <modelfile>
```
The training program will read in the training set and output the trained model.
It will take three parameters as input:
<trainfile> is the training file.
<minfreq> is the minimum number of data points in a node such that the node will keep splitting. Please
experiment with the following values: {1, 5, 10, 20}.
<modelfile> is the file where you will write the learned decision tree. It is up to you to select the format.
Testing
```
dtclassify <modelfile> <testfile> <predictions>
```
The testing program will read in the model that was previously trained by the training program and classify the test set.
It will take three parameters as input:
<modelfile> is the produced model file.
<testfile> is the test file.
<predictions> is the output file with the produced predictions. Note that, there should be two columns in the
predictions file: the first column contains the true labels of the data points and the second column contains the
corresponding predicted labels.
Evaluation
```
showconfmatrix <predictions>
```
This program takes the predictions as input, computes the confusion matrix and calculates the accuracy. The confusion
matrix and the accuracy should be printed in the standard output.
You need to perform experiments using both data representation files and all the given values of minfreq.
Make sure your code can take the whole input files path, not necessarily from your current directory.

Extra credit [25 pts]

You need to implement an algorithm to train random forest classifiers in this section. First of all, you need to write a
program to generate 100 samples from the training set by doing a sampling with replacement. The size of each sample

should be 40% of the size of the training set. Train 100 decision trees, one for each sample. Use the 100 decision trees
to classify the test set and output 100 prediction files with each name "pred%03d" (e.g. "pred001" for the prediction of
the first decision tree). Secondly, you need to write a program to read in the 100 prediction files and generate a single
prediction based on the majority. Finally, use the showconfmatrix program to evaluate the performance of the classifier.
Experiment with both data representations and given values of minfreq.
Make sure your code can take the whole input files path, not necessarily from your current directory.

Report

You should include in your report a description of the implementation of the decision tree classifier. You should also
report the accuracy results on both data representations and all minfreq values provided. Show how the accuracy
varies with different minfreq values and different data representations and explain why.
For the extra credit, you are required to write a description of the implementation of the random forest classifier and
report the accuracy results on both data representations and all minfreq values provided.

Submission

DO FOLLOW ALL OF THESE INSTRUCTIONS. FAIL TO FOLLOW ANY OF THE INSTRUCTIONS WILL LEAD TO
SCORE REDUCTION.
1. You need to submit a directory that contains your code along with the report.
2. The report should be named "report.pdf" and should be in PDF format.
3. Your code should be in five files, named "dtinduce.<extension>", "dtclassify.<extension>", "rf.<extenion>",
"rfmerge.<extension>", and "showconfmatrix.<extension>". The programs should run on the CSELabs Linux
machines.
4. The directory should contain a Makefile such that by simply doing a "make <classifier>" it will build the
corresponding classifier executables where <classifier> refers to "dt" and "rf".
5. The directory should be named as "<username>" and uploaded it as a "<username>.tar.gz" archive. The
<username> should be your x500.

Grading criteria

1. Report: 10%

2. Correctness: 80% (this includes correctness for following the above instructions)
3. Efficiency: 10%
4. Extra credit: 25%

data.tar.gz