B551 Assignment 4: Machine learning

Fall 2019
Due: Sunday December 15, 11:59PM
(You may submit up to 48 hours late for a 10% penalty.)

This assignment will give you a chance to implement machine learning techniques on realistic classification problems.

You'll work in a group of 1-3 people for this assignment. You can choose the group yourself, and please fill out the form (sent out as an Announcement) so that we can create your repository. There is no starter code for this assignment so you can get started even while waiting for your repo to be set up. You should only submit **one** copy of the assignment for your team, through GitHub, as described below. All people on the team will receive the same grade on the assignment, except in unusual circumstances; we will collect feedback about how well your team functioned in order to detect these circumstances. Please read the instructions below carefully; we cannot accept any submissions that do not follow the instructions given here. Most importantly: please **start early,** and ask questions on Piazza or in office hours.

Guidelines for this assignment

Coding requirements. For fairness and efficiency, we use a semi-automatic program to grade your submissions. As usual, we require that: 1. You must code this assignment in Python 3; 2. Make sure to include a #! line at the top of your code; 3. You should test your code on one of the SICE Linux systems; 4. Your code must obey the input and output specifications given below. 5. You may import standard Python modules for routines not related to AI, such as basic sorting algorithms and data structures, as long as they are already installed on the SICE Linux servers; and 6. Make sure to use the program file name we specify.

Coding style and documentation. We will not explicitly grade coding style, but it's important that you write your code in a way that we can easily understand it. Please use descriptive variable and function names, and use comments when needed to help us understand code that is not obvious. For each of these problems, you will face some design decisions along the way. Your primary goal is to write clear code that finds the correct solution in a reasonable amount of time. To encourage innovation, we will conduct a competition among programs to see which can solve the hardest problems in the shortest amount of time.

Report. Please put a report describing your assignment in the Readme.md file in your Github repository. For each problem, please include: (1) a description of how you formulated each problem; (2) a brief description of how your program works; (3) and discussion of any problems you faced, any assumptions, simplifications, and/or design decisions you made. These comments are especially important if your code does not work perfectly, since it is a chance to document the energy and thought you put into your solution.

Academic integrity. We take academic integrity very seriously. To maintain fairness to all students in the class and integrity of our grading system, we will prosecute any academic integrity violations that we discover. *Before beginning this assignment, make sure you are familiar with the Academic Integrity policy of the course, as stated in the Syllabus, and ask us about any doubts or questions you may have.* To briefly summarize, you may discuss the assignment with other people at a high level, e.g. discussing general strategies to solve the problem, talking about Python syntax and features, etc. You may also consult printed and/or online references, including books, tutorials, etc., but you must cite these materials (e.g. in source code comments). We expect that you'll write your own code and not copy anything from anyone else, including online resources. *However, if you do copy something (e.g., a small bit of code that you think is particularly clever), you have to make it explicitly clear which parts were copied and which parts were your own. You can do this by putting a very detailed comment in your code, marking the line above which the copying began, and the line below which the copying ended, and a reference to the source.* Any code that is not marked in this way must be your own, which you personally designed and wrote. You may not share written answers or code with any other students, nor may you possess code written by another student, either in whole or in part, regardless of format.

Image classification

In this assignment we'll study a straightforward image classification task. These days, all modern digital cameras include a sensor that detects which way the camera is being held when a photo is taken. This metadata is then included in the image file, so that image organization programs know the correct orientation — i.e., which way is "up" in the image. But for photos scanned in from film or from older digital cameras, rotating images to be in the correct orientation must typically be done by hand.

Your task in this assignment is to create a classifier that decides the correct orientation of a given image, as shown in Figure 1.

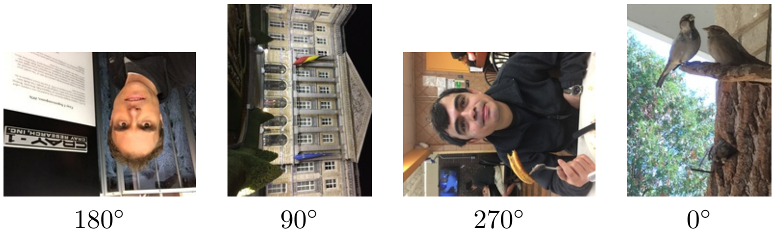
180° 90° 270° 0°

Figure 1: Some sample images, and their correct orientation labels.

Data. To help you get started, we've prepared a dataset of images from the Flickr photo sharing website. The images were taken and uploaded by real users from across the world, making this a challenging task on a very realistic dataset.[1] Since image processing is beyond the scope of this class, we don't expect you to implement any special techniques related to images in particular. Instead, we'll simply treat the raw images as numerical feature vectors, on which we can then apply standard machine learning techniques. In particular, we'll take an $n \times m \times 3$ color image (the third dimension is because color images are stored as three separate planes – red, green, and blue), and append all of the rows together to produce a single vector of size $1 \times 3mn$.

We've done this work for you already, so that you can treat the images simply as vectors and do not have to worry about them being images at all. Thorugh Canvas we've provided two ASCII text files, one for the training dataset and one for testing, that contain the feature vectors. To generate this file, we rescaled each image to a very tiny "micro-thumbnail" of 8×8 pixels, resulting in an $8 \times 8 \times 3 = 192$ dimensional feature vector. The text files have one row per image, where each row is formatted like:

```
photo_id correct_orientation r11 g11 b11 r12 g12 b12 ...
```

where:

- `photo_id` is a photo ID for the image.
- `correct_orientation` is 0, 90, 180, or 270. Note that some small percentage of these labels may be wrong because of noise; this is just a fact of life when dealing with data from real-world sources.
- `r11` refers to the red pixel value at row 1 column 1, `r12` refers to red pixel at row 1 column 2, etc., each in the range 0-255.

[1] All images in this dataset are licensed under Creative Commons, and there are no copyright issues with using them for classroom purposes under fair use guidelines. However, copyrights are still held by the photographers in most cases; you should not post these images online or use these images for any commercial purposes without checking with the original photographer, who can be identified using the Flickr URL described below.

Although you can get away with just using the above text files, you may want to inspect the original images themselves, for debugging or analysis purposes, or if you want to change something about the way we've created the feature vectors (e.g. experiment with larger or smaller "micro-thumbnails"). You can view the images in two different ways:

- You can view the original high-resolution image on Flickr.com by taking just the numeric portion of the photo_id in the file above (e.g. if the photo_id in the file is `test/123456.jpg`, just use 123456), and then visiting the following URL:

  `http://www.flickr.com/photo_zoom.gne?id=`*numeric_photo_id*

- We'll provide a zip file of all the images in JPEG format on Canvas. We've reduced the size of each image to a 75 × 75 square, which still (usually) preserves enough information to figure out the image orientation. The ZIP file also includes UNIX scripts that will convert images in the zip file to the ASCII feature vector file format above. If you want, this lets you experiment with modifying the script to produce other feature vectors (e.g. smaller sized images, or in different color mappings, etc.) and to run your classifier on your own images.

The training dataset consists of about 10,000 images, while the test set contains about 1,000. For the training set, we've rotated each image 4 times to give four times as much training data, so there are about 40,000 lines in the train.txt file (the training images on Flickr and the ZIP file are all oriented correctly already). In test.txt, each image occurs just once (rotated to a random orientation) so there are only about 1,000 lines. If you view these images on Flickr, they'll be oriented correctly, but those in the ZIP file may not be.

What to do. Your goal is to implement and test several different classifiers on this problem: k-nearest neighbors, decision trees, and neural networks. For training, your program should be run like this:

```
./orient.py train train_file.txt model_file.txt [model]
```

where `[model]` is one of `nearest`, `tree`, `nnet`, or `best`. This program uses the data in train_file.txt to produce a trained classifier of the specified type, and save the parameters in model_file.txt. You may use any file format you'd like for model_file.txt; the important thing is that your test code knows how to interpret it.

For testing, your program should be run like this:

```
./orient.py test test_file.txt model_file.txt [model]
```

where `[model]` is again one of `nearest`, `tree`, `nnet`, `best`. This program should load in the trained parameters from model_file.txt, run each test example through the model, display the classification accuracy (in terms of percentage of correctly-classified images), and output a file called output.txt which indicates the estimated label for each image in the test file. The output file should correspond to one test image per line, with the photo_id, a space, and then the estimated label, e.g.:

```
test/124567.jpg 180
test/8234732.jpg 0
```

Here are more detailed specifications for each classifier.

- `nearest`: At test time, for each image to be classified, the program should find the k "nearest" images in the training file, i.e. the ones with the closest Euclidean distance (or other distance, e.g. Manhattan) and have them vote on the correct orientation. (The "training" routine for this classifier will probably just make a copy of the training data into the model file, although you can do other processing also if you prefer.) In your writeup, test and report how accuracy on the test set varies as a function of k.

- `tree`: Uses a decision tree to estimate the orientation. You'll need to define some binary conditions to be tested at each node. As a starting point, you might use nodes that simply compare pairs of individual pixel values, e.g. that test whether the red pixel at position 1,1 is greater than or less than the green pixel at position 3,8. Your decision tree learning process can try all possible combinations (roughly 192^2) or randomly generate some pairs to try. In your report, describe how your accuracy varies as a function of the depth of the tree. A big advantage to decision trees is explainability — it's possible to understand why the classifier made a given decision. If you train a very small tree (e.g., with only 3 levels), what features does it cue on?

- `nnet`: Implement a fully-connected feed-forward network to classify image orientation, and implement the backpropagation algorithm to train the network using gradient descent.

Note that, you have to implement these classifiers from scratch. It's not allowed to use any pre-implemented packages, e.g., scikit-learn. Please check with us before using packages other than os, sys, math, matplotlib, scipy, and numpy. (Yes, we know you could get much better results by using Tensorflow or PyTorch, but that's not the point!)

Each of the above machine learning techniques has a number of parameters and design decisions. It would be impossible to try all possible combinations of all of these parameters, so identify a few parameters and conduct experiments to study their effect on the final classification accuracy. In your report, present neatly-organized tables or graphs showing classification accuracies and running times as a function of the parameters you choose. Which classifiers and which parameters would you recommend to a potential client? How does performance vary depending on the training dataset size, i.e. if you use just a fraction of the training data? Show a few sample images that were classified correctly and incorrectly. Do you see any patterns to the errors?

As in the past, a small percentage of your assignment grade will be based on how accurate your "best" algorithm is with respect to the rest of the class. We will use a separate test dataset, so make sure to avoid overfitting!

Although we expect the testing phase of your classifiers to be relatively fast, we will not evaluate the efficiency of your training program. Please commit your model files for each method to your git repo when you submit your source code, and please name them nearest_model.txt, tree_model.txt, nnet_model.txt, and best_model.txt.

What to turn in

Turn in your source code file(s) and your report by pushing to GitHub (remember to `add`, `commit`, `push`) — we'll grade whatever version you've put there as of 11:59PM on the due date. To make sure that the latest version of your work has been accepted by GitHub, you can log into the github.iu.edu website and browse the code online. Make sure to include your trained model files!